Dream SDK

Ray Dream Studio™

Extension Architecture

Reference Manual

©1995-1997 MetaCreations
All rights reserved

Version 2.0.1

Wednesday, July 02, 199 7

Copyright ©1995-97 MetaCreations, Corp. All rights reserved.

The software described in this manual is furnished under a licensing agreement printed on the inside front cover of the Ray Dream Designer User Manual. It may only be used or copied in accordance with the terms of this license. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical or otherwise, without the prior express written permission of MetaCreations, Corp.
The information in this user guide is provided for informational use only, is subject to change without notice, and should not be construed as a commitment by MetaCreations, Corp. MetaCreations, Corp. assumes no responsibility or liability for any errors or inaccuracies that may appear in this user guide.

Licensee acknowledges that the DreamSDK Development Toolkit may contain bugs, errors and other problems that could cause system failures. Consequently, the DreamSDK is provided to Licensee “AS IS,” and MetaCreations disclaims any warranty or liability obligations to Licensee of any kind. Accordingly, Licensee acknowledges that any research or development that it performs regarding the DreamSDK or any product associated with the DreamSDK is done entirely at Licensee’s own risk.

LICENSEE ACKNOWLEDGES THAT METACREATIONS MAKES NO EXPRESS, IMPLIED, OR STATUTORY WARRANTY OF ANY KIND FOR THE PRODUCT INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY WITH REGARD TO PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE.

METACREATIONS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF REVENUE, LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF INFORMATION OR DATA AND THE LIKE) ARISING OUT OF THE USE OF OR INABILITY TO USE THE PROTOTYPE EVEN IF METACREATIONS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

©1995-1997 MetaCreations. All rights reserved.

Table of Contents

Table of Contents 3

Detailed Table of Contents 7

Chapter 1 -3D Components Reference 21

1.1.I3DExtension Interface 23
1.2.I3DExDataExchanger Interface 25
1.3.I3DExCamera Interface 29
1.4.I3DExGeometricPrimitive Interface 35
1.5.I3DExVolumePrimitive Interface 43
1.6.I3DExLightsource Interface 47
1.7.I3DExLightsourceGel Interface 53
1.8.I3DExAtmosphericShader Interface 55
1.9.I3DExBackground Interface 57
1.10.I3DExBackdrop Interface 59
1.11.I3DExAmbientLight Interface 61
1.12.I3DExImportFilter Interface 63
1.13.I3DExExportFilter Interface 67
1.14.I3DExShader Interface 71
1.15.I3DExPostRenderer Interface 77
1.16.I3DExDeformer Interface 81
1.17.I3DExMotionLink Interface 85
1.18.I3DExTreeBehavior Interface 89
1.19.I3DExTweener Interface 91
1.20.I3DExSceneOperation Interface 95
1.21.I3DExSceneOperation8 Interface 97
1.22.I3DExRenderer Interface 101
1.23.I3DExGBufferRenderer Interface 103

Chapter 2 -3D Shell Reference. 105

2.1.I3DShTreeElement Interface 107
2.2.I3DShGroup Interface 121
2.3.I3DShLightsource Interface 123
2.4.I3DShCamera Interface 129
2.5.I3DShInstance Interface 135

©1995-1997 MetaCreations. All rights reserved.

2.6.I3DShObject Interface 149
2.7.I3DShPrimitive Interface 153
2.8.I3DShScene Interface 157
2.9.I3DShPolygonList Interface 169
2.10.I3DShPolygonArray Interface 175
2.11.I3DShPatchList Interface 181
2.12.I3DShPatchArray Interface 185
2.13.I3DShFlatSurface Interface 191
2.14.IShRasterBuffer Interface 195
2.15.IShRasterBufferGetPut Interface 201
2.16.IShRasterOffscreen Interface 205
2.17.IShLineArt Interface 211
2.18.IShUtilities Interface 213
2.19.I3DShEnvironment Interface 227
2.20.I3DShExtension Interface 231
2.21.I3DShMap Interface 235
2.22.IEnumChunk Interface 237
2.23.IShFileStream Interface 239
2.24.IShGraphicDevice Interface 249
2.25.IShIterator Interface 255

Chapter 3 -Data Structures Reference 257
3.1.NUM3D 259
3.2.VECTOR2D 261
3.3.VECTOR3D 263
3.4.BOX2D 265
3.5.BOX3D 267
3.6.MATRIX3D 269
3.7.TRANSFORM3D 271
3.8.TREETRANSFORM3D 273
3.9.COLOR3D 275
3.10.Basic Geometry Data Structures 277
3.11.Shading Data Structures 279
3.12.Ray Tracing Data Structures 289

Appendix I -Toolkit Libraries 295
I.i.Using the Failure Handling 296

©1995-1997 MetaCreations. All rights reserved.

Appendix II -Using the COM Dynamic Linking 297
II.i.About COM in two words 298
II.ii.Using COM for simple Extensions 299
II.iii.Using COM in Extensions using Shell Interfaces 300

Appendix III -Platforms and Compilers issues 302
III.i.Compiling for Windows 303
III.ii.Compiling for MacOS 304

Appendix IV -Managing the User Interface 306
IV.i.General Information about resources 307
IV.ii.The registration resources 308
IV.i.User Interface resources 310
IV.i.List of types used in the “PMAP” 312
IV.ii.List of Family ID 313
IV.iii.View user interface elements table 314
IV.i.The Component Private resources (“Cmpp”) 319
IV.ii.How to build a .dta file 322
IV.iii.Registering your Class ID from Fractal Design 323

Appendix V -The Ray Dream Designer® File Format . . . 325
V.i.Token Structure 326
V.ii.Base types 328
V.iii.General Structure 329
V.iv.Ray Dream Designer Extensions Support in the Format 330
V.v.3D Document file 331
V.i.Cookbook: Reading a Ray Dream Designer file. 344
V.ii.Cookbook: Writing the Minimum information in a Ray Dream Designer file 345

Index. 347

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

Detailed Table of Contents

Table of Contents 3

Detailed Table of Contents 7

Chapter 1 -3D Components Reference 21

1.1.I3DExtension Interface 23

I3DExtension::ShellUtilitiesInit 23
I3DExtension::Clone 23

1.2.I3DExDataExchanger Interface 25

I3DExDataExchanger::GetExtensionDataMap 25
I3DExDataExchanger::GetExtensionDataBuffer 25
I3DExDataExchanger::ExtensionDataChanged 26
I3DExDataExchanger::HandleEvent 26
I3DExDataExchanger::GetResID 26

1.3.I3DExCamera Interface 29

I3DExCamera::SetTransform 30
I3DExCamera::CreateRay 30
I3DExCamera::Project3DTo2D 31
I3DExCamera::GetPrimitiveID 32
I3DExCamera::Clip3D 32
I3DExCamera::ClipLine3D 33

1.4.I3DExGeometricPrimitive Interface 35

I3DExGeometricPrimitive::IsPatchBased 36
I3DExGeometricPrimitive::EnumPatches 36
I3DExGeometricPrimitive::EnumFacets 37
I3DExGeometricPrimitive::GetBBox 37
I3DExGeometricPrimitive::GetUVSpaceCount 38
I3DExGeometricPrimitive::GetUVSpace 38
I3DExGeometricPrimitive::UV2XYZ 39
I3DExGeometricPrimitive::RayHit 40
I3DExGeometricPrimitive::GetRayHitDetails 41
I3DExGeometricPrimitive::RayAllHits 41

1.5.I3DExVolumePrimitive Interface 43

I3DExVolumePrimitive::GetVolumeDensity 44
I3DExVolumePrimitive::GetVolumeAttenuation 45
I3DExVolumePrimitive::SetDeformedBoundingBox 46

1.6.I3DExLightsource Interface 47

I3DExLightsource::SetTransform 48
I3DExLightsource::GetDirection 48
I3DExLightsource::GetColor 49
I3DExLightsource::IsVisibleInPerspective 50

©1995-1997 MetaCreations. All rights reserved.

I3DExLightsource::ShadowEffect 50
I3DExLightsource::GetNewTrace 51
I3DExLightsource::GetPrimitiveID 51
I3DExLightsource::ForEachShadowBuffer 51
I3DExLightsource::GetLightParameter 52
1.7.I3DExLightsourceGel Interface 53
I3DExLightsourceGel::GetGelValues 53
1.8.I3DExAtmosphericShader Interface 55
I3DExAtmosphericShader::SegmentFilter 55
I3DExAtmosphericShader::DirectionFilter 56
1.9.I3DExBackground Interface 57
I3DExBackground::GetBackgroundColor 57
1.10.I3DExBackdrop Interface 59
I3DExBackdrop::GetBackdropColor 59
1.11.I3DExAmbientLight Interface 61
I3DExAmbientLight::GetColor 61
1.12.I3DExImportFilter Interface 63
I3DExImportFilter::Prepare 63
I3DExImportFilter::WantsOptionDialog 64
I3DExImportFilter::DoImport 64
I3DExImportFilter::WantsTopScene 65
1.13.I3DExExportFilter Interface 67
I3DExExportFilter::Prepare 67
I3DExExportFilter::WantsOptionDialog 68
I3DExExportFilter::NeededDiskSpace 68
I3DExExportFilter::DoExport 69
I3DExExportFilter::WantsPreview 69
I3DExExportFilter::GetPreviewSize 70
I3DExExportFilter::SetPreview 70
1.14.I3DExShader Interface 71
I3DExShader::IsEqualTo 72
I3DExShader::DependsOnAppliedExtent 73
I3DExShader::GetShadingFlags 73
I3DExShader::GetPreferredOutput 73
I3DExShader::DoShade 74
I3DExShader::GetValue 74
I3DExShader::GetColor 75
I3DExShader::GetVector 76
1.15.I3DExPostRenderer Interface 77
I3DExPostRenderer::GetBufferNeeds 77
I3DExPostRenderer::Filter 78
1.16.I3DExDeformer Interface 81
I3DExDeformer::SetBBox 81
I3DExDeformer::DeformPoint 82
I3DExDeformer::DeformFacets 82
I3DExDeformer::DeformPatches 83

©1995-1997 MetaCreations. All rights reserved.

1.17.I3DExMotionLink Interface 85
I3DExMotionLink::GetNbrFreedom 85
I3DExMotionLink::IncrementFreedomValue 86
I3DExMotionLink::GetFreedomRange 86
I3DExMotionLink::GetTransform 87
I3DExMotionLink::GetTransformPartialDerivate 87
1.18.I3DExTreeBehavior Interface 89
I3DExTreeBehavior::Apply 89
I3DExTreeBehavior::SetGlobalTransform 90
1.19.I3DExTweener Interface 91
I3DExTweener::DoTweening 91
I3DExTweener::DoTween 92
I3DExTweener::Draw 92
I3DExTweener::DoValue 93
I3DExTweener::DoVector2 93
I3DExTweener::DoVector3 93
I3DExTweener::DoColor 93
I3DExTweener::DoRotation 94
1.20.I3DExSceneOperation Interface 95
I3DExSceneOperation::Prepare 95
I3DExSceneOperation::DoIt 96
1.21.I3DExSceneOperation8 Interface 97
I3DExSceneOperation::Prepare8 97
I3DExSceneOperation::UndoIt 98
I3DExSceneOperation::RedoIt 99
1.22.I3DExRenderer Interface 101
I3DExRenderer::SetTreeTop 101
I3DExRenderer::SetCamera 102
I3DExRenderer::SetEnvironment 102
1.23.I3DExGBufferRenderer Interface 103
I3DExGBufferRenderer::Set32BitGBuffers 103
I3DExGBufferRenderer::Set8BitGBuffers 103

Chapter 2 -3D Shell Reference 105

2.1.I3DShTreeElement Interface 107
I3DShTreeElement::SetScene 108
I3DShTreeElement::GetScene 109
I3DShTreeElement::SetName 109
I3DShTreeElement::GetName 109
I3DShTreeElement::GetFirst 109
I3DShTreeElement::GetLast 110
I3DShTreeElement::GetLeft 110
I3DShTreeElement::GetRight 110
I3DShTreeElement::GetTop 110
I3DShTreeElement::Unlink 110

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::InsertFirst . . . 111
I3DShTreeElement::InsertLast . . . 111
I3DShTreeElement::InsertLeft . . . 111
I3DShTreeElement::InsertRight . . . 111
I3DShTreeElement::GetTransform . . . 112
I3DShTreeElement::SetTransform . . . 112
I3DShTreeElement::GetGlobalTransform . . . 112
I3DShTreeElement::SetGlobalTransform . . . 113
I3DShTreeElement::GetHotPoint . . . 113
I3DShTreeElement::SetHotPoint . . . 113
I3DShTreeElement::CenterElementOnHotPoint . . . 114
I3DShTreeElement::CenterHotPointOnElement . . . 114
I3DShTreeElement::TranslateHotPoint . . . 114
I3DShTreeElement::Clone . . . 115
I3DShTreeElement::GetBBox . . . 115
I3DShTreeElement::BeginMove . . . 116
I3DShTreeElement::EndMove . . . 116
I3DShTreeElement::GetXYZScaling . . . 116
I3DShTreeElement::SetXYZScaling . . . 116
I3DShTreeElement::GetNbrFreedom . . . 117
I3DShTreeElement::GetFreedomRange . . . 117
I3DShTreeElement::GetFreedomTransform . . . 117
I3DShTreeElement::GetFreedomDerivative . . . 118
I3DShTreeElement::IncrementFreedomValue . . . 118
I3DShTreeElement::GetFullName . . . 119
I3DShTreeElement::GetTransform8 . . . 119
I3DShTreeElement::SetTransform8 . . . 119
I3DShTreeElement::GetGlobalTransform8 . . . 120
I3DShTreeElement::SetGlobalTransform8 . . . 120
2.2.I3DShGroup Interface . . . 121
I3DShGroup::SetBBox . . . 121
I3DShGroup::DoOpenClose . . . 121
I3DShGroup::OpenClose . . . 122
I3DShGroup::IsOpen . . . 122
2.3.I3DShLightsource Interface . . . 123
I3DShLightsource::GetLightGlobalTransform . . . 123
I3DShLightsource::GetDirection . . . 124
I3DShLightsource::GetColor . . . 124
I3DShLightsource::ShadowEffect . . . 125
I3DShLightsource::ForEachShadowBuffer . . . 126
I3DShLightsource::GetLightParameter . . . 126
2.4.I3DShCamera Interface . . . 129
I3DShCamera::GetGlobalToCameraTransform . . . 130
I3DShCamera::Preset . . . 130
I3DShCamera::Project3DTo2D . . . 131
I3DShCamera::CreateRay . . . 131
I3DShCamera::Clip3D . . . 132

©1995-1997 MetaCreations. All rights reserved.

I3DShCamera::GetRenderUVBox 133
2.5.I3DShInstance Interface 135
I3DShInstance::Get3DObject 136
I3DShInstance::Set3DObject 136
I3DShInstance::GetUVSpaceCount 136
I3DShInstance::GetUVSpace 137
I3DShInstance::AddShadingShape 137
I3DShInstance::GetShadingFlags 137
I3DShInstance::DoShade 138
I3DShInstance::GetMainColors 138
I3DShInstance::GetShadingLevel 138
I3DShInstance::GetUVRasterShMap 139
I3DShInstance::GetUVExportShMap 141
I3DShInstance::FreeUVShMap 142
I3DShInstance::GetIRCache 142
I3DShInstance::SetIRCache 142
I3DShInstance::GetFlatness 142
I3DShInstance::Draw 142
I3DShInstance::DrawLA 142
I3DShInstance::GetBoxes 143
I3DShInstance::GetBB 143
I3DShInstance::Dehydrate 143
I3DShInstance::RayHit 143
I3DShInstance::GetAllInfo 144
I3DShInstance::GetDeformed3DObject 144
I3DShInstance::EnumFacetsWithConversion 144
I3DShInstance::SetShadingFromMaps 145
I3DShInstance::GetMappingKind 145
I3DShInstance::SetMappingKind 146
I3DShInstance::GetMappingDirection 146
I3DShInstance::SetMappingDirection 146
I3DShInstance::GetURL 147
I3DShInstance::GetClassSignature 147
I3DShInstance::GetInstanceKind 147
I3DShInstance::GetShadingBBox 148
I3DShInstance::IsVolumeInstance 148
2.6.I3DShObject Interface 149
I3DShObject::GetName 149
I3DShObject::SetName 150
I3DShObject::SetSimpleShading 150
I3DShObject::SetPrimerShader 151
I3DShObject::GetUVSpaceCount 151
I3DShObject::GetUVSpace 151
I3DShObject::AddShadingShape 152
I3DShObject::GetScene 152
2.7.I3DShPrimitive Interface 153
I3DShPrimitive::IsPatchBased 153

©1995-1997 MetaCreations. All rights reserved.

I3DShPrimitive::EnumPatches 153
I3DShPrimitive::EnumFacets 154
I3DShPrimitive::EnumFacetsWithConversion 155
2.8.I3DShScene Interface 157
I3DShScene::Get3DObjectsCount 158
I3DShScene::Get3DObjectByIndex 158
I3DShScene::Get3DObjectByName 158
I3DShScene::Insert3DObject 159
I3DShScene::Remove3DObject 159
I3DShScene::Changed3DObject 159
I3DShScene::GetTreeRoot 160
I3DShScene::SetTreeRoot 160
I3DShScene::CreateTreeRootIfNone 160
I3DShScene::GetTreeElementByName 161
I3DShScene::GetInstancesCount 161
I3DShScene::GetInstanceByIndex 161
I3DShScene::GetLightsourcesCount 161
I3DShScene::GetLightsourceByIndex 162
I3DShScene::GetCamerasCount 162
I3DShScene::GetCameraByIndex 162
I3DShScene::SetProductionFormat 163
I3DShScene::GetProductionFormat 163
I3DShScene::CreateRenderingCameraIfNone 164
I3DShScene::GetRenderingCamera 164
I3DShScene::SetRenderingCamera 164
I3DShScene::GetTime 165
I3DShScene::GetProductionFormatBBox 165
I3DShScene::GetFullPathName 165
I3DShScene::GetTreeElementFromFullName 166
I3DShScene::GetAmbientLight 166
I3DShScene::GetProductionFormatBBox7 166
I3DShScene::GetRenderingCamera7 167
2.9.I3DShPolygonList Interface 169
I3DShPolygonList::Init 170
I3DShPolygonList::SetUVSpace0 170
I3DShPolygonList::PreAllocateFacets 170
I3DShPolygonList::AddFacet 171
I3DShPolygonList::CountFacets 171
I3DShPolygonList::CalcNormals 171
I3DShPolygonList::GetIndFacets 172
I3DShPolygonList::RemoveFacet 172
I3DShPolygonList::SetHasUVSpace 172
I3DShPolygonList::SetUVSpace 173
I3DShPolygonList::GetUVSpace 173
2.10.I3DShPolygonArray Interface 175
I3DShPolygonArray::Init 176
I3DShPolygonArray::SetVertex 176

©1995-1997 MetaCreations. All rights reserved.

I3DShPolygonArray::GetVertex 177
I3DShPolygonArray::MeshSetU 177
I3DShPolygonArray::MeshSetV 178
I3DShPolygonArray::MeshGetU 178
I3DShPolygonArray::MeshGetV 179
I3DShPolygonArray::CalcUV 179
I3DShPolygonArray::CalcNormals 180
2.11.I3DShPatchList Interface 181
I3DShPatchList::Init 181
I3DShPatchList::SetUVSpace0 182
I3DShPatchList::SetHasUVSpace 182
I3DShPatchList::AddPatch 182
I3DShPatchList::CountPatches 183
I3DShPatchList::SetUVSpace 183
I3DShPatchList::GetUVSpace 183
2.12.I3DShPatchArray Interface 185
I3DShPatchArray::Init 186
I3DShPatchArray::SetPatch 186
I3DShPatchArray::GetPatch 187
I3DShPatchArray::MeshSetU 187
I3DShPatchArray::MeshSetV 187
I3DShPatchArray::MeshGetU 188
I3DShPatchArray::MeshGetV 188
I3DShPatchArray::CalcUV 189
2.13.I3DShFlatSurface Interface 191
I3DShFlatSurface::BeginFlatSurface 192
I3DShFlatSurface::EndFlatSurface 192
I3DShFlatSurface::BeginGroup 192
I3DShFlatSurface::EndGroup 192
I3DShFlatSurface::BeginCmpnd 193
I3DShFlatSurface::EndCmpnd 193
I3DShFlatSurface::BeginPath 193
I3DShFlatSurface::LineTo 193
I3DShFlatSurface::BezierTo 194
I3DShFlatSurface::EndPath 194
2.14.IShRasterBuffer Interface 195
IShRasterBuffer::Init 195
IShRasterBuffer::GetParameters 196
IShRasterBuffer::GetChunkInfo 196
IShRasterBuffer::EnumChunks 197
IShRasterBuffer::LockChunk 197
IShRasterBuffer::UnlockChunk 198
IShRasterBuffer::GetChunkData 198
IShRasterBuffer::GetChunkRect 198
2.15.IShRasterBufferGetPut Interface 201
IShRasterBufferGetPut::GetDepth 201
IShRasterBufferGetPut::SetBackground 202

©1995-1997 MetaCreations. All rights reserved.

IShRasterBufferGetPut::Erase . . . 202
IShRasterBufferGetPut::SetPosition . . . 202
IShRasterBufferGetPut::GoRight . . . 202
IShRasterBufferGetPut::GoLeft . . . 203
IShRasterBufferGetPut::GoDown . . . 203
IShRasterBufferGetPut::GoUp . . . 203
IShRasterBufferGetPut::Put . . . 203
IShRasterBufferGetPut::Get . . . 204
IShRasterBufferGetPut::Get32 . . . 204
IShRasterBufferGetPut::Put32 . . . 204
2.16.IShRasterOffscreen Interface . . . 205
IShRasterOffscreen::InitOffscreen . . . 206
IShRasterOffscreen::GetDepth . . . 206
IShRasterOffscreen::BitBlits . . . 206
IShRasterOffscreen::SetBackground . . . 206
IShRasterOffscreen::Erase . . . 207
IShRasterOffscreen::SetPosition . . . 207
IShRasterOffscreen::GoRight . . . 207
IShRasterOffscreen::GoLeft . . . 207
IShRasterOffscreen::GoDown . . . 208
IShRasterOffscreen::GoUp . . . 208
IShRasterOffscreen::Put . . . 208
IShRasterOffscreen::Get . . . 208
IShRasterOffscreen::Save . . . 209
IShRasterOffscreen::Load . . . 209
IShRasterOffscreen::Get32 . . . 210
IShRasterOffscreen::Put32 . . . 210
2.17.IShLineArt Interface . . . 211
2.18.IShUtilities Interface . . . 213
IShUtilities::YieldProcesses . . . 214
IShUtilities::TickCount . . . 214
IShUtilities::Random . . . 215
IShUtilities::RandomSeed . . . 215
IShUtilities::GetSerialNumber . . . 215
IShUtilities::ProgressStart . . . 215
IShUtilities::ProgressIncrement . . . 216
IShUtilities::ProgressNewValue . . . 216
IShUtilities::ProgressDone . . . 217
IShUtilities::Failure . . . 217
IShUtilities::GetGTopHandler . . . 217
IShUtilities::GetSetJmpAddress . . . 217
IShUtilities::Malloc . . . 217
IShUtilities::Calloc . . . 218
IShUtilities::Realloc . . . 218
IShUtilities::Free . . . 218
IShUtilities::CoCreateInstance . . . 219
IShUtilities::GetCOMMap . . . 219

©1995-1997 MetaCreations. All rights reserved.

IShUtilities::GetShellVersion 220
IShUtilities::GetAPINumber 220
IShUtilities::GetIndString 220
IShUtilities::CheckRegistration 221
IShUtilities::GetRegistrationName 222
IShUtilities::GetRegistrationOrganization 222
IShUtilities::GetEffectsDlogScene 222
IShUtilities::CreateKeyFrame 222
IShUtilities::ReadKeyFrame 222
IShUtilities::WriteKeyFrame 223
IShUtilities::Gestalt 223
IShUtilities::CreateTweener 223
IShUtilities::DoAlert 224
IShUtilities::SetupResource 224
IShUtilities::ResetResource 224
IShUtilities::ResetImportReport 225
IShUtilities::AddImportProblem 225
IShUtilities::ShowImportReport 225
2.19.I3DShEnvironment Interface 227
I3DShEnvironment::SegmentFilter 227
I3DShEnvironment::DirectionFilter 227
I3DShEnvironment::GetBackdropColor 228
I3DShEnvironment::GetEnvironmentColor 228
2.20.I3DShExtension Interface 231
I3DShExtension::GetParamsBuffer 231
I3DShExtension::GetResID 231
I3DShExtension::GetViewID 231
I3DShExtension::IsEqual 232
I3DShExtension::GetParameter 232
I3DShExtension::SetParameter 232
I3DShExtension::GetClassSignature 232
I3DShExtension::GetComponentClassName 233
2.21.I3DShMap Interface 235
I3DShMap::GetColor 235
I3DShMap::Invalidate 235
I3DShMap::GetTime 236
I3DShMap::SetTime 236
2.22.IEnumChunk Interface 237
IEnumChunk::Next 237
IEnumChunk::Skip 238
IEnumChunk::Reset 238
IEnumChunk::Clone 238
2.23.IShFileStream Interface 239
IShFileStream::InitFileStream 239
IShFileStream::Tell 240
IShFileStream::Seek 240
IShFileStream::Read 241

©1995-1997 MetaCreations. All rights reserved.

IShFileStream::Write 241
IShFileStream::Append 241
IShFileStream::Get 242
IShFileStream::GetString 242
IShFileStream::GetLine 243
IShFileStream::GetLong 243
IShFileStream::GetDouble 243
IShFileStream::Put 244
IShFileStream::PutString 244
IShFileStream::PutLine 244
IShFileStream::PutEOL 245
IShFileStream::PutLong 245
IShFileStream::PutDouble 245
IShFileStream::PutBack 246
IShFileStream::Good 246
IShFileStream::Fail 246
IShFileStream::Eof 246
IShFileStream::SetMacOSInfo 247
IShFileStream::GetFileSize 247

2.24.IShGraphicDevice Interface 249

IShGraphicDevice::MoveTo 249
IShGraphicDevice::LineTo 250
IShGraphicDevice::SetPen 250
IShGraphicDevice::ResetPen 251
IShGraphicDevice::DrawLine 251
IShGraphicDevice::EraseRect 252
IShGraphicDevice::InvertRect 252
IShGraphicDevice::FrameRect 252
IShGraphicDevice::FillRect 252
IShGraphicDevice::DrawString 253
IShGraphicDevice::DrawOffscreen 254

2.25.IShIterator Interface 255

IShIterator::Next 255
IShIterator::Reset 256
IShIterator::GetInfo 256

Chapter 3 -Data Structures Reference 257

3.1.NUM3D 259
3.2.VECTOR2D 261
3.3.VECTOR3D 263
3.4.BOX2D 265
3.5.BOX3D 267
3.6.MATRIX3D 269
3.7.TRANSFORM3D 271
3.8.TREETRANSFORM3D 273

©1995-1997 MetaCreations. All rights reserved.

3.9.COLOR3D 275
3.10.Basic Geometry Data Structures 277
VERTEX3D 277
FACET3D 277
PATCH3D 278
3.11.Shading Data Structures 279
ShadingIn 279
ShadingFlags 280
ShadingOut 282
UVSpaceInfo 283
ShadingInOut 285
ShadingElem 286
UVShMaps 286
MappingParams 287
3.12.Ray Tracing Data Structures 289
Ray3D 289
RayHit3D 289
RayHitParameters 291
HitInfo 292

Appendix I -Toolkit Libraries 295
I.i.Using the Failure Handling 296

Appendix II -Using the COM Dynamic Linking 297
II.i.About COM in two words 298
II.ii.Using COM for simple Extensions 299
II.iii.Using COM in Extensions using Shell Interfaces 300
Using Shell objects received as parameters 300
Instanciating a Shell object to work on it 300

Appendix III -Platforms and Compilers issues 302
III.i.Compiling for Windows 303
III.ii.Compiling for MacOS 304
Warning: the Creator must be unique ! 304
68K code: using the Far Model 304
M.68K, M.PPC, etc. 304

Appendix IV -Managing the User Interface 306
IV.i.General Information about resources 307
IV.ii.The registration resources 308
The “COMP” resource 308
COM Specific resource “GUID” 308

©1995-1997 MetaCreations. All rights reserved.

IV.i.User Interface resources 310
The “PMAP” resource 310
IV.i.List of types used in the “PMAP” 312
IV.ii.List of Family ID 313
IV.iii.View user interface elements table 314
Editing the ‘View’ resources 314
The Top View 314
Text entry box 314
Number entry box 314
Fixed Number entry box 315
Number entry box with 3D units 315
Number entry box with 2D units 315
Slider 315
Double Slider 316
Triple Slider 316
Angle Slider 316
Limiter Popup 317
Button 317
Check box 317
Radio button 317
Color button 318
Pop up 318
IV.i.The Component Private resources (“Cmpp”) 319
IV.ii.How to build a .dta file 322
IV.iii.Registering your Class ID from Fractal Design 323
How to register 323
Class ID Registration Form 324

Appendix V -The Ray Dream Designer® File Format . . . 325

V.i.Token Structure 326
V.ii.Base types 328
V.iii.General Structure 329
V.iv.Ray Dream Designer Extensions Support in the Format 330
V.v.3D Document file 331
Top Level Sections 331
Header Section 331
Render Settings Section 332
Work Space Section (optional) 333
Background Section (optional) 334
Atmospheric Section (optional) 335
Renderer Section (optional) 336
Preview Section (optional) 336
Scene Section 337
Objects List Section 337

©1995-1997 MetaCreations. All rights reserved.

Object Section 338
Scene Hierarchy Section 339
Hierarchy Element Section 339
Shading Section 341
Paint Shape List Section 342
Paint Shape Section 342
Shader Section 343
Raster Image Section 343
Animation Section 343
Counted Objects Tables Section 343
Time Manager Section 343
V.i.Cookbook: Reading a Ray Dream Designer file. 344
V.ii.Cookbook: Writing the Minimum information in a Ray Dream Designer file 345
Sample File 345

Index. 347

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

Chapter 1 - 3D Components Reference

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.1.I3DExtension Interfac e

Interface ID: IID_I3DExtension
Interface file: I3DExSht.h

The **I3DExtensio n** interface simply adds a Clone method and a generic Shell utilities initialization method to the **IUnknow n** interface. All 3D Component interfaces inherit from this interface.

```
DECLARE_INTERFACE_(I3DExtension, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
 };
```

I3DExtension: :ShellUtilitiesIni t

HRESULT I3DExtension::ShellUtilitiesInit (*shellUtilities***);**
IShUtilities* *shellUtilities*

I3DExtension::ShellUtilitiesIni t is always called by the 3D Shell just after the Component is instanciated to give to the Component access to some Shell services.

Parameter → *shellUtilities*
Pointer on the 3D Shell services object.

Comments Call InitCoFailure(*shellUtilities*). This will allow the failure handling to work between the Extension and the 3D Shell.

I3DExtension::Clone

I3DExtension* I3DExtension::Clone() ;

I3DExtension::Clon e is called when a component needs to be cloned.

Return Valu e A pointer on the cloned object through a I3DExtension interface.

Comments This method must absolutely be implemented. This routine needs to duplicate all private data you may have, and return a new component that has values identical to the former one, but is totally independent. In other words, they should not share any blocks in memory so they can be freed independently.

©1995-1997 MetaCreations. All rights reserved.

Make sure you call AddRef() on the returned extension before giving it.

©1995-1997 MetaCreations. All rights reserved.

1.2.I3DExDataExchanger Interface

Interface ID: IID_I3DExDataExchanger
Interface file: I3DExSht.h

The **I3DExDataExchanger** interface is responsible for handling the user interface of the component and communicating data between the 3D Shell and the 3D Component.

See the “Managing the User Interface” Appendix for all details about implementing the user interface of your 3D Component.

```
DECLARE_INTERFACE_(I3DExDataExchanger, I3DExtension) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE; //
      Return NULL if GetResID is to be used
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;
 };
```

I3DExDataExchanger::GetExtensionDataMap

ExtensionDataMap* I3DExDataExchanger::GetExtensionDataMap() ;

I3DExDataExchanger::GetExtensionDataMap is called when the 3D Shell needs to learn about the data the 3D Component wishes to make public. This allows the 3D Shell to take care of the user interface of the 3D Component.

Return Value Returns a pointer on a **ExtensionDataMap** block. The 3D Shell will be responsible for deleting this block. Return NULL if you want the 3D Shell to use a ‘PMAP’ resource instead.
Reserved for future use. Do not implement if at this time, Make a pMAP or PMAP resource.

I3DExDataExchanger::GetExtensionDataBuffer

void* I3DExDataExchanger::GetExtensionDataBuffer() ;

©1995-1997 MetaCreations. All rights reserved.

I3DExDataExchanger::GetExtensionDataBuffer is called when the 3D Shell needs to get the address of the Component public data. This data must correspond to the mapping provided by the routine **I3DExDataExchanger::GetExtensionDataMap**.

Return Value A pointer on the Data Buffer of the Component. This points to some internal structure inside the Component that was allocated when the Component was instanciated. The Component will free it when freeing itself.

Comments The 3D Shell reads and write data directly in the Data Buffer using the Extension Data Map as a road map to calculate the addresses and to use the right data formatting.

I3DExDataExchanger::ExtensionDataChanged

HRESULT I3DExDataExchanger::ExtensionDataChanged();

I3DExDataExchanger::ExtensionDataChanged is called after the 3D Shell writes any new data in the Data Buffer. This allows the Component to perform any postprocessing or update any internal data based on this new information.

Return value S_OK

I3DExDataExchanger::HandleEvent

HRESULT I3DExDataExchanger::HandleEvent (*sourceID***);**
ULONG *sourceID*

I3DExDataExchanger::HandleEvent is called every time an element of the user interface has been used.

Parameters → *sourceID*
The ID of the element that has been used. This ID is the one used in the 'View' resource to identify the element (button, check box, etc.)

Return value S_OK

Comments Use this call to react to button hits, and more generally to all controls that are action controls and not state controls. State controls like sliders, clusters of radio buttons, check boxes, etc. show the state of some data inside the Data Buffer. Buttons which are action trigger, do not correspond to anything in the Data Buffer because they go back to their original state after being clicked.

I3DExDataExchanger::GetResID

short I3DExDataExchanger::GetResID();

I3DExDataExchanger::GetResID is called when the 3D Shell needs to know the ID of all Component resources: 'COMP', 'PMAP', 'View', etc. Therefore, all resources of a

©1995-1997 MetaCreations. All rights reserved.

component must have the same ID.

Return Value The resource ID number.
It MUST return the same value as the resource ID, and all resources MUST have the same ID.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.3.I3DExCamera Interfac e

Family ID: ’came’
Interface ID: IID_I3DExCamera
Interface file: I3DExCam.h

Cameras define the projection from the 3D world to the 2D screen on which the image is rendered. Like with a photographic camera, one can design different type of cameras with various lens, focal value, zoom effects, etc. Projection is not restricted to the conical projection (the natural projection in your eye), so isometric or fish-eye cameras can be designed.

There are two basic calls. The first one does the projection from the 3D space onto the screen, and the other one creates a 3D ray (for the ray-tracing) from a screen point. The more complicated one is the 3D Clipping call.

```
DECLARE_INTERFACE_(I3DExCamera, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExCamera methods
  STDMETHOD(SetTransform) (THIS_ TRANSFORM3D* transform) PURE;
  STDMETHOD_(BOOLEAN, CreateRay) (THIS_ VECTOR2D* screenPosition,
      VECTOR3D* resultOrigin, VECTOR3D* resultDirection) PURE;
  STDMETHOD_(BOOLEAN, Project3DTo2D) (THIS_ VECTOR3D* position,
      VECTOR2D* resultScreenPosition, NUM3D* resultDistanceToScreen)
      PURE;
  STDMETHOD_(ULONG, GetPrimitiveID) (THIS) PURE;
  STDMETHOD_(ULONG, Clip3D) (THIS_  FACET3D* localFacet, VERTEX3D*
      localVertices, FACET3D* cameraFacet, VERTEX3D* cameraVertices,
      NUM3D* clipBox) PURE;
  STDMETHOD_(BOOLEAN, ClipLine3D) (THIS_ VECTOR3D* P1, VECTOR3D* P2)
      PURE;
 };
```

©1995-1997 MetaCreations. All rights reserved.

I3DExCamera::SetTransform

HRESULT I3DExCamera::SetTransform (*transform***);**
TRANSFORM3D* *transform***;**

I3DExCamera::SetTransfor m is called so the 3D Component can store the camera transformation.

Parameter *transform*
Pointer to the transformation data. Do not keep this pointer. Copy its contents in your component internal data instead.

Return Value S_OK

Comments It is a good idea to do all pre-processing calculations that may speed up the Ray Creation routine. This transformation converts any point from the Screen Coordinate System into the Global Coordinate System.

I3DExCamera::CreateRay

BOOLEAN I3DExCamera::CreateRay (*screenPosition, resultOrigin, resultDirection***);**
VECTOR2D* *screenPosition*
VECTOR3D* *resultOrigin*
VECTOR3D* *resultDirection*

I3DExCamera::CreateRa y is called when a ray (i.e. a straight line) from a screen position needs to be created.

Parameters *screenPosition*
Position on the screen in Points unit. Divide it by 288 to get values in 3D units.

resultOrigin
You must return in resultOrigin the same screen point, in 3D, in the Global Coordinates System. You need to use the transformation provided by SetTransform() to perform those calculations (see example below).

resultDirection
You must return in *resultDirection* the direction vector of the straight line. This must be a unit vector.

Return Value

Value	Meaning
TRUE	A ray could be calculated
FALSE	Calculation was not possible

If you want to be able to do ray-tracing everywhere in your image, you will most probably

©1995-1997 MetaCreations. All rights reserved.

always return TRUE !

Note Note that **I3DExCamera::CreateRa y** works in Global Coordinates where **I3DExCamera::Project3DTo2 D** works in Screen Coordinates. This requires a little bit of extra-work in **I3DExCamera::CreateRa y**, but gives more flexibility.

Example The isometric camera

```
BOOLEAN TIsoCamera::CreateRay(VECTOR2_3D* screenPosition, VECTOR3D*
      resultOrigin, VECTOR3D* resultDirection);
// We assume that 'fTransform' is an object field containing the trans-
      formation

  (*resultOrigin)[0] = fTransform.fR[0][0] * (*screenPosition)[0] +
      fTransform.fR[0][1] * (*screenPosition)[1];
  (*resultOrigin)[1] = fTransform.fR[1][0] * (*screenPosition)[0] +
      fTransform.fR[1][1] * (*screenPosition)[1];
  (*resultOrigin)[2] = fTransform.fR[2][0] * (*screenPosition)[0] +
      fTransform.fR[2][1] * (*screenPosition)[1];

  (*resultOrigin) /= ShortToQuickFix(288);
  (*resultOrigin) += fTransform.fT;

  // The direction vector is orthogonal to the screen in an isometric
      projection
  (*resultDirection)[0] = -fTransform.fR[2][0];
  (*resultDirection)[1] = -fTransform.fR[2][1];
  (*resultDirection)[2] = -fTransform.fR[2][2];

  return TRUE;
}
```

I3DExCamera: :Project3DTo2 D

BOOLEAN I3DExCamera::Project3DTo2D (*position, resultScreenPosition, resultDistanceToScreen***);**
VECTOR3D* *position*
VECTOR2D* *resultScreenPosition*
NUM3D* *resultDistanceToScreen*

I3DExCamera::Project3DTo2 D is called to project a 3D point on the screen.

Parameters

position
the 3D point in the Screen Coordinate System.

resultScreenPosition
Projection on the screen in Points units. Simply multiply the 3D unit by 288 to get Points (1/72 inch).

resultDistanceToScreen
Distance from the point position to the screen. It must be positive. You can simply

©1995-1997 MetaCreations. All rights reserved.

return -(*position)[2] if your screen is a flat plane.

Return Value

Value	Meaning
TRUE	The point is visible
FALSE	The point is not visible

Comments You do not have to do any 3D clipping. Simply return TRUE if the point is in front of the camera.

See the chapter "Database Overview" for more details about the different Coordinate Systems.

The fact that *position* is already in the Screen Coordinate System makes the projection formulae usually simple: a point on the screen has its z coordinate equal to 0.

Example Isometric camera:

```
// Note: Let's pretend that fZoomCoeficient is a camera parameter that
      the user can change through the camera's user interface.
// Note: 3D clipping is not necessary

BOOLEAN TIsoCamera::Project3DTo2D(VECTOR3D* position, VECTOR2D*
      resultScreenPosition, NUM3D* resultDistanceToScreen) {
  (*resultScreenPosition)[0] = (*position)[0] * fZoomCoeficient * Short-
      ToQuickFix(288);
  (*resultScreenPosition)[1] = (*position)[1] * fZoomCoeficient * Short-
      ToQuickFix(288);
  *resultDistanceToScreen = -(*position)[2];
  return (*resultDistanceToScreen > kQuickFixZero);
}
```

I3DExCamera::GetPrimitiveID

ULONG I3DExCamera::GetPrimitiveID() ;

I3DExCamera::GetPrimitiveI D is called to get the primitive ID of the camera.

Return Value The primitive ID.

I3DExCamera::Clip3D

ULONG I3DExCamera::Clip3D (*localFacet, localVertices, cameraFacet, cameraVertices, clipBox***);**
FACET3D* *localFacet*
VERTEX3D* *localVertices*
FACET3D* *cameraFacet*
VERTEX3D* *cameraVertices*

©1995-1997 MetaCreations. All rights reserved.

NUM3D* *clipBox*

I3DExCamera::Clip3 D is called to perform 3D Clipping on a facet.

Clipping have to be performed at the same time in the Object Local Coordinates and in the Camera Coordinates, so the 3D Shell does not have to redo all the inverse transformations to get back to the Object Local Coordinates System.

Parameters

localFacet
Points on the facet to clip. The facet vertices are in the Object Local Coordinates.

localVertices
Points on an array where the clipped vertices should be returned, in Object Local Coordinates. This array will have 64 elements.

cameraFacet
Points on the facet to clip. The facet vertices are in the Camera Coordinates.

cameraVertices
Points on an array where the clipped vertices should be returned, in Camera Coordinates. This array will have 64 elements.

clipBox
Points on an array of 6 elements, describing the 3D Box to use for clipping, in Camera Coordinates.

clipBox[0]	x minimum. Usually the left side of the rendering frame, in 3D units
clipBox[1]	x maximum. Usually the right side of the rendering frame, in 3D units
clipBox[2]	y minimum. Usually the bottom side of the rendering frame, in 3D units
clipBox[3]	y maximum. Usually the top side of the rendering frame, in 3D units
clipBox[4]	z minimum. Usually set to 0.0
clipBox[5]	z maximum. Usually set to the maximum value

Return Value The number of vertices returned in *localVertices* and *cameraVertices.* For example, if no clipping occurred, 3 should be returned, and the facet vertices should just be copied. If the facet is considered invalid for one reason or another, return 0.

Comments *localVertices* and *cameraVertices* is the convex 3D polygon resulting of the clipping.

When coding your 3D clipping, you should base your clipping decisions on *cameraFacet* and *clipBox.* Then when you calculate the new vertices, do it at the same time on *localFacet.*

I3DExCamera::ClipLine3D

BOOLEAN I3DExCamera::ClipLine3D (*P1, P2***);**

©1995-1997 MetaCreations. All rights reserved.

VECTOR3D* *P1*
VERTOR3D* *P2*

I3DExCamera::ClipLine3 D is called when the application wants to draw a line. This function is used to tell the application if this segment is visible in the screen, partally visible or completely unvisible. If the segment is partially visible, I3DExCamera::ClipLine3D modify the points to send back only the visible part of the segment.

Parameters

P1
A point of the segment, modify it if the segment is partially visible.

P2
The other point of the segment, modify it if the segment is partially visible.

Return Value

Value	Meaning
TRUE	The segment is visible (completely or partially)
FALSE	The segment is invisible, P1 and P2 are meaningless

.

Comments Lines are drawn for grids, bounding boxes, wireframes...

©1995-1997 MetaCreations. All rights reserved.

1.4.I3DExGeometricPrimitive Interfac e

Family ID: ‘prim’
Interface ID: IID_I3DExGeometricPrimitive
Interface file: I3DExPrm.h

Geometric primitives describe geometric objects (like a cube, a sphere, etc.). They are rather easy to implement, because there are few calls to write. At the very minimum, the component simply needs to return a list of facets or bicubic patches, and give some information on its UV Space.

They can be extended up to providing the ray-tracing intersection routine so exact ray-tracing calculations can be done instead of relying on facets or patches.

```
DECLARE_INTERFACE_(I3DExGeometricPrimitive, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExGeometricPrimitive methods
  //-- Geometry calls
  STDMETHOD_(BOOLEAN,IsPatchBased)(THIS) PURE;
  STDMETHOD(EnumPatches) (THIS_ EnumPatchesCallback callback, void*
      privData) PURE;
  STDMETHOD(EnumFacets) (THIS_ EnumFacetsCallback callback, void* priv-
      Data, NUM3D fidelity) PURE;
  STDMETHOD(GetBBox) (THIS_ BOX3D* bbox) PURE; // Optional - Return
      E_NOTIMPL if not implemented
  //-- Shading calls
  STDMETHOD_(ULONG, GetUVSpaceCount) (THIS) PURE;
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE; // Optional - Return E_NOTIMPL if not implemented (UVS-
      paceCount = 0)
  STDMETHOD(UV2XYZ) (THIS_ VECTOR2D* uv, ULONG uvSpaceID, VECTOR3D*
      resultPosition, BOOLEAN* inUVSpace) PURE; // Optional - Return
      E_NOTIMPL if not implemented
  //-- Ray Tracing calls
  STDMETHOD(RayHit) (THIS_ BOOLEAN* didHit, Ray3D* ray, RayHitParame-
      ters* hitParams, RayHit3D* hit) PURE; // Optional - Return
      E_NOTIMPL if not implemented
```

©1995-1997 MetaCreations. All rights reserved.

```
    STDMETHOD(GetRayHitDetails) (THIS_ RayHit3D* hit) PURE; // Optional -
        Return E_NOTIMPL if not implemented
    STDMETHOD(RayAllHits) (THIS_ Ray3D* aR, NUM3D tmin, NUM3D tmax,
        RayHit3D* hit, RayHitCallback callback, void* privData) PURE; //
        Optional - Return E_NOTIMPL if not implemented
    };
```

I3DExGeometricPrimitive::IsPatchBased

BOOLEAN I3DExGeometricPrimitive::IsPatchBased() ;

I3DExGeometricPrimitive::IsPatchBased is first called by the 3D Shell to know if the primitive is based on patches or facets.

Return Value

Value	Meaning
TRUE	primitive is based on patches
FALSE	patches cannot be generated

I3DExGeometricPrimitive::EnumPatches

HRESULT I3DExGeometricPrimitive::EnumPatches (*callback, privData***);**
EnumPatchesCallback *callback*
void* *privData*

I3DExGeometricPrimitive::EnumPatche s is called by the 3D Shell to get the primitive as a list of bicubic patches.

Parameters

callback
This callback procedure should be called for each Patch generated by the Primitive. Its interface is as follows:

```
void EnumPatchesCallback(PATCH3D* patch, void* privData);
```

The 3D Shell will not keep the *patch* pointer. It will copy its content.

privData
Points to some private data belonging to the Shell. Pass blankly this pointer to *callback.*

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	patches cannot be generated

Comments The Primitive should build patches, and for each of them call the *callback* procedure.

©1995-1997 MetaCreations. All rights reserved.

I3DExGeometricPrimitive::EnumFacets

HRESULT I3DExGeometricPrimitive::EnumFacets (*callback, privData, fidelity***);**
EnumPatchesCallback *callback*
void* *privData*
NUM3D *fidelity*

I3DExGeometricPrimitive::EnumFacet s is called after **I3DExGeometricPrimitive::EnumPatche s** if it was unsuccessful, to get the primitive as a list of facets.

Parameters

callback
This callback procedure should be called for each facet generated by the Primitive. Its interface is as follows:

```
void EnumFacetsCallback(FACET3D* facet, void* privData);
```

The 3D Shell will not keep the *facet* pointer. It will copy its content.

privData
Points to some private data belonging to the Shell. Pass blankly this pointer to *callback.*

fidelity
This value controls the number of facets generated. The higher the fidelity value, the larger number of facets we get. Standard fidelity is 1.0. If possible, try to be linear: multiplying *fidelity* by 2 should roughly give twice as many facets.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	facets cannot be generated

Comments Either one of EnumFacets or EnumPatches must be implemented.
If both calls are implemented, the 3D Shell always takes patches first.
The Primitive should build facets, and for each of them call the *callback* procedure.

I3DExGeometricPrimitive::GetBBox

HRESULT I3DExGeometricPrimitive::GetBBox (*bbox***);**
BOX3D* *bbox*

I3DExGeometricPrimitive::GetBBo x is called when the 3D Shell needs to get the bounding box of the primitive. This call is optional.

©1995-1997 MetaCreations. All rights reserved.

Parameter *bbox*
The 3D bounding box of the object.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Call not implemented

Comments If this call is not implemented, the bounding box will be calculated automatically by the 3D Shell from the patches or the facets.

I3DExGeometricPrimitive::GetUVSpaceCount

ULONG I3DExGeometricPrimitive::GetUVSpaceCount() ;

I3DExGeometricPrimitive::GetUVSpaceCoun t is called to get the number of UV Spaces of the primitive. See the “Database Overview” chapter for more information about UV Spaces.

Return value Number of UV Spaces for this object. If it is not possible to do direct parametric mapping on this object (no UV values), then return 0.

I3DExGeometricPrimitive::GetUVSpace

HRESULT I3DExGeometricPrimitive::GetUVSpace (*uvSpaceID, uvSpaceInfo***)**
ULONG *uvSpaceID*
UVSpaceInfo* *uvSpaceInfo*

I3DExGeometricPrimitive::GetUVSpac e is called to get more information about the UV Space(s) of the primitive.

Parameters *uvSpaceID*
The UV Space ID on which the Component needs to return information, indexed from 0 to **GetUVSpaceCoun t**()-1.

uvSpaceInfo
Point to a record that the Component must fill-in.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	You do not implement GetUVSpace, if GetUVSpaceCount return 0

Example Example for a primitive with only 1 UV Space that goes from 0 to 1 in both u and v. The UV Space is open on both u and v directions (no wrap-around).

©1995-1997 MetaCreations. All rights reserved.

```
STDMETHODIMP(GetUVSpace) (ULONG uvSpaceID, UVSpaceInfo* uvSpaceInfo)
{
  if (uvSpaceID == 0)
  {
    uvSpaceInfo->fMin.u = kQuickFixZero;
    uvSpaceInfo->fMin.v = kQuickFixZero;
    uvSpaceInfo->fMax.u = kQuickFixOne;
    uvSpaceInfo->fMax.v =  kQuickFixOne;
    uvSpaceInfo->fWraparound[0] = FALSE;
    uvSpaceInfo->fWraparound[1] = FALSE;
    uvSpaceInfo->fIsFlatSurface = FALSE;
  }
  return ResultFromScode(S_OK);
}
```

I3DExGeometricPrimitive::UV2XYZ

HRESULT I3DExGeometricPrimitive::UV2XYZ (*uv, uvSpaceID, resultPosition, inUVSpace***);**
VECTOR2D* *uv*
ULONG *uvSpaceID*
VECTOR3D* *resultPosition*
BOOLEAN* *inUVSpace*

I3DExGeometricPrimitive::UV2XY Z is called when a point on the surface defined only by its u,v coordinates needs to be converted in x, y and z values. This call is optional.

Parameters

uv
Points to the u, v values that are given to the procedure.

uvSpaceID
The identifier of the UV space.

resultPosition
The resulting 3D coordinates in the object Local Coordinates System.

inUVSpace
The procedure must return TRUE is the uv values are within a UV Space boundaries.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Call not implemented

Comments If this call is not implement, default rules will apply using the patches or the facets of the Primitive. When possible, implement **I3DExGeometricPrimitive::UV2XY Z** for optimal performance and accuracy.

©1995-1997 MetaCreations. All rights reserved.

I3DExGeometricPrimitive::RayHit

HRESULT I3DExGeometricPrimitive::RayHit (*didHit, ray, hitParams, hit***)**
BOOLEAN* *didHit*
Ray3D* *ray*
RayHitParameters* *hitParams*
RayHit3D* *hit*

I3DExGeometricPrimitive::RayHi t is called to allow the primitive to perform specific ray-tracing calculations instead of relying on the facets or patches. For example, a sphere can provide its intersection routine so the ray-tracing renders perfect spheres. This call is optional.

Parameters

didHit
Return TRUE in this parameter if a hit occurred.

ray
This is the ray, provided by the 3D Shell.

hitParams
This parameter gives the minimum and the maximum t Ray parameter (see Comments below) above which the hit is allowed to occur.

hit
Points to a record that must be filled-in if a hit occurred. See the description of the data structure for all details. Depending on the flags set in the structure, more or less data should be calculated and stored back in *hit*.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Call not implemented

Comments

The t Ray parameter is the real value that is used in the parametric equation of the ray:

$$P = Origin + t \cdot Direction$$

Call **I3DExGeometricPrimitive::GetRayHitDetail s** if there is a hit and that you want more details on it.

Note

If **I3DExGeometricPrimitive::RayHi t** is implemented, then **I3DExGeometricPrimitive::GetRayHitDetail s** and **I3DExGeometricPrimitive::Ra y-AllHits** must be implemented as well.

If they are not implemented, ray-tracing intersection will be based on the patches or the facets.

©1995-1997 MetaCreations. All rights reserved.

I3DExGeometricPrimitive::GetRayHitDetails

HRESULT I3DExGeometricPrimitive::GetRayHitDetails (*hit***)**
RayHit3D* *hit*

I3DExGeometricPrimitive::GetRayHitDetail s is called when a ray-tracing hit occurred and that the 3D Shell needs to get more information in this hit. This call is optional.

Parameter

hit
Points to a record that must be filled-in.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Call not implemented

Note If **I3DExGeometricPrimitive::GetRayHitDetail s** is implemented, then **I3DExGeometricPrimitive::RayHi t** must be implemented as well.

If they are not implemented, ray-tracing intersection will be based on the patches or the facets.

I3DExGeometricPrimitive::RayAllHits

HRESULT I3DExGeometricPrimitive::RayAllHits (*aR, tmin, tmax, hit, callback, privData***);**
Ray3D* *aR*
NUM3D *tmin*
NUM3D *tmax*
RayHit3D* *hit*
RayHitCallback *callback*
void* *privData*

This call is not yet used by the application, just return **E_NOTIMPL**.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.5.I3DExVolumePrimitive Interfac e

Family ID: ‘prim’
Interface ID: IID_I3DExVolumePrimitive
Interface file: I3DExPrm.h

Volume primitives are primitives that can’t be defined as a simple shading surface like others primitives because they are transparent objects with a variation of color inside. This is for example the fire or the cloud. Generally, you can’t see those objects in preview.

There is two way for making a volume primitive, the first one is by providing a filter color and a glow color for each point of the volume primitive with **I3DExVolumePrimitive::GetVolumedensit y**; the other one is by calculating yourself the color from a ray with **I3DExVolumePrimitive::GetVolumeAttenuatio n**.

This API is quite new. We reserve the right of changing it. Call us directly if you plan to make a volume primitive and diffuse it.

```
DECLARE_INTERFACE_(I3DExVolumePrimitive, I3DExGeometricPrimitive) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExGeometricPrimitive methods
  //-- Geometry calls
  STDMETHOD_(BOOLEAN, IsPatchBased) (THIS) PURE;
  STDMETHOD(EnumPatches) (THIS_ EnumPatchesCallback callback, void*
      privData) PURE;
  STDMETHOD(EnumFacets) (THIS_ EnumFacetsCallback callback, void* priv-
      Data, NUM3D fidelity) PURE;
  STDMETHOD(GetBBox) (THIS_ BOX3D* bbox) PURE; // Optional - Return
      E_NOTIMPL if not implemented
  //-- Shading calls
  STDMETHOD_(ULONG, GetUVSpaceCount) (THIS) PURE;
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE;
  STDMETHOD(UV2XYZ) (THIS_ VECTOR2D* uv, ULONG uvSpaceID, VECTOR3D*
      resultPosition, BOOLEAN* inUVSpace) PURE; // Optional - Return
```

©1995-1997 MetaCreations. All rights reserved.

```
    E_NOTIMPL if not implemented
//-- Ray Tracing calls
STDMETHOD(RayHit) (THIS_ BOOLEAN* didHit, Ray3D* ray, RayHitParame-
    ters* hitParams, RayHit3D* hit) PURE; // Optional - Return
    E_NOTIMPL if not implemented
STDMETHOD(GetRayHitDetails) (THIS_ RayHit3D* hit) PURE; // Optional -
    Return E_NOTIMPL if not implemented
STDMETHOD(RayAllHits) (THIS_ Ray3D* aR, NUM3D tmin, NUM3D tmax,
    RayHit3D* hit, RayHitCallback callback, void* privData) PURE; //
    Optional - Return E_NOTIMPL if not implemented
//-- New Calls for volume primitives
STDMETHOD(GetVolumeDensity) (THIS_ VECTOR3D *point,COLOR3D *fil-
    ter,COLOR3D *glow);
STDMETHOD(GetVolumeAttenuation) (THIS_ VECTOR3D *from,VECTOR3D
    *to,COLOR3D *attenuation);
//-- Deformer support
STDMETHOD(SetDeformedBoundingBox) (THIS_ VECTOR3D *min,VECTOR3D *max);
};
```

I3DExVolumePrimitive::GetVolumeDensity

HRESULT I3DExVolumePrimitive::GetVolumeDensity (*point, filter, glow***);**
VECTOR3D* *point*
COLOR3D* *filter*
COLOR3D* *glow*

I3DExVolumePrimitive::GetVolumeDensity is called during the rendering to calculate a color from a ray going thru the volume primitive. The shell calls many times this method to get a filter color and a glow color for many points on the ray.

Parameters

point
This is the point in the volume primitive in local coordinate where the application asks for colors.

filter
Return here the filter color corresponding to the point. The filter color is the color the ray progressively tend to become.

glow
Return here the glow color corresponding to the point. The glow color is an added color to the color of the ray.

Return Value

Value	Meaning
TRUE	I3DExImportFilter::DoImport will access the entire universe.
FALSE	I3DExImportFilter::DoImport will access an added universe inside the universe.

Comments The application takes many points on the ray that go thru the volume primitive, starting

©1995-1997 MetaCreations. All rights reserved.

from the back of the primitive with a color (the color of an object behind or the color of the background) and process to a set of calculation transforming the color seen behind the primitive to the final color:

newColor = color * filter + glow

Note This is a new API and we reserve the right of changing this behavior. Call us directly if you plan to make a volume primitive and diffuse it.

I3DExVolumePrimitive::GetVolumeAttenuation

HRESULT I3DExVolumePrimitive::GetVolumeAttenuation (*from, to, attenuation***);**
VECTOR3D* *from*
VECTOR3D* *to*
COLOR3D* *attenuation*

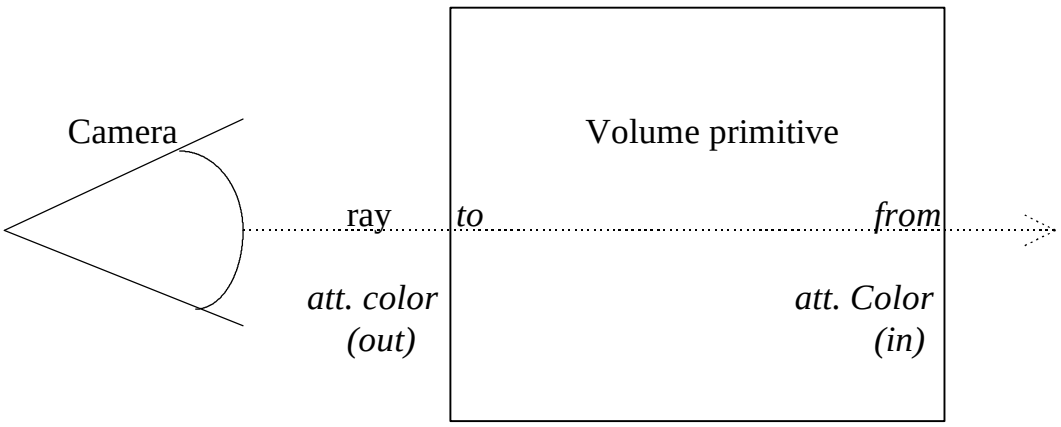
I3DExVolumePrimitive::GetVolumeAttenuation is called by the shell to get a color from a ray going thru a volume primitive.

Parameters

from
This is the intersection of the ray with the back of the volume primitive. This is where the calculation begin.

to
This is the intersection of the ray with the front of the volume primitive. This is where the calculation end.

attenuation
The shell gives here the color behind the object and you return here the resulting color.



©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	This method is not supported, the renderer will have to use the **I3DExVolumePrimitive::GetVolumeDensity.**

Comments The renderer first calculate the color behind the volume primitive and send this color to **I3DExVolumePrimitive::GetVolumeAttenuatio n**. this method calculate the resulting color and return it back.

Note This is a new API and we reserve the right of changing this behavior. Call us directly if you plan to make a volume primitive and diffuse it.

I3DExVolumePrimitive::SetDeformedBoundingBo x

HRESULT I3DExVolumePrimitive::SetDeformedBoundingBox (*min, max***);**
VECTOR3D* *min*
VECTOR3D* *max*

I3DExVolumePrimitive::SetDeformedBoundingBox is called by the shell to set the bounding box of the volume primitive. Usually, a primitive has its original bounding box that don't change, but a volume primitive can be sized up or down when we don't want it to be scaled, like the fire.

Parameters

min
The front lower left border of the bounding box (minimal values).

max
The back upper right border of the bounding box (maximal values).

Return value S_OK

Note This is a new API and we reserve the right of changing this behavior. Call us directly if you plan to make a volume primitive and diffuse it.

©1995-1997 MetaCreations. All rights reserved.

1.6.I3DExLightsource Interfac e

Family ID: 'lite'
Interface ID: IID_I3DExLightsource
Interface file: I3DExLit.h

Light sources define the lighting in a 3D scene. All kind of light sources can be designed: spot lights, bulb lights, distant lights (sun, moon), etc. Various features can be coded in a Light Source Extension, because the Light Source has complete control on the light intensity returned to the 3D Shell.

Light sources create light that illuminate object surfaces. The user can combine lights with gels to make additional lighting effects (see the **I3DExLightsourceGe l** interface).

```
DECLARE_INTERFACE_(I3DExLightsource, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release)
  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExLightsource methods
  STDMETHOD(SetTransform) (THIS_ TRANSFORM3D* transform) PURE
  STDMETHOD(GetDirection) (THIS_ VECTOR3D* position, VECTOR3D* resultDi-
      rection, NUM3D* resultDistance) PURE;
  STDMETHOD_(BOOLEAN, GetColor) (THIS_ VECTOR3D* position, VECTOR3D*
      direction, NUM3D distance, COLOR3D* result, BOOLEAN* callForShad-
      owEffect) PURE;

  STDMETHOD_(BOOLEAN, IsVisibleInPerspective) (THIS) PURE;
  STDMETHOD(ShadowEffect) (THIS_ NUM3D distance, COLOR3D* result) PURE;
  STDMETHOD_(LightTraceElement*, GetNewTrace) (THIS_ short* nb) PURE;
  STDMETHOD_(ULONG, GetPrimitiveID) (THIS) PURE;
  STDMETHOD(ForEachShadowBuffer) (THIS_ ForEachShadowBufferCallback
      proc, void* priv) PURE;
  STDMETHOD(GetLightParameter) (THIS_ long keyword, void* parameter)
      PURE;
};
```

$$y = \frac{Y \cdot F}{F - Z} \cdot Zoom$$

©1995-1997 MetaCreations. All rights reserved.

I3DExLightsource::SetTransform

HRESULT I3DExLightsource::SetTransform (*transform***);**
TRANSFORM3D* *transform*

I3DExLightsource::SetTransfor m is called to tell the light source its position and orientation in space.

Parameter *transform*
Points to the 3D Transformation of the light source.

Return valu e S_OK

Comments You typically store the transformation in the Light Source internal data space so they can be used later during the lighting calculations, in **I3DExLightsource::GetDirection( )**.

For light sources that have an aiming direction, this direction is given by the k vector of *transform*. It is the same Coordinates System as in the **I3DExCamer a** interface.

I3DExLightsource::GetDirection

HRESULT I3DExLightsource::GetDirection (*position, resultDirection, resultDistance***);**
VECTOR3D* *position*
VECTOR3D* *resultDirection*
NUM3D* *resultDistance*

I3DExLightsource::GetDirection is called prior to **I3DExLightsource::GetColo r** to perform preliminary calculations and return the direction and the distance between the surface point and the light source. The surface point is the point on the object where the lighting is about to be calculated.

Parameters *position*
The 3D position of the point on the object surface, in Global Coordinates.

resultDirection
Return the direction vector between the point and the light source, in Global Coordinates. This vector is pointing from the surface point to the light source. It must be a unit vector.

resultDistance
Return the distance between the point and the light source.

Return Valu e S_OK

Comments A light bulb will typically calculate resultDirection by subtracting the position point to the Light Source origin, and normalizing the result. The resultDistance would be the distance between the light source origin and the position point.

©1995-1997 MetaCreations. All rights reserved.

A simulated sun light will typically return its light direction in *resultDirection*, and the maximum possible value in *resultDistance.*

I3DExLightsource::GetColor

BOOLEAN I3DExLightsource::GetColor (*position, direction, distance, result, callForShadowEffect***);**
VECTOR3D* *position*
VECTOR3D* *direction*
NUM3D *distance*
COLOR3D* *result*
BOOLEAN* *callForShadowEffect*

I3DExLightsource::GetColo r is called by the 3D Shell to calculate the lighting. The Component has total control on the color to return.

Parameters

position
The 3D position of the point on the surface, in Global Coordinates, as in **I3DExLightSource::GetDirection() .**

direction
The direction vector between the point and the light source, in Global Coordinates, as calculated by **I3DExLightsource::GetDirection( )**.

distance
The distance between the point and the light source, as calculated by **I3DExLightsource::GetDirection( )**.

result
The Light Source component must return the calculated color here.

callForShadowEffect
The Light Source should return TRUE if the **I3DExLightsource::ShadowEffec t**() method should be called during shadows calculation. You must return TRUE if the light casts any shadows.

Return Value

Value	Meaning
TRUE	The point is within the lighting range
FALSE	The point does not receive any illumination from this light source

Comments **In a spot example, you would use the *position* parameter and the light source position and orientation to figure out if the point is in the light beam or not .**
In a sun example, you would rather use the *direction* parameter to calculate the intensity. Use the *distance* parameter if your lighting is based on distance (for a distance fall-off for example).

©1995-1997 MetaCreations. All rights reserved.

I3DExLightsource::IsVisibleInPerspectiv e

BOOLEAN I3DExLightsource::IsVisibleInPerspective() ;

I3DExLightsource::IsVisibleInPerspectiv e is called by the 3D Shell to know if this light source can be displayed in a 3D perspective display or not.

Return Value

Value	Meaning
TRUE	The light source can be shown in a 3D display
FALSE	The light source should not be shown

Comments Distant lights (sun, moon, etc.) typically answer FALSE.

I3DExLightsource::ShadowEffect

HRESULT I3DExLightsource::ShadowEffect (*distance, result***);**
NUM3D *distance*
COLOR3D* *result*

I3DExLightsource::ShadowEffec t is called after **I3DExLightsource::GetColo r** returns TRUE for *callForShadowEffect.*

Parameters

distance
The distance between the point and the light source, as calculated by **I3DExLightsource::GetDirection( )**.

result
The color is given as an input and output parameter. You need to modify this color based on *distance*.

Return value S_OK

Comments Set *result* to full black if you want solid black shadows.

Example The following example illustrates a bulb light source that creates shadows that are 50 % effective (like if 50% of the light was still going through the solid);

```
STDMETHODIMP TBulbLightSource::ShadowEffect(NUM3D distance, COLOR3D*
      result) {
  result->r *= kQuickFixHalf;
  result->g *= kQuickFixHalf;
  result->b *= kQuickFixHalf;
  return ResultFromScode(S_OK);
}
```

©1995-1997 MetaCreations. All rights reserved.

I3DExLightsource::GetNewTrace

LightTraceElement* I3DExLightsource::GetNewTrace (*nb***);**
short* *nb*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExLightsource::GetPrimitiveID

ULONG I3DExLightsource::GetPrimitiveID() ;

I3DShLightsource::GetPrimitiveID is called to get the primitive ID of the light source.

Return value The primitive identifier.

I3DExLightsource::ForEachShadowBuffe r

HRESULT I3DExLightsource::ForEachShadowBuffer (*proc, priv***);**
ForEachShadowBufferCallback *proc*
void* *priv*

I3DShLightsource::ForEachShadowBuffe r is called when the Shell needs to get a list of Shadow Buffer planes from the light source to calculate shadows.
This call is used by some renderers like the RDD's Production Z-Buffer

Parameters

proc
The callback procedure that you should call for each Shadow Buffer plane of your Light Source.

```
void ForEachShadowBufferCallback(BOOLEAN lightType, TRANSFORM3D* trans-
      form, NUM3D halfAngle, void* priv);
```

lightType
Indicates if halfAngle is meaningful. FALSE for distant lights, TRUE for all others.

transform
The Light source should give the transformation as calculated by **I3DShLightsource::GetLightGlobalTransfor m**.

halfAngle
The half-angle of the light cone. Used only if lightType == TRUE.

priv
Pass back the *priv* parameter

©1995-1997 MetaCreations. All rights reserved.

priv
Give blankly this parameter to the callback procedure.

Return value S_OK

Comments Typically, a bulb light will return 6 Shadow Buffers (as it shines in all directions, a 6-sided Box has to be used). Distant lights and spot lights use only 1 Shadow Buffer.
Do not return any shadow buffer if the light source creates no shadows.

I3DExLightsource::GetLightParamete r

HRESULT I3DExLightsource::GetLightParameter (*keyword*, *parameter***);**
long *keyword*
void* *parameter*

I3DShLightsource::GetLightParamete r is called to get the light parameters from the pMAP.

Parameters *keyword*
The ID of the parameter in the pMAP.

parameter
Points to where to store the light parameters.

Return value S_OK

©1995-1997 MetaCreations. All rights reserved.

1.7.I3DExLightsourceGel Interfac e

Family ID: ‘gel ’
Interface ID: IID_I3DExLightsourceGel
Interface file: I3DExGel.h

The user can combine lights with Gels to make additional lighting effects. A Gel can be thought as a colored slide that is put in front of the light source to modify the color and intensity of the light beam.

There is only one call to implement.

```
DECLARE_INTERFACE_(I3DExLightsourceGel, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExLightsourceGel methods
  STDMETHOD_(BOOLEAN, GetGelValues) (THIS_ VECTOR2D* gelScreenPosition,
      COLOR3D* result) PURE;
  };
```

I3DExLightsourceGel::GetGelValues

BOOLEAN I3DExLightsourceGel::GetGelValues (*gelScreenPosition*, *result***);**
VECTOR2D* *gelScreenPosition*
COLOR3D* *result*

I3DExLightsourceGel::GetGelValue s changes the color of the light beam.

Parameters

gelScreenPosition
2D location where the light beam crosses the gel screen. The gel screen is a 2D screen in front of the light source with boundaries from -1.0 to +1.0 on both axis.

result
Return the factors by which the light beam color is going to be multiplied.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Meaning
TRUE	The light beam should be modified
FALSE	The gel is inactive

Comments The RGB values of the light beam color will be multiplied by the RGB values returned in *result*..

Example The following code illustrates a vertical gradient gel.

```
STDMETHODIMP_(BOOLEAN)  TVerticalGradientGel::GetGelValues(VECTOR2D*
      gelScreenPosition, COLOR3D* result)  {
  result->mode = 0;    // RBG color mode
  result->r = ((*gelScreenPosition)[1] + kQuickFixOne) >> 1;
  result->g = result->r;
  result->b = result->r;
  return TRUE;
}
```

©1995-1997 MetaCreations. All rights reserved.

1.8.I3DExAtmosphericShader Interfac e

Family ID: ‘atmo’
Interface ID: IID_I3DExAtmosphericShader
Interface file: I3DExAB.h

The atmospheric shader is used to create effects such as fog, clouds, etc. The atmospheric shader filters the light beams and can modify it at will.

```
DECLARE_INTERFACE_(I3DExAtmosphericShader, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExAtmosphericShader methods
  STDMETHOD(SegmentFilter) (THIS_ VECTOR3D* beg, VECTOR3D* end, COLOR3D*
      filterOut) PURE;
  STDMETHOD(DirectionFilter) (THIS_ VECTOR3D* origin, VECTOR3D* direc-
      tion, COLOR3D* filterOut) PURE;
};
```

I3DExAtmosphericShader::SegmentFilter

HRESULT I3DExAtmosphericShader::SegmentFilter (*beg, end, filterOut***);**
VECTOR3D* *beg*
VECTOR3D* *end*
COLOR3D* *filterOut*

I3DExAtmosphericShader::SegmentFilte r is called to filter the light beam color between two points.

Parameters

beg
This point is the beginning point of the light beam in Global Coordinates.

end
This point is the ending point of the light beam in Global Coordinates.

©1995-1997 MetaCreations. All rights reserved.

filterOut
The Light beam color is given as an input and output parameter. The component should modify this color to perform an atmospheric effect.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Not implemented

Comments The Atmospheric filtering can be based on the values of the beginning and ending points, the distance between the two points, etc.

I3DExAtmosphericShader::DirectionFilter

HRESULT I3DExAtmosphericShader::DirectionFilter (*beg, direction, filterOut***);**
VECTOR3D* *beg*
VECTOR3D* *direction*
COLOR3D* *filterOut*

I3DExAtmosphericShader::DirectionFilte r works like **I3DExAtmosphericShader::SegmentFilte r**, but is called to filter any light beam going away to the infinite. In this case, there is no ending point, but a direction vector instead.

Parameters

beg
This point is the beginning point of the light beam in Global Coordinates.

direction
The direction of the light beam in Global Coordinates. This is a unit vector.

filterOut
The Light beam color is given as an input and output parameter. The component should modify this color to perform an atmospheric effect.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	Not implemented

Comments The Atmospheric filtering should be based on the values of the beginning point and the direction.

Example Fog at the horizon can be done this way: if the light beam is going to the horizon (i.e. *direction* is horizontal), filtering would be performed; if the light beam is going up to the sky, not filtering would be done. The result is a foggy horizon and a clear sky.

©1995-1997 MetaCreations. All rights reserved.

1.9.I3DExBackground Interfac e

Family ID: ‘back’
Interface ID: IID_I3DExBackground
Interface file: I3DExAB.h

Backgrounds are 3D environment that can be reflected on the 3D objects surfaces (like a sky).

```
DECLARE_INTERFACE_(I3DExBackground, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExBackground methods
  STDMETHOD(GetBackgroundColor) (THIS_ VECTOR3D* direction, COLOR3D*
      resultColor) PURE;

};
```

I3DExBackground::GetBackgroundColo r

HRESULT I3DExBackground::GetBackgroundColor (*direction, resultColor***);**
VECTOR3D* *direction*
COLOR3D* *resultColor*

I3DExBackground::GetBackgroundColo r is called to perform environment backgrounds (sometimes called environment mapping). Environments are reflected on reflecting objects.

Parameters

→ *direction*
The unit vector defining the direction of the light beam.

← *resultColor*
Return the calculated color here.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments The direction *parameter* is enough because one can consider that the environment is an image mapped on the inside surface of a infinite sphere around the scene.

Example This is a white to blue sky, with a gray ground:

```
STDMETHODIMP TWhiteToBlueSky::GetEnvironmentColor(VECTOR3D* direction,
      COLOR3D* resultColor, BOOLEAN* result)  {
  if ((*direction)[2] >= kQuickFixZero)  { // We do it only above the
      horizon
    resultColor->r = (*direction)[2];
    resultColor->g = (*direction)[2];
    resultColor->b = kQuickFixOne;
    }
  else {// Gray ground
    resultColor->r = kQuickFixHalf;
    resultColor->g = kQuickFixHalf;
    resultColor->b = kQuickFixHalf;
  }
  return ResultFromScode(S_OK);
}
```

©1995-1997 MetaCreations. All rights reserved.

1.10.I3DExBackdrop Interfac e

Family ID: ‘drop’
Interface ID: IID_I3DExBackdrop
Interface file: I3DExAB.h

Backdrops are image at infinity (behind every objects) in the axis of the camera. A backdrop can not be reflected on the objects.

```
DECLARE_INTERFACE_(I3DExBackdrop, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExBackground methods
  STDMETHOD(GetBackdropColor) (THIS_ VECTOR2D* screenPoint, BOX2D*
      screenMinMax, COLOR3D* resultColor) PURE;

};
```

I3DExBackdrop::GetBackdropColor

HRESULT I3DExBackdrop::GetBackdropColor (*screenPoint, screenMinMax, resultColor***);**
VECTOR2D* *screenPoint*
BOX2D* *screenMinMax*
COLOR3D* *resultColor*

Parameters

screenPoint
This is the point in 2D screen coordinates. Its boundaries are defined by *screenMinMax*.

screenMinMax
The boundaries (minimum and maximum values) of the *screenPoint* range of values.

resultColor
Return the calculated color here.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

Comments Remember that backdrops are like composing the rendered image with another background image. Backdrops are not reflected on reflective objects. Use backgrounds for this.

Example This is a white to blue backdrop gradient:

```
STDMETHODIMP TWhiteToBlueBackdrop::GetBackdropColor(VECTOR2D* screen-
      Point, BOX2D* screenMinMax, COLOR3D* resultColor)  {
  resultColor->r = ((*screenPoint)[1] - screenMinMax->fMin[1]) /
      (screenMinMax->fMax[1] - screenMinMax->fMin[1]);
  resultColor->g = resultColor->r;
  resultColor->b = kQuickFixOne;
  return ResultFromScode(S_OK);
}
```

©1995-1997 MetaCreations. All rights reserved.

1.11.I3DExAmbientLight Interfac e

Family ID: ’ambi’
Interface ID: IID_I3DExAmbientLight
Interface file: I3DExAB.h

This Ambient Light shines from all directions, and therefore does not cast any shadows.

```
DECLARE_INTERFACE_(I3DExAmbientLight, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExAmbientLight methods
  STDMETHOD(GetColor) (THIS_ COLOR3D* resultColor) PURE;
};
```

I3DExAmbientLight::GetColor

HRESULT I3DExAmbientLight::GetColor (*resultColor***);**
COLOR3D* *resultColor*

Parameters ⇨ *resultColor*
The color of the ambient light to return. This is global for the scene.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.12.I3DExImportFilter Interfac e

Family ID: ’3Din’
Interface ID: IID_ I3DExImportFilter
Interface file: I3DExIO.h

Importers are 3D Components that build a whole scene from a file. They are used to import various 3D formats. They heavily use all the 3D Shell calls to build the 3D database.

Read carefully the description of the ‘cmpp’ resource format for this extension in the “managing the user interface” Appendix

```
DECLARE_INTERFACE_(I3DExImportFilter, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExImportFilter methods
  STDMETHOD_(BOOLEAN, Prepare) (THIS_ char* fullPathName, I3DShScene*
      scene, I3DShTreeElement* fatherTree) PURE;
  STDMETHOD_(BOOLEAN, WantsOptionDialog) (THIS) PURE;
  STDMETHOD(DoImport) (THIS_ char* fullPathName, I3DShScene* scene,
      I3DShTreeElement* fatherTree) PURE;
  STDMETHOD_(BOOLEAN, WantsTopScene) (THIS) PURE;
  };
```

I3DExImportFilter::Prepare

BOOLEAN I3DExImportFilter::Prepare (*fullPathName, scene, fatherTree***);**
char* *fullPathName*
I3DShScene* *scene*
I3DShTreeElement* *fatherTree*

I3DExImportFilter::Prepar e is the first call made by the 3D Shell so that the 3D Component can prepare the work and agree to proceed or not.

©1995-1997 MetaCreations. All rights reserved.

Parameters

fullPathName
The full file pathname.

scene
The scene in which the file should be imported.

fatherTree
The tree element under which the 3D data should be placed. If NULL, then the 3D Importer will have to create a Top Tree (“Universe”) element (See **I3DExImportFilter::DoImpor t**)

Return Value

Value	Meaning
TRUE	The Shell is authorized to proceed with the import
FALSE	Importation should be aborted

Comments The Component can do whatever it wants, even read part of the file to get some data. Typically, **I3DExImportFilter::Prepar e** is used to set up some parameters before displaying an options dialog to the user. For example, in a DXF import, **I3DExImportFilter::Pr epare** would be used to read the list of Coordinate Systems in the file so the user can choose which one to use in the dialog.

I3DExImportFilter::WantsOptionDialog

BOOLEAN I3DExImportFilter::WantsOptionDialog() ;

I3DExImportFilter::WantsOptionDialo g is called by the 3D Shell to learn if the importer needs to open an option dialog or not. This method is called after **I3DExImportFilter::Prepar e**.

Return Value

Value	Meaning
TRUE	Open the options dialog
FALSE	Do not open the options dialog

Comments The dialog will be automatically open by the 3D Shell before calling **I3DExImportFilter::DoImpor t**.

I3DExImportFilter::DoImport

HRESULT I3DExImportFilter::DoImport (*fullPathName, scene, fatherTree***);**
char* *fullPathName*
I3DShScene* *scene*
I3DShTreeElement* *fatherTree*

©1995-1997 MetaCreations. All rights reserved.

I3DExImportFilter::DoImpor t is called after **I3DExImportFilter::WantsOptionDi a-log** to perform the actual import.

Parameters

fullPathName
The full file pathname.

scene
The scene in which the file should be imported.

fatherTree
The tree element under which the 3D data should be placed. If NULL, then the 3D Importer must create a Top Tree (“Universe”) element (using **I3DShScene::Creat e-TreeRootIfNon e**), and hooks its data under it.

Return Valu e S_OK

Comments **I3DExImportFilter::DoImpor t** is responsible for opening, reading and closing the file. The **IShFileStream** interface should be used to read the file.

I3DExImportFilter::WantsTopScene

BOOLEAN I3DExImportFilter::WantsTopScene () ;

I3DExImportFilter::WantsTopScen e is called before **I3DExImportFilter::DoImpor t** to know if you want to import directly in the universe or not. If you answer false, an other universe will be created and you will be able to import into it.

Return Valu e

Value	Meaning
TRUE	I3DExImportFilter::DoImport will access the entire universe.
FALSE	I3DExImportFilter::DoImport will access an added universe inside the universe.

Comments In general, you will return false, except when you explicitly need informations on elements in the universe.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.13.I3DExExportFilter Interfac e

Family ID: ’3Dou’
Interface ID: IID_ I3DExExportFilter
Interface file: I3DExIO.h

Exporters are 3D Components that write files from a scene or a part of a scene. They are used to export in various 3D formats. They use heavily all the 3D Shell calls to browse through the 3D database.

Read carefully the description of the ‘cmpp’ resource format for this extension in the “managing the user interface” Appendix

```
DECLARE_INTERFACE_(I3DExExportFilter, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExExportFilter methods
  STDMETHOD_(BOOLEAN, Prepare) (THIS_ I3DShScene* scene,
      I3DShTreeElement* fatherTree) PURE;
  STDMETHOD_(BOOLEAN, WantsOptionDialog) (THIS) PURE;
  STDMETHOD(NeededDiskSpace) (THIS_ char* fullPathName, I3DShScene*
      scene, I3DShTreeElement* fatherTree, ULONG* diskSpace) PURE;
  STDMETHOD(DoExport) (THIS_ char* fullPathName, I3DShScene* scene,
      I3DShTreeElement* fatherTree) PURE;
  //-- Preview calls
  STDMETHOD_(BOOLEAN, WantsPreview) (THIS) PURE;
  STDMETHOD(GetPreviewSize) (THIS_ short* sizeh, short* sizev) PURE;
  STDMETHOD(SetPreview) (THIS_ I3DShOffscreen* preview) PURE;
  };
```

I3DExExportFilter::Prepare

BOOLEAN I3DExExportFilter::Prepare (*scene, fatherTree***);**
I3DShScene* *scene*
I3DShTreeElement* *fatherTree*

©1995-1997 MetaCreations. All rights reserved.

I3DExExportFilter::Prepar e is the first call made by the 3D Shell so that the 3D Component can prepare the work and agree to proceed or not.

Parameters

scene
The scene to be exported.

fatherTree
The top tree element to export. All data under this tree element should be exported, including *fatherTree*.

Return Value

Value	Meaning
TRUE	The Shell is authorized to proceed with the export
FALSE	Exportation should be aborted

I3DExExportFilter::WantsOptionDialo g

BOOLEAN I3DExExportFilter::WantsOptionDialog() ;

I3DExExportFilter::WantsOptionDialo g is called by the 3D Shell to learn if the exporter needs to open an option dialog or not. This method is called after **I3DExExportFilter::Prepar e**.

Return Value

Value	Meaning
TRUE	Enable the Options button in the standard Save dialog
FALSE	Disable the Options button

Comments Note that the Options dialog will not be opened if the user does not press the Options button. Therefore the Component code should not rely on any initialization done at the user interface level, and should be able to use default values for its export parameters.

I3DExExportFilter::NeededDiskSpac e

HRESULT I3DExExportFilter::NeededDiskSpace (*fullPathName, scene, fatherTree, diskSpace***);**
char* *fullPathName*
I3DShScene* *scene*
I3DShTreeElement* *fatherTree*
ULONG* *diskSpace*

I3DExExportFilter::NeededDiskSpac e is used by the 3D Shell to figure out if there is enough disk space left before trying to export the file.

Parameters

fullPathName
The full file pathname.

©1995-1997 MetaCreations. All rights reserved.

scene
The scene to be exported.

fatherTree
The top tree element to export. All data under this tree element should be exported, including *fatherTree*.

diskSpace
Return here the amount of disk space that will be needed for the export. If this amount is unknown, return 0.

Return Value S_OK

Comments If for some reason the amount of disk space cannot be calculated, return 0. The 3D Shell will still attempt the export by calling **I3DExExportFilter::DoExpor t**. The export Component must fail and delete any created files if running out of disk space.

I3DExExportFilter: :DoExport

HRESULT I3DExExportFilter::DoExport (*fullPathName, scene, fatherTree***);**
char* *fullPathName*
I3DShScene* *scene*
I3DShTreeElement* *fatherTree*

I3DExExportFilter::DoExpor t is called to perform the actual export.

Parameters *fullPathName*
The full file pathname.

scene
The scene to be exported.

fatherTree
The top tree element to export. All data under this tree element should be exported, including *fatherTree*.

Return Value S_OK

Comments The Exporter Component is in charge of creating and opening all necessary files, and closing them when done. Use the **IShFileStream** interface to write the file.

I3DExExportFilter::WantsPreview

BOOLEAN I3DExExportFilter::WantsPreview() ;

©1995-1997 MetaCreations. All rights reserved.

I3DExExportFilter::WantsPrevie w is called by the 3D Shell to learn if the exporter needs any preview image of the data to export. This image can be used by the exporter to build preview icons, thumbnails, etc.

Return Value

Value	Meaning
TRUE	The Exporter needs a preview image
FALSE	

I3DExExportFilter::GetPreviewSize

HRESULT I3DExExportFilter::GetPreviewSize (*sizeh, sizev***);**
short* *sizeh*
short* *sizev*

I3DExExportFilter::GetPreviewSiz e is called by the 3D Shell to get the size of the preview image needed by the exporter.

Parameters

sizeh
Return the horizontal size in pixels.

sizev
Return the vertical size in pixels.

Return Value S_OK

Comments Do not ask for too large images, because they can take quite some RAM.
This procedure is called only if **I3DExExportFilter::WantsPrevie w** returned TRUE.

I3DExExportFilter::SetPreview

HRESULT I3DExExportFilter::SetPreview (*preview***);**
I3DShOffscreen* *preview*

I3DExExportFilter::SetPrevie w is called by the 3D Shell to give the extension the previews that was requested in **I3DExExportFiltre::WantsPrevie w**.

Parameters

preview
The requested 32 bit preview image.

Return Value S_OK

Comments The Exporter should get the bitmap data of *preview*, and build its thumbnail. See the **I3DShOffscreen** interface for more details.

©1995-1997 MetaCreations. All rights reserved.

1.14.I3DExShader Interfac e

Family ID: ‘shdr’
Interface ID: IID_I3DExShader
Interface file: I3DExSha.h

Shaders are very powerful 3D Components used to give photorealistic appearance to the 3D objects surfaces. They define the shading values at each point of an object surface: color, shininess, reflectivity, etc. used to compute the illumination on the surface.

There are two types of Shaders: Shaders and Sub-Shaders. Shaders take care of everything: they take all the input parameters and return all the shading values (see **DoShade**). The advantage is to have complete control on the shading process, the disadvantage is that the shader is more difficult to program and not very flexible. Sub-Shaders are more like little bricks that can be combined together by the user to « build » is own shader. Sub-Shaders return either a value (**GetValue**) or a color (**GetColor**). They are more simple to program and much more flexible.

```
DECLARE_INTERFACE_(I3DExShader, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExShader methods
  STDMETHOD_(BOOLEAN, IsEqualTo) (THIS_ I3DExShader* aShader) PURE;
  STDMETHOD_(BOOLEAN, DependsOnAppliedExtent) (THIS) PURE;
  STDMETHOD(GetShadingFlags) (THIS_ ShadingFlags* theFlags) PURE;
 STDMETHOD_(ULONG, GetPreferredOutput) (THIS) PURE; // Tell which one of
      the following calls is implemented. See kUseGetValue... constants
  //-- Shading calls. Implement only one of those. Return E_NOTIMPL for
      the others
  STDMETHOD(DoShade) (THIS_ ShadingInOut* inOut, ShadingElem* shadin-
      gElem) PURE;
  STDMETHOD(GetValue) (THIS_ NUM3D* result, ShadingIn* in, ShadingElem*
      theShadingElem) PURE;
  STDMETHOD(GetColor) (THIS_ COLOR3D* result, ShadingIn* in, Shadin-
      gElem* theShadingElem) PURE;
  STDMETHOD(GetVector) (THIS_ VECTOR3D* result, ShadingIn* in, Shadin-
      gElem* theShadingElem) PURE; };
```

©1995-1997 MetaCreations. All rights reserved.

You need to implement only one of **DoShade**, **GetValue** or **GetColor**. **DoShade** is a generic call that is used for implementing a “one-shot” shader: such a shader cannot be combined with others to build shader trees, but it can do absolutely any shading effect. **GetValue** or **GetColor** are used to implement sub-shaders: these shaders can be combined in a hierarchy to build a complex shader, but they are more limited because they are basic “bricks”. Sub-shaders are extremely easy to implement.

I3DExShader::IsEqualTo

BOOLEAN I3DExShader::IsEqualTo (*aShader***);**
I3DExShader* *aShader*

I3DExShader::IsEqualTo is used to compare the shader values with other shader values.

Parameter *aShader*
Points to the other shader.

Return Value

Value	Meaning
TRUE	Shaders have identical values
FALSE	Shaders are different

Comments The implementation of **I3DExShader::IsEqualTo** must first get a pointer on the same interface as itself. Then values should be compared one after the other. Finally, the interface should be released.

Example The following code shows an implementation of a Checker shader:

```
DECLARE_INTERFACE_(ICheckerShader, I3DExShader) {
  ... I3DExShader and ICheckerShader methods go here ...

  ULONG  fNbTilesH;  // Number of tiles horizontaly
  ULONG  fNbTilesV;  // Number of tiles verticaly
  };

STDMETHODIMP_(BOOLEAN) ICheckerShader::IsEqualTo(I3DExShader* aShader) {
  ICheckerShader* otherShader = NULL;
  BOOLEAN result = FALSE;

  aShader->QueryInterface(IID_ICheckerShader, & (LPVOID) otherShader);
  if (otherShader == NULL) return FALSE;

  result = (otherShader->fNbTilesH == fNbTilesH) &&
        (otherShader->fNbTilesV == fNbTilesV);
  otherShader->Release();  // Do not forget this
  return result;
}
```

©1995-1997 MetaCreations. All rights reserved.

I3DExShader::DependsOnAppliedExtent

BOOLEAN I3DExShader::DependsOnAppliedExtent() ;

I3DExShader::DependsOnAppliedExten t is called by the 3D Shell to learn if the Shader needs the **ShadingElem** parameter in the shading calls.

Return Value

Value	Meaning
TRUE	The Shader needs the ShadingElem parameter
FALSE	

Comments See **I3DExShader::DoShad e**, **I3DExShader::GetValu e**, **I3DExShader::GetColo r** or **I3DExShader::GetVecto r**.

I3DExShader::GetShadingFlags

HRESULT I3DExShader::GetShadingFlags (*theFlags***);**
ShadingFlags* *theFlags*

I3DExShader::GetShadingFlag s allows the 3D Shell to learn which parameters the shader will need to perform the shading calculations. This way, only the minimal number of parameters is calculated.

Parameter *theFlags*
Points to a **ShadingFlags** record containing all the flags.

Return Value S_OK

Comments If the shader is a *constant shader*, i.e. a shader which values does not depend on any of the ShadingIn parameters, then it should set the fCallOnce flag in *theFlags* in order to be called only once.

See the description of the **ShadingFlags** structure for more details.

I3DExShader::GetPreferredOutput

ULONG I3DExShader::GetPreferredOutput() ;

I3DExShader::GetPreferredOutpu t is called by the 3D Shell to learn which Shading Call the Shader actually implements: **DoShade**, **GetValue**, **GetColor**, etc.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Constant	Value	Meaning
kUsesGetValue	1	**I3DExShader::GetValue** is implemented
kUsesGetColor	2	**I3DExShader::GetColor** is implemented
kUsesGetVector	4	**I3DExShader::GetVector** is implemented
kUsesDoShade	16	**I3DExShader::DoShade** is implemented

Comments Return only one of those values, as you need to implement only one of the Shading Calls. The *kUsesGetVector* value is reserved for a future use.

I3DExShader::DoShade

HRESULT I3DExShader::DoShade (*inOut, shadingElem***);**
ShadingInOut* *inOut*
ShadingElem* *shadingElem*

I3DExShader::DoShad e has total control on the input and output parameters to perform any shading effects. This should be used when the shader needs access to all the lighting channels (Diffuse and specular colors, reflectivity, etc.) at the same time.

Parameters

inOut
Points to the complete input/output shading data structure. Typically **DoShade** fetches input data in *inOut->fIn*, and sets return data in *inOut->fOut*.

shadingElem
The Shading element associated with the shader.

Return Value

Value	Meaning
S_OK	The shading was calculated
E_NOTIMPL	The call is not implemented

Comments See the description of **ShadingInOut** and **ShadingElem** for more details.

I3DExShader::GetValue

HRESULT I3DExShader::GetValue (*result, in, theShadingElem***);**
NUM3D* *result*
ShadingIn* *in*
ShadingElem* *theShadingElem*

I3DExShader::GetValu e should be implemented for shaders that want to return a value between 0.0 and 1.0. Typical examples are noise functions, fractal functions, gradients, etc.

©1995-1997 MetaCreations. All rights reserved.

Parameters *result*
Return the result value between 0.0 and 1.0 here.

in
Points to the input shading data structure. This is where the shader gets the input parameters it requested with **I3DExShader::GetShadingFlag s**.

shadingElem
The Shading element associated with the shader.

Return Value

Value	Meaning
S_OK	The shading was calculated
E_NOTIMPL	The call is not implemented

Comments See the description of **ShadingIn** and **ShadingElem** for more details.

I3DExShader::GetColor

HRESULT I3DExShader::GetColor (*result, in, theShadingElem***);**
COLOR3D* *result*
ShadingIn* *in*
ShadingElem* *theShadingElem*

I3DExShader::GetColo r should be implemented for shaders that want to return a color. Typical examples are constant colors, texture maps, etc.

Parameters *result*
Return the color values here.

in
Points to the input shading data structure. This is where the shader gets the input parameters it requested with **I3DExShader::GetShadingFlag s**.

shadingElem
The Shading element associated with the shader.

Return Value

Value	Meaning
S_OK	The shading was calculated
E_NOTIMPL	The call is not implemented

Comments See the description of **ShadingIn** and **ShadingElem** for more details.

Example Here is a nice “rainbow” shader where the color changes depending the direction the sur-

©1995-1997 MetaCreations. All rights reserved.

face point is facing. We simply map the Normal coordinates to the RGB color space.

```
STDMETHODIMP TRainbowShader::GetColor(ShadingIn* in, ShadingElem*
      theShadingElem, COLOR3D* result)  {
  result->r = (in->fN[0] + kQuickFixOne) >>1;
  result->g = (in->fN[1] + kQuickFixOne) >>1;
  result->b = (in->fN[2] + kQuickFixOne) >>1;
  return ResultFromScode(S_OK);
}
```

I3DExShader::GetVector

HRESULT I3DExShader::GetVector (*result, in, theShadingElem***);**
VECTOR3D* *result*
ShadingIn* *in*
ShadingElem* *theShadingElem*

I3DExShader::GetVecto r is reserved for a future use.

©1995-1997 MetaCreations. All rights reserved.

1.15.I3DExPostRenderer Interfac e

Family ID: ‘post’
Interface ID: IID_I3DExPostRenderer
Interface file: I3DExPos.h

The **I3DExPostRendere r** Interface is used to apply additional effects (filters) to the image after the rendering. First of all, the application asks the extension which informations are needed.

```
DECLARE_INTERFACE_(I3DExPostRenderer, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExPostRenderer methods
  STDMETHOD_(void, Filter) (THIS_ I3DShScene* theScene,IShRasterOff-
      screen* theImage,IUnknown** theBuffers, IShGraphicDevice* gd, /
      *TExternalRenderer*/void* renderer, void* renderHelper) PURE;
  STDMETHOD_(void, GetBufferNeeds) (THIS_ long* needs, /*TExternalRen-
      derer*/void* renderer) PURE;
  };
```

I3DExPostRenderer::GetBufferNeeds

HRESULT I3DExPostRenderer::GetBufferNeeds (*needs, renderer***);**
long* *needs*
void* *renderer*

I3DExPostRenderer::GetBufferneeds is called by the 3D Shell before the rendering to know what informations are needed for the post render filter. The 3D Shell will fill buffers during the rendering and send those buffer the **I3DExPostRenderer::Filte r** method.

©1995-1997 MetaCreations. All rights reserved.

Parameter *needs*

Return here a set of flags for the informations you need.

Constant	Value	Meaning
kDistance	1	Distance from the camera to the object seen at this point (max. val. = background).
kPositionX	2	
kPositionY	4	Position of the point seen (if not background)
kPositionZ	8	
kNormalX	0x10	
kNormalY	0x20	Normal vector of the point seen (if not background)
kNormalZ	0x40	
kAlpha	0x80	Rate of object seen (0 = background, max. val. = object, intermediate val. = frontier)
kIndex	0x100	Index of the UV space seen
kSurfaceU	0x200	UV space position
kSurfaceV	0x400	
kShaderColor	0x800	Shader channels
kShaderSpecularColor	0x1000	
kShaderSpecularSizeValue	0x2000	
kShaderAmbientValue	0x4000	
kShaderLambertValue	0x8000	
kShaderReflectionColor	0x10000	
kShaderTranparencyColor	0x20000	
kShaderRefractionValue	0x40000	
kShaderGlowColor	0x80000	

renderer

This element is not available yet.

Return Value S_OK

I3DExPostRenderer::Filter

HRESULT I3DExPostRenderer::Filter (*theScene, theImage, theBuffers, gd, renderer, renderHelper* **);**
I3DShScene* *theScene*
IShRasterOffscreen* *theImage*
IUnknown** *theBuffers*
IShGraphicDevice* *gd*
void* *renderer*
void* *renderHelper*

I3DExPostRenderer::Filter is called by the 3D Shell after the rendering to process to the filtering. The extension uses *theImage* and the graphic device *gd* to get the colors of the resulting image from the rendering and sets the new ones. It uses also the informations contained in *theBuffers.*

©1995-1997 MetaCreations. All rights reserved.

Parameters

theScene
This is a pointer on the scene, used to get informations on the elements.

theImage
This is a pointer on the image resulting from the rendering or the last filter. This is also used to draw the new image.

theBuffers
This is a table of pointers on buffers for each informations you asked for.

Buffer	Name	Meaning
theBuffers[0]	Distance	Distance from the camera to the object seen at this point (max. val. = background).
theBuffers[1]	Position X	
theBuffers[2]	Position Y	Position of the point seen (if not background)
theBuffers[3]	Position Z	
theBuffers[4]	Normal X	
theBuffers[5]	Normal Y	Normal vector of the point seen (if not background)
theBuffers[6]	Normal Z	
theBuffers[7]	Alpha	Rate of object seen (0 = background, max. val. = object, intermediate val. = frontier)
theBuffers[8]	Index	Index of the UV space seen
theBuffers[9]	Surface U	UV space position
theBuffers[10]	Surface V	
theBuffers[11]	Color	Shader channels
theBuffers[12]	Specular Color	
theBuffers[13]	Specular Size Value	
theBuffers[14]	Ambient Value	
theBuffers[15]	Lambert Value	
theBuffers[16]	Reflection Color	
theBuffers[17]	Transparency Color	
theBuffers[18]	Refraction Value	
theBuffers[19]	Glow Color	

gd
This is a pointer on a graphic device that allows you to process some operations with the image as drawing on screen.

renderer
This element is not available yet.

renderHelper
Keep this pointer and pass it to all the 3D Shell functions you call that ask for it.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.16.I3DExDeformer Interfac e

Family ID: ‘defo’
Interface ID: IID_I3DExDeformer
Interface file: I3DExDfr.h

The **I3DExDeforme r** Interface is used to apply geometric deformations on objects. These deformations can happen on 3D facets or on 3D bicubic patches. Therefore there are two main calls to implement or you can implement a simple function to deform a point and there is default functions to deform patches and facets.

```
DECLARE_INTERFACE_(I3DExDeformer, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExDeformer methods
  STDMETHOD(SetBBox)(BOX3D* bbox) PURE;
  STDMETHOD(DeformPoint) (THIS_ VECTOR3D *point, VECTOR3D *result) PURE;
  STDMETHOD(DeformFacets) (THIS_ IShIterator* iterator, EnumFacetsCall-
      back callback, void* privData) PURE;
  STDMETHOD(DeformPatches) (THIS_ IShIterator* iterator, EnumPatches-
      Callback callback, void* privData) PURE;
  };
```

I3DExDeformer::SetBBox

HRESULT I3DExDeformer::SetBBox (*bbox***);**
BOX3D* *bbox*

I3DExDeformer::SetBBo x is called by the 3D Shell to tell the 3D Deformer in which boundaries the 3D data to deform is going to be.

Parameter *bbox*
The 3D bounding box of the 3D data to be deformed.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DExDeformer::DeformPoint

HRESULT I3DExDeformer::DeformPoint (*point, result***);**
**VECTOR3D ****point*
**VECTOR3D ****result*

I3DExDeformer::DeformPoin t is called by the 3D Shell if the **DeformPatches** and **DeformFacets** are not implemented, to use the default patches and facets deform functions.

Parameter

point
The 3D point to be deformed, in the coordinate relative to the bounding box.

result
The 3D point after the deformation.

Return Value S_OK

I3DExDeformer::DeformFacets

HRESULT I3DExDeformer::DeformFacets (*iterator, callback, privData***);**
IShIterator* *iterator*
EnumFacetsCallback *callback*
void* *privData*

I3DExDeformer::DeformFacet s is called to deform or create 3D facets.

Parameters

iterator
This interface is used to get all the facets to deform. (see IShIterator interface)

callback
Call the *callback* procedure for each deformed or new Facet. Its interface is as follows:

```
void EnumFacetsCallback(FACET3D* facet, void* privData);
```

privData
Give back this pointer to *callback* when calling it.

Return Value S_OK

Comments You have to use the *iterator* (see **IShIterator** interface) to get all the facets one by one. And you have to call the *callback* function for each facet you want to return.

©1995-1997 MetaCreations. All rights reserved.

I3DExDeformer::DeformPatches

HRESULT I3DExDeformer::DeformPatches (*iterator, callback, privData***);**
IShIterator* *iterator*
EnumPatchesCallback *callback*
void* *privData*

I3DExDeformer::DeformPatche s is called to deform or create 3D bicubic patches.

Parameters

iterator
This interface is used to get all the patches to deform. (see IShIterator interface)

callback
Call the *callback* procedure for each Patch. Its interface is as follows:

```
void EnumPatchesCallback(PATCH3D* patch, void* privData);
```

privData
Give back this pointer to *callback* when calling it.

Return Value S_OK

Comments You have to use the *iterator* (see **IShIterator** interface) to get all the patches one by one. And you have to call the *callback* function for each patch you want to return.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.17.I3DExMotionLink Interfac e

Family ID: ‘link’
Interface ID: IID_I3DExMotionLink
Interface file: I3DExLnk.h

The **I3DExMotionLin k** Interface defines motion links between Tree Elements. It allows to define “mechanical” constrains (degrees of freedom) between one Tree Element and its father.

In order to animate the link, you have to put each freedom value (one for each degrees of freedom) in the “pMAP”, even if they not appear in the User Interface.

```
DECLARE_INTERFACE_(I3DExMotionLink, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExMotionLink methods
  STDMETHOD_(short, GetNbrFreedom) (THIS) PURE;
  STDMETHOD(IncrementFreedomValue) (THIS_ short index, NUM3D* increment)
      PURE;
  STDMETHOD(GetFreedomRange) (THIS_ short index, NUM3D* min, NUM3D* max)
      PURE;
  STDMETHOD(GetTransform) (THIS_ TRANSFORM3D* transform) PURE;
  STDMETHOD(GetTransformPartialDerivate)(THIS_ short index, TRANSFORM3D*
      transform) PURE;
  };
```

I3DExMotionLink::GetNbrFreedom

short I3DExMotionLink::GetNbrFreedom() ;

I3DExMotionLink::GetNbrFreedo m is called by the 3D Shell to learn how many degrees of freedom the Motion Link has.

Return Value Number of degrees of freedom, between 0 and 6.

©1995-1997 MetaCreations. All rights reserved.

For example, a link that does not allow any movement, will have 0 degree of freedom. If it allows only the translation along one axis, it will have 1 degree of freedom.

I3DExMotionLink::IncrementFreedomValu e

HRESULT I3DExMotionLink::IncrementFreedomValue (*index, increment***);**
short *index*
NUM3D* *increment*

I3DExMotionLink::IncrementFreedomValu e is called by the 3D Shell to set a new position by incrementing the freedom value

Parameters *index*
Freedom degree index

increment
The Freedom Value will be incremented by *increment*

Return Valu e S_OK

I3DExMotionLink::GetFreedomRang e

HRESULT I3DExMotionLink::GetFreedomRange (*index, min, max***);**
short *index*
NUM3D* *min*
NUM3D* *max*

I3DExMotionLink::GetFreedomRang e is called by the 3D Shell to get the range of possible values for a given freedom degree.

Parameters *index*
Freedom degree index

min
Return the minimum value here.

max
Return the maximum value here.

Return Valu e S_OK

Comments The range is given relatively to the value of the freedom degree. For example, if a freedom degree has a value of 1.0 and the absolute range is -2.0 to 2.0, this function will return min=-3.0 and max=1.0.

©1995-1997 MetaCreations. All rights reserved.

I3DExMotionLink::GetTransform

HRESULT I3DExMotionLink::GetTransform (*transform***);**
TRANSFORM3D* *transform*

I3DExMotionLink::GetTransfor m is called when the shell want to know the new orientation and position of the tree element due to the freedom values.

Parameter *transform*
New transformation to set the position and the orientation of the object. (Note: when the freedom values are initialized the rotation matrix have to be the identity matrix and the translation vector have to be null.)

Return Valu e S_OK

I3DExMotionLink::GetTransformPartialDerivat e

HRESULT I3DExMotionLink::GetTransformPartialDerivate (*index, transform***)**
short *index*
TRANSFORM3D* *transform*

This call is useful to the inverse kinematics. It's allow you to drag an object and see its move on the "path" of your link.

Parameters *index*
Freedom degree index

ransform
This transform is the derivative of the vector and the matrix by the freedom degree given by index.

Return Valu e S_OK

In mathematics the partial derivative is calculated as follow :

If you have the freedom degrees like these : α1, α2, …, αn
the partial derivative for the freedom degree number i, will be :

$$Rot = \begin{bmatrix} \frac{i_x}{i} & \frac{j_x}{i} & \frac{k_x}{i} \\ \frac{i_y}{i} & \frac{j_y}{i} & \frac{k_y}{i} \\ \frac{i_z}{i} & \frac{j_z}{i} & \frac{k_z}{i} \end{bmatrix} \text{ et } Tr = \begin{bmatrix} \frac{t_x}{i} \\ \frac{t_y}{i} \\ \frac{t_z}{i} \end{bmatrix}$$

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.18.I3DExTreeBehavior Interfac e

Family ID: ‘treb’
Interface ID: IID_I3DExTreeBehavior
Interface file: I3DExTBh.h

The **I3DExTreeBehavio r** Interface is used to add any kind of additional behavior to a Tree Element. This is a little like multiple inheritance in C++: new features are added to an object.

```
DECLARE_INTERFACE_(I3DExTreeBehavior, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExTreeBehavior methods
  STDMETHOD(Apply) (THIS_ I3DShTreeElement* tree) PURE;
  STDMETHOD(SetGlobalTransform) (THIS_ I3DShTreeElement *atree,const
      MATRIX3D *RR,const VECTOR3D *TT,short mode) PURE;
};
```

I3DExTreeBehavior::Apply

HRESULT I3DExTreeBehavior::Apply (*tree***);**
I3DShTreeElement* *tree*

I3DExTreeBehavior::Appl y is called by the 3D Shell to apply the Behavior to a Tree Element.

Parameter *tree*
The Tree Element on which to apply the Behavior.

Return Valu e S_OK

Comments **I3DExTreeBehavior::Appl y** is called after a user command or after a change in the time line.

©1995-1997 MetaCreations. All rights reserved.

I3DExTreeBehavior::SetGlobalTransform

HRESULT I3DExTreeBehavior::SetGlobalTransform (*atree, RR, TT, mode***);**
I3DShTreeElement* *atree*
MATRIX3D* *RR*
VECTOR3D* *TT*
short *mode*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

©1995-1997 MetaCreations. All rights reserved.

1.19.I3DExTweener Interfac e

Family ID: ‘twee’
Interface ID: IID_I3DExTweener
Interface file: I3DExTwn.h

The **I3DExTweener** Interface defines Animation Tweeners. Tweeners are 3D Components that describe the interpolation between key frames.

```
DECLARE_INTERFACE_(I3DExTweener, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExTweener methods
  STDMETHOD(DoTweening)(THIS_ NUM3D *result, long time, long time1, long
      time2) PURE;
  STDMETHOD(DoTween)(THIS_ IShKeyFrame *res,long time,XTweenerChainLink
      *alink) PURE;
  STDMETHOD(Draw)(THIS_ IShGraphicDevice *aGD,const LRect *area,const
      LRect *where) PURE;
  STDMETHOD(DoValue)(THIS_ NUM3D *res,long time,XTweenerChainLink
      *alink,I3DShTreeElement *tree) PURE;
  STDMETHOD(DoVector2)(THIS_ VECTOR2D *res,long time,XTweenerChainLink
      *alink,I3DShTreeElement *tree) PURE;
  STDMETHOD(DoVector3)(THIS_ VECTOR3D *res,long time,XTweenerChainLink
      *alink,I3DShTreeElement *tree) PURE;
  STDMETHOD(DoColor)(THIS_ COLOR3D *res,long time,XTweenerChainLink
      *alink,I3DShTreeElement *tree) PURE;
  STDMETHOD(DoRotation)(THIS_ MATRIX3D *res,long time,XTweenerChainLink
      *alink,I3DShTreeElement *tree) PURE;  };
```

I3DExTweener::DoTweening

HRESULT I3DExTweener::DoTweening (*result, time, time1,time2***);**
NUM3D* *result*
long *time*
long *time1*
long *time2*

©1995-1997 MetaCreations. All rights reserved.

I3DExTweener::DoTweenin g is called by the 3D Shell to perform the interpolation at a given time.

Parameters

result
A value that tells the shell where to put the frame by interpolating linearly between the two extreme keyframes.

time
The time when to calculate the animation. Its unit is the ticks

time1
The time of the first keyframe.

time2
The time of the second keyframe.

Return Value S_OK

Comments when *time* is equal to *time1*, you should return *result*=0.0 and when *time* is equal to *time2*, you should return *result*=1.0.

For a linear tweener, the formula is :
**result*=FixedToQuickFix(*time-time1*)/FixedToQuickFix(*time2-time1*);

I3DExTweener::DoTween

HRESULT I3DExTweener::DoTween (*res, time, alink***);**
IShKeyFrame* *res*
long *time*
XTweenerChainLink* *alink*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExTweener::Draw

HRESULT I3DExTweener::Draw (*aGD, area, where***);**
IShGraphicDevice* *aGD*
LRect* *area*
LRect* *where*

This call is not used yet by the application, you must return **E_NOTIMPL**.

©1995-1997 MetaCreations. All rights reserved.

Return Value E_NOTIMPL

I3DExTweener::DoValue

HRESULT I3DExTweener::DoValue (*res, time, alink, tree***);**
NUM3D* *res*
long *time*
XTweenerChainLink* *alink*
I3DShTreeElement* *tree*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExTweener::DoVector2

HRESULT I3DExTweener::DoVector2 (*res, time, alink, tree***);**
VECTOR2D* *res*
long *time*
XTweenerChainLink* *alink*
I3DShTreeElement* *tree*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExTweener::DoVector3

HRESULT I3DExTweener::DoVector3 (*res, time, alink, tree***);**
VECTOR3D* *res*
long *time*
XTweenerChainLink* *alink*
I3DShTreeElement* *tree*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExTweener::DoColor

HRESULT I3DExTweener::DoColor (*res, time, alink, tree***);**
COLOR3D* *res*
long *time*
XTweenerChainLink* *alink*
I3DShTreeElement* *tree*

©1995-1997 MetaCreations. All rights reserved.

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

I3DExTweener::DoRotation

HRESULT I3DExTweener::DoRotation (*res, time, alink, tree***);**
MATRIX3D* *res*
long *time*
XTweenerChainLink* *alink*
I3DShTreeElement* *tree*

This call is not used yet by the application, you must return **E_NOTIMPL**.

Return Value E_NOTIMPL

©1995-1997 MetaCreations. All rights reserved.

1.20.I3DExSceneOperation Interfac e

Family ID : ‘scop’
Interface ID : IID_I3DExSceneOperation
Interface file : I3DExScO.h

The **Scene Operatio n Interface** defines a new feature of the 3D Shell. The user will directly access to this function in the ‘Arrange’ menu of the 3D Shell. It allows you to do everything you want in the scene (create new objects, rotate or translate an object or a list of objects).

```
DECLARE_INTERFACE_(I3DExSceneOperation, I3DExDataExchanger) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExSceneOperation methods
  STDMETHOD(Prepare) (THIS_ I3DShScene* scene, I3DShTreeElement* tree,
      long index, long total) PURE;
  STDMETHOD_(Boolean, DoIt) (THIS_ I3DShScene* scene, I3DShTreeElement*
      tree, long index, long total) PURE;
  };
```

I3DExSceneOperation::Prepare

HRESULT I3DExSceneOperation::Prepare (*scene, tree, index, total***)**
I3DShScene* *scene*
I3DShTreeElement* *tree*
long *index*
long *total*

I3DExSceneOperation::Prepar e allows you to prepare the data of the scene operation before the 3D Shell shows the dialog to set those parameters.

Parameters → *scene*
This is a COM object of the current scene.

©1995-1997 MetaCreations. All rights reserved.

tree
This is a COM object of a tree of the current selection in the scene.

index
Index of the current tree in the multiple selection.

total
Number of sub-tree root selected.

Return Value S_OK

Comments If there is no selection *index* and *total* are equal to 0. When you select an objet in the shell, all the sub-tree are in the same selection.

I3DExSceneOperation::DoIt

BOOLEAN I3DExSceneOperation::DoIt (*scene, tree, index, total***)**
I3DShScene* *scene*
I3DShTreeElement* *tree*
long *index*
long *total*

I3DExSceneOperation::DoI t is called by the 3D Shell for each selected tree.

Parameters

scene
This is a COM object of the current scene.

tree
This is a COM object of a tree of the current selection in the scene.

index
Index of the current tree in the multiple selection.

total
Number of sub-tree root selected.

Return Value

Value	Meaning
TRUE	the scene has changed
FALSE	the scene hasn't changed

Comments If there is no selection, *index* and *total* are equal to 0. When you select an objet in the shell, all the sub-tree are in the same selection.

©1995-1997 MetaCreations. All rights reserved.

1.21.I3DExSceneOperation8 Interfac e

Family ID : ‘scop’
Interface ID : IID_I3DExSceneOperation8
Interface file : I3DExScO.h

The **Scene Operation Interfac e** for API version 8 is the **Scene Operation Interfac e** with **Undo** and **Redo** methods added. You can also use the previous interface if you do not provide **Undo** and **Redo** methods.

```
DECLARE_INTERFACE_(I3DExSceneOperation8, I3DExSceneOperation) {
   // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DExtension methods
  STDMETHOD_(I3DExtension*, Clone) (THIS) PURE;
  STDMETHOD(ShellUtilitiesInit) (THIS_ IShUtilities* shellUtilities)
      PURE;

  // I3DExDataExchanger methods
  STDMETHOD_(ExtensionDataMap*, GetExtensionDataMap) (THIS) PURE;
  STDMETHOD_(void*, GetExtensionDataBuffer) (THIS) PURE;
  STDMETHOD(ExtensionDataChanged) (THIS) PURE;
  STDMETHOD(HandleEvent) (THIS_ ULONG sourceID) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;

  // I3DExSceneOperation methods
  STDMETHOD(Prepare) (THIS_ I3DShScene* scene, I3DShTreeElement* tree,
      long index, long total) PURE;
  STDMETHOD_(Boolean, DoIt) (THIS_ I3DShScene* scene, I3DShTreeElement*
      tree, long index, long total) PURE;

  // I3DExSceneOperation methods API 8
  STDMETHOD(Prepare8) (THIS_ I3DShScene* scene, I3DShTreeElement* tree,
      long index, long total, Boolean* canUndo) PURE;
  STDMETHOD_(Boolean, UndoIt) (THIS_ I3DShScene* scene,
      I3DShTreeElement* tree, long index, long total) PURE;
  STDMETHOD_(Boolean, RedoIt) (THIS_ I3DShScene* scene,
      I3DShTreeElement* tree, long index, long total) PURE;
  };
```

I3DExSceneOperation::Prepare8

HRESULT I3DExSceneOperation::Prepare8 (*scene, tree, index, total, canUndo***)**
I3DShScene* *scene*
I3DShTreeElement* *tree*
long *index*
long *total*
Boolean* *canUndo*

©1995-1997 MetaCreations. All rights reserved.

I3DExSceneOperation::Prepare 8 allows you to prepare the data of the scene operation before the 3D Shell shows the dialog to set those parameters. It also tells to the shell if the **Undo** and **Redo** methods are provided or not.

Parameters

scene
This is a COM object of the current scene.

tree
This is a COM object of a tree of the current selection in the scene.

index
Index of the current tree in the multiple selection.

total
Number of sub-tree root selected.

canUndo
Return TRUE if **Undo** and **Redo** methods are provided.

Return Value S_OK

I3DExSceneOperation::UndoIt

BOOLEAN I3DExSceneOperation::UndoIt (*scene, tree, index, total***)**
I3DShScene* *scene*
I3DShTreeElement* *tree*
long *index*
long *total*

I3DExSceneOperation::UndoI t is called by the 3D Shell for each selected tree when the user undo a scene operation.

Parameters

scene
This is a COM object of the current scene.

tree
This is a COM object of a tree of the current selection in the scene.

index
Index of the current tree in the multiple selection.

total
Number of sub-tree root selected.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Meaning
TRUE	the scene has changed
FALSE	the scene hasn't changed

Comments If there is no selection, *index* and *total* are equal to 0. When you select an objet in the shell, all the sub-tree are in the same selection.

I3DExSceneOperation::RedoIt

BOOLEAN I3DExSceneOperation::RedoIt (*scene, tree, index, total***)**
I3DShScene* *scene*
I3DShTreeElement* *tree*
long *index*
long *total*

I3DExSceneOperation::RedoI t is called by the 3D Shell for each selected tree when the user redo a scene operation.

Parameters

scene
This is a COM object of the current scene.

tree
This is a COM object of a tree of the current selection in the scene.

index
Index of the current tree in the multiple selection.

total
Number of sub-tree root selected.

Return Value

Value	Meaning
TRUE	the scene has changed
FALSE	the scene hasn't changed

Comments If there is no selection, *index* and *total* are equal to 0. When you select an objet in the shell, all the sub-tree are in the same selection.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

1.22.I3DExRenderer Interfac e

Family ID : ‘rend’
Interface ID : IID_I3DExRenderer
Interface file : I3DExRen.h

The **Renderer Interfac e** defines the renderers. A renderer is building a 2D image from a 3D scene. The user will be able to choose the renderer in the scene settings.
Once the image is rendered, the post render filters will be applied.

```
DECLARE_INTERFACE_(I3DExRenderer, I3DExDataExchanger) {
   STDMETHOD(SetTreeTop) (I3DShGroup* treeTop) PURE;
   STDMETHOD(SetCamera) (I3DShCamera* camera) PURE;
   STDMETHOD(SetEnvironment) (I3DShEnvironment* environment) PURE; //
      replace the 3 next functions
   STDMETHOD(SetAmbientLight) (const COLOR3D &ambiantColor) PURE;
   STDMETHOD(Dehydrate) (short level) PURE;
   STDMETHOD_(ULONG, GetRenderingTime) () PURE;
  STDMETHOD(SetOffscreen) (IShRasterOffscreen* offscreen) PURE;
  STDMETHOD(SetFieldRenderingData) (Boolean useFieldRendering,short
      firstFrame) PURE;
  STDMETHOD(GetRenderViewData) (short& viewAndPMapID,// ID of the View
      and PMap resources
                                                    void*& dataBuffer)
      PURE; // Data buffer used to communicate with the view.
   STDMETHOD(PrepareDraw) (const RECT3D &totalarea,const VECTOR2D &off-
      screenOffset,NUM3D zoom, BOOLEAN (*callback) (short vv, void
      *priv),void *priv) PURE;
   STDMETHOD(DrawRect) (short mode,const RECT3D &area) PURE;
   STDMETHOD_(BOOL, IsSMPAware) () PURE;
   STDMETHOD(GetPreferredTiling) (const RECT3D &area, POINT3D & tile-
      Size) PURE;
   STDMETHOD(UpdateBatchQDataList) (RDList* batchQDataList,short
      index,void* dataBuffer) PURE;
  };
```

I3DExRenderer::SetTreeTo p

HRESULT I3DExRenderer::SetTreeTop (*treeTop***)**
I3DShGroup* *treeTop*

I3DExRenderer::SetTreeTo p is called by the 3D shell before rendering to specify which tree needs to be rendered.

Parameters *treeTop*
Store this group, you will have to render it.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DExRenderer::SetCamera

HRESULT I3DExRenderer::SetCamera (*camera***)**
I3DShCamera* *camera*

I3DExRenderer::SetCamer a is called by the 3D shell before rendering to specify the rendering camera that need to be used.

Parameters *camera*
Store this camera, you will have to render from it.

Return Value S_OK

I3DExRenderer::SetEnvironmen t

HRESULT I3DExRenderer::SetEnvironment (*environment***)**
I3DShEnvironment* *environment*

I3DExRenderer::SetEnvironmen t is called by the 3D shell before rendering to give the background, the backdrop and the atmospheric shader of the scene.

Parameters *environment*
The background, the backdrop and the atmospheric shader of the scene.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

1.23.I3DExGBufferRenderer Interfac e

Family ID : ‘rend’
Interface ID : IID_I3DExGBufferRenderer
Interface file : I3DExRen.h

The **GBuffer Renderer Interfac e** is associated with the renderer interface. This interface is used to store the GBuffers the renderer will have to fill.

```
DECLARE_INTERFACE_(I3DExGBufferRenderer, IUnknown) {
   STDMETHOD(Set32BitGBuffers) (RT32Data* bufferData) PURE;
   STDMETHOD(Set8BitGBuffers) (RTBuffer* bufferData) PURE;
   };
```

I3DExGBufferRenderer::Set32BitGBuffer s

HRESULT I3DExRenderer::Set32BitGBuffers (*bufferData***)**
RT32Data* *bufferData*

I3DExRenderer::Set32BitGBuffer s is called by the 3D shell before rendering to provide the buffers that need to be filled.

Parameters → *bufferData*
This is a list of pointers to 32 bits GBuffers. You will have to fill the G Buffers whose pointer is not NULL.

Return Valu e S_OK

I3DExGBufferRenderer::Set8BitGBuffer s

HRESULT I3DExRenderer::Set8BitGBuffers (*bufferData***)**
RTBuffer* *bufferData*

I3DExRenderer::SetBitGBuffer s is called by the 3D shell to provide the 8 bits G buffers.

Parameters → *bufferData*
This is a list of pointers to 8 bits GBuffers. this is working the same way as the 32 bits GBuffers.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

Chapter 2 - 3D Shell Reference

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.1.I3DShTreeElement Interfac e

Interface ID: IID_I3DShTreeElement
Interface file: I3DShTre.h

The **I3DShTreeElemen t** interface is the abstract interface of all elements in the tree (called "Tree Elements"). Those Tree Elements can be groups, object instances, light sources, cameras, sub-scenes, etc.

```
DECLARE_INTERFACE_(I3DShTreeElement, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShTreeElement methods
  STDMETHOD(SetScene) (THIS_ I3DShScene* scene) PURE;// This must be
      called after creating a Tree Element
  STDMETHOD_(I3DShScene*, GetScene) (THIS) PURE;
  //-- Naming
  STDMETHOD(SetName) (THIS_ char* name) PURE;
  STDMETHOD(GetName) (THIS_ char* name) PURE;
  //-- Tree navigation
  STDMETHOD_(I3DShTreeElement*, GetFirst) (THIS) PURE; // Gives first
      son
  STDMETHOD_(I3DShTreeElement*, GetLast) (THIS) PURE;  // Gives last son
  STDMETHOD_(I3DShTreeElement*, GetLeft) (THIS) PURE;  // Gives left
      brother
  STDMETHOD_(I3DShTreeElement*, GetRight) (THIS) PURE; // Gives right
      brother
  STDMETHOD_(I3DShTreeElement*, GetTop) (THIS) PURE;   // Gives father
  //-- Tree manipulation
  STDMETHOD_(I3DShTreeElement*, Unlink) (THIS) PURE;   // Unhook tree
      element from father, and returns old father
  STDMETHOD(InsertFirst) (THIS_ I3DShTreeElement* newSon) PURE;    //
      Insert as the first son
  STDMETHOD(InsertLast) (THIS_ I3DShTreeElement* newSon) PURE;     //
      Insert as the last son
  STDMETHOD(InsertLeft) (THIS_ I3DShTreeElement* newBrother) PURE; //
      Insert as the first left brother
  STDMETHOD(InsertRight) (THIS_ I3DShTreeElement* newBrother) PURE;//
      Insert as the first right brother
  //-- Tree transformations
  STDMETHOD(GetTransform) (THIS_ TREETRANSFORM3D* treeTransform)
      PURE;         // Local transform
  STDMETHOD(SetTransform) (THIS_ TREETRANSFORM3D* treeTransform)
      PURE;         // Local transform
  STDMETHOD(GetGlobalTransform) (THIS_ TREETRANSFORM3D* treeTransform)
      PURE;   // Global transform
  STDMETHOD(SetGlobalTransform) (THIS_ TREETRANSFORM3D* treeTransform)
      PURE;   // Global transform
  //-- Hot point
  STDMETHOD(GetHotPoint) (THIS_ VECTOR3D* hotPoint) PURE;// In local
      coordinates
  STDMETHOD(SetHotPoint) (THIS_ VECTOR3D* hotPoint) PURE;// In local
```

©1995-1997 MetaCreations. All rights reserved.

```
    coordinates
STDMETHOD(CenterElementOnHotPoint) (THIS) PURE; // This moves the tree
    element
STDMETHOD(CenterHotPointOnElement) (THIS) PURE; // This moves the hot
    point
STDMETHOD(TranslateHotPoint) (THIS_ VECTOR3D* delta) PURE;  // Trans-
    lates hotpoint of delta (In local coordinates)
//-- Misc.
STDMETHOD_(I3DShTreeElement*, Clone) (THIS_ BOOLEAN cloneAll) PURE;
STDMETHOD(GetBBox) (THIS_ BOX3D* bbox, BOOLEAN excludeLightsAndCam-
    eras) PURE;
STDMETHOD(BeginMove) (THIS) PURE;
STDMETHOD(EndMove) (THIS) PURE;

//-- XYZ Scaling
STDMETHOD(GetXYZScaling) (THIS_ VECTOR3D* xyzScaling) PURE;
STDMETHOD(SetXYZScaling) (THIS_ VECTOR3D* xyzScaling) PURE;

//IK
STDMETHOD_(short,GetNbrFreedom)(THIS) PURE;
STDMETHOD(GetFreedomRange)(THIS_ short index,NUM3D *min,NUM3D *max)
    PURE;
STDMETHOD(GetFreedomTransform)(THIS_ MATRIX3D *RR,VECTOR3D *TT) PURE;
STDMETHOD(GetFreedomDerivative)(THIS_ short index,MATRIX3D
    *RR,VECTOR3D *TT) PURE;
STDMETHOD(IncrementFreedomValue)(THIS_ short index,NUM3D *value) PURE;

STDMETHOD(GetFullName) (THIS_ char* fullName) PURE;   // Returns the
    "path name" of the tree element (nodes separated by kPathNameSepa-
    rator)

//-- Tree transformations API8 (debugged)
STDMETHOD(GetTransform8) (THIS_ TREETRANSFORM3D* treeTransform) PURE;
STDMETHOD(SetTransform8) (THIS_ TREETRANSFORM3D* treeTransform) PURE;
STDMETHOD(GetGlobalTransform8) (THIS_ TREETRANSFORM3D* treeTransform)
    PURE;
STDMETHOD(SetGlobalTransform8) (THIS_ TREETRANSFORM3D* treeTransform)
    PURE;
};
```

I3DShTreeElement::SetScene

HRESULT I3DShTreeElement::SetScene (*scene***);**
I3DShScene* *scene*

I3DShTreeElement::SetScen e must be called after creating any Tree Element to set which Scene it belongs to.

Parameter → *scene*
Points to the Scene the Tree Element belongs to.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::GetScene

I3DShScene* I3DShTreeElement::GetScene() ;

I3DShTreeElement::GetScen e is used to get the Scene it belongs to.

Return Value The scene the tree element belongs to.

I3DShTreeElement::SetName

HRESULT I3DShTreeElement::SetName (*name***);**
char* *name*

Use I3DShTreeElement::SetName to change the name of a Tree Element.

Parameter → *name*
The new name. 31 characters maximum.

Return Value S_OK

Note The name pointer is not kept: the string content it points to is copied inside the I3DShTreeElement internal space.

I3DShTreeElement::GetName

HRESULT I3DShTreeElement::GetName (*name***);**
char* *name*

Use **I3DShTreeElement::GetNam e** to get the name of a Tree Element.

Parameter ← *name*
Points to where to store the name. 31 characters maximum.

Return Valu e S_OK

I3DShTreeElement::GetFirst

I3DShTreeElement* I3DShTreeElement::GetFirst() ;

Use **I3DShTreeElement::GetFirs t** to get the first son of a Tree Element.

Return Valu e A pointer to the first son of the Tree Element. NULL if there is no son.

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::GetLast

I3DShTreeElement* I3DShTreeElement::GetLast() ;

Use **I3DShTreeElement::GetLas t** to get the last son of a Tree Element.

Return Value A pointer to the last son of the Tree Element. NULL if there is no son.

I3DShTreeElement::GetLeft

I3DShTreeElement* I3DShTreeElement::GetLeft() ;

Use **I3DShTreeElement::GetLef t** to get immediate left brother of a Tree Element.

Return Value A pointer to the immediate left brother of the Tree Element. NULL if there is none.

I3DShTreeElement::GetRight

I3DShTreeElement* I3DShTreeElement::GetRight() ;

Use **I3DShTreeElement::GetRigh t** to get immediate right brother of a Tree Element.

Return Value A pointer to the immediate right brother of the Tree Element. NULL if there is none.

I3DShTreeElement::GetTop

I3DShTreeElement* I3DShTreeElement::GetTop() ;

Use **I3DShTreeElement::GetTo p** to get the father of a Tree Element.

Return Value A pointer to the father of the Tree Element. NULL if there is none.

I3DShTreeElement::Unlink

I3DShTreeElement* I3DShTreeElement::Unlink() ;

Use **I3DShTreeElement::Unlin k** to unhook a Tree Element from its tree.

Return Value A pointer to the former father.

Comments It removes the Tree Element from its father's sons list. The Tree Element has no father anymore (I3DShTreeElement::GetTop will return NULL). Of course, the whole sub-tree under the Tree Element is un-hooked along with it.

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::InsertFirst

HRESULT I3DShTreeElement::InsertFirst (*newSon***);**
I3DShTreeElement* *newSon*

Use **I3DShTreeElement::InsertFirs t** to insert another Tree Element as the first son of the Tree Element.

Parameter *newSon*
Points to the Tree Element to insert.

Return Value S_OK

I3DShTreeElement::InsertLast

HRESULT I3DShTreeElement::InsertLast (*newSon***);**
I3DShTreeElement* *newSon*

Use **I3DShTreeElement::InsertLas t** to insert another Tree Element as the last son of the Tree Element.

Parameter *newSon*
Points to the Tree Element to insert.

Return Value S_OK

I3DShTreeElement::InsertLeft

HRESULT I3DShTreeElement::InsertLeft (*newBrother***);**
I3DShTreeElement* *newBrother*

Use **I3DShTreeElement::InsertLef t** to insert another Tree Element to the immediate left of the Tree Element.

Parameter *newBrother*
Points to the Tree Element to insert.

Return Value S_OK

I3DShTreeElement::InsertRight

HRESULT I3DShTreeElement::InsertRight (*newBrother***);**
I3DShTreeElement* *newBrother*

Use **I3DShTreeElement::InsertRigh t** to insert another Tree Element to the immediate right of the Tree Element.

©1995-1997 MetaCreations. All rights reserved.

Parameter *newBrother*
Points to the Tree Element to insert.

Return Value S_OK

I3DShTreeElement::GetTransform

HRESULT I3DShTreeElement::GetTransform (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::GetTransfor m** to get the Combined Local Transformation of the Tree Element.

Parameters *treeTransform*
Points to where to store the Tree Transformation.

Return Value S_OK

Comments The Local Transformation defines the positioning, orientation, and scaling of the Tree Element in the coordinates system of its ancestor. See **GetTransform8** for more details.

I3DShTreeElement::SetTransform

HRESULT I3DShTreeElement::SetTransform (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::SetTransfor m** to set the Local Transformation of the Tree Element.

Parameters *treeTransform*
Points to the new Tree Transformation.

Return Value S_OK

Comments The Local Transformation defines the positioning, orientation, and scaling of the Tree Element in the coordinates system of its ancestor. SetTransform keep the Hot Point position unchanged relatively to this Tree Element.

I3DShTreeElement::GetGlobalTransform

HRESULT I3DShTreeElement::GetGlobalTransform (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::GetGlobalTransfor m** to get the Combined Global Transformation of the Tree Element.

©1995-1997 MetaCreations. All rights reserved.

Parameter ⇨ *treeTransform*
Points to where to store the Tree Transformation.

Return Value S_OK

Comments The Global Transformation defines the positioning, orientation, and scaling of the Tree Element in the Global Coordinates System. See **GetTransform8** for more details.

I3DShTreeElement::SetGlobalTransform

HRESULT I3DShTreeElement::SetGlobalTransform (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::SetGlobalTransfor m** to set the Global Transformation of the Tree Element.

Parameter ⇦ *treeTransform*
Points to the new Tree Transformation.

Return Value S_OK

Comments The Global Transformation defines the positioning, orientation, and scaling of the Tree Element in the Global Coordinates System. **SetGlobalTransfor m** don't keep the Hot Point position unchanged. The Hot Point is moved to the origin of the object.

I3DShTreeElement::GetHotPoint

HRESULT I3DShTreeElement::GetHotPoint (*hotPoint***);**
VECTOR3D* *hotPoint*

Call **I3DShTreeElement::GetHotPoin t** to get the Hot Point position. The Hot Point is the point around which rotations and scaling occur.

Parameters ⇨ *hotPoint*
Points to where to store the Hot Point coordinates.

Return Value S_OK

Comments The Hot Point coordinates are in the Local Coordinates System of the Tree Element.

I3DShTreeElement::SetHotPoint

HRESULT I3DShTreeElement::SetHotPoint (*hotPoint***);**
VECTOR3D* *hotPoint*

©1995-1997 MetaCreations. All rights reserved.

Call **I3DShTreeElement::SetHotPoin t** to set the Hot Point position. The Hot Point is the point around which rotations and scaling occur.

Parameters → *hotPoint*
Points to the new Hot Point coordinates.

Return Value S_OK

Comments The Hot Point coordinates are in the Local Coordinates System of the Tree Element. Note that changing the Hot Point position of a Tree Element changes the object global coordinates and not the Hot Point position in global coordinates.

I3DShTreeElement::CenterElementOnHotPoin t

HRESULT I3DShTreeElement::CenterElementOnHotPoint() ;

Call **I3DShTreeElement::CenterElementOnHotPoin t** to center the Tree Element on its Hot Point.

Return Value S_OK

Comments This actually moves the Tree Element in space so its bounding box is centered on its Hot Point.

I3DShTreeElement::CenterHotPointOnElemen t

HRESULT I3DShTreeElement::CenterHotPointOnElement() ;

Call **I3DShTreeElement::CenterHotPointOnElemen t** to center the Hot Point on the Tree Element.

Return Value S_OK

Comments This moves the Hot Point in space so it is moved to the center of the Tree Element bounding box.

I3DShTreeElement::TranslateHotPoin t

HRESULT I3DShTreeElement::TranslateHotPoint (*delta***);**
VECTOR3D* *delta*

Call **I3DShTreeElement::TranslateHotPoin t** to translate the Hot Point.

Parameters → *delta*
The translation values, in Local Coordinates.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments This call does the same thing as this:

```
I3DShTreeElement*  treeElement;
VECTOR3D  hpLocation;
VECTOR3D  delta;

treeElement->GetHotPoint(&hpLocation);
hpLocation += delta;
treeElement->SetHotPoint(&hpLocation);
```

Note that the Hot Point global coordinates don't change but the objects contained in this Tree Element are moving.

I3DShTreeElement::Clone

I3DShTreeElement* I3DShTreeElement::Clone() ;

Call **I3DShTreeElement::Clon e** to duplicate a Tree Element.

Return Value The cloned object of the Tree Element.

Comments All the sub-tree under the Tree Element is cloned as well. The new Tree Element is un-hooked from whatever tree the original Tree Element is in, and become an independent tree branch.

I3DShTreeElement::GetBBox

HRESULT I3DShTreeElement::GetBBox(*bbox*, *excludeLightsAndCameras*);
BOX3D* *bbox*
BOOLEAN *excludeLightsAndCameras*

Call **I3DShTreeElement::GetBBo x** to get the bounding box of a tree element.

Parameters *bbox*
Points to where to return the bounding box.

excludeLightsAndCameras
Give TRUE if you need to exclude all lights and cameras under the Tree Element. Give FALSE by default. This is useful to get a bounding box on only the geometric information of a tree hierarchy.

Return Value S_OK

Comments The bounding box is returned in Local Coordinates, and includes the bounding boxes of all sons.

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::BeginMove

HRESULT I3DShTreeElement::BeginMove() ;

I3DShTreeElement::BeginMov e is used by the Shell to know which Tree Element will change to perform some optimization in the drawing of the scene.
It must be called before you change the parameters of the Tree Element.

Return Value S_OK

I3DShTreeElement::EndMove

HRESULT I3DShTreeElement::EndMove() ;

I3DShTreeElement::EndMov e is used by the Shell to know which Tree Element moved to perform some optimization in the drawing of the scene.
It must be called after you have changed the parameters of the Tree Element.

Return Value S_OK

I3DShTreeElement::GetXYZScaling

HRESULT I3DShTreeElement::GetXYZScaling (*xyzScaling***);**
VECTOR3D* *xyzScaling*

Call **I3DShTreeElement::GetXYZScalin g** to get the scaling in the X, Y and Z directions of a tree element.

Parameter ⇨ *xyzScaling*
Points to where to return the X, Y and Z scaling.

Return Value S_OK

I3DShTreeElement::SetXYZScaling

HRESULT I3DShTreeElement::SetXYZScaling (*xyzScaling***);**
VECTOR3D* *xyzScaling*

Call **I3DShTreeElement::SetXYZScalin g** to set the scaling in the X, Y and Z directions of a tree element.

Parameter ⇦ *xyzScaling*
The new X, Y and Z scaling factors.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::GetNbrFreedom

short I3DShTreeElement::GetNbrFreedom() ;

Call **I3DShTreeElement::GetNbrFreedo m** to get the number of freedoms of a tree element. This is the number of possible translations and rotations. A totally free tree element has 6 freedoms.

Return Value Number of freedoms.

I3DShTreeElement::GetFreedomRange

HRESULT I3DShTreeElement::GetFreedomRange (*index, min, max***);**
short *index*
NUM3D* *min*
NUM3D* *max*

Call **I3DShTreeElement::GetFreedomRang e** to get the range of variation for a given freedom.

Parameters

index
Select the freedom.

min
Point to where to return the minimum value for the freedom.

max
Point to where to return the maximum value.

Return Value S_OK

I3DShTreeElement::GetFreedomTransform

HRESULT I3DShTreeElement::GetFreedomTransform (*RR, TT***);**
MATRIX3D* *RR*
VECTOR3D* *TT*

Call **I3DShTreeElement::GetFreedomTransfor m** to get the transformation of a tree element given by a behavior.

Parameters

RR
Points to where to return the rotation matrix.

TT
Points to where to return the translation vector.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

Comments The behavior sets its own transformation calculated with the freedom values.

I3DShTreeElement::GetFreedomDerivative

HRESULT I3DShTreeElement::GetFreedomDerivative (*index, RR,TT***);**
short *index*
MATRIX3D* *RR*
VECTOR3D* *TT*

Call **I3DShTreeElement::GetFreedomDerivativ e** to get the derivative of a tree element given by a behavior. This derivative is used to set the correct value for each freedom value.

Parameters

index
Select the freedom.

RR
Points to where to return the rotation matrix.

TT
Points to where to return the translation vector.

Return Value S_OK

Comments The derivative correspond to the difference of freedom transformation when the given freedom value increase with 1. Those informations are used by the 3D Shell to know which freedom value must be changed when the user drag the Tree Element.

I3DShTreeElement::IncrementFreedomValu e

HRESULT I3DShTreeElement::IncrementFreedomValue (*index, value***);**
short *index*
NUM3D* *value*

Call **I3DShTreeElement::IncreaseFreedomValu e** to increase the value of a given freedom.

Parameters

index
Select the freedom.

value
Points to the value to add to the freedom.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement::GetFullName

HRESULT I3DShTreeElement::GetFullName (*fullName***);**
char* *fullName*

Call **I3DShTreeElement::GetFullNam e** to get the full path name of a tree element. This is the name of the tree element and all the fathers starting from the universe.

Parameters ⇨ *fullName*
Points to where to return the full path name.

Return Value S_OK

Comments The tree element names are separated with the kPathNameSeparator constant.

I3DShTreeElement::GetTransform8

HRESULT I3DShTreeElement::GetTransform8 (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::GetTransform 8** to get the Local Transformation of the Tree Element.

Parameter ⇨ *treeTransform*
Points to where to store the Tree Transformation.

Return Value S_OK

Comments The Local Transformation defines the positioning, orientation, and scaling of the Tree Element in the coordinates system of its ancestor. The origin (the 0,0,0 coordinate in the local coordinate) is in T - S.R.HP in the local coordinate system of his ancestor, where S is the scaling value, R is the rotation matrix and HP is the hot point vector.
T is pointing to the hot point, the tree element is so rotated around this point using the R matrix and scaled using the S value. When using **GetTransform**, T is pointing to the origin.
The **Get** and **Set Transformations 8** have a more intuitive behavior than the older versions. A Set called with the returned values of a Get keep the tree element unchanged.

I3DShTreeElement::SetTransform8

HRESULT I3DShTreeElement::SetTransform8 (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::SetTransform 8** to set the Local Transformation of the Tree Element.

Parameter ⇦ *treeTransform*

©1995-1997 MetaCreations. All rights reserved.

Points to the new Tree Transformation.

Return Value S_OK

Comments The Local Transformation defines the positioning, orientation, and scaling of the Tree Element in the coordinates system of its ancestor. **SetTransform8** keep the Hot Point position unchanged relatively to this Tree Element. See GetTransform8 for more details.

I3DShTreeElement::GetGlobalTransform8

HRESULT I3DShTreeElement::GetGlobalTransform8 (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::GetGlobalTransform 8** to get the Global Transformation of the Tree Element.

Parameters ⇨ *treeTransform*
Points to where to store the Tree Transformation.

Return Value S_OK

Comments The Global Transformation defines the positioning, orientation, and scaling of the Tree Element in the Global Coordinates System. See GetTransform8 for more details.

I3DShTreeElement::SetGlobalTransform8

HRESULT I3DShTreeElement::SetGlobalTransform8 (*treeTransform***);**
TREETRANSFORM3D* *treeTransform*

Call **I3DShTreeElement::SetGlobalTransform 8** to set the Global Transformation of the Tree Element.

Parameters ⇦ *treeTransform*
Points to the new Tree Transformation.

Return Value S_OK

Comments The Global Transformation defines the positioning, orientation, and scaling of the Tree Element in the Global Coordinates System. SetGlobalTransform8 keep the Hot Point position unchanged. See GetTransform8 for more details.

©1995-1997 MetaCreations. All rights reserved.

2.2.I3DShGroup Interfac e

Interface ID: IID_I3DShGroup
Interface file: I3DShTre.h

The **I3DShGroup** interface defines Groups in the tree hierarchy. A Group is a little bit like a directory: its purpose is to gather other Tree Elements together.

Because a Group is a Tree Element, it can return a **I3DShTreeElement Interfac e**.

A Group can be open or closed. At the user interface level, the user can manipulate objects contained in the group as individuals when the group is open. When it is closed, the group behaves like a single object.

```
DECLARE_INTERFACE_(I3DShGroup, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShGroup methods
  STDMETHOD(SetBBox) (THIS_ BOX3D* bbox) PURE;
  STDMETHOD(DoOpenClose) (THIS_ BOOLEAN openIt) PURE;    // Use this call
      to open or close a group
  STDMETHOD(OpenClose) (THIS_ BOOLEAN openIt) PURE;      // This lower
      level call just changes the status
  STDMETHOD_(BOOLEAN, IsOpen) (THIS) PURE;
  };
```

I3DShGroup::SetBBox

HRESULT I3DShGroup::SetBBox (*bbox***);**
BOX3D* *bbox*

Call **I3DShGroup::SetBBo x** to manually set the Bounding Box of a Group.

Parameter *bbox*
The new Bounding Box in local coordinates.

Return Valu e S_OK

Comments You normally never call this procedure.

I3DShGroup::DoOpenClos e

HRESULT I3DShGroup::DoOpenClose (*openIt***);**
BOOLEAN *openIt*

Call **I3DShGroup::DoOpenClos e** to open or closed a Group in the Tree.

©1995-1997 MetaCreations. All rights reserved.

Parameter *openIt*
TRUE to open the Group, FALSE to close it.

Return Value S_OK

Comments Use this procedure rather than I3DShGroup::OpenClose. It tells all clients of the 3D Database that the status of the group changed.

I3DShGroup::OpenClose

HRESULT I3DShGroup::OpenClose (*openIt***);**
BOOLEAN *openIt*

Call **I3DShGroup::OpenClos e** set the open or closed status of a Group.

Parameter *openIt*
TRUE for an open Group, FALSE for a closed one.

Return Value S_OK

Comments This is a low-level procedure that just sets the Group status. Use rather **I3DShGroup::DoOpenClos e** to open or close a Group in a tree.

I3DShGroup::IsOpen

BOOLEAN I3DShGroup::IsOpen() ;

Call **I3DShGroup::IsOpe n** to get the status of a Group.

Return Value

Value	Meaning
TRUE	The Group is open
FALSE	The Group is closed

©1995-1997 MetaCreations. All rights reserved.

2.3.I3DShLightsource Interfac e

Interface ID: IID_I3DShLightsource
Interface file: I3DShTre.h

The **I3DShLightsourc e** interface defines light sources as Tree Elements.

Because a Light Source is a Tree Element, it can return a I3DShTreeElement Interface.

```
DECLARE_INTERFACE_(I3DShLightsource, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShLightsource methods
  STDMETHOD(GetLightGlobalTransform) (THIS_ TRANSFORM3D* transform)
      PURE;    // A little bit like GetGlobalTransform(), but without the
      scaling factor
  STDMETHOD_(BOOLEAN, GetDirection) (THIS_ VECTOR3D* position, VECTOR3D*
      resultDirection, NUM3D* resultDistance) PURE;
  STDMETHOD_(BOOLEAN, GetColor) (THIS_ VECTOR3D* position, VECTOR3D*
      direction, NUM3D distance, COLOR3D* result, BOOLEAN* callForShad-
      owEffect) PURE;
  STDMETHOD(ShadowEffect) (THIS_ NUM3D distanceThru, NUM3D distanceTo-
      tal, COLOR3D* result) PURE;
  STDMETHOD(ForEachShadowBuffer) (THIS_ ForEachShadowBufferCallback
      proc, void* priv) PURE;
  STDMETHOD(GetLightParameter) (THIS_ long keyword, void* parameter)
      PURE;
  };
```

I3DShLightsource::GetLightGlobalTransfor m

HRESULT I3DShLightsource::GetLightGlobalTransform (*transform***);**
TRANSFORM3D* *transform*

Call **I3DShLightsource::GetLightGlobalTransfor m** to get the Light Source Transformation with the combined translation in the Global Coordinates System.

Parameter ⇨ *transform*
Returns the Transformation of the Light Source.

Return Valu e S_OK

Comments Use this rather than **I3DShTreeElement::GetGlobalTransfor m** when specifically dealing with a Light Source, because a **TRANSFORM3 D** is more simple and more adapted to a Light Source than a **TREETRANSFORM3 D** (scaling is meaningless for a light source). Take care that the combined translation points to the origin of the local coordinate and not the hot point. The combined translation is meaningful because this is the real position of the light. See I3DShTreeElement::GetTransform8 for more details.

©1995-1997 MetaCreations. All rights reserved.

I3DShLightsource::GetDirection

BOOLEAN I3DShLightsource::GetDirection (*position, resultDirection, resultDistance***);**
VECTOR3D* *position*
VECTOR3D* *resultDirection*
NUM3D* *resultDistance*

Call **I3DShLightsource::GetDirectio n** before I3DShLightsource::GetColor to perform preliminary calculations and get the direction and the distance between the surface point and the light source.

Parameters

position
Give the 3D position of the point on the surface, in Global Coordinates.

resultDirection
Returns the direction vector between the point and the light source, in Global Coordinates. This is a normalized vector in the surface point pointing to the light source.

resultDistance
Returns the distance between the point and the light source.

Return Value Always TRUE

Comments A light bulb will typically calculate resultDirection by subtracting the position point to the Light Source origin, and normalizing the result. The resultDistance would be the distance between the light source origin and the position point.

A sun light will typically return its light direction in resultDirection, and the maximum value in resultDistance.

This call is used only by renderers.

I3DShLightsource::GetColor

BOOLEAN I3DShLightsource::GetColor (*position, direction, distance, result, callForShadowEffect***);**
VECTOR3D* *position*
VECTOR3D* *direction*
NUM3D *distance*
COLOR3D* *result*
BOOLEAN* *callForShadowEffect*

Call **I3DShLightsource::GetColo r** to calculate the lighting.

Parameters

position
Give the 3D position of the point on the surface, in Global Coordinates.

©1995-1997 MetaCreations. All rights reserved.

direction
The direction vector between the point and the light source, in Global Coordinates, as returned by I3DShLightsource::GetDirection().

distance
The distance between the point and the light source, as returned by I3DShLightsource:: GetDirection().

result
The Light Source will return the calculated color here.

callForShadowEffect
The Light Source will return TRUE if the I3DShLightsource::ShadowEffect() method must be called during shadows calculation. It will return TRUE if the light casts any shadows.

Return Value

Value	Meaning
TRUE	The point is under the light
FALSE	The point is out of the Light Source lighting

Comments This call is used only by renderers.

I3DShLightsource::ShadowEffect

HRESULT I3DShLightsource::ShadowEffect (*distance, distanceTotal, result***);**
NUM3D *distance*
NUM3D *distanceTotal*
COLOR3D* *result*

Call **I3DShLightsource::ShadowEffec t** when **I3DShLightsource::GetColo r** returns TRUE for **callForShadowEffec t** and that the light beam goes through an object.

Parameters

distance
Give the partial distance as returned by GetDirection.

distanceTotal
Future use.

result
The color is given as an input and output parameter. You must give the color as it has been calculated so far. The light source will modify it directly if necessary.

Return value S_OK

Comments I3DShLightsource::GetColor is not enough in most of the cases because it depends on the

©1995-1997 MetaCreations. All rights reserved.

location. This call is used only by renderers.

I3DShLightsource::ForEachShadowBuffe r

HRESULT I3DShLightsource::ForEachShadowBuffer (*proc, priv***);**
ForEachShadowBufferCallback *proc*
void* *priv*

Call **I3DShLightsource::ForEachShadowBuffe r** if you need to get a list of Shadow Buffer planes from a light source to calculate shadows.

Parameters

proc
Give the callback procedure that should be called for each Shadow Buffer plane of the Light Source.

```
void ForEachShadowBufferCallback(BOOLEAN lightType, TRANSFORM3D* trans-
      form, NUM3D halfAngle, void* priv);
```

lightType
Indicates if *halfAngle* is meaningful. FALSE for distant lights, TRUE for all others.

transform
The Light source will give the transformation as calculated by **I3DShLightsource::GetLightGlobalTransfor m**.

halfAngle
The half-angle of the light cone. Used only if *lightType* == TRUE.

priv
You get back any private data you passed to **I3DShLightsource:: ForEachSha dowBuffer**.

priv
Give a pointer on any private data you would like to use in your call-back procedure.

Return value S_OK

Comments This call is used only by renderers such as Z-Buffer renderers, that use the Shadow-Buffer technique to compute shadows.

Typically, a bulb light will return 6 Shadow Buffers (as it shines in all directions, a 6-sided Box has to be used). Distant lights and spot lights use only 1 Shadow Buffer.

I3DShLightsource::GetLightParameter

HRESULT I3DShLightsource::GetLightParameter (*keyword, parameter***);**
long *keyword*

©1995-1997 MetaCreations. All rights reserved.

void* *parameter*

Call **I3DShLightsource::GetLightParamete r** to get a parameter in the pMAP of the light source.

Parameters

keyword
Identifies the parameter needed in the pMAP.

parameter
The returned parameter

Return valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.4.I3DShCamera Interfac e

Interface ID: IID_I3DShCamera
Interface file: I3DShTre.h

The **I3DShCamera** interface describes Cameras as Tree Elements. The **I3DShCamera** methods are used mostly by renderers.

Because a Camera is a Tree Element, it can return the **I3dShTreeElement Interfac e**.

Pipeline of projections:

Say a renderer needs to display a point of an object surface (like the vertex of a facet). The projection pipeline would be like this:

- Get the object point in the object Local Coordinates System.
- Using the object Tree Transformation (by calling I3DShTreeElement::GetGlobal-Transform), transform the point in the Global Coordinates System.
- Transform the point from the Global Coordinates System into the Screen Coordinates System using the transformation returned by I3DShCamera::GetGlobalToCamer-aTransform.
- Perform the camera projection by calling I3DShCamera::Project3DTo2D.
- If we intend to draw in a bitmap, then transform the point by converting it into the Pix-els Space of the offscreen bitmap.

```
DECLARE_INTERFACE_(I3DShCamera, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShCamera methods
  STDMETHOD(GetGlobalToCameraTransform) (THIS_ TRANSFORM3D* transform)
      PURE;          // Gives the transformation from the Global Coordi-
      nate System to the Screen Coordinate System
  STDMETHOD(Preset) (THIS_ ULONG preset, VECTOR3D* aimedPoint, NUM3D
      distance) PURE;  // See preset codes above
  STDMETHOD_(BOOLEAN, Project3DTo2D) (THIS_ VECTOR3D* position,
      VECTOR2D* resultScreenPosition, NUM3D* resultDistanceToScreen)
      PURE;
  STDMETHOD_(BOOLEAN, CreateRay) (THIS_ VECTOR2D* screenPosition,
      VECTOR3D* resultOrigin, VECTOR3D* resultDirection) PURE;
  STDMETHOD_(ULONG, Clip3D) (THIS_  FACET3D* localFacet, VERTEX3D*
      localVertices,
                                    FACET3D* cameraFacet, VERTEX3D* cam-
      eraVertices,
                                    NUM3D* clipBox) PURE;
  STDMETHOD(GetRenderUVBox) (THIS_ BOX2D *uvBox) PURE;
  };
```

©1995-1997 MetaCreations. All rights reserved.

I3DShCamera::GetGlobalToCameraTransfor m

HRESULT I3DShCamera::GetGlobalToCameraTransform (*transform***);**
TRANSFORM3D* *transform*

Call **I3DShCamera::GetGlobalToCameraTransfor m** to get the transformation with the combined translation from the Global Coordinates System to the Screen Coordinates System.

Paramete *transform*
Points to where to store the transformation.

Return Valu e S_OK

Comments This call is typically used by renderers. See note at the description of the interface about the pipeline of projections. Take care that the combined translation points to the origin of the local coordinate and not the hot point. The combined translation is meaningful because this is the real position of the light. See I3DShTreeElement::GetTransform8 for more details. See the "Database Overview" chapter for more details about the different coordinates systems.

I3DShCamera::Preset

HRESULT I3DShCamera::Preset (*preset, aimedPoint, distance***);**
ULONG *preset*
VECTOR3D* *aimedPoint*
NUM3D *distance*

Call **I3DShCamera::Prese t** to automatically put a Camera to a preset position.

Parameters *preset*
Preset code.

Value	Constant	Meaning
0	kReferencePreset	1,1,1 corner of the scene box
1	kTopPreset	Top view
2	kBottomPreset	Bottom view
3	kLeftPreset	Left view
4	kRightPreset	Right view
5	kFrontPreset	Front view
6	kBackPreset	Back view

aimedPoint
The point to look at, in Global Coordinates.

distance
The distance from which to look at aimedPoint.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

I3DShCamera::Project3DTo2D

BOOLEAN I3DShCamera::Project3DTo2D (*position, resultScreenPosition, resultDistanceToScreen***);**
VECTOR3D* *position*
VECTOR2D* *resultScreenPosition*
NUM3D* *resultDistanceToScreen*

Call **I3DShCamera::Project3DTo2 D** to project a 3D point on the screen.

Parameters

position
The 3D point must be given in the Screen Coordinates System.

resultScreenPosition
Resulting projection on the screen in Points units.

resultDistanceToScreen
Distance from the point position to the screen. It will be positive if "in front" of the camera.

Return Value

Value	Meaning
TRUE	The point is visible
FALSE	The point is not visible

Comments This call is typically used by renderers. See note at the description of the interface about the pipeline of projections.

See the chapter "Database Overview" for more details about the different Coordinates Systems.

Points units = 3D units * 288

I3DShCamera::CreateRay

BOOLEAN I3DShCamera::CreateRay (*screenPosition, resultOrigin, resultDirection***);**
VECTOR2D* *screenPosition*
VECTOR3D* *resultOrigin*
VECTOR3D* *resultDirection*

Call **I3DShCamera::CreateRa y** to create a ray (i.e. a straight line) from a screen position.

©1995-1997 MetaCreations. All rights reserved.

Parameters *screenPosition*
Position on the screen in Points unit.

resultOrigin
The camera will return in resultOrigin the same screen point, in 3D, in the Global Coordinates System.

resultDirection
The camera will return in resultDirection the direction vector of the straight line. It will be a unit vector.

Return Value

Value	Meaning
TRUE	A ray could be calculated
FALSE	Calculation was not possible

Comments This call is typically used by renderers, especially ray-tracing renderers.

Points units = 3D units * 288

I3DShCamera::Clip3D

ULONG I3DShCamera::Clip3D (*localFacet, localVertices, cameraFacet, cameraVertices, clipBox***);**
FACET3D* *localFacet*
VERTEX3D* *localVertices*
FACET3D* *cameraFacet*
VERTEX3D* *cameraVertices*
NUM3D* *clipBox*

Call **I3DShCamera::Clip3 D** to have the camera perform 3D Clipping on a facet. Clipping will be performed at the same time in the Object Local Coordinates and in the Camera's Local Coordinates System, so you do not have to redo all the inverse transformations to get back to the Object Local Coordinates System.

Parameters *localFacet*
Points on the facet to clip. The facet vertices are in the Object Local Coordinates.

localVertices
Points on an array where the clipped vertices will be returned, in Object Local Coordinates. This array must have 64 elements.

cameraFacet
Points on the facet to clip. The facet vertices are in the Camera's Local Coordinates System.

©1995-1997 MetaCreations. All rights reserved.

cameraVertices
Points on an array where the clipped vertices will be returned, in Camera Coordinates. This array must have 64 elements.

clipBox
Points on an array of 6 elements, describing the 3D Box to use for clipping, in Camera Coordinates.

clipBox[0]	x minimum. Usually the left side of the rendering frame, in 3D units
clipBox[1]	x maximum. Usually the right side of the rendering frame, in 3D units
clipBox[2]	y minimum. Usually the bottom side of the rendering frame, in 3D units
clipBox[3]	y maximum. Usually the top side of the rendering frame, in 3D units
clipBox[4]	z minimum. Usually set to 0.0
clipBox[5]	z maximum. Usually set to the maximum value

Return Value The number of vertices returned in localVertices and cameraVertices. For example, if no clipping occurred, 3 will be returned. If a value under 3 is returned, the facet should be considered invalid and ignored.

Comments localVertices and cameraVertices describe a convex 3D polygon that can be easily subdivided in sub triangles if needed.

This call is typically used by renderers.

I3DShCamera::GetRenderUVBox

HRESULT I3DShCamera::GetRenderUVBox (*uvBox***);**
BOX2D* *uvBox*

Call **I3DShCamera::GetRenderUVBo x** to get the field seen by the camera.

Parameter *uvBox*
Points to where to return the rectangle, in screen coordinates, seen by the camera.

Return Value S_OK

Comments This field can be not centered.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.5.I3DShInstance Interfac e

Interface ID: IID_I3DShInstance
Interface file: I3DShTre.h

An Instance is a Tree Element that inserts an object in the scene. One can insert several times the same object as several instances of the same I3DShObject. Some calls of I3DShInstance deal with shading the object at the instance level. With this technique, "shading overriding" can be done on objects.

Because an Instance is a Tree Element, it can return a **I3DShTreeElement Interfac e**.

```
DECLARE_INTERFACE_(I3DShInstance, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShInstance methods
  //-- Object referencing
  STDMETHOD_(I3DShObject*, Get3DObject) (THIS) PURE;
  STDMETHOD(Set3DObject) (THIS_ I3DShObject* object) PURE;
  //-- Shading at the Instance level
  STDMETHOD_(ULONG, GetUVSpaceCount) (THIS) PURE;
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE;
  STDMETHOD(AddShadingShape) (THIS_ ShadingShape* shadingShape, ULONG
      uvSpaceID) PURE;
  STDMETHOD(GetShadingFlags) (THIS_ ShadingFlags* theFlags) PURE;
  STDMETHOD(DoShade) (THIS_ ShadingInOut* shadingInOut) PURE;
  STDMETHOD(GetMainColors) (THIS_ COLOR3D* color1, COLOR3D* color2)
      PURE;
  STDMETHOD_(long, GetShadingLevel) (THIS) PURE;
  STDMETHOD(GetUVRasterShMap) (THIS_ long uvSpaceIndex, short& sizeU,
      short& sizeV, UVShMaps* maps, long channels, void *viewPriv,
      UVMapConvertProc convertProc, UVMapDeleteProc deleteProc) PURE;
  STDMETHOD(GetUVExportShMap) (THIS_ long uvSpaceIndex, short& sizeU,
      short& sizeV, FixPtRect *range, UVShMaps* maps, long channels)
      PURE;
  STDMETHOD_(BOOLEAN, FreeUVShMap) (THIS_ long uvSpaceIndex) PURE;
  STDMETHOD_(I3DShIRCache*, GetIRCache) (THIS) PURE;
  STDMETHOD(SetIRCache) (THIS_ I3DShIRCache* theIRCache) PURE;
  //-- For renderers
  STDMETHOD(GetUVSpace) (THIS_  long uvSpaceID,UVSpaceInfo* result)
      PURE;
  STDMETHOD(GetFlatness) (THIS_  NUM3D *flatness) PURE;
  STDMETHOD(Draw) (THIS_  TZBInfo *aZBInfo, short *drawn) PURE;
  STDMETHOD(DrawLA) (THIS_  T2DLineArt *aLineArt,TREETRANSFORM3D
      *aTS,TCamera *acamera,RECT3D *area,short modeD) PURE;
  STDMETHOD(GetBoxes) (THIS_  TZBInfo *aZBInfo,NUM3D *minZ,NUM3D
      *maxZ,BOX2D *clip,BOX3D *box) PURE;
  STDMETHOD(GetBB) (THIS_  BOX3D *bbox) PURE;
  STDMETHOD(Dehydrate) (THIS_  short level) PURE;
  STDMETHOD_(BOOLEAN,RayHit) (THIS_  Ray3D *aR,RayHitParameters
```

©1995-1997 MetaCreations. All rights reserved.

```
    *params,RayHit3D *hit) PURE;
STDMETHOD(GetAllInfo) (THIS_  RayHit3D *hit,short *uvSpaceID) PURE;
//-- New misc. calls (API level 6)
STDMETHOD_(I3DShObject*, GetDeformed3DObject) (THIS) PURE;  // Same as
    Get3DObject(), but with the deformations applied on the object
STDMETHOD(EnumFacetsWithConversion) (THIS_ EnumFacetsCallback call-
    back, void* privData, BOOLEAN doPatches, NUM3D error) PURE;  // Use
    this call when you want facets even for patch based objects, and be
    able to control the error in the facettisation process

STDMETHOD(SetShadingFromMaps) (THIS_ long uvSpaceIndex, UVShMaps*
    maps, MappingParams* mapParams, const FixPtRect& shapeRect) PURE;

STDMETHOD_(short, GetMappingKind) (THIS) PURE;
STDMETHOD(SetMappingKind) (THIS_ short mappingKind) PURE;
STDMETHOD(GetMappingDirection) (THIS_ MATRIX3D* transform) PURE;
STDMETHOD(SetMappingDirection) (THIS_ MATRIX3D* transform) PURE;
STDMETHOD(GetURL) (THIS_ char* name, short *kind) PURE;
STDMETHOD_(long, GetClassSignature) (THIS) PURE;
STDMETHOD_(long, GetInstanceKind) (THIS) PURE;

STDMETHOD(GetShadingBBox) (THIS_ BOX3D* shadingBBox) PURE;
//API8
STDMETHOD_(BOOLEAN, IsVolumeInstance) (THIS) PURE;  };
```

I3DShInstance::Get3DObject

I3DShObject* I3DShInstance::Get3DObject() ;

Call **I3DShInstance::Get3DObjec t** to get the pointer on the 3D object of the instance.

Return Valu e pointer on the 3D object that is instanciated in the scene.

I3DShInstance::Set3DObject

HRESULT I3DShInstance::Set3DObject (*object***);**
I3DShObject* *object*

Call **I3DShInstance::Set3DObjec t** to set the 3D object of an instance.

Parameter *object*
The object to instanciate. The old one is deleted if there is no more reference on it.

Return Valu e S_OK

I3DShInstance::GetUVSpaceCount

ULONG I3DShInstance::GetUVSpaceCount() ;

Call **I3DExGeometricPrimitive::GetUVSpaceCoun t** to get the number of UV Spaces of

©1995-1997 MetaCreations. All rights reserved.

the primitive. See the "Database Overview" chapter for more information about UV Spaces.

Return value Number of UV Spaces for this object. If it is not possible to do direct parametric mapping on this object (no UV values), then 0 will be returned.

I3DShInstance::GetUVSpace

HRESULT I3DShInstance::GetUVSpace(uvSpaceID, uvSpaceInfo)
ULONG uvSpaceI D
UVSpaceInfo* uvSpaceInf o

Call **I3DShInstance::GetUVSpac e** to get more information about the UV Space(s) of the primitive.

Parameters *uvSpaceID*
The UV Space ID on which we want information.

uvSpaceInfo
Points to a record that the Instance will fill-in. See the description of the data structure for more information.

Return Value S_OK

Comments This call is typically used by renderers.

I3DShInstance::AddShadingShape

HRESULT I3DShInstance::AddShadingShape (*shadingShape, uvSpaceID***)**
ShadingShape* *shadingShape*
ULONG *uvSpaceID*

This call is not yet implemented, you musn't call it.

I3DShInstance::GetShadingFlags

HRESULT I3DShInstance::GetShadingFlags (*theFlags***)**
ShadingFlags* *theFlags*

Call **I3DShInstance::GetShadingFlag s** to get the flags on the shading of the instance.

Parameter *theFlags*
Points to where to return the flags.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments See the “Data Structure Reference” chapter for a description of the **ShadingFlags** structure.

I3DShInstance::DoShade

HRESULT I3DShInstance::DoShade (*shadingInOut***);**
ShadingInOut* *shadingInOut*

Call **I3DShInstance::DoShad e** to perform all shading calculations for the Instance.

Parameter *shadingInOut*
Points to the Shading Record. See the description of the data structure for more information, and the example in the Renderer Cookbook.

Return Valu e S_OK

Comments This call is normally used by renderers to calculate the shading on an object. It performs all the shading, combining shading at the Instance level and at the Object level.

I3DShInstance::GetMainColors

HRESULT I3DShInstance::GetMainColors (*color1, color2***);**
COLOR3D* *color1*
COLOR3D* *color2*

Call **I3DShInstance::GetMainColor s** to get two "main colors" of an Instance.

Parameters *color1*
Points to where to store the first main color. This is the real color of the object.

color2
Points to where to store the second main color. This color is not often used.

Return Valu e S_OK

Comments This call is normally used by a renderer when it does not want to call **I3DShInstance::DoShad e**, but still wants to get some colors to make some basic shading. This is for the "shading of the poor". Good for Preview rendering.

I3DShInstance::GetShadingLevel

long I3DShInstance::GetShadingLevel() ;

Call **I3DShInstance::GetShadingLeve l** to get the Shading Level of an Instance.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Constant	Meaning
-1	kShLvlNoChanges	No Changes - Use renderer's default shading level
0	kShLvlInvisible	Invisible - Do not render Instance
1	kShLvlBBox	Render bounding box only
2	kShLvlWireFrame	Wireframe
3	kShLvlPreview	Preview Shading - Fast and crude shading
4	kShLvlFullShading	Full Shading. Call **I3DShInstance::DoShade**

Comments This call is used by renderers that want to use the rendering settings override of each Instance. The Shading level of an Instance is usually set by the user to facilitate his work by hiding or rendering quickly some objects.

I3DShInstance::GetUVRasterShMap

HRESULT I3DShInstance::GetUVRasterShMap (*uvSpaceIndex, sizeU, sizeV, maps, channels, viewPriv, convertProc, deleteProc***)**
long *uvSpaceIndex*
short& *sizeU*
short& *sizeV*
UVShMaps* *maps*
long *channels*
void* *viewPriv*
UVMapConvertProc *convertProc*
UVMapDeleteProc *deleteProc*

Call **I3DShInstance::GetUVRasterShMa p** to get the shading maps of the instance with a conversion you specify applied on them. This can be useful if you wish to get a format of map to be directly used by a hardware.

Parameters

uvSpaceIndex
The identifier of the UV space.

sizeU
Enter the width for the resolution; setting these values to zero will cause values to be chosen automatically (and returned).

sizeV
Enter the height for the resolution; setting these values to zero will cause values to be chosen automatically (and returned).

maps
Points to where to store the shader maps offscreens.

©1995-1997 MetaCreations. All rights reserved.

channels
Specify the shader channels you want to get the offscreen. This is a set of flags.

Value	Constant	Meaning
1	kUVMapColor	Color channel
2	kUVMapHighlight	Highlight channel
4	kUVMapShininess	Shininess channel
8	kUVMapBump	Bump channel
16	kUVMapReflection	Reflection channel
32	kUVMapTransparency	Transparency channel
64	kUVMapRefraction	Refraction channel
128	kUVMapGlow	Glow channel

viewPriv
Simply pass in the fPriv field of the current TZuberBuffer.

convertProc
Pass a pointer to a convert function. This procedure will be called for each TEVOff-screen channel that the programmer has requested. You may choose not to make any alterations to the offscreen; if so you should simply return the pointer to the parameter offscreen. If you do convert the offscreen, you should then delete the offscreen and return the pointer to whatever the offscreen has been converted to.
An example of how to use this would be to convert from a TOffscreen to a pointer to an array of longs for use with a hardware zbufferer.

```
void* UVMapConvertProc(TEVOffscreen *offscreen, long channel);
```

offscreen
The offscreen of the map to be converted.

channel
Specify the kind of map (see above for the constant values).

Return Value a pointer on the modified map.

deleteProc
Pass a pointer to a delete function. The parameter converted will be whatever the result of the above convertProc was.

```
void UVMapDeleteProc(void *convertedUVMap, long channel);
```

convertedUVMap
A pointer on the modified data to be deleted.

channel
Specify the kind of map (see above for the constant values).

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

I3DShInstance::GetUVExportShMap

HRESULT I3DShInstance::GetUVExportShMap (*uvSpaceIndex, sizeU, sizeV, range, maps, channels***);**
long *uvSpaceIndex*
short& *sizeU*
short& *sizeV*
FixPtRect* *range*
UVShMaps *maps*
long *channels*

Call **I3DShInstance::GetUVExportShMa p** to get the shader maps offscreens of a part of the UV space of the Instance.

Parameters

uvSpaceIndex
The identifier of the UV space.

sizeU
Enter the width for the resolution; setting these values to zero will cause values to be chosen automatically (and returned).

sizeV
Enter the height for the resolution; setting these values to zero will cause values to be chosen automatically (and returned).

range
Set the area in UV space coordinates of the shader you want the map.

maps
Points to where to store the shader maps offscreens.

channels
Specify the shader channels you want to get the offscreen. This is a set of flags.

Value	Constant	Meaning
1	kUVMapColor	Color channel
2	kUVMapHighlight	Highlight channel
4	kUVMapShininess	Shininess channel
8	kUVMapBump	Bump channel
16	kUVMapReflection	Reflection channel
32	kUVMapTransparency	Transparency channel
64	kUVMapRefraction	Refraction channel
128	kUVMapGlow	Glow channel

Return Value S_OK

Comments The returned sizeU and sizeV are the resolution of the map. That's mean that range is

©1995-1997 MetaCreations. All rights reserved.

divided by sizeU * sizeV.

I3DShInstance::FreeUVShMap

BOOLEAN I3DShInstance::FreeUVShMap(uvSpaceIndex) ;
long *uvSpaceIndex*

Internal use only.

I3DShInstance::GetIRCache

I3DShIRCache* I3DShInstance::GetIRCache() ;

Internal use only.

I3DShInstance::SetIRCache

HRESULT I3DShInstance::SetIRCache (*theIRCache***);**
I3DShIRCache* *theIRCache*

Internal use only.

I3DShInstance::GetFlatness

HRESULT I3DShInstance::GetFlatness (*flatness***);**
NUM3D* *flatness*

Call **I3DShInstance::GetFlatnes s** tocomment

Parameter *flatness*
Points to where to return the flatness, this is the surface fidelity that is generally used by renderer.

Return Value NOERROR

I3DShInstance::Draw

HRESULT I3DShInstance::Draw (*aZBInfo, drawn***);**
TZBInfo* *aZBInfo*
short* *drawn*

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::DrawLA

HRESULT I3DShInstance::DrawLA (*aLineArt, aTS, acamera, area, modeD***);**

©1995-1997 MetaCreations. All rights reserved.

T2DLineArt* *aLineArt*
TREETRANSFORM3D* *aTS*
TCamera* *acamera*
RECT3D* *area*
short *modeD*

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::GetBoxes

HRESULT I3DShInstance::GetBoxes (*aZBInfo, minZ, maxZ, clip, box***);**
TZBInfo* *aZBInfo*
NUM3D*minZ*
NUM3D* *maxZ*
BOX2D* *clip*
BOX3D* *box*

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::GetBB

HRESULT I3DShInstance::GetBB (*bbox***);**
BOX3D* *bbox*

Call **I3DShInstance::GetB B** to get the bounding box of the instance.

Parameter *bbox*
Points to where to store the bounding box.

Return Value NOERROR

Comments The bounding box is given in the Local Coordinate System.

I3DShInstance::Dehydrate

HRESULT I3DShInstance::Dehydrate (*level***);**
short *level*

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::RayHit

HRESULT I3DShInstance::RayHit (*aR, params, hit***);**
Ray3D* aR
RayHitParameters* *params*
RayHit3D* *hit*

©1995-1997 MetaCreations. All rights reserved.

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::GetAllInfo

HRESULT I3DShInstance::GetAllInfo (*hit, uvSpaceID***);**
RayHit3D* *hit*
short* *uvSpaceID*

At this time, this method is not implemented, you don't have to call it.

I3DShInstance::GetDeformed3DObject

I3DShObject* I3DShInstance::GetDeformed3DObject () ;

Call **I3DShInstance::GetDeformed3DObjec t** to get the object with its deformation.

Return Value The deformed object.

Comments GetDeformed3Dobject is the same thing than **Get3DObject** but **Get3DObject** return the undeformed object. You can't modify the deformed object, you have to handle the original object. This method is only used by renderers and exporters.

I3DShInstance::EnumFacetsWithConversio n

HRESULT I3DShInstance::EnumFacetsWithConversion (*callback, privData, doPatches, error***);**
EnumFacetsCallback *callback*
void* *privData*
BOOLEAN *doPatches*
NUM3D *error*

Call **I3DShInstance::EnumFacetsWithConversio n** to get the Object as a list of triangular facets.

Parameters *callback*
Give the callback procedure that should be called for each facets.

```
void* EnumFacetsCallback(FACET3D* facet, void* privData);
```

facet
The facet being processed.

privData
You get back any private data you passed to **I3DShInstance:: EnumFacetsWit h-Conversion**.

privData
Give a pointer on any private data you would like to use in your call-back procedure.

©1995-1997 MetaCreations. All rights reserved.

doPatches
Give TRUE if you want to process all the patches converted to facets.

error
Give the error for the conversion.

Return Value NOERROR

Comments This method is useful if you want facets even for patches based objects , and be able to control the error in the tesselization process.

I3DShInstance::SetShadingFromMaps

HRESULT I3DShInstance::SetShadingFromMaps (*uvSpaceIndex, maps, mapParams, shapeRect***);**
long *uvSpaceIndex*
UVShMaps *maps*
MappingParams* *mapParams*
const FixPtRect& *shapeRect*

Call **I3DShInstance::SetShadingFromMap s** to set maps to shading channels of an Instance.

Parameters

uvSpaceIndex
The identifier of the UV space.

maps
An array of maps to apply to the instance. One map for each channel and null pointers for those you don't want to apply. See I3DShInstance::SetShadingFromMaps for the list of channels.

mapParams
An array of parameters for the maps. You must specify a map parameters structure for each maps you plan to apply.

shapeRect
The range where to apply the shading.

Return Value S_OK

I3DShInstance::GetMappingKind

short I3DShInstance::GetMappingKind () ;

Call **I3DShInstance::GetMappingKin g** to get the kind of mapping of the Instance.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Constant	Meaning
-1	kDefaultMapping	the default mapping, you don't have to use it
0	kParametricMapping	the mapping using the UV Space information.
1	kBoxMapping	6 faces cube mapping.
2	kCylindricalMapping	cylindrical projection mapping
3	kSphericalMapping	spherical projection mapping
4	kPassThruMapping	top face cube mapping.

Comments The other face cube mapping are obtained by rotation of the shading.

I3DShInstance::SetMappingKind

HRESULT I3DShInstance::SetMappingKind (*mappingKind***);**
short *mappingKind*

Call **I3DShInstance::SetMappingKin d** to select the kind of mapping of an Instance.

Parameter *mappingKind*
The kind of mapping. See I3DShInstance::GetMappingKind for a list of the different kinds of mapping.

Return Value S_OK

I3DShInstance::GetMappingDirection

HRESULT I3DShInstance::GetMappingDirection (*transform***);**
MATRIX3D* *transform*

Call **I3DShInstance::GetMappingTransfor m** to get the rotation matrix of the shading of an Instance.

Parameter *transform*
Points to where to store the rotation matrix.

Return Value S_OK

Comments The shading can be rotated from the local coordinate of the instance, except for the parametric mapping.

I3DShInstance::SetMappingDirection

HRESULT I3DShInstance::SetMappingDirection (*transform***);**
MATRIX3D* *transform*

Call **I3DShInstance::SetMappingTransfor m** to set the rotation of a shading of an

©1995-1997 MetaCreations. All rights reserved.

Instance.

Parameter *transform*
The rotation matrix for the transformation.

Return Value S_OK

Comments The shading can be rotated from the local coordinate of the instance, except for the parametric mapping.

I3DShInstance::GetURL

HRESULT I3DShInstance::GetURL (*name, kind***);**
char* *name*
short* *kind*

Call **I3DShInstance::GetURL** to get the name and the kind of the URL of an Instance.

Parameters *name*
Points to where to store the name.

kind
Points to where to return the kind of URL. 0 is returned for Anchor, 1 for Inline.

Return Value S_OK

I3DShInstance::GetClassSignature

long I3DShInstance::GetClassSignature () ;

In COM, you do not need anymore to get a class signature. This is an obsolete function.

I3DShInstance::GetInstanceKind

long I3DShInstance::GetInstanceKind () ;

Call **I3DShInstance::GetInstanceKind** to get the kind of the Instance.

Return Value

Value	Constant	Meaning
1	kPrimitiveInstance	The instance is a primitive
2	kSceneInstance	a scene
3	kCameraInstance	a camera
4	kLightInstance	a light

©1995-1997 MetaCreations. All rights reserved.

I3DShInstance::GetShadingBBox

HRESULT I3DShInstance::GetShadingBBox (*shadingBBox***);**
BOX3D* *shadingBBox*

Call **I3DShInstance::GetShadingBBo x** to get the bounding box of the shader of an Instance.

Parameter *shadingBBox*
Points to where to return the shading bounding box.

Return Value S_OK

Comments This bounding box is given in the local coordinate of the shading. The shading bounding box is different to the instance when the shading is rotated. See I3DShInstance::GetMapingDirection for details on the rotated shading.

I3DShInstance::IsVolumeInstance

BOOLEAN I3DShInstance::IsVolumeInstance () ;

Call **I3DShInstance::IsVolumeInstanc e** to know if the instance is a volume or not.

Return Value

Value	Meaning
TRUE	The instance is a volume.
FALSE	The instance is not a volume.

Comments A volume instance is an instance that can't be seen as a surface, like the fire, the cloud...

©1995-1997 MetaCreations. All rights reserved.

2.6.I3DShObject Interfac e

Interface ID: IID_I3DShObject
Interface file: I3DShObj.h

The **I3DShObject** interface defines Objects that can be instanciated in the scene. There can be two types of Objects: Primitives (See **I3DShPrimitive**) or Scenes (See **I3DShScene**).

Primitives are basic geometric objects (Sphere, cube, extrusion, external primitives, etc.).

Scenes can be instanciated in the current scene, thus making "complex objects" that can be duplicated without taking more RAM.

I3DShObject are to be considered as "classes" that can be "instanciated" many times in the same scene with different position and orientation. Instanciation happens using **I3DShInstance** Tree Elements. **I3DShObject** are kept in the "3D Objects List" of a scene (see **I3DShScene**).

```
DECLARE_INTERFACE_(I3DShObject, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShObject methods
  //-- Naming
  STDMETHOD(SetName) (THIS_ char* name) PURE;
  STDMETHOD(GetName) (THIS_ char* name) PURE;
  //-- Shading calls
  STDMETHOD(SetSimpleShading) (THIS_ COLOR3D* color, NUM3D specularIn-
      tensity, NUM3D specularIndex, NUM3D reflection, NUM3D transpar-
      ency) PURE;
  STDMETHOD(SetPrimerShader) (THIS_ I3DShShader* primerShader) PURE;
  STDMETHOD_(ULONG, GetUVSpaceCount) (THIS) PURE;
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE;
  STDMETHOD(AddShadingShape) (THIS_ ShadingShape* shadingShape, ULONG
      uvSpaceID) PURE;
  STDMETHOD_(I3DShScene*, GetScene) (THIS) PURE;
};
```

I3DShObject::GetName

HRESULT I3DShObject::GetName (*name***);**
char* *name*

Call **I3DShObject::GetNam e** to get the name of a 3D Object.

Parameter ⇨ *name*
Pointer on a string where to store the name.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

I3DShObject::SetName

HRESULT I3DShObject::SetName (*name***);**
char* *name*

Call **I3DShObject::SetNam e** to set the name of a 3D Object.

Parameter *name*
Pointer on a null terminated string that contains the new name.

Return Value S_OK

I3DShObject::SetSimpleShading

HRESULT I3DShObject::SetSimpleShading (*color, specularIntensity, specularIndex, reflection, transparency***);**
COLOR3D* *color*
NUM3D *specularIntensity*
NUM3D *specularIndex*
NUM3D *reflection*
NUM3D *transparency*

Use **I3DShObject::SetSimpleShadin g** to easily set the main shading parameters of an object.

Parameters *color*
The object color.

specularIntensity
Specular intensity, between 0 and 1. This controls the intensity of the specular spot.

specularIndex
Specular index. This controls the size of the specular spot.

reflection
Reflection, between 0 and 1.

transparency
Transparency, between 0 and 1.

Return Value S_OK

Comments This call will create a constant Shader at the Primer level containing the values. This call

©1995-1997 MetaCreations. All rights reserved.

is typically used by 3D importers that need to set some basic shading parameters easily. See the shading model for more information on the parameters.

I3DShObject::SetPrimerShader

HRESULT I3DShObject::SetPrimerShader(primerShader) ;
I3DShShader* *primerShader*

Call **I3DShObject::SetPrimerShader** to set a shader at the Primer level (the Primer is the default shader that covers the whole object surface).

Parameters *primerShader*
The shader to use as a Primer.

Return Value S_OK

Comments Do not keep or use the primerShader pointer after calling **I3DShObject::SetPrimerShader**, because the call takes over the property of it.

I3DShObject::GetUVSpaceCount

ULONG I3DShObject::GetUVSpaceCount() ;

Call **I3DShObject::GetUVSpaceCount** to get the number of UV Spaces of the object. See the "Database Overview" chapter for more information about UV Spaces.

Return value Number of UV Spaces for this object. If it is not possible to do direct parametric mapping on this object (no UV values), then 0 will be returned.

I3DShObject::GetUVSpace

HRESULT I3DShObject::GetUVSpace (*uvSpaceID, uvSpaceInfo***)**
ULONG *uvSpaceID*
UVSpaceInfo* *uvSpaceInfo*

Call **I3DShObject::GetUVSpace** to get more information about the UV Space(s) of the object.

Parameters *uvSpaceID*
The UV Space ID on which we want information.

uvSpaceInfo
Points to a record that the Object will fill-in. See the description of the data structure for more information.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments This call is typically used by renderers.

I3DShObject::AddShadingShape

HRESULT I3DShObject::AddShadingShape (*shadingShape, uvSpaceID***);**
ShadingShape* *shadingShape*
ULONG *uvSpaceID*

This call is not yet implemented, you musn't call it.

I3DShObject::GetScene

I3DShScene* I3DShObject::GetScene() ;

Call **I3DShObject::GetScene t** to get the scene.

Return value The scene. See the Database Overview chapter for more details.

©1995-1997 MetaCreations. All rights reserved.

2.7.I3DShPrimitive Interfac e

Interface ID: IID_I3DShPrimitive
Interface file: I3DShObj.h

The **I3DShPrimitive** Interface describe Primitives. Primitives are basic geometric objects (Sphere, cube, extrusion, external primitives, etc.). Their main feature is to provide geometric data, either bicubic patches or triangular facets.

Because a Primitive is an Object, it can return a **I3DShObject Interfac e**.

```
DECLARE_INTERFACE_(I3DShPrimitive, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShPrimitive methods
  //-- Geometry calls
  STDMETHOD_(BOOLEAN, IsPatchBased) (THIS) PURE;
  STDMETHOD(EnumPatches) (THIS_ EnumPatchesCallback callback, void*
      privData) PURE;
  STDMETHOD(EnumFacets) (THIS_ EnumFacetsCallback callback, void* priv-
      Data) PURE;
  //-- New geometry calls (API Level 6)
  STDMETHOD(EnumFacetsWithConversion) (THIS_ EnumFacetsCallback call-
      back, void* privData, BOOLEAN doPatches, NUM3D error) PURE;// Use
      this call when you want facets even for patch based objects, and be
      able to control the error in the facetisation process  };
```

I3DShPrimitive::IsPatchBased

BOOLEAN I3DShPrimitive::IsPatchBased() ;

Call **I3DShPrimitive::IsPatchBase d** to learn if the object returns bicubic patches or triangular facets.

Return valu e TRUE if the object returns bicubic patches.

Comments Use this call to decide which one to call between **I3DShPrimitive::EnumPatche s** and **I3DShPrimitive::EnumFacet s**.
This call is typically used by renderers and 3D exporters.

I3DShPrimitive::EnumPatches

HRESULT I3DShPrimitive::EnumPatches (*callback, privData***);**
EnumPatchesCallback *callback*
void* *privData*

Call **I3DShPrimitive::EnumPatche s** to get the Object as a list of bicubic patches.

©1995-1997 MetaCreations. All rights reserved.

Parameters *callback*
This callback procedure will be called for each Patch generated by the Primitive. Its interface is as follows:

```
void EnumPatchesCallback(PATCH3D* patch, void* privData);
```

Do not keep the patch pointer. Copy its content in your own data space.

privData
Give a pointer on your private data here. You will get it back when your callback is called.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	patches cannot be generated

Comments The Object will build patches, and for each of them call the callback procedure.

I3DShPrimitive::EnumFacets

HRESULT I3DShPrimitive::EnumFacets (*callback, privData***);**
EnumPatchesCallback *callback*
void* *privData*

Call **I3DShPrimitive::EnumPatche s** to get the Object as a list of triangular facets.

Parameters *callback*
This callback procedure will be called for each facet generated by the Primitive. Its interface is as follows:

```
void EnumFacetsCallback(FACET3D* facet, void* privData);
```

Do not keep the facet pointer. Copy its content in your own data space.

privData
Give a pointer on your private data here. You will get it back when your callback is called.

Return Value

Value	Meaning
S_OK	
E_NOTIMPL	facets cannot be generated

Comments The Object will build facets, and for each of them call the callback procedure.

©1995-1997 MetaCreations. All rights reserved.

I3DShPrimitive::EnumFacetsWithConversio n

HRESULT I3DShPrimitive::EnumFacetsWithConversion (*callback, privData, doPatches, error***);**
EnumFacetsCallback *callback*
void* *privData*
BOOLEAN *doPatches*
NUM3D *error*

Call **I3DShPrimitive::EnumFacetsWithConversio n** to get the Object as a list of triangular facets.

Parameters

callback
Give the callback procedure that should be called for each facets.

```
void* EnumFacetsCallback(FACET3D* facet, void* privData);
```

facet
The facet being processed.

privData
You get back any private data you passed to I3DShPrimitive:: EnumFacetsWithConversion.

privData
Give a pointer on any private data you would like to use in your call-back procedure.

doPatches
Give TRUE if you want to process all the patches converted to facets.

error
Give the error for the conversion.

Return Value NOERROR

Comments This method is useful if you want facets even for patches based objects , and be able to control the error in the tesselization process.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.8.I3DShScene Interfac e

Interface ID: IID_I3DShScene
Interface file: I3DShScn.h

A Scene is the "master" object that contains all the 3D information to create an image: list of objects, tree hierarchy, light sources, cameras, rendering settings, etc.

Because a Scene is an Object, it can return a **I3DShObject** Interface: a scene can be instanciated in another scene as a "sub-scene", also called sometimes a "Complex Object". This allows the creation and the manipulation of objects that are not limited to the basic primitives.

```
DECLARE_INTERFACE_(I3DShScene, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShScene methods
  //-- 3D Objects list
  STDMETHOD_(ULONG, Get3DObjectsCount) (THIS) PURE;
 STDMETHOD_(I3DShObject*, Get3DObjectByIndex) (THIS_ ULONG index) PURE;
      // index is 1 based
  STDMETHOD_(I3DShObject*, Get3DObjectByName) (THIS_ char* name) PURE;
  STDMETHOD(Insert3DObject) (THIS_ I3DShObject* object) PURE;
  STDMETHOD(Remove3DObject) (THIS_ I3DShObject* object) PURE;
  STDMETHOD(Changed3DObject) (THIS_ I3DShObject* object) PURE;
  //-- Tree root (Universe) setting
  STDMETHOD_(I3DShGroup*, GetTreeRoot) (THIS) PURE;
  STDMETHOD(SetTreeRoot) (THIS_ I3DShGroup* universe) PURE;    // Works
      only if there was no tree root in the scene
  STDMETHOD(CreateTreeRootIfNone) (THIS) PURE;
  //-- Quick tree access
 STDMETHOD_(I3DShTreeElement*, GetTreeElementByName) (THIS_ char* name)
      PURE;
  STDMETHOD_(ULONG, GetInstancesCount) (THIS) PURE;
  STDMETHOD_(I3DShInstance*, GetInstanceByIndex) (THIS_ ULONG index)
      PURE;  // index is 1 based
  STDMETHOD_(ULONG, GetLightsourcesCount) (THIS) PURE;
  STDMETHOD_(I3DShLightsource*, GetLightsourceByIndex) (THIS_ ULONG
      index) PURE;  // index is 1 based
  STDMETHOD_(ULONG, GetCamerasCount) (THIS) PURE;
  STDMETHOD_(I3DShCamera*, GetCameraByIndex) (THIS_ ULONG index) PURE;
      // index is 1 based
  //-- Misc. rendering settings
  STDMETHOD(SetProductionFormat) (THIS_ NUM3D sizeh, NUM3D sizev, NUM3D
      dpi) PURE;  // sizeh and sizev in points (=1/72 inch)
  STDMETHOD(GetProductionFormat) (THIS_ NUM3D* sizeh, NUM3D* sizev,
      NUM3D* dpi) PURE;  // sizeh and sizev in points (=1/72 inch)
  STDMETHOD(CreateRenderingCameraIfNone) (THIS_ ULONG cameraType, BOOL-
      EAN createDistantLight) PURE;
  STDMETHOD_(I3DShCamera*, GetRenderingCamera) (THIS) PURE;
```

©1995-1997 MetaCreations. All rights reserved.

```
STDMETHOD(SetRenderingCamera) (THIS_ I3DShCamera* camera) PURE;
//-- Misc. Animation
STDMETHOD_(ULONG, GetTime) (THIS_ NUM3D* ticks) PURE;
//-- Misc. Utilities (API level 6)
STDMETHOD(GetProductionFormatBBox) (THIS_ BOX2D* prodFormat, NUM3D*
    dpi) PURE; // New call for non centered production frame. Use
    instead of GetProductionFormat
STDMETHOD(GetFullPathName) (THIS_ char* pathName) PURE;
STDMETHOD_(I3DShTreeElement*, GetTreeElementFromFullName)(THIS_ char*
    fullName) PURE; // Returns instance with given full "path name" or
    NULL if no such instance is in the hierarchy
STDMETHOD(GetAmbientLight) (THIS_ COLOR3D* color) PURE;
//-- Better render settings access (API level 7), using 'renderHelper'
STDMETHOD(GetProductionFormatBBox7) (THIS_ BOX2D* prodFormat, NUM3D*
    dpi, void* renderHelper) PURE;// Use this call if you have a ren-
    derHelper (see Post Render Filters). renderHelp=NULL gives same
    behavior as GetProductionFormatBBox()
STDMETHOD_(I3DShCamera*, GetRenderingCamera7) (THIS_ void* render-
    Helper) PURE // Use this call if you have a renderHelper (see Post
    Render Filters). renderHelp=NULL gives same behavior as GetRender-
    ingCamera()  };
```

I3DShScene::Get3DObjectsCount

ULONG I3DShScene::Get3DObjectsCount() ;

Call **I3DShScene::Get3DObjectsCoun t** to get the number of 3D Objects in a scene.

Return Value Number of 3D Objects (I3DShObject interface) in the scene.

Comments Use this in conjunction with I3DShScene::Get3DObjectByIndex.

I3DShScene::Get3DObjectByInde x

I3DShObject* I3DShScene::Get3DObjectByIndex (*index***);**
ULONG *index*

Call **I3DShScene::Get3DObjectByInde x** to get a 3D Object in the Scene.

Parameter → *index*
3D Object index. Must be between 1 and I3DShScene::Get3DObjectsCount

Return Value Pointer on the 3D Object.

Comments Use this call in conjunction with I3DShScene::Get3DObjectsCount to browse through the Objects List of a Scene.

I3DShScene::Get3DObjectByNam e

I3DShObject * I3DShScene::Get3DObjectByName (*name***);**

©1995-1997 MetaCreations. All rights reserved.

char* *name*

Call **I3DShScene::Get3DObjectByNam e** to look for a 3D Object by its name.

Parameter → *name*
The name to find.

Return Value Pointer on the 3D Object, NULL of no object was found.

Note Each 3D Object has a unique name in the 3D Objects List.

I3DShScene::Insert3DObject

HRESULT I3DShScene::Insert3DObject (*object***);**
I3DShObject* *object*

Call **I3DShScene::Insert3DObjec t** to put a new 3D Object in the Scene Objects List.

Parameter → *object*
Pointer on the 3D Object to add to the list.

Return Value S_OK

Comments The 3D Object will have to be referenced with an Instance at least one time in the Scene. Using this call is in fact optional, because if an Instance is added to the Scene Hierarchy, then its Object is automatically added to the Objects List if it was not already in it.

I3DShScene::Remove3DObject

HRESULT I3DShScene::Remove3DObject (*object***);**
I3DShObject* *object*

Call **I3DShScene::Remove3DObjec t** to remove a 3D Object from the Objects List.

Parameter → *object*
Pointer on the 3D Object to remove from the list.

Return Value S_OK

Comments If the Object is still referenced in the tree, it will not be removed, and **I3DShScene::Remove3DObjec t** will not do anything.

I3DShScene::Changed3DObject

HRESULT I3DShScene::Changed3DObject (*object***);**
I3DShObject* *object*

©1995-1997 MetaCreations. All rights reserved.

Call **I3DShScene::Changed3DObjec t** to inform the Scene that a 3D Object has changed.

Parameter *object*
Pointer on the 3D Object to remove from the list.

Return Valu e S_OK

Comments Use this call when any change has been performed on a 3D Object. This will allow the Scene to update any internal data, redraw objects on screen, and inform all 3D Database clients of the change.

I3DShScene::GetTreeRoot

I3DShGroup* I3DShScene::GetTreeRoot() ;

Call **I3DShScene::GetTreeRoo t** to get the topmost Tree Element (sometimes called the "Universe"). It is always a Group.

Return Valu e Pointer on the topmost Group. NULL if there is none yet (use I3DShScene::CreateTree-RootIfNone in that case).

I3DShScene::SetTreeRoot

HRESULT I3DShScene::SetTreeRoot (*universe***);**
I3DShGroup* *universe*

Call **I3DShScene::SetTreeRoo t** to set the topmost Tree Element (sometimes called the "Universe") of a Scene. It must be a Group.

Parameter *universe*
Pointer on the topmost Group.

Return Valu e S_OK

Comments Nothing will happen if there is already a Tree Root in the Scene. This call is rarely used. A preferred technique is to call I3DShScene::CreateTreeRootIfNone, then get the Tree Root by calling **I3DShScene::GetTreeRoo t**.

I3DShScene::CreateTreeRootIfNon e

HRESULT I3DShScene::CreateTreeRootIfNone() ;

Call **I3DShScene::CreateTreeRootIfNon e** to make sure that there is a Tree Root in the Scene.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments It is a good idea to call this just after creating a scene and before starting to insert Tree Elements in the hierarchy. Call I3DShScene::GetTreeRoot afterwards.

I3DShScene::GetTreeElementByNam e

I3DShTreeElement* I3DShScene::GetTreeElementByName (*name***);**
char* *name*

Call **I3DShScene::GetTreeElementByNam e** to find the tree element called name.

Parameter → *name*
The name of the tree element in the C String format (null - terminated)

Return Valu e a Tree Element or NULL if none.

I3DShScene::GetInstancesCoun t

ULONG I3DShScene::GetInstancesCount() ;

Call **I3DShScene::GetInstancesCoun t** to get the number of Instances in a Scene.

Return Valu e Number of Instances in the Scene.

Comments Use this call with I3DShScene::GetInstanceByIndex to browse through the instances of a scene.

I3DShScene::GetInstanceByInde x

I3DShInstance* I3DShScene::GetInstanceByIndex (*index***);**
ULONG *index*

Call **I3DShScene::GetInstanceByInde x** to get an Instance in the Scene.

Parameter → *index*
Instance index. Must be between 1 and I3DShScene::GetInstancesCount.

Return Valu e Pointer on the Instance in the Scene Tree.

Comments Use this call with I3DShScene::GetInstancesCount to browse through the instances of a scene.

I3DShScene::GetLightsourcesCoun t

ULONG I3DShScene::GetLightsourcesCount() ;

©1995-1997 MetaCreations. All rights reserved.

Call **I3DShScene::GetLightsourcesCoun t** to get the number of Light Sources in a Scene.

Return Value Number of Light Sources in the Scene.

Comments Use this call with I3DShScene::GetLightsourceByIndex to browse through the Light Sources of a scene.

I3DShScene::GetLightsourceByInde x

I3DShLightsource* I3DShScene::GetLightsourceByIndex (*index***);**
ULONG *index*

Call **I3DShScene::GetLightsourceByInde x** to get a Light Source in the Scene.

Parameter → *index*
Instance index. Must be between 1 and I3DShScene::GetLightsourcesCount.

Return Value Pointer on the Light Source in the Scene Tree.

Comments Use this call with I3DShScene::GetLightsourcesCount to browse through the Light Sources of a scene.

I3DShScene::GetCamerasCount

ULONG I3DShScene::GetCamerasCount () ;

Call **I3DShScene::GetCamerasCoun t** to get the number of Cameras in a Scene.

Return Value Number of Cameras in the Scene.

Comments Use this call with I3DShScene::GetCameraByIndex to browse through the Cameras of a scene.

I3DShScene::GetCameraByInde x

I3DShCamera* I3DShScene::GetCameraByIndex (*index***);**
ULONG *index*

Call **I3DShScene::GetCameraByInde x** to get a Camera in the Scene.

Parameter → *index*
Instance index. Must be between 1 and I3DShScene::GetCamerasCount.

Return Value Pointer on the Camera in the Scene Tree.

Comments Use this call with I3DShScene::GetCamerasCount to browse through the Cameras of a

©1995-1997 MetaCreations. All rights reserved.

scene.

I3DShScene::SetProductionFormat

HRESULT I3DShScene::SetProductionFormat (*sizeh, sizev, dpi***);**
NUM3D *sizeh*
NUM3D *sizev*
NUM3D *dpi*

Call **I3DShScene::SetProductionForma t** to set the final image size and resolution.

Parameters

sizeh
Horizontal image size in Points.

sizev
Vertical image size in Points.

dpi
Image resolution in dot per inch.

Return Value S_OK.

Comments 1 Point = 1/72 inch.

Number or horizontal pixels = sizeh * dpi / 72
Number or vertical pixels = sizev * dpi / 72

I3DShScene::GetProductionFormat

HRESULT I3DShScene::GetProductionFormat (*sizeh, sizev, dpi***);**
NUM3D* *sizeh*
NUM3D* *sizev*
NUM3D* *dpi*

Call **I3DShScene::GetProductionForma t** to get the current Production Format settings of a Scene.

Parameter

sizeh
Where to return the horizontal image size in Points.

sizev
Where to return the vertical image size in Points.

dpi
Where to return the image resolution in dot per inch.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK.

Comments 1 Point = 1/72 inch.

Number or horizontal pixels = sizeh * dpi / 72
Number or vertical pixels = sizev * dpi / 72

I3DShScene::CreateRenderingCameraIfNon e

HRESULT I3DShScene::CreateRenderingCameraIfNone (*cameraType, createDistantLight***);**
ULONG *cameraType*
BOOLEAN *createDistantLight*

Call **I3DShScene::CreateRenderingCameraIfNon e** to make sure that there is a least a camera we can render from.

Parameters *cameraType*
The Class ID of the camera to create. For a conical camera, enter IDTYPE('c','o','n','i').

createDistantLight
TRUE if a default Distant Light should be created at the same time.

Return Value S_OK.

Comments This call can be used if **I3DShScene::GetRenderingCamer a** returns NULL. It is typically used by Renderers or by 3D Import Filters as a safeguard. It will create a camera if there is none in the Scene, and set it as the Rendering Camera.
The createDistantLight parameter is just a convenient way to create a Light Source at the same time. This is used mostly by 3D Import Filters.

I3DShScene::GetRenderingCamer a

I3DShCamera* I3DShScene::GetRenderingCamera() ;

Call **I3DShScene::GetRenderingCamer a** to get the Rendering Camera, i.e. the camera the final image will be rendered from.

Return Value A pointer on the Camera in the Tree that will be used as the Rendering Camera. NULL if not set yet.

Comments If NULL is returned, you must create a Camera, insert it in the Tree and set it as the Rendering Camera. **I3DShScene::CreateRenderingCameraIfNon e** does all this in one call. This call is typically used by Renderers.

I3DShScene::SetRenderingCamer a

HRESULT I3DShScene::SetRenderingCamera (*camera***);**

©1995-1997 MetaCreations. All rights reserved.

I3DShCamera* *camera*

Use **I3DShScene::SetRenderingCamer a** to set the Rendering Camera.

Parameter *camera*
The Camera to be used as the Rendering Camera.

Return Valu e S_OK

Comments The Rendering Camera must be one of the Camera already in the Scene.

I3DShScene::GetTime

ULONG I3DShScene::GetTime (*ticks***)**
NUM3D* *ticks*

Call **I3DShScene::GetTim e** to know at what time the scene is.

Parameter *ticks*
you get the time in seconds

Return Valu e the time in ticks.

I3DShScene::GetProductionFormatBBo x

HRESULT I3DShScene::GetProductionFormatBBox (*prodFormat, dpi***);**
BOX2D* *prodFormat*
NUM3D* *dpi*

Call **I3DShScene::GetProductionFormatBBo x** to get the current Production Format settings of a Scene.

Parameters *prodFormat*
Where to return the production frame range in Points in the Screen Coordinate System.

dpi
Where to return the image resolution in dot per inch.

Return Valu e S_OK.

Comments This call is useful when the production frame is not centered.

I3DShScene::GetFullPathNam e

HRESULT I3DShScene::GetFullPathName (*pathName***);**
char* *pathName*

©1995-1997 MetaCreations. All rights reserved.

Call **I3DShScene::GetFullPathNam e** to get the full path name of the scene. This is the name of the scene and all the fathers starting from the universe.

Parameters ⇨ *pathName*
Points to where to return the full path name.

Return Valu e S_OK

Comments The tree element names are separated with the kPathNameSeparator constant.

I3DShScene::GetTreeElementFromFullNam e

I3DShTreeElement* I3DShTreeElement::GetTreeElementFromFullName (*fullName***);**
char* *fullName*

Call **I3DShScene::GetTreeElementFromFullNam e** to get the instance with a given full "path name" or NULL if no such instance is in the hierarchy.

Parameter ⇨ *fullName*
The full path name.

Return Valu e The instance or NULL if this instance is not in the hierarchy.

I3DShScene::GetAmbientLight

HRESULT I3DShTreeElement::GetAmbientLight (*color***);**
COLOR3D* *color*

Call **I3DShScene::GetAmbientLigh t** to get the color of the ambient light.

Parameter ⇨ *color*
Where to return the color.

Return Valu e NOERROR.

I3DShScene::GetProductionFormatBBox 7

HRESULT I3DShScene::GetProductionFormatBBox (*prodFormat, dpi, renderHelper***);**
BOX2D* *prodFormat*
NUM3D* *dpi*
void* *renderHelper*

Call **I3DShScene::GetProductionFormatBBox 7** instead of **I3DShScene::GetProduc tionFormatBBo x** if you have a renderHelper.

©1995-1997 MetaCreations. All rights reserved.

Parameter *prodFormat*
Where to return the production frame range in Points.

dpi
Where to return the image resolution in dot per inch.

renderHelper
The pointer to the renderHelper, if used.

Return Value NOERROR.

Comments This call is the same as I3DShScene::GetProductionFormatBBox when the renderHelper is set to NULL.

I3DShScene::GetRenderingCamera7

I3DShCamera* I3DShScene::GetRenderingCamera7 (*renderHelper***);**
void* *renderHelper*

Call **I3DShScene::GetRenderingCamera 7** instead of I3DShScene::GetRenderingCamera if you have a renderHelper.

Parameter *renderHelper*
The pointer to the renderHelper, if used.

Return Value A pointer on the Camera in the Tree that will be used as the Rendering Camera. NULL if not set yet.

Comments This call is the same as I3DShScene::GetRenderingCamera when the renderHelper is set to NULL.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.9.I3DShPolygonList Interfac e

Interface ID: IID_I3DShPolygonList
Interface file: I3DShObj.h

A Polygon List is simply a bag of triangular facets in unsorted order. Of course, it will not return patches, only facets: use I3DShPrimitive::EnumFacets.

A Polygon List is a Primitive. Therefore it supports the **I3DShObject** and **I3DShPrimitive** Interfaces.

A Polygon List can have two additional features: Normals and a UV Space.

When you build a Polygon List, it is highly recommended to provide Normals in the facets. This will ensure best and smooth shading. If Normals cannot be provided, then you must call **I3DShPolygonList::CalcNormal s** to reconstruct the Normals. However this is a costly and long process that is not 100% reliable.

If you can, it is also recommended to provide support for UV Space in the facets. Supporting UV Space will enable Direct Parametric Mapping, which is the best technique to apply 2D Shaders on a surface. Otherwise, Projection Mapping will have to be used, which may result in some texture deformations on the objects.

```
DECLARE_INTERFACE_(I3DShPolygonList, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShPolygonList methods
  //-- Init calls
  STDMETHOD(Init) (THIS_ BOOLEAN hasNormals, BOOLEAN hasUV) PURE;
  STDMETHOD(SetUVSpace0) (THIS_ VECTOR2D* minUV, VECTOR2D* maxUV) PURE;
      // Set UV Space #0. Use SetUVSpace if you have multiple UV Spaces
  //-- Facets handling
  STDMETHOD(PreAllocateFacets) (THIS_ ULONG nbFacets) PURE;  // Use this
      if you know in advance how many facets you will build
  STDMETHOD(AddFacet) (THIS_ FACET3D* facet) PURE;
  STDMETHOD_(ULONG, CountFacets) (THIS) PURE;
 STDMETHOD(CalcNormals) (THIS_ NUM3D smoothAngle) PURE; // Use this call
      if you do not know how to make the normals
  STDMETHOD_(FACET3D*, GetIndFacets) (THIS_ ULONG index) PURE; // (index
      is 1 based)
  STDMETHOD(RemoveFacet) (THIS_ ULONG index) PURE; // (index is 1 based)
  //--Init call
  STDMETHOD(SetHasUVSpace) (THIS_ BOOLEAN hasUV) PURE; // If false, will
      remove all previous UV Space definitions
  //-- New UV Space calls (API level 6)
  STDMETHOD(SetUVSpace) (THIS_ UVSpaceInfo* uvSpaceInfo) PURE; // Set an
      UV Space. The UV space will be added if it was not defined yet
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE; // Get UV Space info
```

©1995-1997 MetaCreations. All rights reserved.

```
};
```

I3DShPolygonList::Init

HRESULT I3DShPolygonList::Init (*hasNormals, hasUV***);**
BOOLEAN *hasNormals*
BOOLEAN *hasUV*

I3DShPolygonList::Ini t must be called right after creating a Polygon List object.

Parameters

hasNormals
TRUE if Normals are provided in the facets.

hasUV
TRUE if this object supports a UV Space.

Return Valu e S_OK

Comments See the explanations in the description of the I3DShPolygonList interface. It is highly recommended to provide Normals, and recommended to provide UV Space information.

I3DShPolygonList::SetUVSpace0

HRESULT I3DShPolygonList::SetUVSpace0 (*minUV, maxUV***);**
VECTOR2D* *minUV*
VECTOR2D* *maxUV*

I3DShPolygonList::SetUVSpace 0 must be called after I3DShPolygonList::Init if there is any UV Space on the object, in order to set the UV Space boundaries.

Parameters

minUV
The minimum values for UV.

maxUV
The maximum values for UV.

Return Valu e S_OK

Comments I3DShPolygonList::SetUVSpace0 is an old version that can set only one UV space. To set multiple UV spaces, I3DShPolygonList::SetUVSpace should be used instead.

I3DShPolygonList::PreAllocateFacets

HRESULT I3DShPolygonList::PreAllocateFacets (*nbFacets***);**
ULONG *nbFacets*

©1995-1997 MetaCreations. All rights reserved.

Call **I3DShPolygonList::PreAllocateFacet s** to pre-allocate the memory space needed for *nbFacets* facets.

Parameter → *nbFacets*
Number of facets that will be added later with I3DShPolygonList::AddFacet.

Return Value S_OK

Comments Use this call if you know in advance how many facets you will build and want to make sure that there will be enough RAM to store them. This allows the code to fail immediately if the 3D Shell is running out of RAM, instead of failing later after many facets may already be calculated and stored. It is also more efficient, because the data block will be allocated only once instead of being regularly re-allocated and copied. The use of this call is optional.

I3DShPolygonList::AddFacet

HRESULT I3DShPolygonList::AddFacet (*facet***);**
FACET3D* *facet*

Use **I3DShPolygonList::AddFace t** to add facets to a Polygon List object one after the other.

Parameter → *facet*
Points to the facet to add to the Polygon List. The facet content is copied, I3DShPolygonList::AddFacet does not take property of the facet pointer.

Return Value S_OK

Comments Fill-in the facet data according to the flags provided in I3DShPolygonList::Init: give Normals and UV Space info as necessary. See the FACET3D structure in the "Database Overview" chapter for more information.
I3DShPolygonList::AddFace t fails if running out of RAM (see **I3DShPolygonList::PreAllocateFacet s** about memory management).

I3DShPolygonList::CountFacets

ULONG I3DShPolygonList::CountFacets() ;

Call **I3DShPolygonList::CountFacet s** to get the number of facets currently stored in the list.

Return Value the number of facets.

I3DShPolygonList::CalcNormals

HRESULT I3DShPolygonList::CalcNormals (*smoothAngle***);**

©1995-1997 MetaCreations. All rights reserved.

NUM3D *smoothAngle*

Use **I3DShPolygonList::CalcNormal s** if Normals could not be provided and therefore must be reconstructed.

Parameter → *smoothAngle*
Angle under which smoothing should occur between facets. This angle is in degrees. 0° means no smoothing, 90° means always smooth. 30° is a good default value.

Return Value S_OK

Comments **I3DShPolygonList::CalcNormal s** must be called if hasNormals was FALSE when calling **I3DShPolygonList::Ini t**.

I3DShPolygonList::GetIndFacets

FACET3D* I3DShPolygonList::GetIndFacets (*index***);**
ULONG *index*

Use **I3DShPolygonList::GetIndFacet s** to get a facet in the list.

Parameter → *index*
Index of the facet, must be between 1 and I3DShPolygonList::CountFacets.

Return Value The facet.

Comments index must be between 1 and **I3DShPolygonList::CountFacet s**.

I3DShPolygonList::RemoveFacet

HRESULT I3DShPolygonList::RemoveFacet (*index***);**
ULONG *index*

Use **I3DShPolygonList::RemoveFace t** to remove a facet in the list.

Parameter → *index*
Index of the facet, starting from 1.

Return Value NOERROR

Comments index must be between 1 and **I3DShPolygonList::CountFacet s**.

I3DShPolygonList::SetHasUVSpace

HRESULT I3DShPolygonList::SetHasUVSpace (*hasUV***);**
BOOLEAN *hasUV*

©1995-1997 MetaCreations. All rights reserved.

Use **I3DShPolygonList::SetHasUVSpac e** to set the flag HasUVSpace.

Parameter → *hasUV*
the flag for HasUVSpace.

Return Valu e NOERROR

Comments if hasUV if false, **I3DShPolygonList::SetHasUVSpac e** will remove all the previous UV spaces.

I3DShPolygonList::SetUVSpace

HRESULT I3DShPolygonList::SetUVSpace (*uvSpaceInfo***);**
UVSpaceInfo* *uvSpaceInfo*

Call **I3DShPolygonList::SetUVSpac e** to add a UV space to the polygon list if it does not exist yet.

Parameters → *uvSpaceInfo*
The information for the added UV space. See the "Data Structure Reference" for more details.

Return Valu e S_OK

Comments **I3DShPolygonList::SetUVSpac e** is the new version that can set multiple UV spaces.

I3DShPolygonList::GetUVSpace

HRESULT I3DShPolygonList::GetUVSpace (*uvSpaceID, uvSpaceInfo***);**
ULONG *uvSpaceID*
UVSpaceInfo* *uvSpaceInfo*

Call **I3DShPolygonList::GetUVSpac e** to get the information on a UV space if the list.

Parameters → *uvSpaceID*
Specify a UV space.

← *uvSpaceInfo*
Where to return the information for the given UV space. See the "Data Structure Reference" for more details.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.10.I3DShPolygonArray Interfac e

Interface ID: IID_I3DShPolygonArray
Interface file: I3DShObj.h

A Polygon Array is an array of 3D points and Normals that describe a surface. See the "3D Database Overview" chapter for more details on Polygon Arrays.

Polygon Arrays are facet-based 3D Objects.

A Polygon Array is a Primitive. Therefore it supports the **I3DShObject** and **I3DShPrimitive** Interfaces.

The "U" 'direction is the direction of the U axis of the UV Space. Same thing for the V direction. For example, do not confuse the uIndex, which is an integer describing the index of a point in the array along the U direction, and the uValue at that index, which is the U value in terms of UV Space for all 3D points at the uIndex index.

Using Polygon Arrays or Patch Arrays when possible is better than using Polygon Lists because it is easier to calculate a UV Space and perform Normal smoothing. There are even calls to do all that for you !

```
DECLARE_INTERFACE_(I3DShPolygonArray, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShPolygonArray methods
  //-- Init calls
  STDMETHOD(Init) (THIS_ ULONG uCount, ULONG vCount, BOOLEAN uClosed,
      BOOLEAN vClosed) PURE;
  //-- Geometry calls
  STDMETHOD(SetVertex) (THIS_ ULONG uIndex, ULONG vIndex, VECTOR3D* ver-
      tex, VECTOR3D* normal) PURE;    // normal can be NULL
  STDMETHOD(GetVertex) (THIS_ ULONG uIndex, ULONG vIndex, VECTOR3D* ver-
      tex, VECTOR3D* normal) PURE;    // normal can be NULL
  //-- (u,v) calls
  STDMETHOD(MeshSetU) (THIS_ ULONG  uIndex, NUM3D uValue) PURE;
  STDMETHOD(MeshSetV) (THIS_ ULONG  vIndex, NUM3D vValue) PURE;
  STDMETHOD(MeshGetU) (THIS_ ULONG  uIndex, NUM3D* uValue) PURE;
  STDMETHOD(MeshGetV) (THIS_ ULONG  vIndex, NUM3D* vValue) PURE;
  //-- Goodies
  STDMETHOD(CalcUV) (THIS) PURE;                      // Calculate for
      you all (u,v) values if you do not know how to make them
 STDMETHOD(CalcNormals) (THIS_ BOOLEAN smooth) PURE;    // Calculate for
      you all normals if you do not know how to make them
  };
```

©1995-1997 MetaCreations. All rights reserved.

I3DShPolygonArray::Init

HRESULT I3DShPolygonArray::Init (*uCount, vCount, uClosed, vClosed***);**
ULONG *uCount*
ULONG *vCount*
BOOLEAN *uClosed*
BOOLEAN *vClosed*

Call **I3DShPolygonArray::Ini t** just after creating a **I3DShPolygonArra y** object to initialize all internal data structures.

Parameters

uCount
Number of vertices in the U direction.

vCount
Number of vertices in the V direction.

uClosed
TRUE if the Array is closed in the U direction.

vClosed
TRUE if the Array is closed in the V direction.

Return Valu e S_OK

Comments **I3DShPolygonArray::Ini t** allocates the Polygon Array in RAM so you can access its points through I3DShPolygonArray::SetVertex and **I3DShPolygonArray::GetVerte x**.

I3DShPolygonArray::SetVertex

HRESULT I3DShPolygonArray::SetVertex (*uIndex, vIndex, vertex, normal***);**
ULONG *uIndex*
ULONG *vIndex*
VECTOR3D* *vertex*
VECTOR3D* *normal*

Use **I3DShPolygonArray::SetVerte x** to set the Vertex and the Normal at a specific point of the Array.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1.

vIndex
Index in the V direction. Index in between 0 and vCount-1.

©1995-1997 MetaCreations. All rights reserved.

vertex
Points to the Vertex coordinates. The values will be copied in the internal Polygon Array data.

normal
Points to the Normal coordinates. The values will be copied in the internal Polygon Array data. The pointer can be NULL if you do not want to provide Normals now, and intend to calculate them later using **I3DShPolygonArray::CalcNormal s**.

Return Value S_OK

Comments It is always better to provide the Normals yourself. Note that because there is only one Normal at each point of the Array, a Array surface is always smooth. If you do not want it to be smooth, do not provide Normals and call **I3DShPolygonArray::CalcNo r-mals(***FALSE***)**.

I3DShPolygonArray::GetVertex

HRESULT I3DShPolygonArray::GetVertex (*uIndex, vIndex, vertex, normal***);**
ULONG *uIndex*
ULONG *vIndex*
VECTOR3D* *vertex*
VECTOR3D* *normal*

Use **I3DShPolygonArray::GetVerte x** to get the Vertex and the Normal at a specific point of the Array.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1.

vIndex
Index in the V direction. Index in between 0 and vCount-1.

vertex
Where to return the Vertex coordinates. The values will be copied from the internal Polygon Array data into this place.

normal
Where to return the Normal coordinates. The values will be copied from the internal Polygon Array data into this place, unless normal is NULL.

Return Value S_OK

I3DShPolygonArray::MeshSetU

HRESULT I3DShPolygonArray::MeshSetU (*uIndex, uValue***);**

©1995-1997 MetaCreations. All rights reserved.

ULONG *uIndex*
NUM3D *uValue*

Use **I3DShPolygonArray::MeshSet U** to set the U coordinate of all points at uIndex.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1 if the Array is open in the U direction, or 0 and uCount if it is closed.

uValue
The U value for all points at uIndex. This value is between 0 and the U value of the last vertex.

Return Value S_OK

Note Note that all points at the same uIndex have the same texture U value. Same thing on V. The UV Space is "parallel" to the array of points: points at the same uIndex have the same uValue.

I3DShPolygonArray::MeshSetV

HRESULT I3DShPolygonArray::MeshSetV (*vIndex, vValue***);**
ULONG *vIndex*
NUM3D *vValue*

Use **I3DShPolygonArray::MeshSet V** to set the V coordinate of all points at vIndex.

Parameters

vIndex
Index in the V direction. Index in between 0 and vCount-1 if the Array is open in the V direction, or 0 and vCount if it is closed.

vValue
The V value for all points at vIndex. This value is between 0 and the V value of the last vertex.

Return Value S_OK

Note Note that all points at the same vIndex have the same texture V value. Same thing on U. The UV Space is "parallel" to the array of points: points at the same vIndex have the same vValue.

I3DShPolygonArray::MeshGetU

HRESULT I3DShPolygonArray::MeshGetU (*uIndex, uValue***);**
ULONG *uIndex*
NUM3D* *uValue*

©1995-1997 MetaCreations. All rights reserved.

Use **I3DShPolygonArray::MeshGet U** to get the U coordinate of all points at uIndex.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1 if the Array is open in the U direction, or 0 and uCount if it is closed.

uValue
Where to copy the U value for all points at uIndex. This value is between 0 and the U value of the last vertex.

Return Value S_OK

Note Note that all points at the same uIndex have the same texture U value. Same thing on V. The UV Space is "parallel" to the array of points.

I3DShPolygonArray::MeshGetV

HRESULT I3DShPolygonArray::MeshGetV (*vIndex, vValue***);**
ULONG *vIndex*
NUM3D* *vValue*

Use **I3DShPolygonArray::MeshGet V** to get the V coordinate of all points at vIndex.

Parameters

vIndex
Index in the V direction. Index in between 0 and vCount-1 if the Array is open in the V direction, or 0 and vCount if it is closed.

vValue
Where to copy the V value for all points at vIndex. This value is between 0 and the V value of the last vertex.

Return Value S_OK

Note Note that all points at the same vIndex have the same texture V value. Same thing on U. The UV Space is "parallel" to the array of points.

I3DShPolygonArray::CalcUV

HRESULT I3DShPolygonArray::CalcUV() ;

Call **I3DShPolygonArray::CalcU V** if you want to have all the UV values calculated for you.

Return Value S_OK

Comments You must call this procedure if you did not set any UV values using **I3DShPolygonArray::MeshSet U** and **I3DShPolygonArray::MeshSet V**.

©1995-1997 MetaCreations. All rights reserved.

UV calculation is based on the distance between vertices in the Array.

I3DShPolygonArray::CalcNormals

HRESULT I3DShPolygonArray::CalcNormals (*smooth***);**
BOOLEAN *smooth*

Use **I3DShPolygonArray::CalcNormal s** to have all Normals calculated for you.

Parameter → *smooth*
TRUE if you want a smooth surface. FALSE if you want to see each Polygon Array facet.

Return Valu e S_OK

Comments You must call this procedure if you did not set any Normals using **I3DShPolygonArray::SetVerte x**.

©1995-1997 MetaCreations. All rights reserved.

2.11.I3DShPatchList Interfac e

Interface ID: IID_I3DShPatchList
Interface file: I3DShObj.h

A Patch List is simply a bag of bicubic patches in unsorted order. Of course, it will not return patches, only facets: use **I3DShPrimitive::EnumPatche s**.

A Patch List is a Primitive. Therefore it supports the **I3DShObject** and **I3DShPrimitive** Interfaces.

A Patch List can have an additional feature: a UV Space.

When you build a Patch List, it is recommended to provide support for UV Space in the patches. Supporting UV Space will enable Direct Parametric Mapping, which is the best technique to apply 2D Shaders on a surface. Otherwise, Projection Mapping will have to be used, which may result in some texture deformations on the objects.

```
DECLARE_INTERFACE_(I3DShPatchList, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShPatchList methods
  //-- Init calls
  STDMETHOD(Init) (THIS_ BOOLEAN hasUV) PURE;
  STDMETHOD(SetUVSpace0) (THIS_ VECTOR2D* minUV, VECTOR2D* maxUV) PURE;
      // Set UV Space #0. Use SetUVSpace if you have multiple UV Spaces
  STDMETHOD(SetHasUVSpace) (THIS_ BOOLEAN hasUV) PURE;// If false, will
      remove all previous UV Space definitions
  //-- Patches handling
  STDMETHOD(AddPatch) (THIS_ PATCH3D* patch) PURE;
  STDMETHOD_(ULONG, CountPatches) (THIS) PURE;
  //-- New UV Space calls (API level 6)
  STDMETHOD(SetUVSpace) (THIS_ UVSpaceInfo* uvSpaceInfo) PURE; // Set an
      UV Space. The UV space will be added if it was not defined yet
  STDMETHOD(GetUVSpace) (THIS_ ULONG uvSpaceID, UVSpaceInfo* uvSpace-
      Info) PURE; // Get UV Space info
  };
```

I3DShPatchList::Init

HRESULT I3DShPatchList::Init (*hasUV***);**
BOOLEAN *hasUV*

I3DShPatchList::Init must be right after creating a Patch List object.

Parameter *hasUV*
TRUE if this object supports a UV Space.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

Comments See the explanations in the description of the I3DShPatchList interface. It is recommended to provide UV Space information.

I3DShPatchList::SetUVSpace0

HRESULT I3DShPatchList::SetUVSpace0 (*minUV, maxUV***);**
VECTOR2D* *minUV*
VECTOR2D* *maxUV*

I3DShPatchList::SetUVSpace 0 must be called after I3DShPatchList::Init if there is any UV Space on the object, in order to set the UV Space boundaries.

Parameters *minUV*
The minimum values for UV.

maxUV
The maximum values for UV.

Return Value S_OK

Comments **I3DShPatchList::SetUVSpace 0** is an old version that can set only one UV space. To set multiple UV spaces, I3DShPatchList::SetUVSpace should be used instead.

I3DShPatchList::SetHasUVSpace

HRESULT I3DShPatchList::SetHasUVSpace (*hasUV***);**
BOOLEAN *hasUV*

Use **I3DShPatchList::SetHasUVSpac e** to set the flag HasUVSpace.

Parameter *hasUV*
the flag for HasUVSpace.

Return Value NOERROR

Comments if hasUV if false, **I3DShPatchList::SetHasUVSpac e** will remove all the previous UV spaces.

I3DShPatchList::AddPatch

HRESULT I3DShPatchList::AddFacet (*patch***);**
PATCH3D* *patch*

Use **I3DShPatchList::AddPatc h** to add patches to a Patch List object one after the other.

©1995-1997 MetaCreations. All rights reserved.

Parameter *patch*
Points to the patch to add to the Patch List. The patch content is copied, I3DShPatchList::AddPatch does not take property of the patch pointer.

Return Value S_OK

Comments Fill-in the patch data according to the flags provided in I3DShPatchList::Init: give a UV Space info as necessary. See the PATCH3D structure for more information. **I3DShPatchList::AddPatc h** fails if running out of RAM.

I3DShPatchList::CountPatches

ULONG I3DShPatchList::CountPatches() ;

Call **I3DShPatchList::CountPatche s** to get the number of patches currently stored in the list.

Return Value the number of patches.

I3DShPatchList::SetUVSpace

HRESULT I3DShPatchList::SetUVSpace (*uvSpaceInfo***);**
UVSpaceInfo* *uvSpaceInfo*

Call **I3DShPatchList::SetUVSpac e** to add a UV space to the patches list if it does not exist yet.

Parameter *uvSpaceInfo*
The information for the added UV space. See the "Data Structure Reference" for more details.

Return Value S_OK

Comments **I3DShPatchList::SetUVSpac e** is the new version that can set multiple UV spaces.

I3DShPatchList::GetUVSpace

HRESULT I3DShPatchList::GetUVSpace (*uvSpaceID, uvSpaceInfo***);**
ULONG *uvSpaceID*
UVSpaceInfo* *uvSpaceInfo*

Call **I3DShPatchList::GetUVSpac e** to get the information on a UV space if the list.

Parameters *uvSpaceID*
Specify a UV space.

©1995-1997 MetaCreations. All rights reserved.

uvSpaceInfo
Where to return the information for the given UV space. See the "Data Structure Reference" for more details.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

2.12.I3DShPatchArray Interfac e

Interface ID: IID_I3DShPatchArray
Interface file: I3DShObj.h

A Patch Array is an array of bicubic patches. It is very similar to a Polygon Array in many aspects, but has some differences like the fact that Normals are not needed. They always generate nice and smooth surfaces. See the "3D Database Overview" chapter for more details on Patch Arrays, especially on the conditions to have Geometric Continuity across patches.

Normals are automatically calculated, you do not have to worry about it.

Patch Arrays are patch-based 3D Objects.

A Patch Array is a Primitive. Therefore it supports the **I3DShObject** and **I3DShPrimitive** Interfaces.

The "U" 'direction is the direction of the U axis of the UV Space. Same thing for the V direction. For example, do not confuse the uIndex, which is an integer describing the index of a point in the array along the U direction, and the uValue at that index, which is the U value in terms of UV Space for all 3D points at the uIndex index.

Using Polygon Arrays or Patch Arrays when possible is better than using Polygon Lists because it is easier to calculate a UV Space and perform Normal smoothing. There are even calls to do all that for you !

```
DECLARE_INTERFACE_(I3DShPatchArray, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShPatchArray methods
  //-- Init calls
  STDMETHOD(Init) (THIS_ ULONG uCount, ULONG vCount) PURE;
  //-- Geometry calls
  STDMETHOD(SetPatch) (THIS_ ULONG uIndex, ULONG vIndex, PATCH3D* patch)
      PURE;
  STDMETHOD(GetPatch) (THIS_ ULONG uIndex, ULONG vIndex, PATCH3D* patch)
      PURE;
  //-- (u,v) calls
  STDMETHOD(MeshSetU) (THIS_ ULONG uIndex, NUM3D uValue) PURE;
  STDMETHOD(MeshSetV) (THIS_ ULONG vIndex, NUM3D vValue) PURE;
  STDMETHOD(MeshGetU) (THIS_ ULONG uIndex, NUM3D* uValue) PURE;
  STDMETHOD(MeshGetV) (THIS_ ULONG vIndex, NUM3D* vValue) PURE;
  //-- Goodies
  STDMETHOD(CalcUV) (THIS) PURE;                        // Calculate for
      you all (u,v) values if you do not know how to make them
  };
```

©1995-1997 MetaCreations. All rights reserved.

I3DShPatchArray::Init

HRESULT I3DShPatchArray::Init (*uCount, vCount***);**
ULONG *uCount*
ULONG *vCount*

Call **I3DShPatchArray::Ini t** just after creating a **I3DShPatchArra y** object to initialize all internal data structures.

Parameters

uCount
Number of patches in the U direction.

vCount
Number of patches in the V direction.

Return Valu e S_OK

Comments Note that there is no notion of closure like with **I3DShPolygonArra y**. You need to provide the closing patch to have a closed surface.

I3DShPatchArray::SetPatch

HRESULT I3DShPatchArray::SetPatch (*uIndex, vIndex, patch***);**
ULONG *uIndex*
ULONG *vIndex*
PATCH3D* *patch*

Use **I3DShPatchArray::SetPatc h** to set Patch values in the Patch Array.

Parameters

uIndex
Patch index in the U direction, between 0 and uCount-1

vIndex
Patch index in the V direction, between 0 and vCount-1

patch
Points to the values for this patch. See the description of PATCH3D for more details (it is an array of 16 3D Points). The values are copied in the Patch Array internal data.

Return Valu e S_OK

Comments Use this call to set patches one after the other in the Patch Array. See the "3D Database Overview" chapter for more details on Patch Arrays, especially on the conditions to have Geometric Continuity across patches.

©1995-1997 MetaCreations. All rights reserved.

I3DShPatchArray::GetPatch

HRESULT I3DShPatchArray::GetPatch (*uIndex, vIndex, patch***);**
ULONG *uIndex*
ULONG *vIndex*
PATCH3D* *patch*

Parameters

uIndex
Patch index in the U direction, between 0 and uCount-1

vIndex
Patch index in the V direction, between 0 and vCount-1

patch
Points to where to return the values for this patch. See the description of **PATCH3D** for more details (it is an array of 16 3D Points). The values are copied from the Patch Array internal data.

Return Value S_OK

Comments See the "3D Database Overview" chapter for more details on Patch Arrays.

I3DShPatchArray::MeshSetU

HRESULT I3DShPatchArray::MeshSetU (*uIndex, uValue***);**
ULONG *uIndex*
NUM3D *uValue*

Use **I3DShPatchArray::MeshSet U** to set the U coordinate of all points at uIndex.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1 if the Array is open in the U direction, or 0 and uCount if it is closed.

uValue
The U value for all points at uIndex. This value is between 0 and the U value of the last point.

Return Value S_OK

Note Note that all points at the same uIndex have the same texture U value. Same thing on V. The UV Space is "parallel" to the array of points.

I3DShPatchArray::MeshSetV

HRESULT I3DShPatchArray::MeshSetV (*vIndex, vValue***);**

©1995-1997 MetaCreations. All rights reserved.

ULONG *vIndex*
NUM3D *vValue*

Use **I3DShPatchArray::MeshSet V** to set the V coordinate of all points at vIndex.

Parameters

vIndex
Index in the V direction. Index in between 0 and vCount-1 if the Array is open in the V direction, or 0 and vCount if it is closed.

vValue
The V value for all points at vIndex. This value is between 0 and the V value of the last point.

Return Value S_OK

Note Note that all points at the same vIndex have the same texture V value. Same thing on U. The UV Space is "parallel" to the array of points.

I3DShPatchArray::MeshGetU

HRESULT I3DShPatchArray::MeshGetU (*uIndex, uValue***);**
ULONG *uIndex*
NUM3D* *uValue*

Use **I3DShPatchArray::MeshGet U** to get the U coordinate of all points at uIndex.

Parameters

uIndex
Index in the U direction. Index in between 0 and uCount-1 if the Array is open in the U direction, or 0 and uCount if it is closed.

uValue
Where to copy the U value for all points at uIndex. This value is between 0 and the U value of the last point.

Return Value S_OK

Note Note that all points at the same uIndex have the same texture U value. Same thing on V. The UV Space is "parallel" to the array of points.

I3DShPatchArray::MeshGetV

HRESULT I3DShPatchArray::MeshGetV (*vIndex, vValue***);**
ULONG *vIndex*
NUM3D* *vValue*

Use **I3DShPatchArray::MeshGet V** to get the V coordinate of all points at vIndex.

©1995-1997 MetaCreations. All rights reserved.

Parameters *vIndex*
Index in the V direction. Index in between 0 and vCount-1 if the Array is open in the V direction, or 0 and vCount if it is closed.

vValue
Where to copy the V value for all points at vIndex. This value is between 0 and the V value of the last point.

Return Value S_OK

Note Note that all points at the same vIndex have the same texture V value. Same thing on U. The UV Space is "parallel" to the array of points.

I3DShPatchArray::CalcUV

HRESULT I3DShPatchArray::CalcUV() ;

Call **I3DShPatchArray::CalcU V** if you want to have all the UV values calculated for you.

Return Value S_OK

Comments You must call this procedure if you did not set any UV values using **I3DShPatchArray::MeshSet U** and **I3DShPatchArray::MeshSet V**.

UV calculation is based on the distance between vectors in the Array.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.13.I3DShFlatSurface Interfac e

Interface ID: IID_I3DShFlatSurface
Interface file: I3DShObj.h

A Flat Surface is a special kind of 3D object that is totally flat and is defined by the boundaries of 2D Bézier curves and polygons (called "Paths"). A closed 2D Path will be filled in 2D holes can be performed using "Compound" paths.

Flat Surfaces can be used to make caps of complex surfaces.

Flat Surfaces are patch-based 3D Objects.

A Flat Surface is a Primitive. Therefore it supports the **I3DShObject** and **I3DShPrimitive** Interfaces.

Here is the grammar for building a Flat Surface:

```
    FLATSURFACE =
      BeginFlatSurface
      GROUP
      EndFlatSurface

    GROUP =
      BeginGroup
      GROUP | COMPOUND | PATH        1 or more times
      EndGroup

    COMPOUND =
      BeginCompound
      GROUP | COMPOUND | PATH        1 or more times
      EndCompound

    PATH =
      BeginPath
      LineTo | BezierTo              1 or more times
      EndPath
```

```
DECLARE_INTERFACE_(I3DShFlatSurface, I3DShObject) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShFlatSurface methods
  //-- Geometry building calls (see grammar above)
  STDMETHOD(BeginFlatSurface) (THIS) PURE;    // Call this before build-
      ing the surface
  STDMETHOD(EndFlatSurface) (THIS) PURE;      // And this when you are
      done
  STDMETHOD(BeginGroup) (THIS) PURE;
  STDMETHOD(EndGroup) (THIS) PURE;
```

©1995-1997 MetaCreations. All rights reserved.

```
STDMETHOD(BeginCmpnd) (THIS) PURE;          // Use compounds to make
    holes
STDMETHOD(EndCmpnd) (THIS) PURE;
STDMETHOD(BeginPath) (THIS_ VECTOR2D* firstPoint) PURE;
STDMETHOD(LineTo) (THIS_ VECTOR2D* secondPoint) PURE;
STDMETHOD(BezierTo) (THIS_ VECTOR2D* secondPoint, VECTOR2D* third-
    Point, VECTOR2D* fourthPoint) PURE;
STDMETHOD(EndPath) (THIS) PURE;
};
```

I3DShFlatSurface::BeginFlatSurface

HRESULT I3DShFlatSurface::BeginFlatSurface() ;

I3DShFlatSurface::BeginFlatSurfac e must be called first when starting to build a Flat Surface.

Return Value S_OK

I3DShFlatSurface::EndFlatSurface

HRESULT I3DShFlatSurface::EndFlatSurface() ;

I3DShFlatSurface::EndFlatSurfac e must be called when building a Flat Surface is finished.

Return Value S_OK

I3DShFlatSurface::BeginGroup

HRESULT I3DShFlatSurface::BeginGroup() ;

I3DShFlatSurface::BeginGrou p is used to group together all following drawing calls until a balancing. EndGroup is done.

Return Value S_OK

Comments Grouping has no impact on the geometric filling. It is more intended for convenience.

I3DShFlatSurface::EndGroup

HRESULT I3DShFlatSurface::EndGroup() ;

Use **I3DShFlatSurface::EndGrou p** to mark the end of a group.

Return Value S_OK

Comments Each **I3DShFlatSurface::BeginGrou p** must be balanced by a **I3DShFlatSurface::En d-**

©1995-1997 MetaCreations. All rights reserved.

Group. Groups can be nested.

I3DShFlatSurface::BeginCmpnd

HRESULT I3DShFlatSurface::BeginCmpnd() ;

I3DShFlatSurface::BeginCmpn d is used to group together all following drawing calls in a Compound Shape.

Return Value S_OK

Comments This is used to make holes: a closed path inside another will create a hole. Two separate paths inside a large one will make two holes. Important: all Paths must not intersect with any other. They have to be separated or nested like Russian dolls.

I3DShFlatSurface::EndCmpnd

HRESULT I3DShFlatSurface::EndCmpnd() ;

Use **I3DShFlatSurface::EndCmpn d** to mark the end of a Compound Shape.

Return Value S_OK

Comments Each **I3DShFlatSurface::BeginCmpn d** must be balanced by a **I3DShFlatSurface::En d-Cmpnd**.

I3DShFlatSurface::BeginPath

HRESULT I3DShFlatSurface::BeginPath (*firstPoint***);**
VECTOR2D* *firstPoint*

Use **I3DShFlatSurface::BeginPat h** to start a Path. A Path is a list of connected Bézier curves and/or straight segments.

Parameter → *firstPoint*
Starting point of the Path.

Return Value S_OK

I3DShFlatSurface::LineTo

HRESULT I3DShFlatSurface::LineTo (*secondPoint***);**
VECTOR2D* *secondPoint*

I3DShFlatSurface::LineT o will draw a straight line from the current point and second-Point.

©1995-1997 MetaCreations. All rights reserved.

Parameter *secondPoint*
Point to draw the line to.

Return Value S_OK

I3DShFlatSurface::BezierTo

HRESULT I3DShFlatSurface::BezierTo (*secondPoint, thirdPoint, fourthPoint***);**
VECTOR2D* *secondPoint*
VECTOR2D* *thirdPoint*
VECTOR2D* *fourthPoint*

I3DShFlatSurface::BezierT o will draw a Bézier curve from the current point to fourth-Point, using secondPoint and thirdPoint as control points.

Parameters *secondPoint*
First control point.

thirdPoint
Second control point.

fourthPoint
Point do draw the curve to.

Return Value S_OK

I3DShFlatSurface::EndPath

HRESULT I3DShFlatSurface::EndPath() ;

Call **I3DShFlatSurface::EndPat h** when you are done building a Path.

Return Value S_OK

Comments Each call to **I3DShFlatSurface::BeginPat h** must be balanced by a **I3DShFlatSurface::EndPat h**. Paths cannot be nested (use Groups or Compounds). For constructing closed paths, make a LineTo or a BezierTo to the first point of the path.

©1995-1997 MetaCreations. All rights reserved.

2.14.IShRasterBuffer Interfac e

Interface ID: IID_IShRasterBuffer
Interface file: I3DShUti.h

The raster buffer is a 2D buffer used by renderers and post render filters to store informations for each point of the image. A particular raster buffer is the result image which is a buffer of colors.
You get access to the elements of the raster buffer in iterating chunks.

```
DECLARE_INTERFACE_(IShRasterBuffer, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShRasterBuffer methods
  STDMETHOD(Init) (THIS_ ULONG sizeH, ULONG sizeV, ULONG depth, ULONG
      planesCount, ULONG background) PURE;
  STDMETHOD(GetParameters) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG*
      depth, ULONG* planesCount, ULONG* background) PURE;
  STDMETHOD(GetChunkInfo) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG* row-
      Bytes, ULONG* depth) PURE;
  STDMETHOD_(IEnumChunk*, EnumChunks) (THIS_ RECT3D* area) PURE;
  STDMETHOD(LockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(UnlockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD_(void*, GetChunkData) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(GetChunkRect) (THIS_ BufferChunk* chunk, RECT3D* rect) PURE;
  };
```

IShRasterBuffer::Init

HRESULT IShRasterBuffer::Init (*sizeH, sizeV, depth, planesCount, background***);**
ULONG *sizeH*
ULONG *sizeV*
ULONG *depth*
ULONG *planesCount*
ULONG *background*

Call **IShRasterBuffer::Ini t** to initialize the raster buffer.

Parameters

sizeH
Number of elements horizontally.

sizeV
Number of elements vertically.

depth
Number of bits for an element, must be 1, 2, 4, 8, 16 or 32.

©1995-1997 MetaCreations. All rights reserved.

planesCount
Number of planes. This is used to store many elements at each position. You have to multiply planesCount with depth to get the size of a cell in bits.

background
Enter here the default value.

Return Value S_OK

IShRasterBuffer::GetParameters

HRESULT IShRasterBuffer::GetParameters (*sizeH, sizeV, depth, planesCount, background***);**
ULONG* *sizeH*
ULONG* *sizeV*
ULONG* *depth*
ULONG* *planesCount*
ULONG* *background*

Call **IShRasterBuffer::GetParameter s** to get back all the parameters that have been sent during the initialization.

Parameters

sizeH
Number of elements horizontally.

sizeV
Number of elements vertically.

depth
Number of bits for an element.

planesCount
Number of planes. This is used to store many elements at each position. You have to multiply planesCount with depth to get the size of a cell in bits.

background
default value.

Return Value S_OK

IShRasterBuffer::GetChunkInfo

HRESULT IShRasterBuffer::GetChunkInfo (*sizeH, sizeV, rowBytes, depth***);**
ULONG* *sizeH*
ULONG* *sizeV*
ULONG* *rowBytes*
ULONG* *depth*

©1995-1997 MetaCreations. All rights reserved.

Call **IShRasterBuffer::GetChunkInf o** to get the informations on the chunks. A chunk is a rectangular area of the raster buffer used to store partially the buffer on disk.

Parameters

sizeH
Number of elements horizontally.

sizeV
Number of elements vertically.

rowBytes
Offset to move from an element to the element bellow.

depth
Number of bits for an element, this is the same depth as returned by IShRaster-Buffer::GetParameters.

Return Value S_OK

IShRasterBuffer::EnumChunks

IEnumChunk* IShRasterBuffer::EnumChunks (*area***);**
RECT3D* *area*

Call **IShRasterBuffer::EnumChunk s** to get an enumerator of chunks. This function is used to read or write in a raster buffer.

Parameter

area
Give here the portion of the raster buffer you want to enumerate on. If this area is larger than the buffer, it will be resized to fit the buffer.

Return Value The enumerator

Comments To enumerate on all the buffer, you can put an area with the maximal values. When you enumerate on an area which don't start in 0,0, you have to take care that you won't get the beginning of your area at the beginning of the first chunk.

IShRasterBuffer::LockChunk

HRESULT IShRasterBuffer::LockChunk (*chunk***);**
BufferChunk* *chunk*

Call **IShRasterBuffer::LockChun k** to lock a chunk before you read or write in a chunk..

Parameter

chunk
The chunk to be locked.

©1995-1997 MetaCreations. All rights reserved.

Return Value S_OK

Comments It is necessary to lock the chunk because the 3D Shell might remove it to disk.

IShRasterBuffer::UnlockChunk

HRESULT IShRasterBuffer::UnlockChunk (*chunk***);**
BufferChunk* *chunk*

Call **IShRasterBuffer::UnlockChun k** to unlock a chunk after reading or writing it.

Parameter → *chunk*
the chunk to be unlocked.

Return Value S_OK

Comments It is necessary to unlock the chunk to let the 3D Shell remove it to disk if free memory is needed.

IShRasterBuffer::GetChunkData

void* IShRasterBuffer::GetChunkData (*chunk***);**
BufferChunk* *chunk*

Call **IShRasterBuffer::GetChunkDat a** to get a pointer on the data of the chunk.

Parameter → *chunk*
The chunk you want the data.

Return Value A pointer on the first element of the chunk. You get the elements from left to right, from top to bottom.

Comments The chunk size is always the same. If you reach the edge of the buffer, you may find elements on your chunk that don't bellow to the raster buffer. Just not consider them. Before you get the pointer, you need to lock your chunk.

IShRasterBuffer::GetChunkRect

HRESULT IShRasterBuffer::GetChunkRect (*chunk, rect***);**
BufferChunk* *chunk*
RECT3D* *rect*

Call **IShRasterBuffer::GetChunkRec t** to get the localization of your chunk on the raster buffer.

©1995-1997 MetaCreations. All rights reserved.

Parameters *chunk*
The chunk on the raster buffer.

rect
The rectangle in the buffer corresponding to the chunk.

Return Value S_OK

Comments The rectangle may go over the buffer if the chunk is on a frontier.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.15.IShRasterBufferGetPut Interfac e

Interface ID: IID_IShRasterBufferGetPut
Interface file: I3DShUti.h

This interface is used to get more access to the raster buffer. It provides all the methods of IShRasterBuffer with some more methods to read, write and navigate through the elements.

```
DECLARE_INTERFACE_(IShRasterBufferGetPut, IShRasterBuffer) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShRasterBuffer methods
  STDMETHOD(Init) (THIS_ ULONG sizeH, ULONG sizeV, ULONG depth, ULONG
      planesCount, ULONG background) PURE;
  STDMETHOD(GetParameters) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG*
      depth, ULONG* planesCount, ULONG* background) PURE;
  STDMETHOD(GetChunkInfo) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG* row-
      Bytes, ULONG* depth) PURE;
  STDMETHOD_(IEnumChunk*, EnumChunks) (THIS_ RECT3D* area) PURE;
  STDMETHOD(LockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(UnlockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD_(void*, GetChunkData) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(GetChunkRect) (THIS_ BufferChunk* chunk, RECT3D* rect) PURE;

  // IShRasterBufferGetPut methods
  STDMETHOD_(short, GetDepth) (THIS) PURE;
  STDMETHOD(SetBackground) (THIS_ void* buffer) PURE;
  STDMETHOD(Erase) (THIS) PURE;
  STDMETHOD(SetPosition) (THIS_ short h, short v) PURE;
  STDMETHOD(GoRight) (THIS) PURE;
  STDMETHOD(GoLeft) (THIS) PURE;
  STDMETHOD(GoDown) (THIS) PURE;
  STDMETHOD(GoUp) (THIS) PURE;
  STDMETHOD(Put) (THIS_ void* buffer) PURE;
  STDMETHOD(Get) (THIS_ void* buffer) PURE;
  STDMETHOD_(unsigned long,Get32) (THIS) PURE;
  STDMETHOD(Put32) (THIS_ long value) PURE;
  };
```

IShRasterBufferGetPut::GetDepth

short IShRasterBufferGetPut::GetDepth() ;

Call **IShRasterBufferGetPut::GetDept h** to get the number of bits for an element.

Return Value The number of bits of an element.

©1995-1997 MetaCreations. All rights reserved.

IShRasterBufferGetPut::SetBackground

HRESULT IShRasterBufferGetPut::SetBackground (*buffer***);**
void* *buffer*

Call **IShRasterBufferGetPut::SetBackgroun d** to change the background value.

Parameter → *buffer*
Give a pointer on the new value for the background.

Return Value ERROR_CALL_NOT_IMPLEMENTED

Note At this time, this method is not yet implemented.

IShRasterBufferGetPut::Erase

HRESULT IShRasterBufferGetPut::Erase() ;

Call **IShRasterBufferGetPut::Eras e** to erase the entire content of the raster buffer.

Return Value S_OK

Comments All the elements will have the background value.

IShRasterBufferGetPut::SetPosition

HRESULT IShRasterBufferGetPut::SetPosition (*h, v***);**
short *h*
short *v*

Call **IShRasterBufferGetPut::SetPositio n** to set your position on the raster buffer. From this position, you will be able to operate a Get and a Put.

Parameters → *h*
The horizontal position.

→ *v*
The vertical position.

Return Value S_OK

IShRasterBufferGetPut::GoRight

HRESULT IShRasterBufferGetPut::GoRight() ;

Call **IShRasterBufferGetPut::GoRight** to move from your original position to the next

©1995-1997 MetaCreations. All rights reserved.

element on the right.

Return Value S_OK

IShRasterBufferGetPut::GoLeft

HRESULT IShRasterBufferGetPut::GoLeft() ;

Call **IShRasterBufferGetPut::GoLef t** to move from your original position to the next element on the left.

Return Value S_OK

IShRasterBufferGetPut::GoDown

HRESULT IShRasterBufferGetPut::GoDown() ;

Call **IShRasterBufferGetPut::GoDow n** to move from your original position to the next element under.

Return Value S_OK

IShRasterBufferGetPut::GoUp

HRESULT IShRasterBufferGetPut::GoUp() ;

Call **IShRasterBufferGetPut::GoU p** to move from your original position to the next element under.

Return Value S_OK

IShRasterBufferGetPut::Put

HRESULT IShRasterBufferGetPut::Put (*buffer***);**
void* *buffer*

Call **IShRasterBufferGetPut::Pu t** to enter a value at the current location.

Parameter *buffer*
Points to the element to enter. Must be correctly sized to correspond to the depth.

Return Value S_OK

Comments If you are using a raster buffer with a depth to 32, you should rather use **IShRasterBuffe r-GetPut::Put32** which is much faster.

©1995-1997 MetaCreations. All rights reserved.

IShRasterBufferGetPut::Get

HRESULT IShRasterBufferGetPut::Get (βυφφερ**);**
void* *buffer*

Call **IShRasterBufferGetPut::Ge t** to read a value at the current location.

Parameter ⇨ *buffer*
Points to where to store the element. This buffer must be correctly sized to correspond to the depth.

Return Valu e S_OK

Comments If you are using a raster buffer with a depth to 32, you should rather use **IShRasterBuffe r-GetPut::Get32** which is much faster.

IShRasterBufferGetPut::Get32

long IShRasterBufferGetPut::Get32() ;

Call **IShRasterBufferGetPut::Get3 2** to read a value at the current location.

Return Valu e The element read.

Comments This function is faster than IShRasterBufferGetPut::Get because it does not need to check the depth. You need to use a depth of 32.

IShRasterBufferGetPut::Put32

HRESULT IShRasterBufferGetPut::Put32 (*value***);**
long *value*

Call **IShRasterBufferGetPut::Put3 2** to write a value at the current location.

Parameter ⇦ *value*
The element you want to write.

Return Valu e S_OK

Comments This function is faster than IShRasterBufferGetPut::Put because it does not need to check the depth. You need to use a depth of 32.

©1995-1997 MetaCreations. All rights reserved.

2.16.IShRasterOffscreen Interfac e

Interface ID: IID_IShRasterOffscreen
Interface file: I3DShUti.h

This interface is a raster buffer used to store images. It provides the same tools that are in the IShRasterBufferGetPut interface specialized for images. It also provides methods to load and save on disk the image.

```
DECLARE_INTERFACE_(IShRasterOffscreen, IShRasterBuffer) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShRasterBuffer methods
  STDMETHOD(Init) (THIS_ ULONG sizeH, ULONG sizeV, ULONG depth, ULONG
      planesCount, ULONG background) PURE;
  STDMETHOD(GetParameters) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG*
      depth, ULONG* planesCount, ULONG* background) PURE;
  STDMETHOD(GetChunkInfo) (THIS_ ULONG* sizeH, ULONG* sizeV, ULONG* row-
      Bytes, ULONG* depth) PURE;
  STDMETHOD_(IEnumChunk*, EnumChunks) (THIS_ RECT3D* area) PURE;
  STDMETHOD(LockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(UnlockChunk) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD_(void*, GetChunkData) (THIS_ BufferChunk* chunk) PURE;
  STDMETHOD(GetChunkRect) (THIS_ BufferChunk* chunk, RECT3D* rect) PURE;

  // IShRasterOffscreen methods
  //-- Init
  STDMETHOD(InitOffscreen) (THIS_ short sizeH, short sizeV, short depth)
      PURE;
  //-- Queries
  STDMETHOD_(short, GetDepth) (THIS) PURE;
  //-- Drawing
  STDMETHOD(BitBlits) (THIS_ SDGraphicDevice* destGD, RECT3D* source,
      RECT3D* dest) PURE;
  STDMETHOD(SetBackground) (THIS_ void* buffer) PURE;
  STDMETHOD(Erase) (THIS) PURE;
  STDMETHOD(SetPosition) (THIS_ short h, short v) PURE;
  STDMETHOD(GoRight) (THIS) PURE;
  STDMETHOD(GoLeft) (THIS) PURE;
  STDMETHOD(GoDown) (THIS) PURE;
  STDMETHOD(GoUp) (THIS) PURE;
  STDMETHOD(Put) (THIS_ void* buffer) PURE;
  STDMETHOD(Get) (THIS_ void* buffer) PURE;
  STDMETHOD(Save) (THIS_ char* fileName, long id) PURE;
  STDMETHOD(Load) (THIS_ char* fileName, Boolean storeExternally) PURE;
  STDMETHOD_(unsigned long,Get32) (THIS) PURE;
  STDMETHOD(Put32) (THIS_ long value) PURE;
  };
```

©1995-1997 MetaCreations. All rights reserved.

IShRasterOffscreen::InitOffscreen

HRESULT IShRasterOffscreen::InitOffscreen (*sizeH, sizeV, depth***);**
short *sizeH*
short *sizeV*
short *depth*

Call **IShRasterOffscreen::InitOffscree n** to initialize your offscreen.

Parameters

sizeH
Number of pixels horizontally.

sizeV
Number of pixels vertically.

depth
Number of bits to code the color, must be 1, 2, 4, 8, 16 or 32.

Return Value S_OK

Comments Using this method, you initialize your offscreen with a black background, which will be the default color returned for non allocated pixels.

IShRasterOffscreen::GetDepth

short IShRasterOffscreen::GetDepth() ;

Call **IShRasterOffscreen::GetDept h** to get the number of bits for a pixel.

Return Value The number of bits of an element.

IShRasterOffscreen::BitBlits

HRESULT IShRasterOffscreen::BitBlits (*destGD, source, dest***);**
SDGraphicDevice* *destGD*
RECT3D* *source*
RECT3D* *dest*

At this time, this method is not yet implemented.

IShRasterOffscreen::SetBackground

HRESULT IShRasterOffscreen::SetBackground (*buffer***);**
void* *buffer*

Call **IShRasterOffscreen::SetBackgroun d** to change the background color.

©1995-1997 MetaCreations. All rights reserved.

Parameter *buffer*
Points to the new background color.

Return Value S_OK

Comments The background color is used to set the color of the pixels that are not yet allocated (they haven't been yet written).

IShRasterOffscreen::Erase

HRESULT IShRasterOffscreen::Erase() ;

Call **IShRasterOffscreen::Eras e** to erase the entire offscreen.

Return Value S_OK

IShRasterOffscreen::SetPosition

HRESULT IShRasterOffscreen::SetPosition (*h, v***);**
short *h*
short *v*

Call **IShRasterOffscreen::SetPositio n** to set your position on the offscreen. From this position, you will be able to operate a Get and a Put.

Parameters *h*
The horizontal position

v
The vertical position.

Return Value S_OK

IShRasterOffscreen::GoRight

HRESULT IShRasterOffscreen::GoRight() ;

Call **IShRasterOffscreen::GoRigh t** to move from your original position to the next pixel on the right.

Return Value S_OK

IShRasterOffscreen::GoLeft

HRESULT IShRasterOffscreen::GoLeft() ;

©1995-1997 MetaCreations. All rights reserved.

Call **IShRasterOffscreen::GoLef t** to move from your original position to the next pixel on the left.

Return Valu e S_OK

Comments tocomment.

IShRasterOffscreen::GoDown

HRESULT IShRasterOffscreen::GoDown() ;

Call **IShRasterOffscreen::GoDow n** to move from your original position to the next pixel under.

Return Valu e S_OK

IShRasterOffscreen::GoUp

HRESULT IShRasterOffscreen::GoUp() ;

Call **IShRasterOffscreen::GoU p** to move from your original position to the next pixel under.

Return Valu e S_OK

IShRasterOffscreen::Put

HRESULT IShRasterOffscreen::Put (*buffer***);**
void* *buffer*

Call **IShRasterOffscreen::Pu t** to enter a color at the current location.

Parameter → *buffer*
tocomment

Return Valu e S_OK

Comments If you are using an offscreen with a depth to 32, you should rather use **IShRasterOff-screenGetPut::Put3 2** which is much faster.

IShRasterOffscreen::Get

HRESULT IShRasterOffscreen::Get (*buffer***);**
void* *buffer*

©1995-1997 MetaCreations. All rights reserved.

Call **IShRasterOffscreen::Ge t** to read a color at the current location.

Parameter *buffer*
Points to where to store the color. This buffer must be correctly sized to correspond to the depth.

Return Valu e S_OK

Comments If you are using an offscreen with a depth to 32, you should rather use **IShRasterOff-screenGetPut::Get3 2** which is much faster.

IShRasterOffscreen::Save

HRESULT IShRasterOffscreen::Save (*fileName, id***);**
char* *fileName*
long *id*

Call **IShRasterOffscreen::Sav e** to save the image on disk.

Parameters *fileName*
The name of the file.

id
The type of the image.

ID	Type
EPSF	EPS 3.0
GIFf	GIF
JPEG	Jpeg
PNTG	MacPaint
8BIM	Photoshop 2.0
8BPS	Photoshop 2.5
PICT	Pict
TIFF	Tiff
BINA	BMP
PCX	PCX
TGA	Targa

For example, set *id* to **IDTYPE('P', 'C', 'X', ' ')** to save in PCX format.

Return Valu e S_OK

IShRasterOffscreen::Load

HRESULT IShRasterOffscreen::Load (*fileName, storeExternally***);**
char* *fileName*
Boolean *storeExternally*

Call **IShRasterOffscreen::Loa d** to load an image on disk.

©1995-1997 MetaCreations. All rights reserved.

Parameters *fileName*
The name of the file.

storeExternally
Set it to TRUE if you want this image stored in the rd4 file stored as a reference and not integrally included in the file. This is used especially with maps.

Return Value S_OK

IShRasterOffscreen::Get32

unsigned long IShRasterOffscreen::Get32() ;

Call **IShRasterOffscreen::Get3 2** to read a color at the current location.

Return Value The color read

Comments This function is faster than IShRasterOffscreenGetPut::Get because it does not need to check the depth. You need to use a depth of 32.

IShRasterOffscreen::Put32

HRESULT IShRasterOffscreen::Put32 (*value***);**
long* *value*

Call **IShRasterOffscreen::Put3 2** to write a color at the current location.

Parameter *value*
The color you want to write

Return Value S_OK

Comments This function is faster than **IShRasterOffscreenGetPut::Pu t** because it does not need to check the depth. You need to use a depth of 32.

©1995-1997 MetaCreations. All rights reserved.

2.17.IShLineArt Interfac e

Interface ID: IID_IShLineArt
Interface file: I3DShUti.h

This interface is used internally, you don’t have to use it.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.18.IShUtilities Interfac e

The **IShUtilities** Interface groups a few low-level OS calls.

```
DECLARE_INTERFACE_(IShUtilities, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShUtilities methods
  STDMETHOD(YieldProcesses) (THIS_ ULONG ticks) PURE;
  STDMETHOD_(ULONG, TickCount) (THIS) PURE;
  STDMETHOD_(ULONG, Random) (THIS) PURE;
  STDMETHOD(RandomSeed) (THIS_ ULONG newSeed) PURE;
  STDMETHOD(GetSerialNumber) (THIS_ char* serialNumber) PURE;
  //-- Progression bar
  STDMETHOD(ProgressStart) (THIS_ char* prompt, ULONG maxValue) PURE;
  STDMETHOD(ProgressIncrement) (THIS_ ULONG increment) PURE;
  STDMETHOD(ProgressNewValue) (THIS_ ULONG newValue) PURE;
  STDMETHOD(ProgressDone) (THIS) PURE;
  //-- Failure Handling
  STDMETHOD(Failure) (THIS_ short error, long message) PURE;
  STDMETHOD_(void*, GetGTopHandler) (THIS) PURE;
  STDMETHOD_(void*, GetSetJmpAddress) (THIS) PURE;
  //-- Memory Management
  STDMETHOD_(void*, Malloc) (THIS_ unsigned long  size) PURE;
  STDMETHOD_(void*, Calloc) (THIS_ unsigned long nmemb, unsigned long
      size) PURE;
 STDMETHOD_(void*, Realloc) (THIS_ void* ptr, unsigned long  size) PURE;
  STDMETHOD(Free) (THIS_ void* ptr) PURE;
  //-- Instanciation
  STDMETHOD(CoCreateInstance) (THIS_ REFCLSID rclsid, LPUNKNOWN pUnk-
      Outer,
                    DWORD dwClsContext, REFIID riid, LPVOID* ppv) PURE;
  //-- Get COM helper (API level 6)
   STDMETHOD_(void*, GetCOMMap) (THIS_ void *map) PURE;
  //-- Misc. Utilities (API level 6)
  STDMETHOD(GetShellVersion) (THIS_ ULONG* major, ULONG* minor, char*
      stage, ULONG* stageCount) PURE;
  STDMETHOD_(ULONG, GetAPINumber) (THIS) PURE;
  STDMETHOD(GetIndString) (THIS_ char* str, short strID, short itemNo)
      PURE;
  STDMETHOD_(BOOLEAN, CheckRegistration) (THIS_ ULONG familyID, ULONG
      classID, char* productKey, char* productName) PURE;
  STDMETHOD(GetRegistrationName) (THIS_ char* name) PURE;
  STDMETHOD(GetRegistrationOrganization) (THIS_ char* name) PURE;
  STDMETHOD_(I3DShScene* ,GetEffectsDlogScene) (THIS) PURE; // Use only
      while the Render Effects dialog is open, and only if you know what
      you are doing.
  //--KeyFrame generation
  STDMETHOD_(IShKeyFrame *,CreateKeyFrame)(long itsType) PURE;
  STDMETHOD_(IShKeyFrame *,ReadKeyFrame)(Riostream *stream) PURE;
  STDMETHOD(WriteKeyFrame)(const IShKeyFrame *,Riostream *stream) PURE;
  //--Gestalt
```

©1995-1997 MetaCreations. All rights reserved.

```
    STDMETHOD(Gestalt)(THIS_ void *data,ULONG id) PURE;
    //--Tweener
    STDMETHOD_(IShTweener *,CreateTweener)(long itsType) PURE;
    STDMETHOD_(short, DoAlert)(short alertID, char* arg0, char* arg1,
        char* arg2) PURE;
    STDMETHOD(SetupResource) (THIS_ long* oldMap) PURE;
    STDMETHOD(RestoreResource) (THIS_ long* oldMap) PURE;
    //Import Reports
    STDMETHOD(ResetImportReport) (THIS_ ) PURE;
    STDMETHOD(AddImportProblem) (THIS_ short msgIndex, char* substitution-
        String, Boolean allowMultipleEntries) PURE;
    STDMETHOD(ShowImportReport) (THIS_ short dialogID,short strListID)
        PURE;
};
```

IShUtilities::YieldProcesses

HRESULT IShUtilities::YieldProcesses (*ticks***);**
ULONG *ticks*

Call **IShUtilities::YieldProcesse s** to allow background processes to receive processing time.

Parameter → *ticks*
The number of ticks (a tick is approximately 1/60th of a second) indicating the amount of time you want to stay in your code before allowing processing time to other processes.

Comments Time between calls to Yield are accumulated until ticks is reached. Then switching to other processes occurs at the next Yield, and the counter is reset when coming back to your code.

This call is often used by renderers to allow rendering in the background.

Examples Here are some examples:

```
Yield(0);   // You switch immediately to other processes, you lose the
      CPU.
Yield(15);  // You can call Yield() as much as you want, you are guaran-
      tied that you will keep the CPU for at least 15 ticks.
Yield(0x7FFF);  // You give processing time to other processes only every
      9 minutes. Not very nice to others...
```

Note This call can trigger a Failure if the current process is about to be killed. In that case, you should perform whatever clean-up is necessary before transmitting the Failure. See the Appendix on Toolkit Libraries for more information on the Failure Handling system.

IShUtilities::TickCount

ULONG IShUtilities::TickCount() ;

©1995-1997 MetaCreations. All rights reserved.

Call **IShUtilities::TickCount** to get an arbitrary time in 1/60 of a second (depending on the platform, it can be the number of ticks since the last boot or something else).

Return Value Number of ticks (1/60th of a second).

Comments Use this call to perform any time-based calculations.

IShUtilities::Random

ULONG IShUtilities::Random() ;

Call **IShUtilities::Random** to get a random long integer.

Return Value A random long integer.

Comments Use **IShUtilities::RandomSeed** if you need to change the seed of the random function.

IShUtilities::RandomSeed

HRESULT IShUtilities::RandomSeed (*newSeed***);**
ULONG *newSeed*

Use **IShUtilities::RandomSeed** if you need to change the seed of the random function

Parameter → *newSeed*
New seed value.

Return Value S_OK

IShUtilities::GetSerialNumber

HRESULT IShUtilities::GetSerialNumber (*serialNumber***);**
char* *serialNumber*

Use **IShUtilities::GetSerialNumber** to get the Shell Serial Number as a string.

Parameter ← *serialNumber*
The string where the serial number will be returned.

Comments Use this call if your 3D Component needs to perform any registration checking.

IShUtilities::ProgressStart

HRESULT IShUtilities::ProgressStart (*prompt, maxValue***);**
char* *prompt*

©1995-1997 MetaCreations. All rights reserved.

ULONG *maxValue*

Call **IShUtilities::ProgressStar t** to initialize a gauge at the bottom of the window.

Parameters *prompt*
Points to the string you want to display in front of the gauge.

maxValue
Enter the maximum value for the progress bar.

Return Valu e S_OK

Comments You can use **IShUtilities::ProgressIncremen t** or **IShUtilities::ProgressNewValu e** to manage the progress bar and you close it using IShUtilities::ProgressDone.

IShUtilities::ProgressIncremen t

HRESULT IShUtilities::ProgressIncrement (*increment***);**
ULONG *increment*

Call **IShUtilities::ProgressIncremen t** to change the value of the gauge.

Parameter *increment*
The value to add to the gauge.

Return Valu e S_OK

Comments You can use **IShUtilities::ProgressIncremen t** or **IShUtilities::ProgressNewValu e** to manage the progress bar.

IShUtilities::ProgressNewValu e

HRESULT IShUtilities::ProgressNewValue (*newValue***);**
ULONG *newValue*

Call **IShUtilities::ProgressIncremen t** to change the value of the gauge.

Parameter *newValue*
The new value to set to the gauge. This value must be between 0 and the maximum value you set in the initialization.

Return Valu e S_OK

Comments You can use **IShUtilities::ProgressIncremen t** or **IShUtilities::ProgressNewValu e** to manage the progress bar.

©1995-1997 MetaCreations. All rights reserved.

IShUtilities::ProgressDone

HRESULT IShUtilities::ProgressDone() ;

Call **IShUtilities::ProgressDon e** to close the gauge.

Return Valu e S_OK

IShUtilities::Failure

HRESULT IShUtilities::Failure (*error, message***);**
short *error*
long *message*

Call **IShUtilities::Failur e** to signal a failure to the OS.

Parameter

error
The error number of the failure. This is the standard error numbers list that is used. If you want a custom error number, set a number bellow -15000.

message
Must be 0.

Return Valu e S_OK

IShUtilities::GetGTopHandler

void* IShUtilities::GetGTopHandler() ;

This function is used in the macro **TRY** defined in 3DCoFail.h. You must use the **TRY** macro.

IShUtilities::GetSetJmpAddress

void* IShUtilities::GetSetJmpAddress() ;

This function is used in the macro **TRY** defined in 3DCoFail.h. You must use the **TRY** macro.

IShUtilities::Malloc

void* IShUtilities::Malloc (*size***);**
unsigned long *size*

Call **IShUtilities::Malloc** to allocate memory.

©1995-1997 MetaCreations. All rights reserved.

Parameter *size*
The size of the memory you want to allocate.

Return Value A pointer on the memory allocated.

IShUtilities::Calloc

void* IShUtilities::Calloc (*nmemb, size***);**
unsigned long *nmemb*
unsigned long *size*

Call **IShUtilities::Callo c** to allocate consecutively nmemb elements with the given size in memory.

Parameters *nmemb*
The number of elements to allocate.

size
The size of the memory you want to allocate.

Return Value A pointer on the memory allocated.

Comments **IShUtilities::Callo c** calls **IShUtilities::Mallo c** with the size nmemb*size.

IShUtilities::Realloc

void* IShUtilities::Realloc (*ptr, size***);**
void* *ptr*
unsigned long *size*

Call **IShUtilities::Reallo c** to change the size of an already allocated field.

Parameters *ptr*
The pointer of the allocated field.

size
The new size to attribute to the field.

Return Value A new pointer on the memory allocated with the new size.

Comments **IShUtilities::Reallo c** keeps the old content of the field. It keeps the entire content if the field is sized up and it keeps the beginning of the content if the field is sized down.

IShUtilities::Free

HRESULT IShUtilities::Free (*ptr***);**

©1995-1997 MetaCreations. All rights reserved.

void* *ptr*

Call **IShUtilities::Free** to release an allocated field in memory.

Parameter *ptr*
The pointer of the allocated field to be released.

Return Value S_OK

IShUtilities::CoCreateInstance

HRESULT IShUtilities::CoCreateInstance (*rclsid, pUnkOuter, dwClsContext, riid, ppv***);**
REFCLSID *rclsid*
LPUNKNOWN *pUnkOuter*
DWORD *dwClsContext*
REFIID *riid*
LPVOID* *ppv*

Call **IShUtilities::CoCreateInstance** to create an instance of a COM object.

Parameters *rclsid*
Class identifier (CLSID) of the object.

pUnkOuter
If null, the object is not an aggregate, if not, points to the aggregate object's IUnknown interface (the controlling unknown).

dwClsContext
Context for running executable code.

riid
Context for running executable code.

ppv
Points to requested interface pointer.

Return Value NOERROR if the instanciation was successful.

Comments See documentation on COM for more details.

IShUtilities::GetCOMMap

void* IShUtilities::GetCOMMap (*map***);**
void* *map*

Call **IShUtilities::GetCOMMap** to get a map interface **I3DShMap** of a given map.

©1995-1997 MetaCreations. All rights reserved.

Parameter *map*
The map you want to get the interface.

Return Value The map interface, see **I3DShMap** Interface for details.

IShUtilities::GetShellVersion

HRESULT IShUtilities::GetShellVersion (*major, minor, stage, stageCount***);**
ULONG* *major*
ULONG* *minor*
char* *stage*
ULONG* *stageCount*

Call **IShUtilities::GetShellVersio n** to get the shell version number.

Parameters *major*
The major version number, for 1.2.3, this is 1.

minor
The two minors numbers, for 1.2.3, this is 23, for 1.2, this is 20.

stage
This is a character specifying the stage. It can be the null character if none. For example, 1.2.3r45, the stage is 'r'.

stageCount
This is the stage counter, zero if there is no stage. For example, 1.2.3b45, stageCount is 45.

Return Value S_OK

IShUtilities::GetAPINumber

ULONG IShUtilities::GetAPINumber() ;

Call **IShUtilities::GetAPINumbe r** to get the current version number of the API.

Return Value The API version number.

CommentsYou can see in the header files some API version numbers for a group of methods. You can read the API version number to know if you can call those methods.

IShUtilities::GetIndString

HRESULT IShUtilities::GetIndString (*str, strID, itemNo***);**

©1995-1997 MetaCreations. All rights reserved.

char* *str*
short *strID*
short *itemNo*

Call **IShUtilities::GetIndStrin g** to get a string from the resource file.

Parameters

str
Points to where to return the string. If there is no element in the resource file corresponding, an empty string is returned.

strID
The string identifier number.

itemNo
The item number in the list, starting from 1.

Return Value S_OK

IShUtilities::CheckRegistration

BOOLEAN IShUtilities::CheckRegistration (*familyID, classID, productKey, productName***);**
ULONG *familyID*
ULONG *classID*
char* *productKey*
char* *productName*

Call **IShUtilities::CheckRegistratio n** to know if the registration is valid for the extension. This enables you to put a protection on the extension.

Parameters

familyID
Reserved for a future use.

classID
Reserved for a future use.

productKey
A two character extension code.

productName
The name of the (group of) extension(s) that will be displayed if the registration is not valid.

Return Value

Value	Meaning
TRUE	The registration number is valid for this extension.
FALSE	The registration number is not valid for this extension.

©1995-1997 MetaCreations. All rights reserved.

IShUtilities::GetRegistrationNam e

HRESULT IShUtilities::GetRegistrationName (*name***);**
char* *name*

Call **IShUtilities::GetRegistrationNam e** to get the name under what the user is registered.

Parameter ⇨ *name*
Points to where to return the name.

Return Valu e S_OK

IShUtilities::GetRegistrationOrganizatio n

HRESULT IShUtilities::GetRegistrationOrganization (*name***);**
char* *name*

Call **IShUtilities::GetRegistrationOrganizatio n** to get the organization name under what the user is registered.

Parameter ⇨ *name*
Points to where to return the organization name.

Return Valu e S_OK

IShUtilities::GetEffectsDlogScen e

I3DShScene* IShUtilities::GetEffectsDlogScene() ;

This is an internal function, you don't have to use it.

IShUtilities::CreateKeyFram e

IShKeyFrame* IShUtilities::CreateKeyFrame (*itsType***);**
long *itsType*

This is an internal function, you don't have to use it.

IShUtilities::ReadKeyFram e

IShKeyFrame* IShUtilities::ReadKeyFrame (*stream***);**
Riostream* *stream*

This is an internal function, you don't have to use it.

©1995-1997 MetaCreations. All rights reserved.

IShUtilities::WriteKeyFrame

HRESULT IShUtilities::WriteKeyFrame (*keyframe, stream***);**
IShKeyFrame* *keyframe*
Riostream* *stream*

This is an internal function, you don't have to use it.

IShUtilities::Gestalt

HRESULT IShUtilities::Gestalt (*data, id***);**
void* *data*
ULONG *id*

Call **IShUtilities::Gestal t** to get an information on the system.

Parameters

data
Points to where to store the information.

id
The ID for the kind of information.

ID	Type of data	Version	Meaning
Vers	long	1	The actual version of Gestalt
FN83	Boolean	1	True if the file name is 8 + 3 characters

For example, set *id* to **IDTYPE('F', 'N', '8', '3')** to know if the file names are 8 characters long plus 3 characters for the extension.

Return Value

Value	Meaning
S_OK	The ID is valid
S_FALSE	The ID is not valid

Comments If you want to access an information with a version number greater than 1, you can get the Gestalt version before to know if this version is recent enough to provide you information, or you can just call **IShUtilities::Gestal t** and check the return value.

IShUtilities::CreateTweener

HRESULT IShUtilities::CreateTweener (*itsType***);**
long *itsType*

This method is used internally, you don't have to call it.

©1995-1997 MetaCreations. All rights reserved.

IShUtilities::DoAlert

short IShUtilities::DoAlert (*alertID, arg0, arg1, arg2***);**
short *alertID*
char* *arg0*
char* *arg1*
char* *arg2*

Call **IShUtilities::DoAlert** to display an alert dialog box on the screen.

Parameters

alertID
Identify the string in the resource file to be displayed. This string can contain characters 0, 1 and 2 that will be replaced by the following arguments.

arg0
The string which will replace the characters 0 in the string.

arg1
The string which will replace the characters 1 in the string.

arg2
The string which will replace the characters 2 in the string.

Return Value The identifier of the pressed button.

IShUtilities::SetupResource

HRESULT IShUtilities::SetupResource (*oldMap***);**
long* *oldMap*

Call **IShUtilities::SetupResource** before reading your resources. You have to call **IShUtilities::RestoreResource** after reading your resources.

Parameter

oldMap
Points to the ID of the map that was used before. You will have to give back this ID when restoring the resource.

Return Value S_OK

IShUtilities::ResetResource

HRESULT IShUtilities::ResetResource (*oldMap***);**
long* *oldMap*

Call **IShUtilities::ResetResource** after reading your resources.

©1995-1997 MetaCreations. All rights reserved.

Parameter *oldMap*
Give back the ID of the map that was used before. See also **IShUtilities::SetupR e-source**.

Return Value S_OK

IShUtilities::ResetImportReport

HRESULT IShUtilities::ResetImportReport() ;

Call **IShUtilities::ResetImportRepor t** before doing an import/export if you plan to report any possible problems.

Return Value S_OK

IShUtilities::AddImportProble m

HRESULT IShUtilities::AddImportProblem (*msgIndex, substitutionString, allowMultipleEntries***);**
short *msgIndex*
char* *substitutionString*
BOOLEAN *allowMultipleEntries*

Call **IShUtilities::AddImportProble m** tocomment

Parameters *msgIndex*
This is an index in the array of errors strings.

substitutionString
Send here the string you want to replace the character zero by in the import/export dialog.

allowMultipleEntries
Send TRUE if you want to display many problems. Send FALSE if you want to display only the first problem.

Return Value S_OK

IShUtilities::ShowImportReport

HRESULT IShUtilities::ShowImportReport (*dialogID, strListID***);**
short *dialogID*
short *strListID*

Call **IShUtilities::ShowImportRepor t** to display the problems that occurred during the import/export, if problems occurred.

©1995-1997 MetaCreations. All rights reserved.

Parameters *dialogID*
This is the identifier of the dialog to be display in case of problems.

strListID
This is the identifier of the list of errors to be displayed in case of problems.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

2.19.I3DShEnvironment Interfac e

Interface ID: IID_I3DShEnvironment
Interface file: I3DShAb.h

The Environment Interface lets you get some information on the environment of the scene. This interface is especially used by renderers.

```
DECLARE_INTERFACE_(I3DShEnvironment, IUnknown) {
  STDMETHOD(SegmentFilter) (const VECTOR3D &beg,const VECTOR3D
      &end,COLOR3D &filterOut)=0;
  STDMETHOD(DirectionFilter) (const VECTOR3D &origin,const VECTOR3D
      &direction, COLOR3D &filterOut)=0;
  STDMETHOD(GetBackdropColor) (const VECTOR2D &screenPoint,const BOX2D
      &screenMinMax, COLOR3D &resultColor)=0;
  STDMETHOD(GetEnvironmentColor) (const VECTOR3D &direction, COLOR3D
      &resultColor, BOOLEAN &result)=0;
  };
```

I3DShEnvironment::SegmentFilte r

HRESULT I3DShEnvironment::SegmentFilter (*beg, end, filterOut***);**
VECTOR3D& *beg*
VECTOR3D& *end*
COLOR3D& *filterOut*

Call **I3DShEnvironment::SegmentFilte r** to get the color or a ray when you know the color at one point of the ray. For example, you may know the color of an object where the ray hit and you want to get the color seen because the ray may go thru a fog or a volume primitive.

Parameters

beg
The point where you want to get the color.

end
The point where you are supposed to know the color.

filterOut
Give the color of the ray at the point *end*, you will get back the color at the point *beg* of the ray.

Return Valu e S_OK.

Comments See **I3DExVolumePrimitive::GetVoolumeAttenuatio n** for a kind of implementation.

I3DShEnvironment::DirectionFilte r

HRESULT I3DShEnvironment::DirectionFilter (*origin, direction, filterOut***);**

©1995-1997 MetaCreations. All rights reserved.

VECTOR3D& *origin*
VECTOR3D& *direction*
COLOR3D& *filterOut*

Call **I3DShEnvironment::DirectionFilte r** to get the color of a ray that reach no object.

Parameters *origin*
The point where the ray starts from.

direction
The direction of the ray.

filterOut
Give the color of the background and you will get back the resulting color.

Return Valu e S_OK.

I3DShEnvironment::GetBackdropColo r

HRESULT I3DShEnvironment::GetBackdropColor (*screenPoint, screenMinMax, resultColor***);**
VECTOR2D& *screenPoint*
BOX2D& *screenMinMax*
COLOR3D& *resultColor*

Call **I3DShEnvironment::GetBackdropColo r** to get the backdrop color on a given point of the screen.

Parameters *screenPoint*
The screen point, in the Screen Coordinate System.

screenMinMax
The range of the rendered image, in the Screen Coordinate System.

resultColor
The color of the backdrop at this point.

Return Valu e S_OK.

I3DShEnvironment::GetEnvironmentColo r

HRESULT I3DShEnvironment::GetEnvironmentColor (*direction, resultColor, result***);**
VECTOR3D& *direction*
COLOR3D& *resultColor*
BOOLEAN& *result*

Call **I3DShEnvironment::GetEnvironmentColo r** to get the color of the background.

©1995-1997 MetaCreations. All rights reserved.

Parameters *direction*
The direction of the ray.

resultColor
The color of the background seen in this direction.

result
TRUE if there is a background assigned to the scene.

Return Value S_OK.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.20.I3DShExtension Interfac e

Interface ID: IID_I3DShExtension
Interface file: I3DShTre.h

This interface is used to get access to the parameters of an other extension.

```
DECLARE_INTERFACE_(I3DShExtension, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShExtension methods
  STDMETHOD_(char*, GetParamsBuffer) (THIS) PURE;
  STDMETHOD_(short, GetResID) (THIS) PURE;
  STDMETHOD_(short, GetViewID) (THIS) PURE;
  STDMETHOD_(BOOLEAN, IsEqual) (THIS_ I3DShExtension* extension) PURE;
  STDMETHOD(GetParameter) (THIS_ long param,void *) PURE;
  STDMETHOD(SetParameter) (THIS_ long param,void *) PURE;
  STDMETHOD_(long, GetClassSignature) (THIS) PURE;
  STDMETHOD(GetComponentClassName) (THIS_ char* name) PURE;
  };
```

I3DShExtension::GetParamsBuffe r

char* I3DShExtension::GetParamsBuffer() ;

Call **I3DShExtension::GetParamsBuffe r** to get a pointer on the buffer of the extension.

Return Valu e A pointer on the buffer of the extension.

I3DShExtension::GetResID

short I3DShExtension::GetResID() ;

Call **I3DShExtension::GetResI D** to get the resource ID of the extension.

Return Valu e The resource ID.

I3DShExtension::GetViewID

short I3DShExtension::GetViewID() ;

Call **I3DShExtension::GetViewI D** to get the view ID of the extension used to display the parameters.

Return Valu e The view ID.

©1995-1997 MetaCreations. All rights reserved.

I3DShExtension::IsEqual

BOOLEAN I3DShExtension::IsEqual (*extension***);**
I3DShExtension* *extension*

This function is not yet implemented, you musn't call it.

I3DShExtension::GetParameter

HRESULT I3DShExtension::GetParameter (*param, buffer***);**
long *param*
void* *buffer*

Call **I3DShExtension::GetParamete r** to get a parameter from the extension's parameters.

Parameters → *param*
The keyword of the parameter.

← *buffer*
Points to where to store the parameter.

Return Value S_OK.

I3DShExtension::SetParameter

HRESULT I3DShExtension::SetParameter (*param, buffer***);**
long *param*
void* *buffer*

Call **I3DShExtension::SetParamete r** to change the value of a parameter.

Parameters → *param*
The keyword of the parameter.

→ *buffer*
Points to the new value for the parameter.

Return Value S_OK.

I3DShExtension::GetClassSignature

long I3DShExtension::GetClassSignature() ;

Call **I3DShExtension::GetClassSignature** to get the class signature of the extension.

©1995-1997 MetaCreations. All rights reserved.

Return Value The class signature of the extension.

I3DShExtension::GetComponentClassName

HRESULT I3DShExtension::GetComponentClassName (*name***);**
char* *name*

Call **I3DShExtension::GetComponentClassName** to get the class name of the component.

Parameter ⇨ *name*
Points to where to store the class name.

Return Value S_OK.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.21.I3DShMap Interfac e

Interface ID: IID_I3DShMap
Interface file: I3DShMap.h

This interface is used to read the color on a map. This map can be a movie, so you can set the time.

```
DECLARE_INTERFACE_(I3DShMap, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DShMap methods
  STDMETHOD(GetColor) (THIS_ BOX2D *mapBox,BOX2D *uvBox,VECTOR2D *posi-
      tion, COLOR3D *result) PURE;
  STDMETHOD(Invalidate) (THIS) PURE;
  STDMETHOD_(LONG, GetTime) (THIS) PURE;
  STDMETHOD(SetTime) (THIS_ LONG time) PURE;
  };
```

I3DShMap::GetColor

HRESULT I3DShMap::GetColor (*mapBox, uvBox, position, result***);**
BOX2D* *mapBox*
BOX2D* *uvBox*
VECTOR2D* *position*
COLOR3D* *result*

Call **I3DShMap::GetColo r** to get a color on a position of the map.

Parameters

mapBox
The position of the map, in uvSpace coordinates.

uvBox
The range of the uvSpace.

position
The position on the uvSpace where you want to get the color.

result
The color at this point.

Return Valu e S_OK

I3DShMap::Invalidate

HRESULT I3DShMap::Invalidate() ;

©1995-1997 MetaCreations. All rights reserved.

This function is used internally, you musn’t call it.

I3DShMap::GetTime

LONG I3DShMap::GetTime() ;

Call **I3DShMap::GetTim e** to get the time setted to the map.

Return Valu e The time, in ticks.

I3DShMap::SetTime

HRESULT I3DShMap::SetTime (*time***);**
LONG *time*

Call **I3DShMap::SetTim e** to set a new time to the map. This is used to get a color on a movie for example.

Parameter → *time*
The new time, in ticks.

Return Valu e S_OK

©1995-1997 MetaCreations. All rights reserved.

2.22.IEnumChunk Interfac e

Interface ID: IID_IIEnumChunk
Interface file: I3DShUti.h

tocomment.

```
DECLARE_INTERFACE_(IEnumChunk, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // I3DEnumPatch methods
  STDMETHOD(Next) (THIS_ ULONG celt, BufferChunk** rgelt, ULONG* pceltFetched) PURE;
  STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
  STDMETHOD(Reset) (THIS) PURE;
  STDMETHOD(Clone) (THIS_ IEnumChunk** ppenm) PURE;
  };
```

IEnumChunk::Next

HRESULT IEnumChunk::Next (*celt, rgelt, pceltFetched***);**
ULONG *celt*
BufferChunk** *rgelt*
ULONG* *pceltFetched*

Call **IEnumChunk::Nex t** to get the next chunk.

Parameters

celt
Number of chunks to be read, must be 1.

rgelt
Points to where to return a pointer on the next chunk, send a NULL pointer if you don't want to get it back.

pceltFetched
Points to where to return the number of chunks read, it's always 1. Send a NULL pointer if you don't want to get it back.

Return Valu e

Value	Meaning
S_OK	The next chunk has been correctly returned.
S_FALSE	There is no more chunk.

Comments This is a COM implementation, but this function don't allow to read many chunks at a time.

©1995-1997 MetaCreations. All rights reserved.

The first call of **IEnumChunk::Next** return the first chunk. See the COM documentation for more details.

IEnumChunk::Skip

HRESULT IEnumChunk::Skip (*celt***);**
ULONG *celt*

Call **IEnumChunk::Skip** to skip many chunks without reading them.

Parameter *celt*
Number of chunks to be skipped.

Return Value

Value	Meaning
S_OK	The celt chunks have been correctly skipped.
S_FALSE	There is not enough chunks left to skip.

Comments See the COM documentation for more details.

IEnumChunk::Reset

HRESULT IEnumChunk::Reset() ;

Call **IEnumChunk::Reset** to restore the enumerator on the first chunk.

Return Value S_OK

Comments See the COM documentation for more details.

IEnumChunk::Clone

HRESULT IEnumChunk::Clone (*ppenm***);**
IEnumChunk** *ppenm*

This call is not yet implemented, do not call it.

©1995-1997 MetaCreations. All rights reserved.

2.23.IShFileStream Interfac e

Interface ID: IID_IShFileStream
Interface file: I3DShUti.h

This interface is used especially for import export. It provides some methods to handle text files.

```
DECLARE_INTERFACE_(IShFileStream, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShFileStream methods
  STDMETHOD(InitFileStream) (THIS_ char* fileName, ULONG mode) PURE;
  //-- Seek
  STDMETHOD_(ULONG, Tell) (THIS) PURE;
  STDMETHOD(Seek) (THIS_ ULONG offset, short direction) PURE;
  //-- Binary io
  STDMETHOD(Read) (THIS_ char* s, ULONG n) PURE;
  STDMETHOD(Write) (THIS_ char* s, ULONG n) PURE;
  STDMETHOD(Append) (THIS_ ULONG n) PURE;
  //-- Text io
  STDMETHOD(Get) (THIS_ char* c) PURE;
  STDMETHOD(GetString) (THIS_ char* str, ULONG count, char delim) PURE;
  STDMETHOD(GetLine) (THIS_ char* str, ULONG count) PURE;
  STDMETHOD(GetLong) (THIS_ long* num) PURE;
  STDMETHOD(GetDouble) (THIS_ Double* num) PURE;
  STDMETHOD(Put) (THIS_ char c) PURE;
  STDMETHOD(PutString) (THIS_ char* str) PURE;
  STDMETHOD(PutLine) (THIS_ char* str) PURE;
  STDMETHOD(PutEOL) (THIS) PURE;
  STDMETHOD(PutLong) (THIS_ long num) PURE;
  STDMETHOD(PutDouble) (THIS_ Double num) PURE;
  STDMETHOD(PutBack) (THIS_ char c) PURE;
  //-- Error
  STDMETHOD_(BOOLEAN, Good) (THIS) PURE;
  STDMETHOD_(BOOLEAN, Fail) (THIS) PURE;
  STDMETHOD_(BOOLEAN, Eof) (THIS) PURE;
  //-- Misc.
  STDMETHOD(SetMacOSInfo) (THIS_ ULONG creator, ULONG type) PURE;
  STDMETHOD_(ULONG, GetFileSize) (THIS) PURE;
  };
```

IShFileStream::InitFileStream

HRESULT IShFileStream::InitFileStream (*fileName, mode***);**
char* *fileName*
ULONG *mode*

Call **IShFileStream::InitFileStrea m** to initialize a file stream for reading or writing.

©1995-1997 MetaCreations. All rights reserved.

Parameters *fileName*
The name of the file.

mode
The mode for the stream.

Constant	Value	Meaning
in	0x0001	input.
out	0x0002	output.
nocreate	0x0040	fail if already exist.
resFork	0x0008	open the resource fork instead of the data fork (mac only).

Return Value S_OK

IShFileStream::Tell

ULONG IShFileStream::Tell() ;

Call **IShFileStream::Tell** to get the current position in the file stream.

Return Value The position in the file stream, starting from zero.

IShFileStream::Seek

HRESULT IShFileStream::Seek (*offset, direction***);**
ULONG *offset*
short *direction*

Call **IShFileStream::Seek** to position the stream at the given offset.

Parameters *offset*
Enter the offset to process. offset starts from 0.

direction
Select the kind of offset.

Constant	Value	Meaning
beg	0	offset from the beginning of the file.
cur	1	offset from the current location.
end	2	offset from the end of the file.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

©1995-1997 MetaCreations. All rights reserved.

IShFileStream::Read

HRESULT IShFileStream::Read (*s, n***);**
char* *s*
ULONG *n*

Call **IShFileStream::Rea d** to read a number of characters.

Parameters

s
Points to where to store the characters.

n
Number of character to read.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::Write

HRESULT IShFileStream::Write (*s, n***);**
char* *s*
ULONG *n*

Call **IShFileStream::Writ e** to write a number of characters.

Parameters

s
The string to be written.

n
The number of characters to write..

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::Append

HRESULT IShFileStream::Append (*n***);**
ULONG *n*

Call **IShFileStream::Appen d** to put additional characters at the end of the file.

©1995-1997 MetaCreations. All rights reserved.

Parameter *n*
Number of characters to add.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::Get

HRESULT IShFileStream::Get (*c***);**
char* *c*

Call **IShFileStream::Ge t** to read a character.

Parameter *c*
Points to where to return the character.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::GetString

HRESULT IShFileStream::GetString (*str, count, delim***);**
char* *str*
ULONG *count*
char *delim*

Call **IShFileStream::GetStrin g** to read a string.

Parameters *str*
Points to where to store the string.

count
Maximum number of characters to read..

delim
Character of end of string. This character is not read.

©1995-1997 MetaCreations. All rights reserved.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::GetLine

HRESULT IShFileStream::GetLine (*str, count***);**
char* *str*
ULONG *count*

Call **IShFileStream::GetLin e** to read a line.

Parameters

str
Points to where to store the string.

count
Maximum number of characters to read.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::GetLong

HRESULT IShFileStream::GetLong (*num***);**
long* *num*

Call **IShFileStream::GetLon g** to read a long.

Parameter

num
Points to where to store the long.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::GetDouble

HRESULT IShFileStream::GetDouble (*num***);**
Double* *num*

©1995-1997 MetaCreations. All rights reserved.

Call **IShFileStream::GetDoubl e** to read a double..

Parameter *num*
Points to where to store the double.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::Put

HRESULT IShFileStream::Put (*c***);**
char *c*

Call **IShFileStream::Pu t** to write a character..

Parameter *c*
The character to write.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutString

HRESULT IShFileStream::PutString (*str***);**
char* *str*

Call **IShFileStream::PutStrin g** to write a string.

Parameter *str*
The null-terminated string to write.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutLine

HRESULT IShFileStream::PutLine (*str***);**
char* *str*

©1995-1997 MetaCreations. All rights reserved.

Call **IShFileStream::PutLin e** to write a string and a line feed..

Parameter *str*
The null-terminated string to write.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutEOL

HRESULT IShFileStream::PutEOL() ;

Call **IShFileStream::PutEO L** to write a line feed.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutLong

HRESULT IShFileStream::PutLong (*num***);**
long *num*

Call **IShFileStream::PutLon g** to write a long.

Parameter *num*
The long to write.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutDouble

HRESULT IShFileStream::PutDouble (*num***);**
Double *num*

Call **IShFileStream::PutDoubl e** to write a double.

©1995-1997 MetaCreations. All rights reserved.

Parameter *num*
The double to write.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::PutBack

HRESULT IShFileStream::PutBack (*c***);**
char *c*

Call **IShFileStream::PutBac k** to cancel a character reading. The next get character will return this character. This method don't write into the file.

Parameter *c*
The character to write back.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::Good

BOOLEAN IShFileStream::Good() ;

Call **IShFileStream::Goo d** to know if the reading can continue, if there is no file failure and if the end of the file isn't reached yet.

Return Value TRUE if the file stream is good.

IShFileStream::Fail

BOOLEAN IShFileStream::Fail() ;

Call **IShFileStream::Fai l** to know if a failure occured while reading/writing in the file.

Return Value TRUE if the last file operation failed.

IShFileStream::Eof

BOOLEAN IShFileStream::Eof() ;

©1995-1997 MetaCreations. All rights reserved.

Call **IShFileStream::Eo f** to know if you reached the end of the file.

Return Value

Value	Meaning
TRUE	end of file reached or error.
FALSE	

IShFileStream::SetMacOSInfo

HRESULT IShFileStream::SetMacOSInfo (*creator, type***);**
ULONG *creator*
ULONG *type*

Call **IShFileStream::SetMacOSInf o** to set the MacOS informations.

Parameters

creator
The creator of the file.

type
The type of the file.

Return Value

Value	Meaning
S_OK	
ERROR_FILE_NOT_FOUND	File not initialized.

IShFileStream::GetFileSize

ULONG IShFileStream::GetFileSize() ;

Call **IShFileStream::GetFileSiz e** to get the size of the file.

Return Value the size of the file, 0 if error.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

2.24.IShGraphicDevice Interfac e

Interface ID: IID_IShGraphicDevice
Interface file: I3DShUti.h

This interface provides some methods to draw on the windows.

```
DECLARE_INTERFACE_(IShGraphicDevice, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID* ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShGraphicDevice methods
  STDMETHOD(MoveTo)(short hh, short vv) PURE;
  STDMETHOD(LineTo)(short hh, short vv) PURE;
  STDMETHOD(SetPen)(short aPenMode, short aPenType, short size, unsigned
      short rr, unsigned short gg, unsigned short bb) PURE;
  STDMETHOD(ResetPen)() PURE;
  STDMETHOD(DrawLine)(short h1, short v1, short h2, short v2) PURE;
  STDMETHOD(EraseRect)(const RECT3D& theRect) PURE;
  STDMETHOD(InvertRect)(const RECT3D& theRect) PURE;
  STDMETHOD(FrameRect)(const RECT3D& theRect) PURE;
  STDMETHOD(FillRect)(const RECT3D& theRect, unsigned short rr, unsigned
      short gg, unsigned short bb) PURE;
  STDMETHOD(DrawString)(const RECT3D& theRect, const RDString &astring,
      short just, short mode) PURE;
  STDMETHOD(DrawOffscreen)(const IShRasterOffscreen *aOffscrn, const
      RECT3D &srcRect, const RECT3D &dstRect) PURE;
  };
```

IShGraphicDevice::MoveTo

HRESULT IShGraphicDevice::MoveTo (*hh, vv***);**
short *hh*
short *vv*

Call **IShGraphicDevice::MoveTo** to set the current position before the **IShGraphicDevice::LineTo**.

Parameters

→ *hh*
Horizontal position.

→ *vv*
Vertical position.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

IShGraphicDevice::LineTo

HRESULT IShGraphicDevice::LineTo (*hh, vv***);**
short *hh*
short *vv*

Call **IShGraphicDevice::LineTo** to draw a line from the current position. The specified point become the current point.

Parameters

hh
Horizontal position.

vv
Vertical position.

Return Value S_OK

IShGraphicDevice::SetPen

HRESULT IShGraphicDevice::SetPen (*aPenMode, aPenType, size, rr, gg, bb***);**
short *aPenMode*
short *aPenType*
short *size*
unsigned short *rr*
unsigned short *gg*
unsigned short *bb*

Call **IShGraphicDevice::SetPe n** to set a new drawing pen with a new color.

Parameters

aPenMode
Mode for the pen. They are standard Windows pen modes.

Constant	Value
kNotXor	3
kCopyPen	4
kBicPen	5

aPenType
Type of the pen. They are standard Windows pen types.

Constant	Value
kSolidPen	0
kDashPen	1
kDotPen	2

©1995-1997 MetaCreations. All rights reserved.

size
Size of the pen.

rr
Red value for the color.

gg
Green value for the color.

bb
Blue value for the color.

Return Value S_OK

IShGraphicDevice::ResetPen

HRESULT IShGraphicDevice::ResetPen() ;

Call **IShGraphicDevice::ResetPe n** to reset the default pen which is black.

Return Value S_OK

IShGraphicDevice::DrawLine

HRESULT IShGraphicDevice::DrawLine (*h1, v1, h2, v2***);**
short *h1*
short *v1*
short *h2*
short *v2*

Call **IShGraphicDevice::DrawLin e** To draw a line.

Parameters

h1
Horizontal position of the first point.

v1
Vertical position of the first point.

h2
Horizontal position of the second point.

v2
Vertical position of the second point.

Return Value S_OK

©1995-1997 MetaCreations. All rights reserved.

Comments The second point become the current point.

IShGraphicDevice::EraseRect

HRESULT IShGraphicDevice::EraseRect(theRect) ;
Rect3D& theRec t

Call **IShGraphicDevice::EraseRec t** to erase a rectangle.

Parameter → *theRect*
The rectangle to be erased.

Return Valu e S_OK

IShGraphicDevice::InvertRect

HRESULT IShGraphicDevice::InvertRect (*theRect***);**
Rect3D& *theRect*

Call **IShGraphicDevice::InvertRec t** to make an inverse video on a portion of the screen.

Parameter → *theRect*
The rectangle to be inverted.

Return Valu e S_OK

IShGraphicDevice::FrameRect

HRESULT IShGraphicDevice::FrameRect (*theRect***);**
Rect3D& *theRect*

Call **IShGraphicDevice::FrameRec t** to draw an empty rectangle on the screen.

Parameter → *theRect*
The rectangle to draw.

Return Valu e S_OK

IShGraphicDevice::FillRect

HRESULT IShGraphicDevice::FillRect (*theRect, rr, gg, bb***);**
RECT3D& *theRect*
unsigned short *rr*
unsigned short *gg*
unsigned short *bb*

©1995-1997 MetaCreations. All rights reserved.

Call **IShGraphicDevice::FillRec t** to fill a rectangle on screen with a given color.

Parameters

theRect
The rectangle to be filled.

rr
Red value for the color.

gg
Green value for the color.

bb
Blue value for the color.

Return Value S_OK

IShGraphicDevice::DrawString

HRESULT IShGraphicDevice::DrawString (*theRect, astring, just, mode***);**
RECT3D& *theRect*
const RDString& *astring*
short *just*
short *mode*

Call **IShGraphicDevice::DrawStrin g** to draw text on screen.

Parameters

theRect
The rectangle where to display the text.

astring
The string to be displayed on screen.

just
The justification of the text.

Constant	Value	Meaning
kJustLeft	0	justified on left.
kJustCenter	1	centered.
kJustRight	2	justified on right.
kMiddleCut	3	suspension points in the middle of the string.
kEndCut	4	suspension points at the end of the string.

you can use **kMiddleCut** or **kEndCut** to display a string larger than the rectangle. A part of the string will be replaced by suspension points.

©1995-1997 MetaCreations. All rights reserved.

mode
The mode for the text. This is a set of flags.

Constant	Value	Meaning
kTMNone	0	nothing.
kTMEraseFirst	1	the field is cleaned before.
kTMDoNotWrap	2	no carriage return.
kTMCenterVert	4	the text is vertically centered.
kTMTranslateMetaChar	8	style conversion.

Return Value S_OK

Comments At this time, this method is not yet implemented.

IShGraphicDevice::DrawOffscreen

HRESULT IShGraphicDevice::DrawOffscreen (*aOffscrn, srcRect, dstRect***);**
IShRasterOffscreen* *aOffscrn*
const RECT3D& *srcRect*
const RECT3D& *dstRect*

Call **IShGraphicDevice::DrawOffscree n** to draw an offscreen on the screen.

Parameters

aOffscrn
The offscreen where to get the bitmap.

srcRect
The rectangle on the offscreen where to get the bitmap.

dstRect
The rectangle on the graphic device where to draw the bitmap.

Return Value S_OK

Comments If the srcRect and the dstRect haven't got the same dimensions, the bitmap will be zoomed in or out to fit the dstRect.

©1995-1997 MetaCreations. All rights reserved.

2.25.IShIterator Interfac e

Interface ID: IID_IShIterator
Interface file: I3DShIte.h

This interface is a COM iterator. You may find more informations about it on a documentation of COM.

```
DECLARE_INTERFACE_(IShIterator, IUnknown) {
  // IUnknown methods
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
      PURE;
  STDMETHOD_(ULONG, AddRef) (THIS) PURE;
  STDMETHOD_(ULONG, Release) (THIS) PURE;

  // IShIterator methods:
  STDMETHOD(Next) (THIS_ ULONG celt, void** rgelt, ULONG* pceltFetched)
      PURE;
  STDMETHOD(Reset) (THIS) PURE;
  STDMETHOD_(void*, GetInfo) (THIS_ short index) PURE;
  };
```

IShIterator::Next

HRESULT IShIterator::Next (*celt, rgelt, pceltFetched***);**
ULONG *celt*
void** *rgelt*
ULONG* *pceltFetched*

Call **IShIterator::Nex t** to get the next element.

Parameters

celt
Number of elements to be read, must be 1.

rgelt
Points to where to return a pointer on the next element.

pceltFetched
Points to where to return the number of elements read. In the application, it is not handled.

Return Valu e

Value	Meaning
S_OK	The next element has been correctly returned.
E_NOTIMPL	There is no more element, or there is no defined iterator.

Comments See the COM documentation for more details.

©1995-1997 MetaCreations. All rights reserved.

IShIterator::Reset

HRESULT IShIterator::Reset() ;

Call **IShIterator::Reset** to restore the enumerator on the first element.

Return Value

Value	Meaning
S_OK	The iterator has been correctly reseted.
E_NOTIMPL	There is no defined iterator.

Comments See the COM documentation for more details.

IShIterator::GetInfo

void* IShIterator::GetInfo (*index***);**
short *index*

This function is not yet documented.

©1995-1997 MetaCreations. All rights reserved.

Chapter 3 - Data Structures Reference

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.1.NUM3D

All geometric data is in 32-bit fixed point format. This is similar to the FIXED data type in Windows, and identical to the Fixed data type on the Macintosh. As far as the C compiler is concerned, it is a `long`.

```
3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
^                               ^
Here is the sign           Here is the decimal point: between bit 15 & 16
```

Examples of NUM3D values:

```
   1.0      0x00010000
   0.5      0x00008000
   0.0      0x00000000
   -1.0     0xFFFF0000
   -1.5     0xFFFE8000
```

Range it limited to ± 32767.99998, accuracy is limited to 1.5259E-5 (=1/65536).

Additions, subtractions, multiplications by 2 and divisions by 2 are very efficient: the regular `long` operators do the job.

Examples To convert from a NUM3D to a double:

```
double Num3DToDouble(NUM3D f) {
      return (f / 65536.0);
}
```

To convert from a double to a NUM3D:

```
NUM3D DoubleToNUM3D(double d) {
      return (long) (d * 65536.0);
}
```

Note If you use C++, it is highly recommended to use the **QuickMath** library provided in the development toolkit. It defines a C++ class called **QuickFix** that matches the NUM3D type, and that contains all kind of optimized operators and functions. You will save a lot of time and efforts by using this library, and your code will be easier to read. See the development toolkit notes for details on how to use this library on your specific platform and compiler.

For C fanatics, there are low-level calls that can be used to perform the basic operations efficiently. See the QLib.h file.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.2.VECTOR2D

VECTOR2D is used to manipulate two-dimensional data: 2D points or 2D vectors. When using QuickMath, it is mapped to the **QuickFixVector2** class, which provides all kinds of useful operators.

VECTOR2D data is organized as an array of two **NUM3D** data.

```
typedef NUM3D VECTOR2D[2];
```

Let *point* be a variable of type VECTOR2D, point[0] is the X value and point[1] is the Y value.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.3.VECTOR3D

VECTOR3D is similar to **VECTOR2D**, but for three-dimensional data: 3D points or 3D vectors. When using QuickMath, it is mapped to the **QuickFixVector3** class, which provides all kinds of useful operators.

VECTOR3D data is organized as an array of three **NUM3D** data.

```
typedef NUM3D VECTOR3D[3];
```

Let *point* be a variable of type VECTOR2D, point[0] is the X value, point[1] is the Y value and point[2] the Z value.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.4.BOX2D

BOX2D is used for 2D Bounding Boxes. A 2D Bounding Box is a rectangular area that encloses a set of 2D points.

```
typedef struct BOX2D
{
  VECTOR2DfMin;
  VECTOR2DfMax;
} BOX2D;
```

Fields

fMin
The minimum X and Y values of the Bounding Box.

fMax
The maximum X and Y values of the Bounding Box.

Examples Here is how to calculate the center of a 2D Bounding Box:

```
void CalcBox2DCenter(BOX2D* box, VECTOR2D* center)
{
  *center = (box->fMin + box->fMax)>>1;
}
```

Here is how to test if a point is inside a 2D Bounding Box:

```
BOOLEAN IsInsideBox2D(BOX2D* box, VECTOR2D* point)
{
  return (box->fMin[0] <= (*point)[0]) && (box->fMin[1] <= (*point)[1])
      && ((*point)[0] <= box->fMax[0]) && ((*point)[1] <= box->fMax[1]);
}
```

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.5.BOX3D

BOX3D is used for 3D Bounding Boxes. A 3D Bounding Box is a cube volume that encloses a set of 3D points.

```
typedef struct BOX3D
{
  VECTOR3DfMin;
  VECTOR3DfMax;
} BOX3D;
```

Fields

fMin
The minimum X, Y and Z values of the Bounding Box.

fMax
The maximum X, Y and Z values of the Bounding Box.

Examples Here is how to calculate the center of a 3D Bounding Box:

```
void CalcBox3DCenter(BOX3D* box, VECTOR3D* center)
{
  *center = (box->fMin + box->fMax)>>1;
}
```

Here is how to test if a point is inside a 3D Bounding Box:

```
BOOLEAN IsInsideBox3D(BOX3D* box, VECTOR3D* point)
{
  return   (box->fMin[0] <= (*point)[0]) && (box->fMin[1] <= (*point)[1])
       &&
       (box->fMin[2] <= (*point)[2]) &&
       ((*point)[0] <= box->fMax[0]) && ((*point)[1] <= box->fMax[1]) &&
       ((*point)[2] <= box->fMax[2]);
}
```

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.6.MATRIX3D

MATRIX3D is used for 3D rotation.

```
typedef struct MATRIX3D {
  NUM3D fix, fjx, fkx,
        fiy, fjy, fky,
        fiz, fjz, fkz;
  } MATRIX3D;
```

A convenient way to look at such a 3 x 3 Matrix is to consider it as three column vectors I, J and K. In this perspective, here is how one could write the transformation of an (x, y, z) point in the mathematical form:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} ix & jx & kx \\ iy & jy & ky \\ iz & jz & kz \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

The reason of its convenience is that these vectors represent the unit vectors of the Object Local System defined by this matrix, and that the Ix, Iy, etc. values are the coordinates of these vectors in the Global Coordinates System.

Example Here is how a point transformation is written in C++:

```
void TransformPoint(MATRIX3D* mat, VECTOR3D* point, VECTOR3D* result)
{
  (*result)[0] = mat->fix * (*point)[0] +
                 mat->fjx * (*point)[1] +
                 mat->fkx * (*point)[2];

  (*result)[1] = mat->fiy * (*point)[0] +
                 mat->fjy * (*point)[1] +
                 mat->fky * (*point)[2];

  (*result)[2] = mat->fiz * (*point)[0] +
                 mat->fjz * (*point)[1] +
                 mat->fkz * (*point)[2];
}
```

Comments To define a 3D rotation, such a matrix must be normalized.
All row and column vectors of the matrix must be unit vectors:

$ix^2 + iy^2 + iz^2 = 1.0$
$jx^2 + jy^2 + jz^2 = 1.0$
$kx^2 + ky^2 + kz^2 = 1.0$

$ix^2 + jx^2 + kx^2 = 1.0$
$iy^2 + jy^2 + ky^2 = 1.0$
$iz^2 + jz^2 + kz^2 = 1.0$

©1995-1997 MetaCreations. All rights reserved.

All row and column vectors of the matrix must be orthogonal to each other:

ix*jx + iy*jy + iz*jz = 0.0
ix*kx + iy*ky + iz*kz = 0.0
jx*kx + jy*ky + jz*kz = 0.0

ix*iy + jx*jy + kx*ky = 0.0
ix*iz + jx*jz + kx*kz = 0.0
iy*iz + jy*jz + ky*kz = 0.0

If the matrix is normalized, then inverting it to get the inverse transformation is easy: the inverse matrix is obtained by simple transposition:

```
                  ix    iy    iz
Inverse Matrix = jx    jy    jz
                  kx    ky    kz
```

Example Here is a rotation around the Z axis of an alpha angle:

```
cos(alpha)    -sin(alpha)    0
sin(alpha)     cos(alpha)    0
0              0             1
```

And the inverse rotation is written like this after transposition:

```
cos(alpha)     sin(alpha)    0
-sin(alpha)    cos(alpha)    0
0              0             1
```

Which does correspond to a -alpha rotation because cos(alpha) = cos(-alpha) and -sin(alpha) = sin(-alpha):

```
cos(-alpha)    sin(-alpha)    0
sin(-alpha)    cos(-alpha)    0
0              0              1
```

©1995-1997 MetaCreations. All rights reserved.

3.7. TRANSFORM3D

TRANSFORM3D is used to define a combination of a Rotation and a Translation.

```
typedef struct TRANSFORM3D
{
  MATRIX3D  fR;   // Rotation
  VECTOR3D  fT;   // Translation
} TRANSFORM3D;
```

Fields

fR
The 3D rotation matrix.

fT
The 3D Translation.

Comments This kind of transformation can be written as this:

$$P' = [R] P + T$$

when P is the original 3D point and P' the same point after transformation.

Example Here is how this is written in C:

```
void TransformPoint(TRANSFORM3D* trsf, VECTOR3D* point, VECTOR3D*
      result)
{
  (*result)[0] = trsf->fR.fix * (*point)[0] +
      trsf->fR.fjx * (*point)[1] +
      trsf->fR.fkx * (*point)[2] +
      trsf->fT[0];

  (*result)[1] = trsf->fR.fiy * (*point)[0] +
      trsf->fR.fjy * (*point)[1] +
      trsf->fR.fky * (*point)[2] +
      trsf->fT[1];

  (*result)[2] = trsf->fR.fiz * (*point)[0] +
      trsf->fR.fjz * (*point)[1] +
      trsf->fR.fkz * (*point)[2] +
      trsf->fT[2];
}
```

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.8. TREETRANSFORM3 D

TREETRANSFORM3 D is used to define a Tree Element Transformation.

```
typedef struct TREETRANSFORM3D {
  MATRIX3D    fR;    // Rotation 3 x 3 matrix
  VECTOR3D    fT;    // Translation
  NUM3D       fS;    // Uniform scaling
} TREETRANSFORM3D;
```

Fields

fR
The 3D rotation matrix.

fT
The 3D Translation.

fS
The Uniform scaling. This scaling is applied in all directions.

Comments This kind of transformation can be written as this:

P’ = S * ([R] P) + T

when P is the original 3D point and P’ the same point after transformation.

Example Here is how this is written in C:

```
void TransformPoint(TREETRANSFORM3D* trsf, VECTOR3D* point, VECTOR3D*
      result)
{
  (*result)[0] = trsf->fR.fix * (*point)[0] +
      trsf->fR.fjx * (*point)[1] +
      trsf->fR.fkx * (*point)[2] +
      trsf->fT[0];

  (*result)[1] = trsf->fR.fiy * (*point)[0] +
      trsf->fR.fjy * (*point)[1] +
      trsf->fR.fky * (*point)[2] +
      trsf->fT[1];

  (*result)[2] = trsf->fR.fiz * (*point)[0] +
      trsf->fR.fjz * (*point)[1] +
      trsf->fR.fkz * (*point)[2] +
      trsf->fT[2];

  *result = (*result) * trsf->fS + trsf->fT;
}
```

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.9.COLOR3D

COLOR3D is used to define RGB and CMYK colors.

```
typedef struct COLOR3D {
  long  Mode;    // Color model. 0=RGB, 1=CMYK
  NUM3D  R;      // Red or Cyan
  NUM3D  G;      // Green or Magenta
  NUM3D  B;      // Blue or Yellow
  NUM3D  A;      // N/A or Black
} COLOR3D;
```

Fields

Mode
Color Model. 0 for RGB, 1 for CMYK

R
The Red value in RGB mode, or the Cyan value in CMYK. From 0.0 to 1.0.

G
The Green value in RGB mode, or the Magenta value in CMYK. From 0.0 to 1.0.

B
The Blue value in RGB mode, or the Yellow value in CMYK. From 0.0 to 1.0.

A
Not used in RGB mode, or the Black value in CMYK. From 0.0 to 1.0.

Comments All rendering and shading calculations are done in RGB mode.

Example White = (1.0, 1.0, 1.0)
Black = (0.0, 0.0, 0.0)

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

3.10.Basic Geometry Data Structure s

VERTEX3D

VERTEX3D defines the facets vertices (see **FACET3D**).

```
typedef struct VERTEX3D {
  VECTOR3D    fVertex;    // x, y, z vertex coordinates
  VECTOR3D    fNormal;    // Nx, Ny, Nz normal values at that vertex
  NUM3D       fu,fv;      // Texture u,v values at that vertex
} VERTEX3D;
```

Fields

- *fVertex*
  The coordinates of the vertex, in the Object Local Coordinate System.

- *fNormal*
  The Surface Normal at this vertex, in the Object Local Coordinate System.

- *fu*
- *fv*
  The u and v texture coordinates at this vertex.

Comments See the “Database Overview” chapter for detailed explanations about Surface Normals and UV Spaces.

FACET3D

FACET3D defines the triangular facets used to describe the surface of an object.

```
typedef struct FACET3D{
  VERTEX3D  fVertices[3];   // The facet three vertices
  short     fUVSpace;       // UV Space ID this facet belongs to
  short     fReserved;      // Reserved
  } FACET3D;
```

Fields

- *fVertices*
  Array of the 3 vertices of the facet.

- *fUVSpace*
  The UV Space ID this facet belongs to.

- *fReserved*
  This field can be used by renderers to store any private data. It is not used otherwise.

Comments See the “Database Overview” chapter for detailed explanations about Surface Normals and UV Spaces.

©1995-1997 MetaCreations. All rights reserved.

PATCH3D

PATCH3D defines the bicubic patches used to describe the surface of an object with a *much better* quality than with facets.

```
typedef struct PATCH3D{
  VECTOR3D   fVertices[4][4];    // The patch 16 vertices
  NUM3D      fu[2];              // u values at the patch boundaries
  NUM3D      fv[2];              // v values at the patch boundaries
  short      fUVSpace;           // UV Space ID this patch belongs to
  short      fReserved;          // Reserved - 0
  } PATCH3D;
```

Fields

fVertices
Array of the 16 vertices of the Patch, in the Object Local Coordinate System.

fu
Array of the minimum and maximum U values of this patch. fu[0] if the U value of all fVertices[0][x] vertices, and fu[1] is the U value of all fVertices[3][x] vertices.

fv
Array of the minimum and maximum V values of this patch. fv[0] if the V value of all fVertices[x][0] vertices, and fv[1] is the V value of all fVertices[x][3] vertices.

fUVSpace
The UV Space ID this facet belongs to.

fReserved
This field can be used by renderers to store any private data. It is not used otherwise.

Comments

See the “Database Overview” chapter for detailed explanations about Bicubic Patches and UV Spaces.

Normals do not need to be stored: they are calculated directly from the Patch vertices.

©1995-1997 MetaCreations. All rights reserved.

3.11.Shading Data Structure s

ShadingIn

The ShadingIn data structure is used in conjunction with the other Shading data structures to perform shading. It contains all the input parameters provided to the Shader according to how the Shading Flags were set earlier by the Shader.

```
typedef struct ShadingIn  {
  COLOR3D   fColor;       // Color - Not used - For future use
  VECTOR3D  fPoint;       // Surface point in Global Coordinates
  VECTOR3D  fNormal;      // Surface Normal in Global Coordinates
  VECTOR3D  fIsoU;       // U iso-parametric vector in Local Coordinates
  VECTOR3D  fIsoV;       // V iso-parametric vector in Local Coordinates
  VECTOR2D  fUV;          // u,v texture values
  ULONG     fUVSpaceID;   // UV Space ID
  VECTOR3D  fPointLoc;    // Surface point in Local Coordinates
  VECTOR3D  fNormalLoc;   // Surface Normal in Local Coordinates
  NUM3D     fPixelRatio;  // Pixel Ratio - Not used - For future use
} ShadingIn;
```

Fields

fColor
This field is reserved for future use.

fPoint
Surface point in Global Coordinates. Examples of such shaders are volumetric procedural shaders for objects that are “cut” in a global matter. An example would be a “Mountain” texture where the rock color depends on the altitude.

fNormal
The Normal to the surface, in Global Coordinates System. An example of such a shader could be a “Snowy Mountains shader”: when the Normal is looking up vertically (which mean that the surface is horizontal), the shader would return a snow color. When the Normal is horizontal, it would return a gray rocky color (because the snow does not hold on steep slopes).

fIsoU
fIsoV
The U or V iso-parametric vector in Local Coordinates. See the “Database Overview” chapter for more details on these vectors. Briefly, they can be described as the vectors that are tangent to the u,v Iso-parametric curves. These curves are the curves on the surface that have a constant u (or v) value. For example, the meridians and parallels on a globe are Iso-parametric curves.
These vectors are necessary to perform Bump effects: if the Normal is to be perturbed based on the u and v values, one needs to know in which direction to perturb it: that is what *fIsoU* and *fIsoV* are for.

©1995-1997 MetaCreations. All rights reserved.

fUV
The u,v texture values. See the “Database Overview” chapter for more details on UV Space. Briefly, a UV Space is a 2D coordinate system wrapped on the object surface (like the longitude and latitude parameters of a globe). There can be several UV Spaces on an object: each one has a unique UV Space ID.

fUVSpaceID
The UV Space ID of the UV Space the *fUV* values belong to.

fPointLoc
The surface point in Local Coordinates. Examples of such shaders are volumetric procedural shaders like a wood function. The shading will not depend on the position of the object in the Scene.

fNormalLoc
The surface Normal in Local Coordinates. See the Rainbow shader example in the toolkit. A shader can use *fNormalLoc* to perform shading that do not depend on the orientation of the object in the Scene.

fPixelRatio
The Pixel Ratio. This field is not used at this time, and reserved for future Pixel Ratio support.

Comments Fields that were not requested in the **ShadingFlags** record have undefined values. Do not use them.

See the **I3DExShader** Interface documentation for further reference. Also read the description of the other Shading data structures, especially **ShadingFlags** and **ShadingOut**.

ShadingFlags

Shading flags are used by the 3D Shell to learn from a Shader which kind of data the Shader requires. This allows the 3D Shell to always calculate the minimum of data for best speed.

See the **ShadingIn** data structure for complete details on the meaning of each field

```
typedef struct ShadingFlags{
  BOOLEAN  fCallOnce;         // Constant Shader: The shader needs to be
      called only once for the whole object or shading area
  BOOLEAN  fNeedsColor;        // Needs Color - Not used
  BOOLEAN  fNeedsPoint;        // Needs surface point in Global Coordi-
      nates
  BOOLEAN  fNeedsNormal;       // Needs surface Normal in Global Coordi-
      nates
  BOOLEAN  fNeedsIsoU;        // Needs U iso-parametric vector in Local
      Coordinates
  BOOLEAN  fNeedsIsoV;        // Needs V iso-parametric vector in Local
```

©1995-1997 MetaCreations. All rights reserved.

```
        Coordinates
    BOOLEAN  fNeedsUV;           // Needs u,v texture values
    BOOLEAN  fNeedsPointLoc;     // Needs surface point in Local Coordi-
        nates
    BOOLEAN  fNeedsNormalLoc;    // Needs surface Normal in Local Coordi-
        nates
    BOOLEAN  fNeedsPixelRatio;   // Needs Pixel Ratio - Not used yet -
        Future use
    BOOLEAN  fChangesNormal;     // The shader changes the Normal. The new
        Normal will have to be read in ShadingOut::fChangedNormal
} ShadingFlags;
```

Fields

fCallOnce
Constant Shader flag. If the Shader is constant, i.e. if it does not depend on any fields of the **ShadingIn** structure, then it should return TRUE in this field. Then its shading procedure will be called only once. Such shaders are a regular Color Shader or a Value Shader.

fNeedsColor
This field is for future use.

fNeedsPoint
The Shader needs the surface point in Global Coordinates (**ShadingIn::fPoint**).

fNeedsNormal
The Shader needs the Normal to the surface, in the Global Coordinates System (**ShadingIn::fNormal**).

fNeedsIsoU
fNeedsIsoV
The Shader needs the U or V iso-parametric vector in Local Coordinates (**ShadingIn::fIsoU** and **ShadingIn::fIsoV**).

fNeedsUV
The Shader needs the u,v texture values (**ShadingIn::fUV**).

fNeedsPointLoc
The Shader needs the surface point in Local Coordinates (**ShadingIn::fPointLoc**).

fNeedsNormalLoc
The Shader needs the surface Normal in Local Coordinates (**ShadingIn::fNormalLoc**).

fNeedsPixelRatio
The Shader needs the Pixel Ratio. This field is not used at this time, and reserved for future Pixel Ratio support.

©1995-1997 MetaCreations. All rights reserved.

fChangesNormal
The Shader will change the Normal. This allows the Shader to tell the 3D Shell that a new Normal (**ShadingOut::fChangedNorma l**) will have to be read after calling the Shader. This is intended for Shaders that changes the Normal like Bump mapping Shaders.

Comments See the **I3DExShader** Interface documentation for further reference. Also read the description of the other Shading data structures, especially **ShadingIn** and **ShadingOut**.

ShadingOut

The **ShadingOut** data structure is filled out by Shaders that implement the **I3DExShader::DoShad e** call. It is also used by renderers that call **I3DShInstance:: DoShade** to get the shading parameters of an Instance in order to calculate the lighting on the surface (using a Phong Illumination Model for example). In both cases, the **Shad-ingOut** data structure is included in a **ShadingInOut** data structure.

```
typedef struct ShadingOut{
  COLOR3D    fColor;            // Diffuse Color (Kd)
  COLOR3D    fSpecularColor;    // Specular Color (Ks)
  NUM3D      fSpecularSize;     // Specular highlight size (Ns)
  NUM3D      fAmbient;          // Ambient factor (Fa)
  NUM3D      fLambert;          // Diffuse factor (Fd)
  NUM3D      fReflection;       // Reflectivity (Rx)
  NUM3D      fTransparency;     // Transparency (Tx)
  NUM3D      fRefractiveIndex;  // Refraction index (Nr)
  COLOR3DfGlow;            // Glow color
  VECTOR3D   fChangedNormal;    // Changed Normal in Global Coordinates
} ShadingOut;
```

Fields

fColor
Diffuse Color (Kd).

fSpecularColor
Specular Color (Ks).

fSpecularSize
Specular highlight size (Ns).

fAmbient
Ambient factor (Fa).

fLambert
Diffuse factor (Fd).

fReflection
Reflectivity (Rx).

©1995-1997 MetaCreations. All rights reserved.

fTransparency
Transparency (Tx).

fGlow
Glow color (G)

fRefractiveIndex
Refraction index (Nr)

fChangedNormal
Changed Normal in Global Coordinates. The Shader will set this field if it told the 3D Shell that it will change the Normal (see **ShadingFlags::fChangesNorma l**).

Comments See the description of the Illumination Model in the “Database Overview” chapter.

UVSpaceInfo

The **UVSpaceInfo** record is filled out by **I3DExGeometricPrimitive::GetUVSpac e**, **I3DShObject::GetUVSpac e**, and **I3DShInstance::GetUVSpac e** to give all the necessary information on the UV Space(s) of the Objects and Instances.

```
typedef struct UVSpaceInfo  {
  ULONG     fID;     // UV Space ID
  VECTOR2D  fMin;    // UV Space minimum boundaries
  VECTOR2D  fMax;    // UV Space maximum boundaries
  BOOLEAN   fWraparound[2];    // Tells is the UV Space is closed in the
      U or V direction

  //-- Flat surfaces handling:
  BOOLEAN   fIsFlatSurface;    // TRUE if the object is a flat surface
      only
  // for xyz to uv
  VECTOR2D  fUVOffset;
  VECTOR3D  fProjU;
  VECTOR3D  fProjV;
  // for uv to xyz
  VECTOR3D  fOffset;
  VECTOR2D  fProjX;
  VECTOR2D  fProjY;
  VECTOR2D  fProjZ;
  } UVSpaceInfo;
```

Fields

fID
The UV Space ID identifying this UV Space.

fMin
The minimum u and v values of the UV Space

fMax
The maximum u and v values of the UV Space

©1995-1997 MetaCreations. All rights reserved.

fWraparound
This array of two boolean values indicate if the UV Space is closed in the u and v directions. For example, a hollow cylinder would be closed in one direction (for instance the v). In this case, Shading Elements can cross over the v=vmax boundary.

fIsFlatSurface
TRUE if a direct linear relationship between the surface points and the UV Space can be done. In this case, calculating UV from XYZ and XYZ from UV becomes very easy and efficient using the next fields.

Comments The Flat Surface formulae are as follows:

Point to UV:

Let *ploc* be a surface point in Local Coordinates and *uv* the result:

```
VECTOR3D  pLoc;
VECTOR2D  uv;
UVSpaceInfo*  info;

uv[0] = info->fUVOffset[0]  +
      pLoc[0] * info->fProjU[0] +
      pLoc[1] * info->fProjU[1] +
      pLoc[2] * info->fProjU[2];
uv[1] = info->fUVOffset[1] +
      pLoc[0] * info->fProjV[0] +
      pLoc[1] * info->fProjV[1] +
      pLoc[2] * info->fProjV[2];
```

UV to Point:

Let *uv* be the UV coordinates and *pLoc* the result:

```
VECTOR3D  pLoc;
VECTOR2D  uv;
UVSpaceInfo*  info;

pLoc[0] = info->fOffset[0] +
      uv[0] * info->fProjX[0] +
      uv[1] * info->fProjX[1];
pLoc[1] = info->fOffset[1] +
      uv[0] * info->fProjY[0] +
      uv[1] * info->fProjY[1];
pLoc[2] = info->fOffset[2] +
      uv[0] * info->fProjZ[0] +
      uv[1] * info->fProjZ[1];
```

Note *fIsFlatSurface* is used only by very specific objects (like Flat Surfaces).

©1995-1997 MetaCreations. All rights reserved.

ShadingInOut

The **ShadingInOut** data structure is the “top” record used for all **DoShade** calls. It gathers all the Shading data structures so **DoShade** has all parameters in hands to compute the Illumination parameters and put them in the **ShadingOut** record.

```
typedef struct ShadingInOut{
  ShadingIn*    fIn;      // Shading input parameters
  ShadingOut*   fOut;     // Shading output parameters
  UVSpaceInfo*  fUVInfo;  // UV Space information
  ULONG         fCurrentCompletionMask;  // Internal flags for shaders.
      Must be set to 0x7F before calling DoShade
} ShadingInOut;
```

Fields

fIn

Points to the Shading Input parameters. See the **ShadingIn** data structure above.

fOut

Points to the Shading Output parameters. This is were **DoShade** will return all Illumination parameters.

fUVInfo

Points to the UV Space information record for the current UV Space. See **UVSpaceInfo**.

fCurrentCompletionMask

Internal flags used by the 3D Shell when combining and computing sub-shaders in the different Illumination Channels. Each channel calculation is marked as computed by zeroing a specific bit. Renderers should set this to 0x7F before calling **DoShade** and not worry any further about it.

Comments

Typically, when a Renderer needs to compute the shading on an Instance, it would follow these steps:

- Get the UV Space ID from the facet currently rendered
- Call **I3DShInstance::GetUVSpace** with this UV Space ID to have a **UVSpaceInfo** record filled in by the Instance.
- Call **I3DShInstance::GetShadingFlags** to learn which parameters need to be calculated
- Then for each surface point to render:
  - Interpolate or calculate somehow the needed parameters
  - Fill in a **ShadingIn** record with them
  - Build a ShadingInOut record by making its fields point to the **ShadingIn** and **UVSpaceInfo** constructed above. Make its *fOut* field point to a **ShadingOut** structure for receiving the results.
  - Call **I3DShInstance::DoShade**
  - Read the *fOut* field and compute the Illumination from that: you get your pixel color.

©1995-1997 MetaCreations. All rights reserved.

Note that if *fCallOnce* was TRUE in the Shading Flags, you need to perform the sub-steps only once.

ShadingElem

A pointer to a **ShadingElem** data structure is passed by the 3D Shell to **I3DExShader** shading calls to describe the **Shading Element** in which the Shader is calculated. A Shading Element can be a Shading Area (a specific sub-region on the object surface) or the Shading Primer (the default shading done at the whole object level). It defines in which UV boundaries the shading is to be calculated.

The area is defined in the surface UV Space.
fBBox and fShaderBox define the same rectangle in two different systems. fShaderBox defines the boundaries of the UV values that will be passed to the Shader. Therefore, moving or resizing the Shading Element on the object surface will not change the shading inside the Shading Element (provided that the Shader bases its calculations on the UV values. Otherwise the discussion is pointless).

```
typedef struct ShadingElem {
  ShadingElemRect  fBBox;       // Extent of the Shading Element in the
      UV Space
  ShadingElemRect  fShaderBox;  // Extent of the Shading Element as seen
      by the shader
  NUM3D            fOpacity;    // Opacity of the Shading Element vs.
      whatever other Shader Elements may be under
 } ShadingElem;
```

Fields

fBBox
The Shading Element extent in the UV Space, and thus define the position and size of the Shading Element on the Object surface.

fShaderBox
The UV boundaries of the Shading Element as seen by the Shader. The Shader should base its uv calculations on this field, rather than *fBBox*.

fOpacity
Opacity of the Shading Element vs. whatever other Shader Elements may be under. This field is used by the 3D Shell, the Shader has little use for it.

Example A Checker sub-shader will base its modulo calculations on *fShaderBox* See the Checker example provided in the Toolkit.

UVShMaps

A pointer to a **UVShMaps** data structure is used in **I3DShInstance::GetUVRasterS h-Map**, **I3DShInstance::GetUVExportShMa p** and **I3DShInstance::SetShadingFro m-Maps**.This structure contain all the maps for each shading channel.

©1995-1997 MetaCreations. All rights reserved.

```
typedef struct UVShMaps {
  IShRasterOffscreen* fColorMap;
  IShRasterOffscreen* fHighlightMap;
  IShRasterOffscreen* fShininessMap;
  IShRasterOffscreen* fBumpMap;
  IShRasterOffscreen* fReflectionMap;
  IShRasterOffscreen* fTransparencyMap;
  IShRasterOffscreen* fRefractionMap;
  IShRasterOffscreen* fGlowMap;
  } UVShMaps;
```

Fields

- *fColorMap*
- *fHighlightMap*
- *fShininessMap*
- *fBumpMap*
- *fReflectionMap*
- *fTransparencyMap*
- *fRefractionMap*
- *fGlowMap*

  Those fields are pointers on map offscreens for each shader channel.

Example A Checker sub-shader will base its modulo calculations on *fShaderBox* See the Checker example provided in the Toolkit.

MappingParams

A pointer to a **MappingParams** data structure is used in **I3DShInstance::SetShading-FromMaps**.This structure contain the informations for mapping on a channel.

```
typedef struct _MappingParams {
  Boolean  fFlipVert;          // true if and only if the map shader
      should be flipped FROM ITS ORIGINAL orientation
  Boolean  fFlipHoriz;
  QuickFix fBrightness;        // A quickfix number between zero and one
      (65536)
  short    fNinetyDegreeCount; //number between 0 and 3 indicating the
      number of rotations to do on the map
  } MappingParams;
```

Fields

- *fFlipVert*

  The map is vertically flipped.

- *fFlipHoriz*

  The map is horizontally flipped.

- *fBrightness*

  The brightness of the map (0.00 for black, 1.00 for the original map).

©1995-1997 MetaCreations. All rights reserved.

fNinetyDegreeCount
The number of fourth right rotations applied on the map.

Comments You can set those attributes in the shader editor when you apply a map on a channel.

©1995-1997 MetaCreations. All rights reserved.

3.12.Ray Tracing Data Structure s

Ray3D

The Ray3D structure is used to describe a light ray. This structure is mostly used by ray-tracing algorithms.

```
typedef struct Ray3D{
  VECTOR3D  fOrigin;        // Origin of the ray
  VECTOR3D  fDirection;     // Direction of the ray
  NUM3D     fFocalPoint;    // Coordinate of the focal point of the light
      beam
  NUM3D     fAngle;         // Angle of the light beam
  NUM3D     fWidthAtFocal;  // width of the light beam
} Ray3D;
```

Fields

- *fOrigin*
  Origin of the ray in Global Coordinates.

- *fDirection*
  Direction of the ray in Global Coordinates. It must be a unit vector.

- *fFocalPoint*
  Coordinate of the focal point of the light beam. This coordinate is a coordinate along the ray's line, i.e. the focal point is at: *fOrigin* + *fFocalPoint* * *fDirection*

- *fAngle*
  Angle of the light beam. In fact, this is not really an angle, but more the sinus of the light beam angle. It controls how the light beam spreads out. See the formula below.

- *fWidthAtFocal*
  Width of the light beam at the focal point.

Comments

fFocalPoint, *fAngle* and *fWidthAtFocal* are made to calculate the "pixel size", i.e. the size that the light beam covers at a specific point if it is considered as a cone of light rather than a straight line. The formula works like this:

$$Size(t) = fWidthAtFocal + (t - fFocalPoint) * fAngle$$

Where 't' is the "ray parameter": any point on the ray can be written as:

$$fOrigin + t * fDirection \quad \text{with t as a real value.}$$

RayHit3D

The RayHit3D structure is used by **I3DExGeometricPrimitive::RayHi t** and **I3DExGeometricPrimitive::RayAllHit s**. Its purpose it to return ray-tracing hit informa-

©1995-1997 MetaCreations. All rights reserved.

tion when a hit occurs between a ray and a Primitive's surface.

```
typedef struct RayHit3D{
  VECTOR3D    fPosition;        // Intersection point in Global Coordi-
      nates
  VECTOR3D    fNormal;         // Normal at that point in Global Coordi-
      nates
  VECTOR2D    fUV;              // (u,v) texture values at that point
  NUM3D      ft;              // t parameter along the ray: fPosition =
      RayOrigin + t * RayDirection
  VECTOR3D    fIsoU;            // Isoparametric U vector in the Object
      Local Coordinates
  VECTOR3D    fIsoV;            // Isoparametric V vector in the Object
      Local Coordinates
  void*       fPrivate1;         // Private data for renderer.
  void*       fPrivate2;         // Private data for renderer
  void*       fPrivate3;         // Private data for renderer
  BOOLEAN     fShouldSetUV;      // TRUE if fUV should be set
  BOOLEAN     fShouldSetIsoUV;   // TRUE if fIsoU and fIsoV should be set
  BOOLEAN     fHasSetPosition;   // TRUE if fPosition was set
  BOOLEAN     fHasSetNormal;     // TRUE if fNormal was set
  BOOLEAN     fHasSetUV;         // TRUE if fUV was set
  BOOLEAN     fHasSetIsoUV;      // TRUE if fIsoU and fIsoV were set
} RayHit3D;
```

Fields

fPosition
Intersection point in Global Coordinates.

fNormal
Normal at that point in Global Coordinates.

fUV
(u,v) texture values at that point. To be filled in only if *fShouldSetUV* was TRUE.

ft
Ray parameter of *fPosition*. *fPosition* = RayOrigin + *ft* * RayDirection

fIsoU
Isoparametric U vector in the Object Local Coordinates at the intersection point. To be filled in only if *fShouldSetIsoUV*was TRUE.

fIsoV
Isoparametric V vector in the Object Local Coordinates at the intersection point. To be filled in only if *fShouldSetIsoUV*was TRUE.

fShouldSetUV
TRUE if *fUV* should be returned

fShouldSetIsoUV
TRUE if *fIsoU* and *fIsoV* should be returned.

©1995-1997 MetaCreations. All rights reserved.

fHasSetPosition
Should return TRUE if *fPosition* was calculated.

fHasSetNormal
Should return TRUE if *fNormal* was calculated.

fHasSetUV
Should return TRUE if *fUV* was calculated.

fHasSetIsoUV
Should return TRUE if *fIsoU* and *fIsoV* was calculated.

Comments See the sphere example in the geometric primitive cookbook for more information on this structure.

RayHitParameters

The RayHitParameters structure is used by **I3DShInstance::RayHi t** and **I3DExGeometricPrimitive::RayHi t**. to calculate the hit of a ray on an object.

```
typedef struct RayHitParameters {
  NUM3D tmin;// minimum t value
  NUM3D tmax;// maximum t value
  BOOLEANfShouldSetS;
  BOOLEANfShouldSetUV;
  BOOLEANfZBShading;
  BOOLEANfNoGrid;
  void *fFilterInstance;
} RayHitParameters;
```

Fields

tmin
Minimum value for the parametric variable t.

tmax
Maximum value for the parametric variable t, that mean that for this value, the ray hit an object and don’t go thru. If no object have been found yet, the maximum value is set instead.

fShouldSetS
tocomment.

fShouldSetUV
tocomment.

fZBShading
tocomment.

©1995-1997 MetaCreations. All rights reserved.

fNoGrid
tocomment.

fFilterInstance
tocomment

Comments See **Ray3D** structure for more details.

HitInfo

The HitInfo data structure works in conjunction with the RayHit3D structure. If a hit occurs and that the ray-tracing needs all the information on this hit, then **I3DExGeometricPrimitive::GetRayHitDetail s** is called.

```
typedef struct HitInfo {
  //-- Output data:
  I3DShInstance*  fInstance;    // Instance that was hit by the ray
  VECTOR2D     fUV;             // (u,v) texture values at the intersec-
      tion point
  ULONG        fUVSpaceID;      // UV Space ID
  VECTOR3D     fPosition;       // Intersection point in Global Coordi-
      nates
  VECTOR3D     fNormal;        // Normal at that point in Global Coordi-
      nates
  COLOR3D     fColor;         // Rendered final color (all lighting and
      shading  included)
  NUM3D        fAlpha;          // Alpha-channel value
  ULONG        fIndex;          // Instance unique index
  NUM3D       fDistance;      // Distance from the point of view to the
      intersection point

  //-- Input flags:
  BOOLEAN  fNeedsColor;          // Needs fColor
  BOOLEAN  fNeedsDistance;       // Needs fDistance
  BOOLEAN  fNeedsPosition;       // Needs fPosition
  BOOLEAN  fNeedsNormal;         // Needs Normal
  BOOLEAN  fNeedsAlpha;          // Needs Alpha
  BOOLEAN  fNeedsIndex;          // Needs Instance index
  BOOLEAN  fNeedsUV;             // Needs (u,v) values
  BOOLEAN  fZBShading;           // Ask for Z-Buffer-like shading: no
      transparencies, no reflections and no shadows
  } HitInfo;
```

Fields

fInstance
Instance that was hit by the ray.

fUV
(u,v) texture values at the intersection point. To be filled only if *fNeedsUV* is TRUE.

fUVSpaceID
UV Space ID at the intersection point

©1995-1997 MetaCreations. All rights reserved.

fPosition
Intersection point in Global Coordinates. To be filled only if *fNeedsPosition* is TRUE.

fNormal
Normal at the intersection point in Global Coordinates. To be filled only if *fNeedsNormal* is TRUE.

fColor
Rendered final color (all lighting and shading included). To be filled only if *fNeedsColor* is TRUE.

fAlpha
Alpha-channel value. To be filled only if *fNeedsAlpha* is TRUE.

fIndex
Instance unique index. To be filled only if *fNeedsIndex* is TRUE.

fDistance
Distance from the point of view to the intersection point. To be filled only if *fNeedsDistance* is TRUE.

fNeedsColor
fNeedsDistance
fNeedsPosition
fNeedsNormal
fNeedsAlpha
fNeedsIndex
fNeedsUV
These flags control the calculation of the returned data.

fZBShading
Ask for Z-Buffer-like shading: no transparencies, no reflections and no shadows.

Comments See the sphere example in the geometric primitive cookbook for more information on this structure.

©1995-1997 MetaCreations. All rights reserved.

©1995-1997 MetaCreations. All rights reserved.

Appendix I - Toolkit Libraries

©1995-1997 MetaCreations, All rights reserved.

I.i.Using the Failure Handlin g

The API for this section does exist, but is not documented yet. The documentation will come in a future version of the DreamSDK. If you need more information, please contact MetaCreations directly.

©1995-1997 MetaCreations, All rights reserved.

Appendix II - Using the COM Dynamic Link-ing

This chapter described in more details the COM Dynamic Linking supported in Ray Dream Designer.

Before reading this chapter, it is highly recommended to read the Introduction chapter, especially the section called "how the whole thing works".

©1995-1997 MetaCreations, All rights reserved.

II.i.About COM in two word s

COM (Component Objects Model) is the low-level dynamic linking system on which OLE is built. It is a very widely spread industry standard, and is actively promoted by major industry players. So if you are already familiar with OLE, you will find it easy to be started with this documentation. COM offers a nice and clean C++ like interfacing, and it is highly recommended to read the excellent book “Inside OLE 2” by Kraig Brockschmidt from Microsoft Press to learn everything about it. You can also find all the necessary information in any edition of the MSDN (Microsoft Developers Network) CDs.

©1995-1997 MetaCreations, All rights reserved.

II.ii.Using COM for simple Extension s

"Simple" 3D Components are extensions that do not call back the Shell API to perform their duties. Such extensions are Primitives, Shaders, Light sources, Ambient lights, Gels, Atmospheric Shaders, Background and Backdrop Shaders, and Cameras.

Like any regular COM component, the entry key is the **DllGetClassObjec t**() procedure. This is where your Class Factory will be instanciated. You will find the implementation of this procedure in the *DLL.CPP file in each sample of the DreamSDK.

In your Class Factory object, you will instanciate your Component in **CreateInstanc e**(). See the *FACT.CPP file of each sample.

In your Component object, you will implement the usual calls of the **IUnknown** interface, using a reference counter, etc.

Regarding the I3DExtension and I3DExDataExchanger interfaces, make sure you implement the **Clone**(), **GetResID**() and **GetExtensionDataBuffe r**() procedures. These are vital for your Extension health. Generally you will want to enable failure handling, so call **InitCoFailur e**() in **ShellUtilitiesIni t**().

Make sure you choose a CLSID and that you copy its value in the GUID resource (see the "Managing the User Interface" Appendix for more details. Make sure that all your COMP, pMAP, GUID, View, etc. resources have the same ID number and that this number is the one returned by **GetResID**. You still need to choose a 4-letters Class ID and put it in your COMP resource.

And, once again, Windows programmers do not need to enter their IID and CLSID in the Windows Registry Database. Simply give your Component the .RDX suffix, and put it in the EXT directory (or in any sub-directory of the EXT directory). Ray Dream Designer will find it automatically.

Use the samples of the DreamSDK as a framework to start your own extensions.

Apart from the special and important role played by the resources in the .DTA (or Data on MacOS) file, all this will seem very common and simple for the COM/OLE programmer. Well, it is !

©1995-1997 MetaCreations, All rights reserved.

II.iii.Using COM in Extensions using Shell Interface s

First, you must follow all the steps described in the previous section.

Then some extensions need to call back the Shell. For example, a 3D Import Filter will need to create 3D objects and put them in the scene. There can be two types of situations:

- A Shell object pointer is passed to you as a parameter of one of your Extension procedure.
- You need to instanciate a new Shell object and work on it.

Using Shell objects received as parameter s

This is a very simple case. The Shell already allocated an object and gives you its pointer as the parameter of one of your Component procedure.

Example:

```
HRESULT I3DExtension::ShellUtilitiesInit(IShUtilities* shellUtilities);
```

The *shellUtilities* parameter is a Shell object of the **IShUtilities** Interface. You simply call its methods as you wish. There is no need to call **AddRef**() on it. Do not call **Release**() either, the Shell will take care of its own children.

Example:

```
char* someBuffer;
someBuffer = shellUtilities->Malloc(128); // Let's allocate a buffer
```

Instanciating a Shell object to work on i t

This situation appears when you need to create an object, like creating a 3D Primitive to put it in the Scene.

The procedure a little bit unusual for COM programmers. Instead of calling the global procedure **CoCreateInstanc e**(), you will call **IShUtilities::CoCreateInstanc e**() on the *shellUtilities* pointer passed in **I3DExtension::ShellUtilitiesIni t**().

The *shellUtilities* object is your key back to the Shell. It is a little bit like if the Shell was an In-Proc server from the Extension's point of view. There are other basic methods that you will find useful in the **IShUtilities** Interface, like the ones managing memory allocation (**IShUtilities::Mallo c**, **IShUtilities::Fre e**, etc.). Using these calls instead of the regular malloc() libraries is highly recommended to benefit from Ray Dream Designer internal memory management and failure handling.

Note that it is OK to store the *shellUtilities* object in a global variable or in the your Component for later use. This pointer is never disposed of. In fact, if you called **InitCoFailure**(), the **gShellUtilitie s** global variable will be initialized and ready to be used.

©1995-1997 MetaCreations, All rights reserved.

Example:

We create a Group and put it in the Scene:

```
HRESULT TDXFImporter::DoImport(THIS_ char* fullPathName, I3DShScene*
      scene, I3DShTreeElement* fatherTree)  {

 I3DShGroup*       group;
 I3DShTreeElement* groupTree;// The same as group, but as a Tree Element
.......

 gShellUtilities->CoCreateInstance(CLSID_StandardGroup, NULL,
      CLSCTX_INPROC_SERVER, IID_I3DShGroup, (LPVOID*) &group);

 // Do whatever we want on the group
 ......

 // Use it as a Tree Element to put it in the scene
  group->QueryInterface(I3DShTreeElement, (LPVOID*) &grouptTree);
  groupTree->SetScene(scene);
  fatherTree->InsertLast(groupTree);
  groupTree->Release();// We do not need this interface anymore
```

©1995-1997 MetaCreations, All rights reserved.

Appendix III - Platforms and Compilers issues

This chapter describes details about specific issues related to compilers and platforms.

©1995-1997 MetaCreations, All rights reserved.

III.i.Compiling for Window s

The supported compiler is Visual C++ 2.2 or better from Microsoft.

COM extensions are simply standard 32-bit DLLs. Take a look at the sample make files in the DreamSDK. There are no special tricks there.

©1995-1997 MetaCreations, All rights reserved.

III.ii.Compiling for MacO S

The supported compilers are Code Warrior 7 Gold MPW C/C++ and IDE from Metrowerks.

The use of the excellent TheDebugger™ from Steve Jasik is recommended for debugging. This debugger is worth the learning curve to do any serious Macintosh programming.

COM extensions are standard **applications**. They just have the RDEX type instead of APPL. They are launched like any regular MacOS Application, but their heap will be located inside the Ray Dream Designer heap, unless Jasik's TheDebugger is running. In this case, they are launched outside to allow easy debugging (just make sure you do not run out of System RAM to launch your extensions).

Warning: the Creator must be unique !

The Creator of each extension file must be unique. In other words, there should not be several RDEX files with the same creator. Failing to do so may result in a **lock-out** when your extension is launched. Ray Dream Designer will get stucked if another extension with the same creator was already loaded, because it will be totally confused to decide if the extension was launched properly or not.

You can ask Apple Computer to give you a unique Creator (this is recommended if you plan to distribute your Component). Apple does it by e-mail for free.

Be carefull that this situation may happen to you if you start from one of the DreamSDK samples. Make sure you change the creator in the Make file !

68K code: using the Far Model

When compiling 68K code, the Far Model must be used. Do not generate code that uses A5, because A5 will not be yours when your procedures are called (it will be the Shell's A5).

Be carefull when you use 3rd parties libraries: they may use global variables and calls relative to A5. Make sure you use libraries that were compiled with the Far Model.

M.68K, M.PPC, etc.

Because we are dealing with two platforms (68K and PowerPC), the following suffix convention has been used in the DreamSDK Samples.

	68K	PowerPC
Metrowerks compilers	M.68K	M.PPC

©1995-1997 MetaCreations, All rights reserved.

For example, `COMATMOS.M.PPC.MAK` is the make file to compile the COMATMOS extension in PowerPC code using CodeWarrior compilers. The compiled binary will end up in the `:OBJS:M.PPC:` sub-folder.

©1995-1997 MetaCreations, All rights reserved.

Appendix IV - Managing the User Interface

There are two ways to create the User Interface for an extension, you can create it on Macintosh with tools like ResEdit or AdLib. The resource file is then converted to a .dta file which is then used on both Mac & Windows platforms. Or if you are on Windows, you can use the Windows resources but in this case you can only use your User Interface on the Windows Version.

If you do not use the Windows resources, you must have a .DTA file that contains a view of your Interface and some other resources. In this part, we will describe all the needed resources on both platform (Macintosh or Windows).

Building a .dta file is the preferred solution because it is a cross-platform solution. The same .dta file will give you a native look-and-feel on each platform: buttons will look like Windows buttons on Windows, and like Macintosh buttons on MacOS. The 3D Shell is managing all this for you.

Make sure you are familiar with the main concepts of Components before reading this chapter. Specifically, be sure you read the paragraph called “How the whole thing works” in the “Introduction” chapter.

©1995-1997 MetaCreations, All rights reserved.

IV.i.General Information about resources

For each extension, you need some resources with the same ID as the one you give in the function **GetResID()** of the API.
In all case you need a resource "*COMP*" that describes your extension.
In most of them, you need a resource "*pMAP*" that describes the different parameters that the shell can access.
And if you have an User Interface, you must have a resource "*View*" for the DTA file or "*Dialog*" on the Windows side.
If you have created a COM extensions, do not forget to add the "*GUID*" resource.
And some extensions require some additional resources like a "Cmpp" resource for Import & Export Filters, or a "CICN" resource for Geometric Primitives. But the 3D Shell do not yet support the Windows Icon resources for a Geometric Primitives, so you have to use the default icon for the Geometric Primitives, to do so do not create a Cmpp resource.

©1995-1997 MetaCreations, All rights reserved.

IV.ii.The registration resource s

When you create an extension, you have to tell the Shell what kind of extension it is. You have to give a name to put in the right menu. And you have to tell where it can find the different methods of the extension.
You have to create a “COMP” resource that contains the type of extension and the name, because the link with the methods is made at run-time in the *RDDLLMain* function. In addition to the COMP resource, you have to create the “GUID” resource to tell the Shell that the extension is COM and to give the interface ID of the object and its class ID.

The “*COMP*” resource

The resource “COMP” describe the type of extension and the extension.

If you create the resource on Mac, it will be like this:

```
resource ‘COMP’ (128) /* resource ID = 128*/
{ ‘shdr’, /* type of extension, the family name */
  ‘COch’, /* Class ID, to recognize your extension in the family */
  “Checker”, /* Name of your extension, it will appears in the Shell to
      describe and select your object */
  “COM Examples”, /* Sub Family Name, for the shaders, it will create a
      new submenu call “COM Examples” in the menu “Component” of the
      Shader Editor */
  0x00030103, /* Version */
  “3.1.3”, /* Version String */
  “Examples”, /* Comments (it will appear in the About extension window)
      */
  3      /* API Version the Extension is based on */
}
```

The windows resource is very similar:

```
128 COMP DISCARDABLE
BEGIN
   “shdr”,
   “WinC”,
   “Win Checker\0”, /* C String, so do not forget \0 */
   “COM Examples\0”,
   “3.1.3\0”
   “TOOLKIT Examples\0”,
   0x0003L
END
```

COM Specific resource “GUID”

When you create a COM Object, you have to create an unique GUID. And to tell the 3D Shell that your extension can be called with the COM mechanism, you have to create a GUID resource that contains the IID (Interface ID) and the CLSID (Class ID) of the COM object.

©1995-1997 MetaCreations, All rights reserved.

So the “GUID” resource will look like this:

```
data ‘GUID’ (128)
{ $”06DD 2161 2ED7 101C 885F 0402 1C00 7002” /* IID_I3DExShader */
  $”1C2D AC60 2ED7 101C 885F 0402 1C00 7002” /* CLSID_Checker */
}
```

Or

```
128 GUID DISCARDABLE
BEGIN
  0x06DD2161L,0x2ED7,0x101C,”\x88\x5f\x04\x02\x1c\x00\x70\x02”,
  0x1C2DAC60L,0x2ED7,0x101C,”\x88\x5f\x04\x02\x1c\x00\x70\x02”
END
```

See the end of this chapter for a complete list of Interface Ids.

©1995-1997 MetaCreations, All rights reserved.

IV.i.User Interface resource s

You need two resources to manage the User Interface of an extension. You need a view or a dialog resource for the visual appearance of the Interface, and a parameters map to tell the shell where to put the different value of the controls you put in your view.

To create a view on your Macintosh (that can be use on the Windows version of Ray Dream Designer) you can use AdLib for example. If you have decided to use Windows resources, Microsoft Visual C++ have a resource editor that allows you to create the dialogs.
But do not forget the different ID of the controls, because they will be used in the “PMAP” resource.

The “PMAP” resource

There are two types of “PMAP”, the first one does not include any features for animation and it is called “PMAP”, the second one allows you to give more information for the animation and it is called “pMAP”.

If you create your resource on Mac to have a cross platform resource, you will create your “PMAP” like this:

```
resource ‘PMAP’ (128)
{ /* First Array Element */
  ‘SIZH’, /* ID of the Control in the view */
  ‘shrt’, /* type of the value */
  /* Second Array Element */
  ‘SIZV’,
  ‘shrt’,
}
```

Or

```
resource ‘pMAP’ (128)
{ /* First Array Element */
  ‘SIZH’, /* ID of the Control in the view */
  ‘shrt’, /* type of the value */
  interpolate,/* long that allow to tell if the parameters can be inter-
      polate */
  “# Hori.”, /* String that will appear in the timeline to describe the
      parameter (P String = 1 char for length and 31 characters) */
  /* Second Array Element */
  ‘SIZV’,
  ‘shrt’,
  interpolate,
  “# Vert.”,
}
```

There is some differences, if you want to make the resource with Windows. The resource is called “PMAP” but includes the features for animation.

©1995-1997 MetaCreations, All rights reserved.

```
128 PMAP DISCARDABLE
BEGIN
   0x0002, /* short that gives the number of entry in the PMAP */
   “NBHS”,  /* ID of the Parameter \ Windows Control */
   “shrt”, /* Type of value return by the control */
   1L,       /* Flag to set if the parameter is interpolable (long) */
   “Nb Sq Hori.\0”, /* string for the time line (C String)*/
   0L,       /* minimum value of the Windows Control (long) */
   40L,  /* maximum value of the Control (long) */

   “NBVS”,
   “shrt”,
   1L,
   “Nb Sq Vert.\0”
   0L,
   40L
END
```

In the Windows “PMAP”, the ID parameter is given as 4 characters, but when you define our Windows control the ID must be a short. So to get the control ID from those 4 characters, you use the two last. For example, if you use “NBHS” as an PMAP ID, the control ID will be 0x4853 (0x48 is the ASCII code of ‘H’ and 0x53 of ‘S’).
Those four letters are used in the file format, so be sure you have 4 characters and to have unique set of the 2 last characters (“12AB” and “34AB” are not valid, they will be attached to the same Windows Control).

©1995-1997 MetaCreations, All rights reserved.

IV.i.List of types used in the “PMAP ”

Type	ID	Comments
Boolean	bool	= unsigned char
short	shrt	
long	long	
NUM3D	qfix	
VECTOR2D	qfv2	
VECTOR3D	qfv3	
COLOR3D	colo	
char[256]	c255	C string of 255 characters with a terminal 0
Windows Only:		
NUM3D	qf16	allows you to use the NUM3D in a slider (16 bits), the range will be -128.0 to 127.996. (if you use qfix it will be -0.5 to 0.49998)

©1995-1997 MetaCreations, All rights reserved.

IV.ii.List of Family ID

Family	Family ID	API Version #	COM IID
Ambient light	“ambi”	4	IID_I3DExAmbient
Atmospheric Shader	“atmo”	3	IID_I3DExAtmosphericShader
Backdrop	“drop”	3	IID_I3DExBackdrop
Background	“back”	3	IID_I3DExBackground
Behavior	“treb”	4	IID_I3DExTreeBehavior
Camera	“came”	3	IID_I3DExCamera
Deformer	“defo”	4	IID_I3DExDeformer
Exporter	“3Dou”	3	IID_I3DExExportFilter
Gel	“gel “	3	IID_I3DExLightGel
Geometric Primitive	“prim”	3	IID_I3DExGeometricPrimitive
Importer	“3Din”	3	IID_I3DExImportFilter
Light	“lite”	3	IID_I3DExLightsource
Motion Link	“link”	4	IID_I3DExMotionLink
Scene Operation	“scop”	4	IID_I3DExSceneOperation
Shader	“shdr”	3	IID_I3DExShader
Tweener	“twee”	4	IID_I3DExTweener

©1995-1997 MetaCreations, All rights reserved.

IV.iii.View user interface elements table

'View' resources are constructed using AdLib on the Macintosh. They allow the construction of sophisticated user interfaces. See the « Building a .dta file » section below for more details on how to convert your Macintosh resource file into a cross-platform .dta file.

Editing the 'View' resources

The 'View' resources are resources that were defined by Apple Computer, Inc. for the MacApp¨ development toolkit. This format is used in Ray Dream Designer because it is much more powerful that the regular 'DLOG'/'DITL' resources.

There are several View resource editors available on the market. The excellent AdLib 2.0 from Nick Nallick is currently shipping with the MacApp toolkit from Apple Computer. Earlier versions of MacApp were providing another editor, called ViewEdit.

All the user interface elements described below are 'View' items. Some are customized versions of the standard base classes. If these cases, there are two class names: the **Base Class Name** indicates which standard item should be created in AdLib, and the **Class Name** indicates what should be put in the *Class* box of the item's attributes.

The Top View

When you create the view of your user interface, your top view must have the 'EDIT' ID and its class must be a **TComponentView**, except for shaders that use the **TExternalShView** class name.

Text entry box

Base Class Name TEditText

pMAP Type ID **"**c255"

Comments This control allows to edit a C String (null terminated string) with a maximum of 255 characters.

Number entry box

Base Class Name TNumberText

pMAP Type ID "long", "shrt"

Comments This control allows to edit an integer.

©1995-1997 MetaCreations, All rights reserved.

Fixed Number entry box

Class Name TFixedNumberText
Base Class Name TNumberText

pMAP Type ID “qfix”

Comments This control allows to edit a Fixed.

Number entry box with 3D units

Class Name T3DUnitNumberText
Base Class Name TNumberText

pMAP Type ID qfix

Comments Min, max and default values are long integers that are interpreted as Fixed values (i.e. 65536 -> 1.0). To be used in an extension, the ME3D resources should be copied in the extension resource file. The 3D Units are inches(in.), millimeters(mm), centimeters(cm), meters(m) and points(pts).

Number entry box with 2D units

Class Name TUnitNumberText
Base Class Name TNumberText

pMAP Type ID qfix

Comments Min, max and default values are long integers that are interpreted as Fixed values (i.e. 65536 -> 1.0). To be used in an extension, the ME2D resources should be copied in the extension resource file. The 2D Units are inches(in.), centimeters(cm) and points(pts).

Slider

Class Name TSlider
Base Class Name TNumberText

Comments Min, max and default values are long integers that are interpreted as Fixed values (i.e. 65536 -> 1.0). The User Value give the type of slider (slider with long, with Fixed, with a percent, etc.)

©1995-1997 MetaCreations, All rights reserved.

User Value	pMap Type ID	Slider Type	
2000	qfix, long, shrt	Simple Slider	
2002	qfix	Fixed * 100 (with %)	
2003	long, shrt	Long/short	
2004	qfix	Fixed (integer part)	
2005	qfix	Fixed * 100	
2008	qfix	Fixed with decimal point	

Double Slider

Class Name RDNoOverlapSlider
Base Class Name TControl

pMAP Type ID “qfv2”

Comments This slider allows to edit two Fixed values. The first value will always be lower or equal to the second one.

Triple Slider

Class Name RDMinMaxValueSlider
Base Class Name TControl

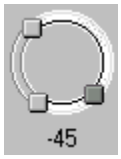
pMAP Type ID “qfv3”

Comments This slider is used to set a value between two limits. The first value of the Vector is the minimum, the third value is the maximum and the second one is a value between the 2 previous.

This slider is generally associated with the **Limiter Popup** which allows to lock, to limit or to free the value of the slider. To associate the **Limiter Popup** and the **Triple Slider**, you have to put them in a same TCluster.

Angle Slider

Class Name RDMinMaxValueAngle
Base Class Name TControl



pMAP Type ID “qfv3”

Comments This slider is used to set a value between two angular limits. The first value of the

©1995-1997 MetaCreations, All rights reserved.

Vector is the minimum, the third value is the maximum and the second one is a value between the 2 previous.

This slider is generally associated with the **LimiterPopup** which allows to lock, to limit or to free the value of the slider. To associate the **Limiter Popup** and the **Angle Slider**, you have to put them in a same TCluster.

Limiter Popup

Class Name LimiterPopup
Base Class Name TPopup

pMAP Type ID “shrt”

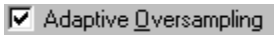
Comments This Popup is only used with a Vector Slider (**Angle Slider** and **Triple/Double Slider**). You must include the MENU resource :

Button

Base Class Name TButton

Check box

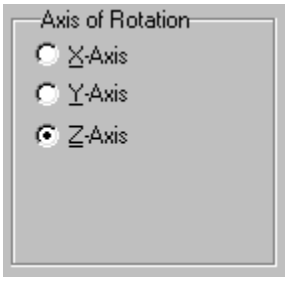
Base Class Name TCheckBox



pMAP Type ID “bool”, ”shrt”, “long”

Radio button

Base Class Name TRadioButton



pMAP Type ID “long”

Comments You have to put the Radio Buttons in a cluster (TCluster). The ID of the selected Radio Button will be return in the long. The pMAP ID is the ID of the TCluster.

©1995-1997 MetaCreations, All rights reserved.

Color button

Class Name TColorButton
Base Class Name

pMAP Type ID “colo”

Comments This control allows to edit a RGB Color.

Pop up

Base Class Name TPopup

pMAP Type ID “shrt”

Comments It will use a MENU resource in your extension resource file. The result of this Control is the number item of this MENU resource.

©1995-1997 MetaCreations, All rights reserved.

IV.i.The Component Private resources (“Cmpp”)

Some external component like the exporter, the importer and the geometric primitives have an extra resource called “Cmpp”. This resource have a different structures for each family.

Exporter

Importer

The “Cmpp” resource have the following structure for those two families :

```
type 'Cmpp'{   /* 'RDin' private data */
   longint;    /* Family Signature */
   longint;    /* Class Signature */
   longint;    /* flags */
   literal longint;/* creator (for Macintosh exporters only) */
// The following is the 'FTYP' format of RFrame, with only 1 entry
   FTYPDATA
};
```

Where FTYPDATA is

```
#define FTYPDATA\
   integer = $$CountOf(TypeArray); /* Number of types*/ \
   wide array TypeArray{\
      literal longint; /* Reference */\
      cstring;/* Name */ \
      integer = $$CountOf(MacTypeArray); \
      wide array MacTypeArray{ \
         literal longint; /* Mac type  */ \
         }; \
      integer = $$CountOf(ExtTypeArray); \
      wide array ExtTypeArray{ \
         literal longint;/* Extension (last char ignored)  */ \
         }; \
      };
```

And the different flag values are :

```
#define isAlienType 0
#define isMainType 1
#define hasExportOptions 2
```

For example, the DXF Importer and Exporter “Cmpp” resources are :

```
resource 'Cmpp' (128) {
   '3Din',
   'dxf ',
   isAlienType,
   'RD3A',        // creator
   {
```

©1995-1997 MetaCreations, All rights reserved.

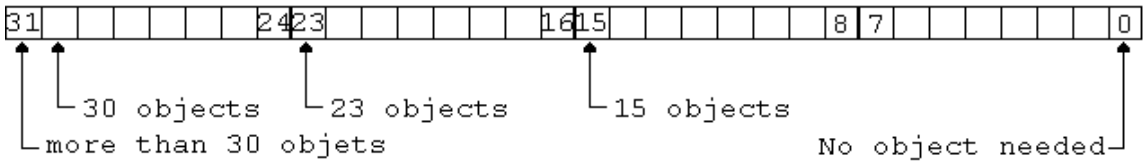
```
            'DXF ',  // Reference
            "DXF",       // Name
            {'TEXT' },// List of MacOS types
            {'DXF ' },// List of extensions
        };
    };


    resource 'Cmpp' (129) {
        '3Dou',
        'dxf ',
        isAlienType,
        'ttxt',         // creator
        {
            'DXF ',  // Reference
            "DXF",       // Name
            {'TEXT' },// List of MacOS types
            {'DXF ' },// List of extensions
        };
    };
```

Primitives

This resource contains the ID (a short) of the main icon of the primitive (it will be shown in the hierarchy window). The two icons in the toolbar (not selected and selected) have their ID calculated like this : not selected ID = main ID + 50, and selected ID = main ID + 100.

Because the Shell do not support Windows Icon, you must not define a “Cmpp” resource in a Windows resource file.

Scene Operations
This resource is used to know it what case the Scene Operation can be called by the Shell. This resource only contains a long (4 bytes) in which each bit has a different signification:



For example, if your Scene Operation can only work when there are 2 or 4 objects selected, you have to put this value in the Cmpp resource :

```
00000000 00000000 00000000 00010100 = 20
```

Or if your Scene Operation can be used when there is 1 or more object selected, you will have :

```
11111111 11111111 11111111 11111110 = 0xFFFFFFFE
```

©1995-1997 MetaCreations, All rights reserved.

And if you Scene Operation do not need any selection, it will be :

```
00000000 00000000 00000000 00000001 = 1
```

If you do not define the Cmpp resource, your Scene Operation can be used in every case (by default the value is 0xFFFFFFFF).

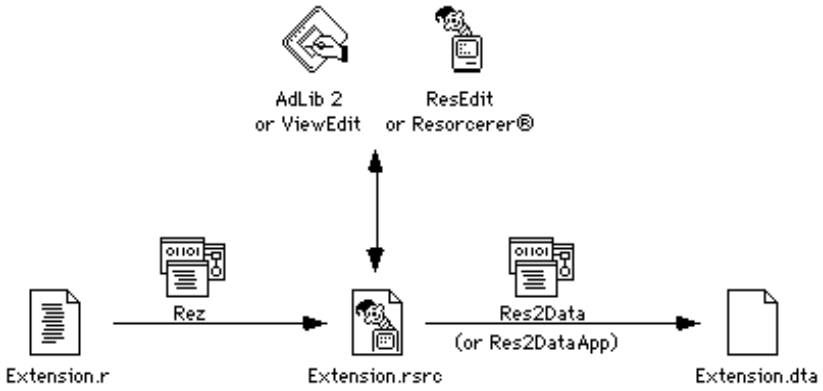
This resource can be created on both cross-Platform and Windows-only sides, but for the Windows resource do not forget to put the long in the Motorola format (Highest byte first)

$$y = \frac{Y \cdot F}{F - Z} \cdot Zoom$$

©1995-1997 MetaCreations, All rights reserved.

IV.ii.How to build a .dta file

A .dta file is constructed from a Macintosh resource file. It is simply the MacOS resource fork of a resource file copied blankly in the data fork.



The resource file (called extension.rsrc here) can be created and edited directly using ResEdit (for the COMP, Cmpp, pMap, etc. resources) and using AdLib or ViewEdit for the “View” resources (ResEdit Templates are provided in the toolkit for these resources. See DreamSDK.Mac:Tools:RDD.TMPL). You can also define some of these resources in a MPW .r file that will be compiled with Rez, but this is only if you prefer to do so. Then you compile the resource file into a .dta file using the MPW tool Res2Data. If you do not have MPW, you can use the Res2DataApp application found in DreamSDK.Mac:Tools:.

The MPW syntax is simply:

```
Res2Data extension.rsrc extension.dta
```

©1995-1997 MetaCreations, All rights reserved.

IV.iii.Registering your Class ID with MetaCreation s

As explained in the section “How the whole thing works” in the Introduction chapter, it is vital that you make sure that your Class ID is unique before distributing your software outside, even as a Freeware or Shareware. The reason is that if someone else uses the same Class ID in his own extension, the 3D Shell will be confused when trying to instanciate and initialize your 3D Component, causing a failure or a crash.

Registering your Class ID is free. There are no obligations. After acceptation, your Class ID will be yours forever and will never be assigned to someone else.

- A Class ID is a set of 4 ASCII characters.
- Ranges of Class ID will **not** be registered by MetaCreations (like “all IDs from ‘AAAA’ to ‘AAAZ’”).
- Class IDs are case-sensitive.
- Class IDs with all lowercase letters are reserved to MetaCreations (like ‘abcd’). Combos are fine, though (like ‘Abcd’).
- Try avoiding special characters (ASCII code > 128), because these characters are different between MacOS and Windows.

How to register

Simply fill-in the form below, and send it to MetaCreations by Fax or e-mail (e-mail is preferred) to the following address. If you register by e-mail, make sure you include the word “registration” in your e-mail title to allow faster processing.

DreamSDK Component Registration / R&D Dept.
MetaCreations, Corp.
P.O. Box 66959
Scotts Valley, CA 95067-6959
U.S.A.

Fax: (1) 408-438-9673
E-Mail: tech.dreamsdk@metacreations.com

You will then receive a confirmation message from MetaCreations by fax or e-mail. If a conflict occurs, MetaCreations will offer an alternate ID.

©1995-1997 MetaCreations, All rights reserved.

Class ID Registration Form

Name: _____

Company: _____

Address: _____

_____

Phone: _____

Fax: _____

E-Mail: _____

Component Name	Family ID	Class ID

Please include a description of your Component(s) if you like (optional):

©1995-1997 MetaCreations, All rights reserved.

Appendix V - The Ray Dream Designer® File Format

This chapter contains general informations about the file format of Ray Dream Designer 3 and 4.
We will describe the Main document file structure and the shader file structure. It will be helpful, if you want to create programs which will create scene to be rendered with Ray Dream Designer. Or if you want to use shaders in other programs.

Before you start writing code that read or write Ray Dream Designer files, you should ask yourself if writing a 3D Import/Export Extension would not be a better option to your problem. Usually one needs to convert Ray Dream Designer files to/from another file format. It is often easier and more flexible to code an Importer or Exporter: this way you code only one file format, the one you know. Furthermore using the API is more easy and flexible than reading or writing Ray Dream Designer files. And if you already have some code for the other file format, then it is really a breeze.

The RDD file format contains a lot of information that are not necessary relevant to your case. Because of the file structure it is however very easy to access only the information you wantand skipping anything else.

The files used in Ray Dream Designer are mostly text except for the preview and texture map

©1995-1997 MetaCreations, All rights reserved.

V.i.Token Structur e

The files used in Ray Dream Designer are mostly text except for the preview and texture map data which are binary. They can be edited in all text editor.

The data in the file are organized like this:

keyword value [# comments]

(Terms in *italic* are definition names to be replaced by actual values).

They keyword is a 1 to 4 letters token.
The keyword gives the type and the use of the following value. The value can be a string, a boolean, an integer, a fixed, a Rectangle, a 2D or 3D point or vector, an array, etc. These are **Base Types**.
The value can also be a **list** of other data. In this case we use brackets '{' and '}' to give the limits of the block of data. Inside the block, we find another list of *keyword value* pairs. Therefore, blocks can be nested.

Characters separating tokens are the space, tab, end of line, {, }, # and ". Note that values made of several items (like the 3 coordinates of a 3D points) have their items separated by commas **without any spaces:**

```
p3    1.0,2.0,3.0    # Note: no spaces between the 3 coordinates !
```

Comments start at the # character until the next end of line. Therefore they can contain any characters, including separating characters.

```
vs "1.0"       # This is a version number string
```

String are enclosed by double quotes ". They can contain separating characters. To include a double quote in a string, use the backslash \ as an escape character. To do a backslash, put two of them: \\.

```
"Hello World "
"This is \"legal\" too !"
```

The end of line character(s) can follow the Macintosh (0x0D), DOS/Windows (0x0D/0x0A) or Unix (0x0A) convention. Ray Dream Designer can read any of these files on all platforms.

The great convenience of this grammar is that one can always skip the value of a keyword if needed. If the value is a base type, then just skip the token. If it is a list of data, then skip all tokens, counting opening and closing brackets until they balance. This is a great savior for ensuring backward and forward file compatibility, and for quickly browing through a file until you get to the data you are interested in. You do not need to understand 100% of the file if you are interested in only 1%.

To help the reading of a file in a text editor, indentation can be used. For example:

©1995-1997 MetaCreations, All rights reserved.

```
3DC {
 vs "1.0"
 mapf "bin"
 sf "Ray Dream Designer(TM) 3"
}
```

Sometimes, data is optional. This means that Ray Dream Designer will still be able to read the file even if the data is not provided. A default value will be used instead.

©1995-1997 MetaCreations, All rights reserved.

V.ii.Base type s

There are different base types for values:

Type	Meaning
string31	a string limited to 31 characters
string255	a string limited to 255 characters
boolean	0 (False) or 1 (True)
integer	a 32 bit signed integer (from -2 147 483 648 to 2 147 483 647)
float	a 32 bit signed fixed point value (the maximum is 32767.999984, with precision 1.5E-5)
rect	array of 4 float values defining a 2D rectangle (two opposite corners) separated by a “,”
point2D	coordinates of a 2D point: array of 2 float values separated by a “,”
point3D	coordinates of a 3D point: 3array of 3 float values separated with a “,”
FARRAYxx	array of float values separated only with “,”. xx is the number of elements.
IARRAYxx	same as FARRAYxx but for integer.
IDType	a 32 bit unsigned integer describing a 4-letters ID value. For example, ‘ABCD’ would give 1094861636 (the decimal value of 0x41424344).
Time	Like IARRAY2: two integer values. The first value is the time in ‘ticks’, and the second one is the number of ‘ticks’ per seconds. Divide the first values by the second one to get seconds.
Hexa	Binary chunk of data in ascii hexadecimal. If there are more than 64 byte to write, the remaining part will be written in as many ‘MORE’ tokens as necessary.

©1995-1997 MetaCreations, All rights reserved.

V.iii.General Structur e

a default Ray Dream Designer file looks like:

```
Header
3DC {
 vs "1.0"
..etc..
}

Workspace Info
wkmg {
}
prvw {
 w 64
..etc..
}
Scene Itself
sc {
 nm "Tower Base"
 ...etc...
Object List
 ob {
  ex {
   }
...etc...
  }
Scene tree
 tr {
  }
 }
```

©1995-1997 MetaCreations, All rights reserved.

V.iv.Ray Dream Designer Extensions Support in the Forma t

Ray Dream Designer support extensions written by third party developers in more than 20 areas of the application. The format therefore provide support for saving a reading information specific to each of these extensions.

TBD

©1995-1997 MetaCreations, All rights reserved.

V.v.3D Document fil e

Top Level Section s

Keyword	Meaning
3DC	Header, this section of the file contains all the information about rendering (i.e.: the preview and the artwork settings, the ambient light, the version of the file...)
wkmg	Work space, This section contains all the information of the perspective window (size, location, active camera...)
back	Background, this section is optional in a file, it gives the background attribute
atmo	Atmosphere, this section is optional, when it is present, it gives the atmospheric component used in this file.
rart	Artwork renderer, this section is optional, it gives the attribute of the specific artwork renderer.
rprv	Preview renderer, this section is optional, it gives the attribute of the specific preview renderer.
prvw	Preview, this section gives a preview of the scene and is optional.
sc	Scene, this is the main section of the file.

Header Sectio n

The header section is organized as follow:

```
3DC {
 asci boolean
 vs string31
 sf string31
 prvw boolean
 comm string255
 RSet {...
 }
 CamL {...
 }
}
```

©1995-1997 MetaCreations, All rights reserved.

Keyword	Type	Meaning
asci	boolean	gives the ASCII character set used (0 is Macintosh and 1 is ANSI Windows) the default value is 0. Optional.
vs	string31	the string in quotes following this keyword gives the file format version number. Default value is "1.0". Optional.
sf	string31	this string contains the name of the software that created this file. Optional.
prvw	boolean	if the file contains a preview section of the scene the boolean value will be 1 otherwise it is 0. Optional.
comm	string255	The string following this keyword is a comment of the file, it can be 255 characters long.
Rset	section	RenderSettings Section (see below)
CamL	section	Camera list. Each name is a string in quotes after the keyword CamN.

Render Settings Section

This section contains the current render settings.

```
RSet {
 pltf IDType
 name string
 comt string
 widt float
 heig float
 dpi  float
 dept integer
 C    boolean
 A    boolean
 D    boolean
 I    boolean
 N    boolean
 P    boolean
 S    boolean
 came string31
 OSTy IDType
 hour integer
 min  integer
 rbat { #private data of the chosen renderer
  ...
  }
 oopt integer
 Ofil string
 movi boolean
 tchf boolean
 anim { # Animation settings
  begT Time
  endT Time
  fSec integer
  MOVW IDType
  WOPT {
   SIZE integer
   DATA hexa
```

©1995-1997 MetaCreations, All rights reserved.

```
  }
 }
}
```

Keyword	Type	Meaning
pltf	IDType	Platform on which this setting was defined: 'MACr' for MacOS, 'WINr' for Windows.
name	string31	Name of this render settings set
comt	string	Comments
widt	float	Width of the rendered image in points (not pixels)
heig	float	Height of the rendered image in points (not pixels)
dpi	float	Resolution in dpi
kpro	boolean	Keep proportions flag
dept	integer	Channel depth (8 bits per pixel by default)
C, A, D, I, N, P, S	boolean	All the different channel to calculate. C is the color channel (Red, Green, Blue). A is the Mask channel. D is the distance. I is the object index channel. N is the normal coordinates channel. P is the 3D location and S is the surface location. A 1 after
came	string31	Name of the camera used to render.
OSTy	IDType	MacOS file type of the file to render.
hour	integer	Number of rendering hours spent so far.
min	integer	Number of rendering minutes spent so far.
rbat		This keyword starts a new subsection containing all the private data of the chosen renderer. The next token will be the ClassID of the renderer, followed by the list of its parameters.
oopt	integer	Output Option. Save with default name = 1, Save with specific name = 2
Ofil	string	File name
movi	boolean	Render a movie instead of a still image
tchf	boolean	true if the rendering format should be set at each time change. 1 by default.
anim		Animation settings
begT	Time	Start time of the animation
endT	IArray2	End time of the animation
fSec	integer	Number of frames per second
MOVW	IDType	ID of the type of file to render the animation to.
WOPT		2D Output filter private options
SIZE	integer	Size of the private options in bytes that follows
DATA	Hexa	Bytes dump of the private options in hexa

Work Space Section (optional)

The Work Space Section describes the different opened windows of the documents

```
wkmg {
 wlst {# Workspace list
  work {# 1 workspace
   name string31
   wind { # Window info list
    winf { # 1 window info
```

©1995-1997 MetaCreations, All rights reserved.

```
      type OSType
      locH integer
      locV integer
      sizH integer
      sizV integer
      flot boolean
      stat boolean
      ???
      }
     ...
    }
   }
  ...
 }
}
```

Background Section (optional)

The Background section is organized as follow:

```
back {
 (type of background) {
  background's parameters
 }
}
```

There are initially 3 types of background: color, bitmap, and bi-gradient.

Color background:

the color background is very simple:

```
colo {
 colo red,green,blue
}
```

The color is given with the 3 parameters red, green, and blue separated with a ',' without space. Each value is a fixed value between 0.0 and 1.0.

Bitmap background:

The picture used for this background is entirely in the file. So you do not have to keep the original file with the 3D file.
The bitmap background is certainly the most difficult to understand. Its structure is:

```
bmap {
 bkmp {
  w width          # Width in pixel unit
  h height         # Height in pixel unit
  dpth (integer)   # Color Depth, 8 correspond to 256 colors
  ctab $colortablelength
      Binary color table data$
```

©1995-1997 MetaCreations, All rights reserved.

```
  # The length of the color table is written in hexadecimal and the
      binary color table data are written just after, without any separa-
      tion.
  tils x,y
  tdta {
   tile $length-tile data
  }
  tile (boolean)
  seam (boolean)
  cnth number of Horizontal tile
  cntv number of Vertical tile
  flph (boolean)
  flpv (boolean)
  rotc (boolean)
  intp (boolean)
 }
}
```

bi-gradient background:

the bi-gradient is not very more complicate than then color one:

```
bigr {
 tsco red,green,blue  #Top Start COlor
 teco red,green,blue  #Top End COlor
 bsco red,green,blue  #Bottom Start COlor
 beco red,green,blue  #Bottom End COlor
 limi angle           #limit between top and bottom in degrees
}
```

tsco, teco, bsco, and beco are like colo in the color background. Each color component is a fixed value between 0.0 and 1.0.

Atmospheric Section (optional)

The Atmospheric section is organized as follow:

```
atmo {
 (type of atmospheric effect) {
  atmospherics' parameters
 }
}
```

There are initially only one Atmospheric effect: fog.

The fog section is:

```
fog {
 colo red,green,blue
 dist DistanceFromTheCamera
 dvis DistanceOfVisibility
}
```

©1995-1997 MetaCreations, All rights reserved.

Keyword	Type	Meaning	Unit
colo	FARRAY3	Color of the fog	3 float between 0.0 and 1.0
dist	integer	distance to the fog from the camera	1 integer = inches x 16384
dvis	integer	distance of visibility in the fog	1 integer = inches x 16384

Renderer Section (optional)

This section include the Artwork renderer (rart {}) and the Preview renderer (rprv {}).

Those sections is useful only if you do not use the internal Ray Tracer. Initially, there is only one other renderer: Z-Buffer. Its parameters are organized as follow:

```
rart {
 jfsZ {
  COLR red,green,blue
  COLT red,green,blue
 }
}
```

Keyword	Type	Meaning
COLR	FARRAY3	Reflected color
COLT	FARRAY3	Transparent color

Preview Section (optional)

You can have a preview of the scene in this section. The structure of this section is:

```
prvw {
 w width
 h height
 dpth depth
 ctab $length+BinaryColorTableData$
 tils width,height
 tdta {
  tile $length+BinaryBitmapRLEData$
  ...
 }
}
```

The width and the height of the preview is given in pixels.

The depth is the number of bits per pixel.

The ctab gives the color table. A color table is a binary bloc.
A binary bloc is between two ‘$’, the first eight characters give the length of the bloc in hexadecimal.
Each color in the color table is made of 4 bytes (red, green, blue, and zero for word alignment).

©1995-1997 MetaCreations, All rights reserved.

The tdta sub section contains all the bitmap data. The preview can be made of several binary blocs. Each bloc begin with the keyword: ‘tile’.
A bloc begins end stops with a ‘$’. The first eight characters are the length of the bloc in the hexadecimal format. Then the bitmap data follow, these data are in the RLE format.

TBD

Scene Section

This is the most critical section of the file. This section contains all the information of the scene: the objects list section, the hierarchy of the scene, plus some other informations.

```
sc {
 nm string31
 nxtL integer  # NextLightNumber
 nxtC integer  # NextCameraNumber
 nxtG integer  # NextGroupNumber
 nxtO integer  # NextObjectNumber
 ob {
  Object list
 }
 tr {
   Hierarchy
 }
}
```

Keyword	Type	Meaning
nm	string31	Name of the scene
nxtL	integer	Number to use next time a Light Source is created
nxtC	integer	Number to use next time a Camera is created
nxtG	integer	Number to use next time a Group is created
nxtO	integer	Number to use next time an Object is created
ob	section	List of objects in the scene (See Objects list)
tr	section	Hierarchy of the scene (See Scene Hierarchy)
TABL	section	Counted Objects Tables Section
TMgr	section	Time Manager Section

Objects List Section

```
ob {
xxxx {
   }
xxxx {
   }
…etc…
}
```

The different class of objects are extended by adding 3D primitives extensions. Here is a list of the main object classes shipping by default with Ray Dream Designer:

©1995-1997 MetaCreations, All rights reserved.

Keyword	Meaning
o2	RDD 2.x object
tx	3D Text
ex	Extrusion
di	Flat Surface
im	Polygon List
pa	Patch Array
po	Polygon Array
sc	Enclosed scene (“Complex object”)
cube	Cube
sp	Sphere
co	Cone
cyli	Cylinder

Because there can be primitives defined by future extensions, this list cannot be complete.

Object Section

Object have common tokens that are:

Keyword	Type	Meaning
nm	string31	Name of the object
fidl	integer	object fidelity (default is 1)
sh	section	object shading (see object shading section)

Each class of objects then has a set of specific tokens. These are described in the following sections, at least for the predefined 3D primitives extensions shipping with Ray Dream Designer.

TBD: cube{prim{cube{aber

o2 RDD 2.x object

TBD

tx 3D Text

The specific tokens that can be found in text object are:

ex Extrusion

TBD

di Flat Surface

TBD

im Polygon List

TBD

©1995-1997 MetaCreations, All rights reserved.

pa Patch Array

TBD

po Polygon Array

TBD

sc Enclosed scene (“Complex object”)

TBD

cube Cube

No specific tokens.

sp Sphere

No specific tokens.

co Cone

No specific tokens.

cyli Cylinder

No specific tokens.

Scene Hierarchy Section

The scene heiarchys contains the following kinsd of tree elements

Keyword	Meaning
gr	Group
in	3D Object Instance
ca	Camera
li	Lightsource

The top of the hierarchy is always group called “Universe”.

Hierarchy Element Section

the keywords common to all hierarchy elements are:

Keyword	Type	Meaning
nm	string31	Name of the element of the Hierarchy
ma	FARRAY16	3D transformation factors (see 3D Transform)
hp	point3D	Hot point
so	section	List of sub hierarchy elements

©1995-1997 MetaCreations, All rights reserved.

The 3D Transformation factors are organized as follow:

ix,jx,kx,tx,iy,jy,ky,ty,iz,jz,kz,tz,sx,sy,sz,s

Each factors is used in the matrix formula:

TBD: Local=s.(Mijk)*(sx.Globalx...)+t

Group Elements

a group is declared as follow:

```
gr {
  Treefields
  boxM (point3D)
  boxm (point3D)
  open (boolean)
}
```

Keyword	Type	Meaning
boxM	point3D	Maximum Values of the Boundary Box in the local coordinate system.
boxm	point3D	Minimum Values of the Boundary Box in the local coordinates system.
open	boolean	Is-it an open group ?

Camera Elements

a camera is declared as follow:

```
ca {
  TreeFields
  proj {
    ExtCameraComponentAttribute {
      Component Fields
    }
  }
  cc (boolean)
}
```

The “cc” parameters tells you if this is the current camera (the active camera)

The proj subsection contains the type of camera and its parameters. In the standard version of Ray Dream Designer, there is only two types of camera: conic and isometric.
Each camera have its own parameters:

conic camera

```
coni {
  foca (integer)
  zoom (integer)
}
```

©1995-1997 MetaCreations, All rights reserved.

isometric camer a

```
???? {
  ????...
}
```

Keyword	Type	Meaning	Unit
foca	integer	!!!focal lentgth of the camera!!!	
zoom	integer	zoom factor of the camera	no unit

Lightsources Elements

A light source is defined like this:

```
li {
  ExtLightsourceComponentAttribute {
    Component Fields

  }
}
```

Instance Elements

an instance is very simple, because all the declarations of the object is in the objects list. Its section is like this:

```
in {
  TreeFields
  ob (string31)
}
```

The keyword “ob” gives the reference to the object in the objects list.

If the name (“nm”) is omitted in the ***TreeFields***, you have to use the name given by “ob” as the name of the object in the hierarchy.

Shading Section

```
0
    elms {
     sprm {
```

Keyword	Type	Meaning
mapk	integer	mapping mode 0 for parametric mapping, 1 for !!! TOFINISH !!!
elms	section	Paint Shape list Section

©1995-1997 MetaCreations, All rights reserved.

Paint Shape List Section

```
elms {
xxxx {
   }
xxxx {
   }
…etc…
}
```

the Paint shapes conbtained in the paint shape list can be :

Keyword	Meaning
sprm	shading primer
srct	rectangular paint shape
sovl	oval paint shape
sply	oval paint shape
srgn	raster paint shape

Paint Shape Section

All paint shape classes have common tokens that are:

Keyword	Type	Meaning
uvsp	integer	identifier of uv space in which paint-shape is defined (default is 0)
aexp	boolean	auto-expand flag (default is 0 for all paint shape except primer shape)
bbox	FARRAY4	paint shape visible bounding box in uv space.
mbox	FARRAY4	paint shape shading bounding box in uv space.
shdr	section	shader assigned to the paint shape
Ashe	section	animation information

Notes: The visible Bounding Box and the Shading Bounding Box are very often equal but can be different to allow for cropping of for example a texture map.

TBD: figure

Each class of paint shape then has a set of specific tokens. These are described in the following sections

sprm,srct,sovl primer paint shape, rectangular paint shape,oval paint shape

no specific token

sply polygonal paint shape

TBD

©1995-1997 MetaCreations, All rights reserved.

`srgn` raster paint shape

TBD

Shader Section

Raster Image Section

this section can be found in a lot of places. These include scene preview, texture map shaders, texture map backgrounds, texture map gels, raster paint shapes

Animation Section

This section contains the animation information for a given element.

Counted Objects Tables Section

Time Manager Section

©1995-1997 MetaCreations, All rights reserved.

V.i.Cookbook: Reading a Ray Dream Designer file .

The format is safe in terms of skipping unknown data.

©1995-1997 MetaCreations, All rights reserved.

V.ii.Cookbook: Writing the Minimum information in a Ray Dream Designer fil e

Sample File

This a minimum file containing a sphere no shading, no camera, no lights:

```
3DC {
 asci 1
 vs "1.0"
}
sc {
 ob {
  sp {
   nm "Sphere"
   }
  }
 tr {
  gr {
   nm "Universe"
   so {
    inpr {
     ob "Sphere"
     }
    }
   }
  }
 }
```

©1995-1997 MetaCreations, All rights reserved.

©1995-1997 MetaCreations, All rights reserved.

Index

A

AddFacet() 171
AddImportProblem() 225
AddPatch() 182
AddShadingShape()
  I3DShInstance 137
  I3DShObject 152
angle slider 316
Append() 241
Apply() 89

B

BeginCmpnd() 193
BeginFlatSurface() 192
BeginGroup() 192
BeginMove() 116
BeginPath() 193
BezierTo() 194
BitBlits() 206
BOX2D 265
BOX3D 267
button 317

C

CalcNormals()
  I3DShPolygonArray 180
  I3DShPolygonList 171
CalcUV()
  I3DShPatchArray 189
  I3DShPolygonArray 179
Calloc() 218
CenterElementOnHotPoint() 114
CenterHotPointOnElement() 114
Changed3DObject() 159
check box 317
CheckRegistration() 221
class ID
  registering 323
Clip3D()
  I3DExCamera 32
  I3DShCamera 132
ClipLine3D() 33
Clone()
  I3DExtension 23
  I3DShTreeElement 115
  IEnumChunk 238
Cmpp resource 319
CoCreateInstance() 219
color button 318
COLOR3D 275
COM 297
COMP resource 308
compilers 302
CountFacets() 171
CountPatches() 183
CreateKeyFrame() 222
CreateRay()
  I3DExCamera 30
  I3DShCamera 131
CreateRenderingCameraIfNone() 164
CreateTreeRootIfNone() 160
CreateTweener() 223
creator 304

D

data file 322
DeformFacets() 82
DeformPatches() 83
DeformPoint() 82
Dehydrate() 143
DependsOnAppliedExtent() 73
DirectionFilter()
  I3DExAtmosphericShader 56
  I3DShEnvironment 227
DoAlert() 224
DoColor() 93
DoExport() 69
DoImport() 64
DoIt() 96
DoOpenClose() 121
DoRotation() 94
DoShade()
  I3DExShader 74

©1995-1997 MetaCreations. All rights reserved.

I3DShInstance 138
DoTween() 92
DoTweening() 91
double slider 316
DoValue() 93
DoVector2() 93
DoVector3() 93
Draw()
I3DExTweener 92
I3DShInstance 142
DrawLA() 142
DrawLine() 251
DrawOffscreen() 254
DrawString() 253
dynamic linking 297

E

EndCmpnd() 193
EndFlatSurface() 192
EndGroup() 192
EndMove() 116
EndPath() 194
EnumChunks() 197
EnumFacets()
I3DExGeometricPrimitive 37
I3DShPrimitive 154
EnumFacetsWithConversion()
I3DShInstance 144
I3DShPrimitive 155
EnumPatches()
I3DExGeometricPrimitive 36
I3DShPrimitive 153
Eof() 246
Erase()
IShRasterBufferGetPut 202
IShRasterOffscreen 207
EraseRect() 252
ExtensionDataChanged() 26

F

FACET3D 277
Fail() 246
failure handling 296
Failure() 217
family ID 313
FillRect() 252
Filter() 78
fixed number entry box 315
ForEachShadowBuffer()
I3DExLightsource 51
I3DShLightsource 126
FrameRect() 252
Free() 218
FreeUVShMap() 142

G

geometry data structures 277
Gestalt() 223
Get()
IShFileStream 242
IShRasterOffscreen 208
IShRsaterBufferGetPut 204
Get32()
IShRasterBufferGetPut 204
IShRasterOffscreen 210
Get3DObject() 136
Get3DObjectByIndex() 158
Get3DObjectByName() 158
Get3DObjectsCount() 158
GetAllInfo() 144
GetAmbientLight() 166
GetAPINumber() 220
GetBackdropColor()
I3DExBackdrop 59
I3DShEnvironment 228
GetBackgroundColor() 57
GetBB() 143
GetBBox()
I3DExGeometricPrimitive 37
I3DShTreeElement 115
GetBoxes() 143
GetBufferNeeds() 77
GetCameraByIndex() 162
GetCamerasCount() 162
GetChunkData() 198
GetChunkInfo() 196
GetChunkRect() 198
GetClassSignature()
I3DShExtension 232
I3DShInstance 147
GetColor()

©1995-1997 MetaCreations. All rights reserved.

I3DExAmbientLight 61
I3DExLightsource 49
I3DExShader 75
I3DShLightsource 124
I3DShMap 235
GetCOMMap() 219
GetComponentClassName() 233
GetDeformed3DObject() 144
GetDepth()
IShRasterBufferGetPut 201
IShRasterOffscreen 206
GetDirection()
I3DExLightsource 48
I3DShLightsource 124
GetDouble() 243
GetEffectsDlogScene() 222
GetEnvironmentColor() 228
GetExtensionDataBuffer() 25
GetExtensionDataMap() 25
GetFileSize() 247
GetFirst() 109
GetFlatness() 142
GetFreedomDerivative() 118
GetFreedomRange()
I3DExMotionLink 86
I3DShTreeElement 117
GetFreedomTransform() 117
GetFullName() 119
GetFullPathName() 165
GetGelValues() 53
GetGlobalToCameraTransform() 130
GetGlobalTransform() 112
GetGlobalTransform8() 120
GetGTopHandler() 217
GetHotPoint() 113
GetIndFacets() 172
GetIndString() 220
GetInfo() 256
GetInstanceByIndex() 161
GetInstanceKind() 147
GetInstancesCount() 161
GetIRCache() 142
GetLast() 110
GetLeft() 110
GetLightGlobalTransform() 123
GetLightParameter()
I3DExLightsource 52
I3DShLightsource 126
GetLightsourceByIndex() 162
GetLightsourcesCount() 161
GetLine() 243
GetLong() 243
GetMainColors() 138
GetMappingDirection() 146
GetMappingKind() 145
GetName()
I3DShObject 149
I3DShTreeElement 109
GetNbrFreedom()
I3DExMotionLink 85
I3DShTreeElement 117
GetNewTrace() 51
GetParameter() 232
GetParameters() 196
GetParamsBuffer() 231
GetPatch() 187
GetPreferredOutput() 73
GetPreviewSize() 70
GetPrimitiveID()
I3DExCamera 32
I3DExLightsource 51
GetProductionFormat() 163
GetProductionFormatBBox() 165
GetProductionFormatBBox7() 166
GetRayHitDetails() 41
GetRegistrationName() 222
GetRegistrationOrganization() 222
GetRenderingCamera() 164
GetRenderingCamera7() 167
GetRenderUVBox() 133
GetResID()
I3DExDataExchanger 26
I3DShExtension 231
GetRight() 110
GetScene()
I3DShObject 152
I3DShTreeElement 109
GetSerialNumber() 215
GetSetJmpAddress() 217
GetShadingBBox() 148
GetShadingFlags()
I3DExShader 73
I3DShInstance 137
GetShadingLevel() 138
GetShellVersion() 220
GetString() 242

©1995-1997 MetaCreations. All rights reserved.

GetTime()
I3DShMap 236
I3DShScene 165
GetTop() 110
GetTransform()
I3DExMotionLink 87
I3DShTreeElement 112
GetTransform8() 119
GetTransformPartialDerivate() 87
GetTreeElementByName() 161
GetTreeElementFromFullName() 166
GetTreeRoot() 160
GetURL() 147
GetUVExportShMap() 141
GetUVRasterShMap() 139
GetUVSpace()
I3DExGeometricPrimitive 38
I3DShInstance 137
I3DShObject 151
I3DShPatchList 183
I3DShPolygonList 173
GetUVSpaceCount()
I3DExGeometricPrimitive 38
I3DShInstance 136
I3DShObject 151
GetValue() 74
GetVector() 76
GetVertex() 177
GetViewID() 231
GetVolumeAttenuation() 45
GetVolumeDensity() 44
GetXYZScaling() 116
GoDown()
IShRasterBufferGetPut 203
IShRasterOffscreen 208
GoLeft() 203
IShRasterOffscreen 207
Good() 246
GoRight()
IShRasterBufferGetPut 202
IShRasterOffscreen 207
GoUp()
IShRasterBufferGetPut 203
IShRasterOffscreen 208
GUID resource 308

H

HandleEvent() 26
HitInfo 292

I

I3DExAmbientLight interface 61
I3DExAtmosphericShader interface 55
I3DExBackdrop interface 59
I3DExBackground interface 57
I3DExCamera interface 29
I3DExDataExchanger interface 25
I3DExDeformer interface 81
I3DExExportFilter interface 67
I3DExGeometricPrimitive interface 35
I3DExImportFilter interface 63
I3DExLightsource interface 47
I3DExLightsourceGel interface 53
I3DExMotionLink interface 85
I3DExPostRenderer interface 77
I3DExSceneOperation interface 95
I3DExSceneOperation8 interface 97, 101
I3DExShader interface 71
I3DExtension interface 23
I3DExTreeBehavior interface 89
I3DExTweener interface 91
I3DExVolumePrimitive interface 43
I3DShCamera interface 129
I3DShEnvironment interface 227
I3DShExtension interface 231
I3DShFlatSurface interface 191
I3DShGroup interface 121
I3DShInstance interface 135
I3DShLightsource interface 123
I3DShMap interface 235
I3DShObject interface 149
I3DShPatchArray interface 185
I3DShPatchList interface 181
I3DShPolygonArray interface 175
I3DShPolygonList interface 169
I3DShPrimitive interface 153
I3DShScene interface 157
I3DShTreeElement interface 107
IEnumChunk interface 237
IncrementFreedomValue()
I3DExMotionLink 86

©1995-1997 MetaCreations. All rights reserved.

I3DShTreeElement 118
Init()
I3DShPatchArray 186
I3DShPatchList 181
I3DShPolygonArray 176
I3DShPolygonList 170
IShRasterBuffer 195
InitFileStream() 239
InitOffscreen() 206
Insert3DObject() 159
InsertFirst() 111
InsertLast() 111
InsertLeft() 111
InsertRight() 111
Invalidate() 235
InvertRect() 252
IsEqual() 232
IsEqualTo() 72
IShFileStream interface 239
IShGraphicDevice interface 249
IShIterator interface 255
IShLineArt interface 211
IShRasterBuffer interface 195
IShRasterBufferGetPut interface 201
IShRasterOffscreen interface 205
IShUtilities interface 213
IsOpen() 122
IsPatchBased()
I3DExGeometricPrimitive 36
I3DShPrimitive 153
IsVisibleInPerspective() 50
IsVolumeInstance() 148

L

limiter popup 317
LineTo()
I3DShFlatSurface 193
IShGraphicDevice 250
Load() 209
LockChunk() 197

M

Malloc() 217
MappingParams 287
MATRIX3D 269
MeshGetU()
I3DShPatchArray 188
I3DShPolygonArray 178
MeshGetV()
I3DShPatchArray 188
I3DShPolygonArray 179
MeshSetU()
I3DShPatchArray 187
I3DShPolygonArray 177
MeshSetV()
I3DShPatchArray 187
I3DShPolygonArray 178
MoveTo() 249

N

NeededDiskSpace() 68
Next()
IEnumChunk 237
IShIterator 255
NUM3D 259
number entry box 314
number entry box with 2D units 315
number entry box with 3D units 315

O

OpenClose() 122

P

PATCH3D 278
platforms 302
PMAP
list of types 312
PMAP resource 310
pop up 318
PreAllocateFacets() 170
Prepare()
I3DExExportFilter 67
I3DExImportFilter 63
I3DExSceneOperation 95
Prepare8() 97
Preset() 130
primitive icon 320
ProgressDone() 217

©1995-1997 MetaCreations. All rights reserved.

ProgressIncrement() 216
ProgressNewValue() 216
ProgressStart() 215
Project3DTo2D()
I3DExCamera 31
I3DShCamera 131
Put()
IShFileStream 244
IShRasterBufferGetPut 203
IShRasterOffscreen 208
Put32()
IShRasterBufferGetPut 204
IShRasterOffscreen 210
PutBack() 246
PutDouble() 245
PutEOL() 245
PutLine() 244
PutLong() 245
PutString() 244

R

radio button 317
Random() 215
RandomSeed() 215
ray tracing data structures 289
Ray3D 289
RayAllHits() 41
RayHit()
I3DExGeometricPrimitive 40
I3DShInstance 143
RayHit3D 289
RayHitParameters 291
Read() 241
ReadKeyFrame() 222
Realloc() 218
RedoIt() 99
Remove3DObject() 159
RemoveFacet() 172
Reset()
IEnumChunk 238
IShIterator 256
ResetImportReport() 225
ResetPen() 251
ResetResource() 224
resource 307
Cmpp resource 319
COMP resource 308
GUID resource 308
PMAP resource 310
view 314

S

Save() 209
Seek() 240
SegmentFilter()
I3DExAtmosphericShader 55
I3DShEnvironment 227
Set3DObject() 136
SetBackground()
IShRasterbufferGetPut 202
IShRasterOffscreen 206
SetBBox()
I3DExDeformer 81
I3DShGroup 121
SetDeformedBoundingBox() 46
SetGlobalTransform()
I3DExTreeBehavior 90
I3DShTreeElement 113
SetGlobalTransform8() 120
SetHasUVSpace()
I3DShPatchList 182
I3DShPolygonList 172
SetHotPoint() 113
SetIRCache() 142
SetMacOSInfo() 247
SetMappingDirection() 146
SetMappingKind() 146
SetName()
I3DShObject 150
I3DShTreeElement 109
SetParameter() 232
SetPatch() 186
SetPen() 250
SetPosition()
IShRasterBufferGetPut 202
IShRasterOffscreen 207
SetPreview() 70
SetPrimerShader() 151
SetProductionFormat() 163
SetRenderingCamera() 164
SetScene() 108
SetShadingFromMaps() 145

©1995-1997 MetaCreations. All rights reserved.

SetSimpleShading() 150
SetTime() 236
SetTransform()
I3DExCamera 30
I3DExLightsource 48
I3DShTreeElement 112
SetTransform8() 119
SetTreeRoot() 160
SetupResource() 224
SetUVSpace()
I3DShPatchList 183
I3DShPolygonList 173
SetUVSpace0()
I3DShPatchList 182
I3DShPolygonList 170
SetVertex() 176
SetXYZScaling() 116
shading data structures 279
ShadingElem 286
ShadingFlags 280
ShadingIn 279
ShadingInOut 285
ShadingOut 282
ShadowEffect()
I3DExLightsource 50
I3DShLightsource 125
shell interface 300
ShellUtilitiesInit() 23
ShowImportReport() 225
Skip() 238
slider 315
angle slider 316
double slider 316
triple slider 316

T

Tell() 240
text entry box 314
TickCount() 214
top view 314
TRANSFORM3D 271
TranslateHotPoint() 114
TREETRANSFORM3D 273
triple slider 316

U

UndoIt() 98
Unlink() 110
UnlockChunk() 198
User Interface 306
resources 310
view 314
UV2XYZ() 39
UVShMaps 286
UVSpaceInfo 283

V

VECTOR2D 261
VECTOR3D 263
VERTEX3D 277
view 314

W

WantsOptionDialog()
I3DExExportFilter 68
I3DExImportFilter 64
WantsPreview() 69
WantsTopScene() 65
Write() 241
WriteKeyFrame() 223

Y

YieldProcesses() 214

©1995-1997 MetaCreations. All rights reserved.