Carrara SDK

Extension Architecture

Reference Manual

©2001-2002 Eovia
All rights reserved

Version 1.1
July 2002

Copyright ©2001-2002 Eovia, Corp. All rights reserved.

The software described in this manual is furnished under a licensing agreement contained in the license.txt file in the SDK folder. 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 Eovia, 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 Eovia, Corp. Eovia, Corp. assumes no responsibility or liability for any errors or inaccuracies that may appear in this user guide.

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

LICENSEE ACKNOWLEDGES THAT Eovia 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.

Eovia 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 Eovia HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

©2001-2003 Eovia. All rights reserved.

Chapter 1 - Table of Contents

Chapter 1 - Table of Contents 3

Chapter 2 - MCCommon IEx* Reference 17

2.1 IExDataExchanger Interface 17

IExDataExchanger::Clone(IExDataExchanger**, IMCUnknown*) 17
IExDataExchanger::IsEqual(IExDataExchanger*) 17
IExDataExchanger::GetParameter(IDType, void*) 18
IExDataExchanger::SetParameter(IDType, void*) 18
IExDataExchanger::GetExtensionDataBuffer() 18
IExDataExchanger::ExtensionDataChanged() 18
IExDataExchanger::HandleEvent(MessageID, IMFResponder*, void*) 18
IExDataExchanger::GetResID() 19

2.2 I3DExExportFilter Interface 21

I3DExExportFilter::Prepare(I3DShScene*, I3DShElement*) 21
I3DExExportFilter::WantsOptionDialog() 21
I3DExExportFilter::NeededDiskSpace(IMCFile*, I3DShScene*, I3DShTreeElement*, uint32*) 21
I3DExExportFilter::DoExport(IMCFile*,I3DShScene*, I3DShElement*) 22
I3DExExportFilter::WantsPreview() 22
I3DExExportFilter::GetPreviewSize(int16*, int16*) 22
I3DExExportFilter::SetPreview(TMCPixelBucket*) 22

2.3 I3DExImportFilter Interface 25

I3DExImportFilter::Prepare(IMCFile*, I3DShScene*, I3DShElement*) 25
I3DExImportFilter::WantsOptionDialog() 25
I3DExImportFilter::DoImport(IMCFile*,I3DShScene*, I3DShElement*) 25
I3DExImportFilter::WantsTopScene() 26

2.4 IExRegisterer Interface 27

IExRegisterer::GetComponentCount() 27
IExRegisterer::GetComponentData(uint32, APICOMPData&, APIPrivateData&, APIGUIDData&) . . . 27
IExRegisterer::UnregisterComponents() 27

Chapter 3 - Data Structures Reference. 29

TVECTOR2 29
TVECTOR3 31
TBBOX2D 31
TBBOX3D 32
TMATRIX33 32
TTRANSFORM3D 35
TTREETRANSFORM 36
TMCCOLORRGB 38
TMCGradient 40
ETreeDeformBBoxModes 41

©2001-2003 Eovia. All rights reserved.

Chapter 4 - Shading Data Structures Reference 43

ShadingIn 43
ShadingFlags 45
ShadingOut 47
UVSpaceInfo 48
ShadedArea 48

Chapter 5 - Geometry Data Structures Reference. 49

TVERTEX3D 49
TFACET3D 49
TPATCH3D 49
FacetMesh 50

Chapter 6 - Ray Tracing Data Structures Reference 51

Ray3D 51
RayHit3D 52
RayHitParameters 53
HitInfo 54

Chapter 7 - 3D IEx* Reference 57

7.1 I3DExAmbientLight Interface 59

I3DExAmbientLight::GetColor 59

7.2 I3DExAtmosphericShader Interface 61

I3DExAtmosphericShader::SegmentFilter 61
I3DExAtmosphericShader::DirectionFilter 61

7.3 I3DExBackground Interface 63

I3DExBackground::GetBackgroundColor 63

7.4 I3DExBackdrop Interface 65

I3DExBackdrop::GetBackdropColor 65

7.5 I3DExCamera Interface 67

I3DExCamera::SetTransform(constTTransform3D&) 67
I3DExCamera::CreateRay(const TVector2&, TVector3&, TVector3&) 68
I3DExCamera::CreateRay(const TVector2&, const TVector2&, Ray3D&) 68
I3DExCamera::IsProjectionAffine(void) 68
I3DExCamera::PreProject3DTo2D(const TVector3&, TVector4&) 69
I3DExCamera::Project3DTo2D(const TVector3&, TVector3&) 69
I3DExCamera::GetPrimitiveID() 69
I3DExCamera::GetStandardCameraInfo(TStandardCameraInfo&) 69
I3DExCamera::SetHalfWidth(real) 70

7.6 I3DExConstraint 73

I3DExConstraint::GetNbrFreedom() 73
I3DExConstraint::IncrementFreedomValue(int16 , real&) 73
I3DExConstraint::GetMotionLimit (int16, ELimitType&) 73
I3DExConstraint::GetFreedomRange(int16, real&, real&) 74
I3DExConstraint::GetTransform(TTransform3D&) 74
I3DExConstraint::GetTransformPartialDerivate(int16, TTransform3D&) 74
I3DExConstraint::GetTransformSecondPartialDerivative (int16, int16, TTransform3D&) 75
I3DExConstraint::SetTransform(TTransform3D&) 75
I3DExConstraint::IsTransformResetRequired() 75

©2001-2003 Eovia. All rights reserved.

I3DExConstraint::ResetConstraint() 75
7.7 I3DExDataComponent 77
I3DExDataComponent:: GetPostRenderList(TMCArray<IDType> &) 77
I3DExDataComponent::IsActive(I3DShTreeElement*) 77
7.8 I3DExExportFilter Interface 79
I3DExExportFilter::Prepare 79
I3DExExportFilter::WantsOptionDialog 79
I3DExExportFilter::NeededDiskSpace 79
I3DExExportFilter::DoExport 80
I3DExExportFilter::WantsPreview 80
I3DExExportFilter::GetPreviewSize 80
I3DExExportFilter::SetPreview 80
I3DExExportFilter::WantsTempFile 81
7.9 I3DExFinalRenderer Interface 85
I3DExFinalRenderer::SetTreeTop 85
I3DExFinalRenderer::SetCamera 85
I3DExFinalRenderer::SetEnvironment 86
I3DExFinalRenderer::SetAmbientLight 86
I3DExFinalRenderer::SetFieldRenderingData 86
I3DExFinalRenderer::GetRenderStatistics 86
I3DExFinalRenderer::PrepareDraw 87
I3DExFinalRenderer::FinishDraw 87
I3DExFinalRenderer::IsSMPAware 88
I3DExFinalRenderer::Abort 88
I3DExFinalRenderer::RenderVolumetrics 88
I3DExFinalRenderer::RenderBackGround 88
I3DExFinalRenderer::OptimizeTileSize 88
I3DExFinalRenderer::GetTileRenderer 89
7.10 I3DExImportFilter Interface 91
I3DExImportFilter::Prepare 91
I3DExImportFilter::WantsOptionDialog 91
I3DExImportFilter::DoImport 92
I3DExImportFilter::WantsTopScene 92
7.11 I3DExInvertibleTweener Interface 97
I3DExTweener::Inverse() 97
7.12 I3DExLightsource Interface 99
I3DExLightsource::SetTransform(TTransform3D*) 99
I3DExLightsource::GetDirection(TVector3&, TVector3&, real&) 99
I3DExLightsource::GetColor(TVector3&, TVector3&, real, TMCColorRGB&, real&) 100
I3DExLightsource::IsVisibleInPerspective() 100
I3DExLightsource::GetPrimitiveID() 100
I3DExLightsource::ForEachShadowBuffer(ForEachShadowBufferCallback, void*) 101
I3DExLightsource::GetLightParameter(int32, void*) 101
I3DExLightsource::GetLightInfo(boolean&, boolean&, real&, TTransform3D&) 101
I3DExLightsource::GetUVfromDirection(TVector2&, const TVector3&) 101
I3DExLightsource::GetNewTrace(int16*) 102
I3DExLightsource::GetStandardLight(TStandardLight&) 102
7.13 I3DExLightsourceGel Interface 103
I3DExLightsourceGel::GetGelValues(constTVector2&, TMCColorRGB&) 103
7.14 I3DExModifier Interface 105
I3DExModifier::Apply(I3DShTreeElement) 105
I3DExModifier::GetMaxTimeWithoutComputation (I3DShTreeElement*) 105
I3DExModifier::SetBoundingBox(const TBBox3D&) 105

©2001-2003 Eovia. All rights reserved.

I3DExModifier::DeformPoint(TVector3&, TVector3&) 106
I3DExModifier::DeformVertex3D(TVertex3D&, TVertex3D&) 106
I3DExModifier::DeformFacetMesh(real, FacetMesh*, FacetMesh**) 106
I3DExModifier::DeformBBox(const TBBox3D&, TBox3D&) 106
I3DExModifier::GetNeededLOD(const real, real&) 107
I3DExModifier::GetDeformerInfo(real, TDeformInfo&) 107
7.15 I3DExModule Interface 109
I3DExModule::Initialize(IMCUnknown*) 110
I3DExModule::Destroy() 110
I3DExModule::Hydrate() 110
I3DExModule::Dehydrate() 111
I3DExModule::Activate() 111
I3DExModule::Deactivate() 111
I3DExModule::Import(IMCUnknown*, void*) 111
I3DExModule::BuildMenuBar() 111
I3DExModule::GetPreferredRoom(IDType&) 112
I3DExModule::WantsToBeInCurrentRoom() 112
I3DExModule::SubModuleClosedMainWindow(I3DShModule*) 112
I3DExModule::SubModuleAboutToBeDestroyed(I3DShModule*) 112
I3DExModule:: SubModuleAboutToBeHydrated(I3DShModule*) 112
I3DExModule::AboutToCloseMainWindow() 113
I3DExModule::SavesInNativeFormat() 113
7.16 I3DExPostRenderer Interface 115
I3DExPostRenderer::GetBufferNeeds(uint32&, void*) 115
I3DExPostRenderer::PrepareDraw(IShRasterLayer*, const TMCRect&, IShChannel*, I3DShScene*, I3DShCamera*, TBBox2D&, TBBox2D&) 116
I3DExPostRenderer::DrawRect(const TMCRect&, const TChannelDataBucket*,TBBox2D&) 117
I3DExPostRenderer::FinishDraw() 117
I3DExPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean) 117
I3DExPostRenderer::EndRender() 117
7.17 I3DExGeometricPrimitive Interface 119
I3DExGeometricPrimitive::GetBoundingBox(TBox3D*) 119
I3DExGeometricPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean&) 120
I3DExGeometricPrimitive::GetNbrLOD(int16&) 120
I3DExGeometricPrimitive::GetLOD(int16, real&) 121
I3DExGeometricPrimitive::GetFacetMesh(uint32, FacetMesh**) 121
I3DExGeometricPrimitive::GetFacetMesh(real, FacetMesh**) 121
I3DExGeometricPrimitive::CanBeSplit() 121
I3DExGeometricPrimitive::SplitPrimitive(TMCCountedPtrArray<I3DExGeometricPrimitive>&, TMCArray<TTransform3d>&) 122
I3DExGeometricPrimitive::GetUVSpaceCount() 122
I3DExGeometricPrimitive::GetUVSpace(uint32, UVSpaceInfo*) 122
I3DExGeometricPrimitive::UV2XYZ(TVector2*, uint32, TVEctor3*, boolean*) 122
I3DExGeometricPrimitive::GetUVSpaceRDS5(uint32, UVSpaceInfoRDS5*) 123
I3DExGeometricPrimitive::AppendToRenderables(const TTransform3D&, TRenderableAndTfmArray&) 123
I3DExGeometricPrimitive::AutoSwitchToModeler() 124
I3DExGeometricPrimitive::IsBBoxPickable() 124
I3DExGeometricPrimitive::GetDefaultShader 124
I3DExGeometricPrimitive::ChangedData 124
7.18 I3DExReflectionFeature 125
I3DExReflectionFeature::GetReflectionNeeds(uint32 &) 125
I3DExReflectionFeature::CalcReflection(RenderFeatureIO&) 125
7.19 I3DExRefractionFeature 127
I3DExRefractionFeature::GetRefractionNeeds(uint32 &) 127

©2001-2003 Eovia. All rights reserved.

I3DExRefractionFeature::CalcRefraction(RenderFeatureIO&) 127
7.20 I3DExRendererBox Interface 129
7.21 I3DExShadowFeatureFeature 135
I3DExShadowFeature::BeginRenderFrame 135
I3DExShadowFeature::EndRenderFrame 135
I3DExShadowFeature::GetShadowNeedFlags 135
I3DExShadowFeature::CalcShadow 135
7.22 I3DExSceneCommand Interface 137
I3DExSceneCommand::GetMenuCallBack 137
I3DExSceneCommand::Init 137
I3DExSceneCommand::Prepare 138
I3DExSceneCommand::CanUndo 138
I3DExSceneCommand::Do 138
I3DExSceneCommand::Undo 138
I3DExSceneCommand::Redo 138
I3DExSceneCommand::PostPrepare 138
7.23 I3DExShader Interface 143
I3DExShader::IsEqualTo(I3DExShader*) 143
I3DExShader::GetShadingFlags(ShadingFlags&) 144
I3DExShader::GetImplementedOutput() 144
I3DExShader::DoShade(ShadingOut&, ShadingIn&) 145
I3DExShader::GetValue(real&, boolean, ShadingIn&) 145
I3DExShader::GetColor(TMCColorRGB&, boolean, ShadingIn&) 145
I3DExShader::GetVector(TVector3&, ShadingIn&) 146
I3DExShader::ConvertToPoly(TMCArray<TVector2>*) 146
I3DExShader::OutlineInPreviewMode() 146
I3DExShader::Contains(ShadingIn&, TVector2&) 146
7.24 I3dExSkinable Interface 147
I3dExSkinable::AttachSkeleton 147
I3dExSkinable::DettachAllJoints 147
I3dExSkinable::IsAttached 148
I3dExSkinable::GetJoints 148
I3dExSkinable::InvalidateSkining 148
I3dExSkinable::GetInstance 148
I3dExSkinable::GetInstanceGlobalReferenceTransform 148
I3dExSkinable::SetInstanceGlobalReferenceTransform 148
7.25 I3DExTileRenderer Interface 149
I3DExTileRenderer::Init 149
I3DExTileRenderer::GetNextSubTile 149
I3DExTileRenderer::RenderSubTile 149
II3DExTileRenderer::FinishRender 150
7.26 I3DExTweener Interface 155
I3DExTweener::SimpleTween(real&, int32, int32, int32) 155
I3DExTweener::Tween(I3DShKeyFrame*,int32, I3DShTweenerChainLink*) 155
7.27 I3DExTweenerEdition Interface 157
I3DExTweener::PrepareEdit(I3DShTweenerChainLink*) 157
7.28 I3DExVertexPrimitive Interface 159
I3dExVertexPrimitive::GetNbPolymeshes 159
I3dExVertexPrimitive::GetNbEdges 159
I3dExVertexPrimitive::GetNbPolygons 159
I3dExVertexPrimitive::GetPolymesh 160
I3dExVertexPrimitive::GetBoundingBox 160

©2001-2003 Eovia. All rights reserved.

I3dExVertexPrimitive::Pick 160
I3dExVertexPrimitive::DeleteSelectedEdges 160
7.29 I3DExVolumetricEffect Interface 161
I3DExVolumetricEffect::Initialise 161
I3DExVolumetricEffect::DirectionFilter 161
7.30 I3DExWireFrame Interface 163
I3DExWireFrame::TrackWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*, int16, const TRACKINFO&, const TRACKINFO&, const TRACKINFO&) 163
I3DExWireFrame::DataToWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*) 164
I3DExWireFrame::WireFrameToData(I3DShWireFrame*, int16, I3DShTreeElement*) 164
I3DExWireFrame::GetWireFrameBBox(I3DShTweenerChainLink*) 164
I3DExWireFrame::Activate() 165
I3DExWireFrame::Deactivate() 165
I3DExWireFrame::HandlesTool(int16) 165
7.31 IRaytracablePrimitive Interface 167
IRaytracablePrimitive::RayHit(boolean*, Ray3D*, RayHitParameters*, RayHit3D*) 167
IRaytracablePrimitive::GetRayHitDetails(RayHit3D*) 168
IRaytracablePrimitive::IsInfiniteBB(boolean&) 168
7.32 ISceneDocument Interface 169
ISceneDocument::GetScene(I3DShScene**) 169
ISceneDocument::GetSceneSelection(ISceneSelection**) 169
ISceneDocument::GetSceneSelectionChannel(IChangeChannel**) 169
ISceneDocument::GetTreeHierarchyChannel(IChangeChannel**) 170
ISceneDocument::GetDocumentWindowByID(IMFPart**, IDType) 170
ISceneDocument::GetModuleByID(I3DShModule**, IDType) 170
ISceneDocument::GetSceneRenderingModule(I3DRenderingModule**) 170
ISceneDocument::StopAnimPreview() 170
ISceneDocument::StartAnimPreview() 170
7.33 IVolumePrimitive Interface 173
IVolumePrimitive::GetVolumeDensity(const TVector3&, TMCColorRGB&, TMCColorRGB&) 173
IVolumePrimitive::GetVolumeAttenuation(const TVector3&, constTVector3&, TMCColorRGB&) . . 174

Chapter 8 - MCCommon ISh* Reference 175
8.1 IShAction Interface 177
IShAction::Do() 177
IShAction::Undo() 177
IShAction::Redo() 177
IShAction::WillCauseChange() 177
IShAction::CanUndo() 178
IShAction::GetPartToRedraw(IMFPart**, int32) 178
IShAction::GetName(TMCString&) 178
IShAction::NeedsToPrepareMenus(void) 178
8.2 IShActionManager Interface 179
IShActionManager::PostAction(IShAction*, int32, IMFResponder*) 179
IShActionManager::PostMouseAction(IShMouseAction*, int32, IMFResponder* IMFPart*, const TMCPoint&) 179
IShActionManager::IsOptionDown() 180
IShActionManager::IsShiftDown() 180
IShActionManager::IsCommandDown() 180
IShActionManager::IsControlDown() 180
IShActionManager::IsLongStaticClick() 180
IShActionManager::IsShortStaticClick() 180

©2001-2003 Eovia. All rights reserved.

8.3 IShComponent Interface 181

IShComponent::GetFamilySignature() 181
IShComponent::GetClasSignature() 181
IShComponent::GetClassName(TMCString&) 181
IShComponent::IsDefaultComponent() 181
IShComponent::InvalidatePreview() 182

8.4 IShComponentUtilities Interface 183

IShComponentUtilities::CoCreateInstance(const MCCLSID&, IMCUnknown*, uint32, const MCIID&, void**) 183
IShComponentUtilities::CreateComponent(IDType, IDType, IShComponent**) 184
IShComponentUtilities::CreateIUnknown(IDType, IDType, IMCUnknown*, IMCUnknown**) 184
IShComponentUtilities::CreateDefaultComponent(IDType, IShComponent**) 184
IShComponentUtilities::GetComponentClassInfoArrayByFamilyID(IDType,TMCArray<TComponentClassInfo>&) 185
IShComponentUtilities::WriteComponent(IShTokenStream*, IShParameterComponent*) 185
IShComponentUtilities::WriteComponent(IShTokenStream*, IDType, IShParameterComponent*) . . . 185
IShComponentUtilities::ReadComponent(IShTokenStream*, IShParameterComponent**) 185
IShComponentUtilities::ReadComponent(IShTokenStream*, IShParameterComponent**, IDType) . . . 186
IShComponentUtilities::ReadComponentData(TMCiostream*, IShParameterComponent*) 186
IShComponentUtilities::AskForParams(IShParameterComponent*) 186

8.5 IShParameterComponent Interface 187

IShParameterComponent::GetFamilySignature() 187
IShParameterComponent::GetClasSignature() 187
IShParameterComponent::GetClassName(TMCString&) 187
IShParameterComponent::GetParameter(IDType, void*) 188
IShParameterComponent::SetParameter(IDType, void*, boolean) 188
IShParameterComponent::GetParameterType(IDType, IDType&) 188
IShParameterComponent::GetParameterCount() 188
IShParameterComponent::GetParameterInfo(uint32, TPMapElement&) 188
IShParameterComponent::Inval(boolean, boolean) 189

8.6 IShUtilities Interface 191

IShUtilities::BeginProgress(TMCString&, IMCUnknown**, real32) 192
IShUtilities::EndProgress(IMCUnknown*) 192
IShUtilities::SetProgressPrompt(TMCString&, IMCUnknown*) 192
IShUtilities::IncrementProgress(real32, IMCUnknown*) 192
IShUtilities::SetProgressValue(real32, IMCUnknown*) 193
IShUtilities::SetProgressValue0to1(real32, IMCUnknown*) 193
IShUtilities::YieldProcesses(uint32) 193
IShUtilities::TickCount() 194
IShUtilities::Random() 194
IShUtilities::SetRandomSeed(uint32) 194
IShUtilities::RandomStep(int32) 194
IShUtilities::GetAPINumber() 194
IShUtilities::Gestalt(void*, uint32) 194
IShUtilities::DoAlert(EAlerts, TMCString&, TMCString&, TMCString&, TMCString&, int32, TMCString&, TMCString&, TMCString&) 195
IShUtilities::ErrorToString(MCCOMErr, TMCString&) 196
IShUtilities::BuildAlertMessage(MCCOMErr, TMCString&, TMCString&) 196
IShUtilities::GetResource(int32, int32, int32, void**) 196
IShUtilities::SetupResources(int32, int32, void**) 197
IShUtilities::RestoreResources(void*) 197
IShUtilities::BlitUIImage(IMCGraphicContext*, TMCRect&, TMCRect&, uint32) 197
IShUtilities::GetUIImageInfo(uint32, int32&, int32&, int32) 197
IShUtilities::GetUIImageBucket(uint32, TMCPixelBucket*) 198
IShUtilities::GetUIImage(uint32, TMCUIImage**) 198

©2001-2003 Eovia. All rights reserved.

ISh Utilities::CreateCompatiblePixelBucket(int32, int32, int32, TMCCompatiblePixelBucket**) 198
ISh Utilities::DestroyCompatiblePixelBucket(TMCCompatiblePixelBucket*) 199
ISh Utilities::GetAsciiTranslationTable() 199
ISh Utilities::GetOpenDocumentsCount() 199
ISh Utilities::GetOpenDocumentByIndex(IMFDocument**, index) 199
ISh Utilities::GetLastOpenDocPath(TMCString&) 199
ISh Utilities::GetLastSaveDocPath(TMCString&) 199
ISh Utilities::SetLastOpenDocPath(TMCString&) 200
ISh Utilities::SetLastSaveDocPath(TMCString&) 200
ISh Utilities::UseMMX() 200
ISh Utilities::IsPentiumProcessor() 200
ISh Utilities::IsPentiumIIProcessor() 200
ISh Utilities::IsPentiumIIIProcessor() 200
ISh Utilities::StringWidth(TMCTextStyle&, TMCString&) 200
ISh Utilities::GetTextMetrics(TMCTextStyle&, TMCTextMetrics&) 201
ISh Utilities::GetCreator() 201
ISh Utilities::SetCursor(int32) 201
ISh Utilities::GetFontList(IMCFontList**) 201
ISh Utilities::CheckForUserCancel() 201
ISh Utilities::GetCurrentModifiers() 201
ISh Utilities::CheckKeyDown(int16) 202

Chapter 9 - 3D ISh* Reference 203

9.1 I3DShAtmosphericShader Interface 205

I3DShAtmosphericShader::GetScene(I3DShScene**) 205

9.2 I3DShBackdrop Interface. 207

I3DShBackdrop::GetScene(I3DShScene**) 207

9.3 I3DShBackground Interface 209

I3DShBackground::GetScene(I3DShScene**) 209

9.4 I3DShCamera Interface 211

I3DShCamera::GetGlobalToCameraTransform(TTransford3d*) 211
I3DShCamera::Preset(uint32, const TVector3*, real) 211
I3DShCamera::Project3DTo2D(const TVector3*, TVector2*, real*) 212
I3DShCamera::CreateRay(const TVector2*, TVector3*, TVector3*) 212
I3DShCamera::GetRenderUVBox(TBBox2d*, uint32, uint32, real) 213
I3DShCamera::SetCameraComponent(int32) 213
I3DShCamera::SetCameraComponent(IShComponent*, ECloneMode) 213
I3DShCamera::GetCameraComponent(IShComponent**) 213
I3DShCamera::PreProject3DTo2D(const TVector3&, TVector4&) 214
I3DShCamera::IsProjectionAffine(void) 214
I3DShCamera::GetStandardCameraInfo(TStandardCameraInfo&) 214
I3DShCamera::SetCanonicalCameraType(ECanonicalCameraType) 215
I3DShCamera::GetCanonicalCameraType() 215
I3DShCamera::SetProductionOffset(const TVector2&) 216
I3DShCamera::GetProductionOffset() 216
I3DShCamera::SetProductionZoom(real) 216
I3DShCamera::GetProductionZoom() 216

9.5 I3DShConstraint 217

9.6 I3DShEnvironment Interface. 219

I3DShEnvironment::SegmentFilter(const TVector3& ,const TVector3&, TMCColorRGB&, boolean) . 219
I3DShEnvironment::SegmentFilter(const TVector3&,const TVector3&,TMCColorRGB&) 220
I3DShEnvironment::DirectionFilter(const TVector3 ,const TVector3&, TMCColorRGB&) 220

©2001-2003 Eovia. All rights reserved.

I3DShEnvironment::GetBackdropColor(const TVector2&,boolean&,const TBBox2D&, TMCColorRGB&) 220
I3DShEnvironment::GetEnvironmentColor(const TVector3&, boolean&, TMCColorRGB&) 221
I3DShEnvironment::HasAtmosphere() 221
I3DShEnvironment::HasBackdrop() 221
9.7 I3DShExternalPrimitive Interface. 223
I3DShExternalPrimitive::SetPrimitiveComponent(IShComponent*, ECloneMode) 223
I3DShExternalPrimitive::GetPrimitiveComponent(IShComponent**) 223
9.8 I3DShObject Interface 225
I3DShObject::SetName(const TMCString&) 225
I3DShObject::GetName(TMCString&) 225
I3DShObject::GetInstancesCount() 226
I3DShObject::GetInstanceArray(TMCCountedPtrArray<I3DShInstance>&) 226
I3DShObject::GetUVSpaceCount() 226
I3DShObject::GetUVSpace(uint32, UVSpaceInfo*) 226
I3DShObject::GetScene(I3DShScene**) 226
I3DShObject::IsInfiniteObject() 227
I3DShObject::GetBoundingBox(TBBox3D&) 227
I3DShObject::GetFacetBoundingBox(TBBox3D&) 227
I3DShObject::Clone(I3DShObject**, ECloneMode) const 227
I3DShObject::GetEditor() 227
I3DShObject::GetObjectID() 228
9.9 I3DShPatchArray 229
I3DShPatchArray::Init(uint32, uint32) 229
I3DShPatchArray::SetPatch(uint32, uint32, TPatch3D*) 229
I3DShPatchArray::GetPatch(uint32, uint32, TPatch3D*) 230
I3DShPatchArray::MeshSetU(uint32, real) 230
I3DShPatchArray::MeshSetV(uint32, real) 231
I3DShPatchArray::MeshGetU(uint32, real*) 231
I3DShPatchArray::MeshGetV(uint32, real*) 231
I3DShPatchArray::MeshCalcUV() 231
I3DShPatchArray::GetSize(uint32&, uint32&) 232
I3DShPatchArray::BuildFacetMesh(real, FacetMesh**) 232
9.10 I3DShPatchList 233
I3DShPatchList::Init(boolean) 233
I3DShPatchList::AddPatch(TPatch3D*) 233
I3DShPatchList::GetIndPatches(uint32) 234
I3DShPatchList::CountPatches() 234
I3DShPatchList::SetHasUVSpace(boolean) 234
I3DShPatchList::SetUVSpace(UVSpaceInfo*) 234
I3DShPatchList::GetUVSpace(uint32, UVSpaceInfo*) 235
I3DShPatchList::GetUVSpaceCount() 235
I3DShPatchList::BuildFacetMesh(real, FacetMesh**) 235
9.11 I3DShPolygonArray 237
I3DShPolygonArray::Init(uint32, uint32, boolean, boolean) 237
I3DShPolygonArray::MeshSetVertex(uint32, uint32, TVector3*, TVector3*) 238
I3DShPolygonArray::MeshGetVertex(uint32, uint32, TVector3*, TVector3*) 238
I3DShPolygonArray::MeshSetU(uint32, real) 239
I3DShPolygonArray::MeshSetV(uint32, real) 239
I3DShPolygonArray::MeshGetU(uint32, real*) 240
I3DShPolygonArray::MeshGetV(uint32, real*) 240
I3DShPolygonArray::MeshCalcUV() 240
I3DShPolygonArray::MeshCalcNormals(boolean) 241
I3DShPolygonArray::GetSize(uint32&, uint32&, boolean&, boolean&) 241
9.12 I3DShPolygonList Interface 243

©2001-2003 Eovia. All rights reserved.

I3DShPolygonList::Init(boolean, boolean) 243
I3DShPolygonList::PreAllocateFacets(uint32) 244
I3DShPolygonList::AddFacet(TFacet3D*) 244
I3DShPolygonList::CountFacets() 244
I3DShPolygonList::CalcNormals(real) 245
I3DShPolygonList::GetIndFacets(uint32) 245
I3DShPolygonList::SetHasUVSpace(boolean) 245
I3DShPolygonList::SetUVSpace(UVSpaceInfo*) 245
I3DShPolygonList::GetUVSpace(uint32, UVSpaceInfo*) 245
I3DShPolygonList::GetUVSpaceCount() 246

9.13 I3DShPostRenderer Interface 247

I3DShPostRenderer::SetScene(I3DShScene*) 247
I3DShPostRenderer::GetBufferNeeds(uint32&) 247
I3DShPostRenderer::PrepareDraw(IShRasterLayer*,const TMCRect&,IShChannel*,I3DShScene*,TBox2D&,TBox2D&)
248
I3DShPostRenderer::DrawRect(const TMCRect&, const TChannelDataBucket*, TBBox2D) 249
I3DShPostRenderer::FinishDraw() 249
I3DShPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean) 250
I3DShPostRenderer::EndRender() 250

9.14 I3DShMasterShader Interface 251

I3DShMasterShader::SetName(const TMCString&) 251
I3DShMasterShader::GetName(TMCString&) 251
I3DShMasterShader::SetShader(I3DShShader*, ECloneMode) 251
I3DShMasterShader::GetShader(I3DShShader**) 252
I3DShMasterShader::Clone(I3DShMasterShader**, ECloneMode) 252
I3DShMasterShader::GetScene(I3DShScene**) 252
I3DShMasterShader::IsDefaultMaster() 252

9.15 I3DShPrimitive Interface 253

I3DShPrimitive::IsPatchBased() 253
I3DShPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean) 253
I3DShPrimitive::ChangedData() 254
I3DShPrimitive::GetFMesh(real, FacetMesh**) 254
I3DShPrimitive::IsVolumic() 254
I3DShPrimitive::UV2XYZ(const TVector2&, int16, TVector3&, TVector3*) 254

9.16 I3DShPrimitiveComponent Interface 257

I3DShPrimitiveComponent::GetPrimitive(I3DShPrimitive**) 257
I3DShPrimitiveComponent::GetExternalPrimitive(IMCUnknown**) 257

9.17 I3DShShader Interface 259

I3DShShader::Clone(I3DShShader**, ECloneMode) const 259
I3DShShader::IsConstant() 259
I3DShShader::AssignsChannels() 260
I3DShShader::GetShadingFlags(ShadingFlags&) 260
I3DShShader::DoShade(ShadingOut&, ShadingIn&) 260
I3DShShader::GetValue(real&, boolean, ShadingIn&) 260
I3DShShader::GetColor(TMCColorRGB&, boolean, ShadingIn&) 261
I3DShShader::GetVector(TVector3&, ShadingIn&) 261
I3DShShader::GetBump(TVector3&, ShadingIn&) 261
I3DShShader::GetImplementedOutput() 262
I3DShShader::GetOriginalParametricTextureMap(uint32&,uint32&, IShTextureMap**) 262

9.18 I3DShTreeElement Interface 263

I3DShTreeElement::SetScene(I3DShScene*) 265
I3DShTreeElement::GetScene(I3DShScene**) 265
I3DShTreeElement::GetSceneNoAddRef() 265

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::SetName(const TMCString&) 265
I3DShTreeElement::GetName(TMCString&) 265
I3DShTreeElement::GetFirst(I3DShTreeElement**) 265
I3DShTreeElement::GetLast(I3DShTreeElement**) 266
I3DShTreeElement::GetLeft(I3DShTreeElement**) 266
I3DShTreeElement::GetRight(I3DShTreeElement**) 266
I3DShTreeElement::GetTop(I3DShTreeElement**) 266
I3DShTreeElement::Unlink(I3DShTreeElement**, RelinkTreeElementInfo*) 266
I3DShTreeElement::Relink(RelinkTreeElementInfo*) 266
I3DShTreeElement::InsertFirst(I3DShTreeElement*) 267
I3DShTreeElement::InsertLast(I3DShTreeElement*) 267
I3DShTreeElement::InsertLeft(I3DShTreeElement*) 267
I3DShTreeElement::InsertRight(I3DShTreeElement*) 267
I3DShTreeElement::GetHotPoint(TVector3&) 267
I3DShTreeElement::SetHotPoint(const TVector3&) 267
I3DShTreeElement::CenterElementOnHotPoint() 268
I3DShTreeElement::CenterHotPointOnElement() 268
I3DShTreeElement::TranslateHotPoint(TVector3&) 268
I3DShTreeElement::ComClone(I3DShTreeElement**, ECloneMode, boolean) 268
I3DShTreeElement:: InvalidateBoundingBox() 268
I3DShTreeElement::GetBoundingBoxWithoutLightsAndCameras(TBBox3D&) 269
I3DShTreeElement::GetBoundingBox(TBBox3D&, ETreeDeformBBoxModes) 269
I3DShTreeElement::CanBeDistorted() 269
I3DShTreeElement::BeginMove() 269
I3DShTreeElement::PostMoveChange(boolean) 269
I3DShTreeElement::EndMove(boolean) 270
I3DShTreeElement::GetXYZScaling(TVector3&) 270
I3DShTreeElement::SetXYZScaling(TVector3&) 270
I3DShTreeElement::GetNbrFreedom() 270
I3DShTreeElement::GetFreedomRange(int16, real&, real&) 270
I3DShTreeElement::GetFreedomTransform(TMatrix33&, TVector3&) 271
I3DShTreeElement::GetFreedomDerivative(int16, TMatrix33&, TVector3&) 271
I3DShTreeElement::GetFreedomSecondDerivative(int16, int16, TMatrix33&, TVector3&) 271
I3DShTreeElement::IncrementFreedomValue(int16 , real*) 272
I3DShTreeElement::GetFullName(TMCString&) 272
I3DShTreeElement::InsertModifier(I3DShModifier* , uint32, ECloneMode) 272
I3DShTreeElement::RemoveModifier(uint32) 272
I3DShTreeElement::GetModifiersCount() 272
I3DShTreeElement::GetModifierByIndex(I3DShModifier**, uint32) 273
I3DShTreeElement::SetModifiers(const TMCCountedPtrArray<I3DShModifier>*, ECloneMode) . . . 273
I3DShTreeElement::LinksChanged() 273
I3DShTreeElement::BeginGetRenderables(const TRenderableAndTfmArray*&, boolean) 273
I3DShTreeElement::EndGetRenderables() 274
I3DShTreeElement::GetConstraint(I3DShConstraint**) 274
I3DShTreeElement::SetConstraint(I3DShConstraint*, ECloneMode) 274
I3DShTreeElement::SetConstraintByID(int32) 274
I3DShTreeElement::GetDataComponentsCount() 274
I3DShTreeElement::GetDataComponentByIndex(I3DShDataComponent**, uint32) 274
I3DShTreeElement::SetDataComponentByID(I3DShDataComponent*, ECloneMode) 275
I3DShTreeElement::GetDataComponentByID(I3DShDataComponent**, IDType) 275
I3DShTreeElement::InsertDataComponent(I3DShDataComponent*) 275
I3DShTreeElement::InsertDataComponentByID(int32, int32) 275
I3DShTreeElement::RemoveDataComponent(uint32) 276
I3DShTreeElement::SetData(TDataKey, IMCUnknown*) 276
I3DShTreeElement::GetData(TDataKey, IMCUnknown**) 276
I3DShTreeElement::ReleaseData(TDataKey) 276

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::IsOpen() 276
I3DShTreeElement::GetEditor() 276
I3DShTreeElement::GetAnimationMethod(I3DShAnimationMethod**) 277
I3DShTreeElement::SetAnimationMethod(I3DShAnimationMethod*, ECloneMode, real) 277
I3DShTreeElement::GetLocalTreeTransform(TTreeTransform&) 277
I3DShTreeElement::SetLocalTreeTransform(const TTreeTransform&) 277
I3DShTreeElement::GetGlobalTreeTransform(TTreeTransform&) 277
I3DShTreeElement::SetGlobalTreeTransform(const TTreeTransform&, int16, boolean) 278
I3DShTreeElement::GetGlobalTreeTransform3D(TTreeTransform3D&) 278
I3DShTreeElement::SetGlobalTransform3D(const TTransform3D&, int16) 278
I3DShTreeElement::GetFatherGlobalTransform3D(TTransform3D&) 278
I3DShTreeElement::CloneSubTree(ECloneMode, I3DShTreeElement**) 279
I3DShTreeElement::GetUndoInfo(TTreeUndoInfo&) 279
I3DShTreeElement::SetFromUndoInfo(const TTreeUndoInfo&, const TTreeUndoInfo*) 279
I3DShTreeElement::GetTopIfConstrained(I3DShTreeElement**) 279
I3DShTreeElement::AddWireFrame(I3DExWireFrame*) 279
I3DShTreeElement::RemoveWireFrame(I3DExWireFrame*) 280
I3DShTreeElement::WireFrameArrayFree() 280
I3DShTreeElement::GetWireFrameArray(TMCClassArray<TWireFrameInfo>&) 280
I3DShTreeElement:: IsPickable() 280
I3DShTreeElement:: IsBBoxPickable() 280
I3DShTreeElement::RemoveConstraint(I3DShConstraint**) 280
I3DShTreeElement::RestoreConstraint(I3DShConstraint*) 280
I3DShTreeElement::GetExists() 281
I3DShTreeElement::SetExists(boolean) 281
I3DShTreeElement::GetIsShown() 281
I3DShTreeElement::SetIsShown(boolean) 281
I3DShTreeElement::SetShadowCasting(boolean) 281
I3DShTreeElement::SetShadowCatching(boolean) 281
I3DShTreeElement::GetShadowCasting() 282
I3DShTreeElement::GetShadowCatching() 282
I3DShTreeElement::GetAnimateTree() 282
I3DShTreeElement::SetAnimateTree(boolean) 282
I3DShTreeElement::ClearInstanceCache() 282

9.19 I3DShTweener Interface. 283

I3DShTweener::Tween(I3DShKeyFrame*, int32, I3DShTweenerChainLink) 283
I3DShTweener::Edit(I3DShTweener**,I3DShTweenerChainLink*) 283
I3DShTweener::SetLink(I3DShTweenerChainLink*) 284
I3DShTweener::GetLinkNoAddRef() 284

9.20 I3DShTweenerChainLink Interface 285

I3DShTweenerChainLink::Clone(I3DShTweenerChainLink**, I3DShParamTimeLine* . . . . .) 285
I3DShTweenerChainLink::CopyData(I3DShTweenerChainLink*) 286
I3DShTweenerChainLink::GetTweenerNoAddRef(boolean, boolean) 286
I3DShTweenerChainLink::GetTweener(I3DShTweener**, boolean, boolean) 286
I3DShTweenerChainLink::SetTweener(I3DShTweener*) 286
I3DShTweenerChainLink::GetKeyFrameNoAddRef() 287
I3DShTweenerChainLink::GetKeyFrame(I3DShKeyFrame**) 287
I3DShTweenerChainLink::SetKeyFrame(I3DShKeyFrame**) 287
I3DShTweenerChainLink::GetTime() 287
I3DShTweenerChainLink::SetTime(MicroTick) 287
I3DShTweenerChainLink::GetNextNoAddRef() 287
I3DShTweenerChainLink::GetNext(I3DShTweenerChainLink**) 288
I3DShTweenerChainLink::SetNext(I3DShTweenerChainLink*) 288
I3DShTweenerChainLink::GetPrevNoAddRef() 288
I3DShTweenerChainLink::GetPrev(I3DShTweenerChainLink**) 288

©2001-2003 Eovia. All rights reserved.

I3DShTweenerChainLink::SetPrev(I3DShTweenerChainLink*) 288
I3DShTweenerChainLink::GetFatherNoAddRef() 289
I3DShTweenerChainLink::RemoveFromChain() 289
I3DShTweenerChainLink::InsertAfter(I3DShTweenerChainLink*) 289
I3DShTweenerChainLink::OffsetLink(MicroTick) 289
I3DShTweenerChainLink::GetTimeInfo(TTimeInfo&) 289
I3DShTweenerChainLink::IsTweenerSelected() 289
I3DShTweenerChainLink::IsKeyFrameSelected() 289
I3DShTweenerChainLink::SelectKeyFrame(boolean) 290
I3DShTweenerChainLink::SelectTweener(boolean) 290

9.21 I3DShWireFrame Interface 291

I3DShWireFrame::AllocateFMesh(int32, int32, int32 . . . . .) 291
I3DShWireFrame::GetFMesh(FacetMesh**) 291
I3DShWireFrame::AllocateWireFrame(int16, int16) 292
I3DShWireFrame::GetWireFrame(WFData&) 292
I3DShWireFrame::Connect(int16, int16) 292
I3DShWireFrame::SetPoint(int16, EWFHandleType, const TVector3&) 292
I3DShWireFrame::Transform(const TTreeProjectionTransform&) 293
I3DShWireFrame::TestHit(TVector2&, real) 293
I3DShWireFrame::AddRenderables(TMCPtrArray<I3DShRenderable>&) 293
I3DShWireFrame::GetColor() 293
I3DShWireFrame::SetColor(const TMCColorRGB&) 293
I3DShWireFrame::ExcludeFromZBuffer(boolean, boolean) 294
I3DShWireFrame::SetSelectedState(const boolean) 294
I3DShWireFrame::ExcludeFromProjection(boolean) 294

Chapter 10 - DataBase Overview 295

10.1 Scene 295
10.2 Coordinate Systems 297
10.3 Geometry 302
10.4 Color 309

Chapter 11 - Index 311

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

Chapter 2 - MCCommon IEx* Reference

2.1 IExDataExchanger Interface

Interface ID: IID_IExDataExchanger

Interface file: IExtension.h

Description

The **IExDataExchanger** Interface contains utility methods that are complements to IMCUnknown, like clone(). It allows the Shell to handle the UI of the Component and the Component's public data.
It also handles the communication between the Component and the Shell.

Methods Index

"IExDataExchanger::Clone(IExDataExchanger**, IMCUnknown*)"
"IExDataExchanger::IsEqual(IExDataExchanger*)"
"IExDataExchanger::GetParameter(IDType, void*)"
"IExDataExchanger::SetParameter(IDType, void*)"
"IExDataExchanger::GetExtensionDataBuffer()"
"IExDataExchanger::ExtensionDataChanged()"
"IExDataExchanger::HandleEvent(MessageID, IMFResponder*, void*)"
"IExDataExchanger::GetResID()"

IExDataExchanger::Clone(IExDataExchanger**, IMCUnknown*)

void MCCOMAPI Clone(IExDataExchanger,IMCUnknown* pUnkOuter)**

Clone() creates a copy of an external component. You only need to implement this method if your object has data that are not stored in the pmap.

Parameter

IExDataExchanger**
A pointer to where the pointer to the cloned component should be stored.

IMCUnknown* *pUnkOuter*
A pointer to the internal component corresponding to the new clone. (Most external components are aggregated to an internal component that takes care of the reference counting). If this parameter is null, your component will have to do his own reference counting.

IExDataExchanger::IsEqual(IExDataExchanger*)

boolean MCCOMAPI IsEqual(IExDataExchanger * otherDataExchanger)

IsEqual() returns True if the two pointers are referring to the same data exchanger.

Parameter

IExDataExchanger*otherDataExchanger*
A pointer to the data exchanger.

©2001-2003 Eovia. All rights reserved.

IExDataExchanger::GetParameter(IDType, void*)

MCCOMErr MCCOMAPI GetParameter(IDType keyword, void *parameter)

GetParameter() returns the value of a given parameter of this dataExchanger. This function is called when the shell cannot find the given parameter in the Parameter Map (Pmap) of the component.

Parameter

IDType *keyword*
The four character ID corresponding to the parameter.

void* *parameter*
A pointer to a bloc of memory where the value of the parameter will be stored.

IExDataExchanger::SetParameter(IDType, void*)

MCCOMErr MCCOMAPI SetParameter(IDType keyword, void *parameter)

SetParameter() sets the value of a parameter of this dataExchanger .

Parameter

IDType *keyword*
The four character ID corresponding to the parameter.

void* *parameter*
A pointer to a bloc of memory where the value of the parameter is stored.

IExDataExchanger::GetExtensionDataBuffer()

void* MCCOMAPI GetExtensionDataBuffer()

GetExtensionDataBuffer() returns a pointer to the extension data buffer (Pmap). This function should always be implemented by your component otherwise the base address of the object will be used as the address of the Pmap.

IExDataExchanger::ExtensionDataChanged()

MCCOMErr MCCOMAPI ExtensionDataChanged()

ExtensionDataChanged() is called by the shell each time the parameter map (Pmap) of your component is modified. This call can be used to update caches for instance.

IExDataExchanger::HandleEvent(MessageID, IMFResponder*, void*)

MCCOMErr MCCOMAPI HandleEvent(MessageID message, IMFResponder* source, void* data)

HandleEvent() handles a message event.

Parameter

MessageID *message*
All MessageIDs are preceeded with "kMsg".

©2001-2003 Eovia. All rights reserved.

IMFResponder* *source*
A pointer that specifies the source of the event.

void* *data*
A pointer that specifies the data.

IExDataExchanger::GetResID()

int16 MCCOMAPI GetResID()

GetResID() returns the resource ID of the node part that should be used to display the UI of your component. If this function is not implemented, the resource ID of the 'COMP' resource associated with your component will be used instead.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

2.2 I3DExExportFilter Interface

Interface ID: IID_I3DExExportFilter

Interface file: I3DExImportExport.h

Description

The **I3DExExportFilter** Interface filters Carrara data for exporting, and allows you to set up preview information that gets stored with the file.

Methods Index

"I3DExExportFilter::Prepare(I3DShScene*, I3DShElement*)"
"I3DExExportFilter::WantsOptionDialog()"
"I3DExExportFilter::NeededDiskSpace(IMCFile*, I3DShScene*, I3DShTreeElement*, uint32*)"
"I3DExExportFilter::DoExport(IMCFile*,I3DShScene*, I3DShElement*)"
"I3DExExportFilter::WantsPreview()"
"I3DExExportFilter::GetPreviewSize(int16*, int16*)"
"I3DExExportFilter::SetPreview(TMCPixelBucket*)"

I3DExExportFilter::Prepare(I3DShScene*, I3DShElement*)

boolean MCCOMAPI Prepare(I3DShScene* scene, I3DShTreeElement* fatherTree)

Prepare() prepares to export the specifed tree element in the specified scene . Returns True when <zot needs more description>

Parameter

I3DShScene* *scene*
A pointer that specifies which scene to prepare to export.

I3DShTreeElement* *fatherTree*
A pointer to the top or father of the tree element for the item being prepared for export.

I3DExExportFilter::WantsOptionDialog()

boolean MCCOMAPI WantsOptionDialog()

WantsOptionDialog() returns True if the Options dialog box is needed to specify options before the scene can be saved.

I3DExExportFilter::NeededDiskSpace(IMCFile*, I3DShScene*, I3DShTreeElement*, uint32*)

MCCOMErr MCCOMAPI NeededDiskSpace(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree, uint32* diskSpace)

NeededDiskSpace() determines the amount of disk spaced needed to store a file containing the specified scene and specified tree element .

©2001-2003 Eovia. All rights reserved.

Parameter

IMCFile* *file*
A pointer that specifies the file name.

I3DShScene* *scene*
A pointer that specifies which scene to prepare to export.

I3DShTreeElement* *fatherTree*
A pointer to the top or father of the tree element for the item being prepared for export.

uint32* *diskSpace*
A pointer to the amount of disk space needed to store the file.

I3DExExportFilter::DoExport(IMCFile*,I3DShScene*, I3DShElement*)

MCCOMErr MCCOMAPI DoExport(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree)

DoExport() performs the export to the specfied file of the specified scene and tree element.

Parameter

IMCFile* *file*
A pointer that specifies the file name.

I3DShScene* *scene*
A pointer that specifies which scene to prepare to export.

I3DShTreeElement* *fatherTree*
A pointer to the top or father of the tree element for the item being prepared for export.

I3DExExportFilter::WantsPreview()

boolean MCCOMAPI WantsPreview()

WantsPreview() returns True when preview information should be saved with the file.

I3DExExportFilter::GetPreviewSize(int16*, int16*)

MCCOMErr MCCOMAPI GetPreviewSize(int16* sizeh, int16* sizev)

GetPreviewSize() calculates the preview size in pixels .

Parameter

int16* *sizeh*
A pointer that specifies the horizontal preview dimension in pixels.

int16* *sizev*
A pointer that specifies the vertical preview dimension in pixels.

I3DExExportFilter::SetPreview(TMCPixelBucket*)

MCCOMErr MCCOMAPI SetPreview(TMCPixelBucket* preview)

©2001-2003 Eovia. All rights reserved.

SetPreview() displays the specified preview image .

Parameter **TMCPixelBucket*** *preview*
A pointer that specifies which preview to display.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

2.3 I3DExImportFilter Interface

Interface ID: IID_I3DExImportFilter

Interface file: I3DExImportExport.h

Description

The **I3DExImportFilter** Interface filters data for importing into Carrara.

Methods Index

"I3DExImportFilter::Prepare(IMCFile*, I3DShScene*, I3DShElement*)"
"I3DExImportFilter::WantsOptionDialog()"
"I3DExImportFilter::DoImport(IMCFile*,I3DShScene*, I3DShElement*)"
"I3DExImportFilter::WantsTopScene()"

I3DExImportFilter::Prepare(IMCFile*, I3DShScene*, I3DShElement*)

boolean MCCOMAPI Prepare(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree)

Prepare() prepares to export the specifed tree element in the specified scene . Returns True when <zot needs more description>

Parameter

IMCFile* *file*
A pointer that specifies the file to import.

I3DShScene* *scene*
A pointer that specifies which scene into which to import.

I3DShTreeElement* *fatherTree*
A pointer to the top or father of the tree element for the item being prepared for import.

I3DExImportFilter::WantsOptionDialog()

boolean MCCOMAPI WantsOptionDialog()

WantsOptionDialog() returns True if the Options dialog box is needed to specify options before the scene can be imported.

I3DExImportFilter::DoImport(IMCFile*,I3DShScene*, I3DShElement*)

MCCOMErr MCCOMAPI DoImport(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree)

DoImport() performs the import of the specified file to the specified scene and tree element.

Parameter

IMCFile* *file*
A pointer that specifies the file name.

I3DShScene* *scene*
A pointer that specifies which scene to prepare to export.

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement* *fatherTree*
A pointer to the top or father of the tree element for the item being prepared for export.

I3DExImportFilter::WantsTopScene()

boolean MCCOMAPI WantsTopScene()

WantsTopScene() returns True when <zot need more info>.

©2001-2003 Eovia. All rights reserved.

2.4 IExRegisterer Interface

Interface ID: IID_IExRegisterer

Interface file: IExIRegisterer.h

Description

The **IExRegisterer** Interface can register multiple components, typically when the existance of components is conditionned by other installed software.

Methods Index

"IExRegisterer::GetComponentCount()"
"IExRegisterer::GetComponentData(uint32, APICOMPData&, APIPrivateData&, APIGUIDData&)"
"IExRegisterer::UnregisterComponents()"

IExRegisterer::GetComponentCount()

uint32 MCCOMAPI GetComponentCount()

GetComponentCount() returns the number of components to be registered. <zot needs more description>

IExRegisterer::GetComponentData(uint32, APICOMPData&, APIPrivateData&, APIGUIDData&)

MCCOMErr MCCOMAPI GetComponentData(uint32 index, APICOMPData& compData, APIPrivateData& privateData, APIGUIDData& guidData)

GetComponentData() gets all of the information needed to register a component. This includes the component data, any private data needed by that component, and the guid data.

Parameter

uint32 *index*
<zot needs description>.

APICOMPData& *compData*
A reference to the component data.

APIPrivateData& *privateData*
A reference to any private data needed by that type of component.

APIGUIDData& *guidData*
A reference to the guid data for the component.

IExRegisterer::UnregisterComponents()

MCCOMErr MCCOMAPI UnregisterComponents()

UnregisterComponents() should be called when a component is uninstalled <zot need more info>.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

Chapter 3 - Data Structures Reference

Basic Data Structures

These are common data structures found throughout the SDK.

TVECTOR2

TVECTOR2 is used to manipulate two-dimensional data: 2D points or 2D vectors.

Constructors

```
TVector2::TVECTOR2(void)
TVector2::TVECTOR2(real, real)
```

Constants

Name	X value	Y value
TVector2::kZero	0.0	0.0
TVector2::kOnes	1.0	1.0
TVector2::kUnitX	1.0	0.0
TVector2::kUnitY	0.0	1.0
TVector2::kNegativeX	-1.0	0.0
TVector2::kNegativeY	0.0	-1.0

Element Access Operators

```
TVector2::TVector2()
```
TVector2(0) returns the X value and TVector2(1) returns the Y value.

```
TVecotr2::TVector2[]
```
TVector2[0] returns the X value and TVector2[1] returns the Y value.

```
TVector2::TVector2.X
TVector2::TVector2.Y
```

Methods

```
TVector2::SetValues(real,real)
```
Assigns the X and Y values.

```
TVector2::GetDimension(void)
```
Returns the dimensions of a TVector2.

©2001-2003 Eovia. All rights reserved.

`TVector2::GetMagnitude()`

Returns $\sqrt{x^2 + y^2}$

`TVector2::GetMagnitudeSquared()`
Returns the square of the magnitude:

$$x^2 + y^2$$

`TVector2::GetDistance(const TVector2& )`
Returns the distance of vector.

`TVector2::SetMagnitude(const real)`
Sets the magnitude of the vector equal to the real given in the parameter.

`TVector2::SetToCenter(const TVector2&, const TVector2&)`
Sets the point at the center of the two points given in parameters.

`TVector2::Normalize(const real)`
Determines if the magnitude of the vector is greater than the real given in the parameter. If so, it normalizes the vector by setting the magnitude equal to 1. Returns True if it normalized the vector.

`TVector2::Normalize(TVector2& )`
Normalizes the vector if it is not (0,0). Returns True if it normalized the vector.

`TVector2::NormalizeOrReplaceWith(const TVector2& ,const real)`
If the magnitude of the vector is greater than the real, then the vector is normalized. Otherwise, it is replaced by the vector given in the parameter.

`IsEqual(const TVector2& , const real)`
Returns True if the vector and the vector given in the paramter are the same or within the tolerance specified by the real.

`ClampValue(const TVector2& , const TVector2& )`
Bounds the point within the range fixed by the two points given in the parameters. The first is the mimimum value, the second represents the upper limit.

`Negate(void)`
Returns the negation of "this".

`Interpolate(const TVector2& , const TVector2& , const float )`
Interpolates the point "this" between the two points given in parameters; if hte float is equal to 0, "this" will be equal to the first paramter. If the float is equal to 1, "this will be equal to the second parameter.

`Swap(TVector2& )`
Swaps "this" and the vector given in the paramter.

zot There was more in Vector2.h not sure how much to document.

Operators

These are used for fast vector operations.

```
inline TVector2&operator/=(const real scalar);
inline TVector2&operator*=(const real scalar);
inline TVector2&operator+=(const TVector2& v);
inline TVector2&operator-=(const TVector2& v);
```

©2001-2003 Eovia. All rights reserved.

TVECTOR3

TVECTOR3 is similar to **TVECTOR2** but for three-dimensional data: 3D points or 3D vectors.

TBBOX2D

TBBOX2D is used for 2D Bounding Boxes. A 2D Bounding Box is a rectangular area that encloses a set of 2D points.

Constructors

```
TBBox2D();
TBBox2D(const TVector2& min, const TVector2& max);
TBBox2D(const TVector2& max);
```

Methods

```
void SetMax(const TVector2& max) {fMax = max;}
```
Sets the upper right corner of the bounding box.

```
void SetMin(const TVector2& min) {fMin = min;}
```
Sets the bottom left corner of the bounding box.

```
void GetCenter(TVector2& center) const
```
Returns the coordinates for the center of the bounding box.

```
void GetDelta(TVector2& delta) const
```
Gets the difference between the minimum and maximum values.

```
real GetMax() const
```
Returns either the width of the bounding box if it is greater than the length, or the length if it is greater than or equal to the width.

```
boolean IsEmpty() const
```
Returns True if the bounding box is not set to a value.

```
void AddPoint(const TVector2 &PP)
```
Resets the bounding box if the point PP wasn't within. PP becomes either hte upper right or bottom left corrner of the new bounding box.

```
void Inset(const TVector2 &PP)
```
Shrinks the bounding box of the vector given in the parameter.

```
TVector2 operator[](int16 ii) const
```
Transforms the bounding box in an array. The first element is the bottom-left corner, the second is the upper-left corner, the third is the upper-right corner, and the forth is the bottom-right corner.

```
void FixInvertedFields()
```
If hte bottom-left corner of the bounding box is located further to the right than the upper-right corner, then both corner x-coordinates are set to 0. If the bottom-left corner is greater than the upper-left, then both of their Y coordinates are set to 0.

```
inline real GetWidth( void )
       { return fMax.x - fMin.x; }
```
Returns the width of the bounding box.

©2001-2003 Eovia. All rights reserved.

```
inline real GetHeight( void )
   { return fMax.y - fMin.y; }
```

Returns the height of the bounding box.

Constants

TVector2 fMin

Bottom-left corner of the bounding box.

TVector2 fMax

Upper-right corner of the bounding box.

TBBOX3D

TBBOX3D is similar to TBBox 2D except that it is used for 3D Bounding Boxes. A 3D Bounding Box is a cubic volume that encloses a set of 3D points.

TMATRIX33

TMATRIX33 is used for 3D rotation.

If you are porting code from RDS, be careful about converting code from TMatrix3D to TMatrix33 since the void ctor for TMatrix33 does not set the matrix to identity. Use explicit initialization to the constant TMatrix33::kIdentity if necessary.

zot the section that follows is from the old sdk. How much of this is stil relevant?

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.

Comments To define a 3D rotation, such a matrix must be normalized.

©2001-2003 Eovia. All rights reserved.

All row and column vectors of the matrix must be unit vectors:

$ix^2 + iy^2 + iz^2 = 1.0$

$jx^2 + jy^2 + jz^2 = 1.0$

$kx^2 + ky^2 + kz^2 = 1.0$

$ix^2 + jx^2 + kx^2 = 1.0$

$iy^2 + jy^2 + ky^2 = 1.0$

$iz^2 + jz^2 + kz^2 = 1.0$

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
```

zot end old material

Constants

`kIdentity` is the matrix identity.

©2001-2003 Eovia. All rights reserved.

kZero is Matrix 0.

Constructors

```
inline TMatrix33 (void);
inline TMatrix33 (const TMatrix33& original);
inline TMatrix33 (const real m00, const real m01, const real m02, const real m10,
      const real m11, const real m12, const real m20, const real m21, const real
      m22);
```

Methods

`inline uint32 GetRowDimension (void)`
Gets the number of rows.

`inline uint32 GetColumnDimension (void)`
Gets the number of columns.

`inline real operator() (const uint32 row, const uint32 col)`
Use this for element access, since there is no [] [] operator.

`inline real& operator() (const uint32 row, const uint32 col)`
Use this for element access, since there is no [] [] operator.

`inline const real* operator[] (const uint32 row)`
Use this method for [][] access only; the second index is unchecked.

`inline real* operator[] (const uint32 row)`
Use this method for [][] access only; the second index is unchecked.

`void GetRow (const uint32 row, TVector3& v)`
Gets a row for row access:

`void SetRow (const uint32 row, const TVector3& v)`
Sets a row of the matrix.

`void GetColumn (const uint32 col, TVector3& v)`
Gets a column for column access:

`void SetColumn (const uint32 col, const TVector3&v)`
Sets a column of the matrix.

`void SetToTranspose (void)`
Use this method for matrix conversion. Transposes the matrix "this".

`void SetToTransposeOf (const TMatrix33& source)`
Use this method for matrix conversion. Sets the matrix equal to the transpose of the matrix source.

`TMatrix33 GetTranspose (void)`
Returns the transpose of the matrix "this" without changing it.

`TMatrix33 GetAdjoint ()`
Returns the adjoint of the matrix "this".

`TMatrix33 GetOrthogonal (boolean* success)`
Returns the rotation associated with the matrix ; an orthogonal matrix returns itself.

©2001-2003 Eovia. All rights reserved.

```
real GetTrace (void)
```
Returns the trace of the matrix "this".

```
real GetDeterminant (void)
```
Returns the determinant of the matrix "this".

```
void Multiply (const TMatrix33& m, const TMatrix33& n)
```
Sets the matrix "this" equal to the multiplication of the parameters m and n.

The following methods are used for fast matrix operation:

```
void operator*=   (const real scalar);
TMatrix33& operator= (const TMatrix33& original);
```
These routines were added to support existing code in RealMatrix33:

```
void SetColumns   (const TVector3& c0, const TVector3& c1, const TVector3& c2);
void GetColumns   (TVector3& c0, TVector3& c1, TVector3& c2) const;
void SetToDiagonalTfm (const real x, const real y, const real z);
boolean IsEqual   (const TMatrix33& rhs, const real zeroTolerance = 0.0f) const;
```
These are the depricated names from RealMatrix33, use the direct equivalents when writing new code:

```
inline void SetIJK (const TVector3& c0, const TVector3& c1, const TVector3& c2)
          { SetColumns( c0, c1, c2 ); }

inline void GetIJK (TVector3& c0, TVector3& c1, TVector3& c2) const
          { GetColumns( c0, c1, c2 ); }

inline void SetDiagonal (const real x, const real y, const real z)
          { SetToDiagonalTfm( x, y, z ); }

inline void Reset ()
          { *this = TMatrix33::kIdentity; }

inline boolean IsIdentity () const
          {
          return ( this == &TMatrix33::kIdentity ) ||
              this->IsEqual( TMatrix33::kIdentity );

inline TMatrix33 Transpose () const
          { return GetTranspose(); }
```

Use TMtxMath::SolveInverseGeneralCR instead of this:

```
TMatrix33 Inverse(boolean* success) const;
```

TTRANSFORM3D

TTRANSFORM3D is used to define a combination of a Rotation and a Translation.

Constants

fRotationAndScale

Matrix composed of: UniformScaling, XYZScaling, and Rotation around local 0,0,0.

©2001-2003 Eovia. All rights reserved.

Constructor

```
TTransform3D()
```

Methods

```
TVector3 TransformPoint(const TVector3 &PP) const;
```
Transforms a three dimensional point.

```
TVector3 TransformVector(const TVector3 &VV) const;
```
Transforms a three dimensional vector.

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.

TTREETRANSFORM

TREETRANSFORM is used to define a Tree Element Transformation.

Constructors and Destructor

```
TTreeTransform();
TTreeTransform(const TTreeTransform &aT);
~TTreeTransform();
```

Methods

```
void      Replicate(const TTreeTransform& repTransform)
boolean   IsIdentity()
boolean   IsDirect()
void      Reset()
void      AlignOnGravity()
void      PointAt(const TVector3 &point)
void      Expand(real scale)
void      Invert()
TTreeTransformInverse()
void      Align()
void      Orthogonalize()
```

Operators

```
boolean operator==(const TTreeTransform& tt)
void   operator= (const TTreeTransform& tt)
TTreeTransformoperator*(const TTreeTransform& b)
TVector3  operator*(const TVector3& point)
```

Accessors

```
void   SetOffset(const TVector3& offset){fOffset= offset;}
void   SetRotation(const TMatrix33& rotation){fRotation= rotation;}
void   SetUniformScaling(const real& scaling){fUniformScaling    = scaling;}
void   SetHotPoint(const TVector3& hotPoint){fHotPoint= hotPoint;}
void   SetXYZScaling(const TVector3& XYZScaling){fXYZScaling= XYZScaling;}
void   SetPhyThetaPsy(real phy, real theta, real psy, boolean isDirect);
const TVector3& GetOffset() const {return fOffset;}
```

©2001-2003 Eovia. All rights reserved.

```
const TMatrix33& GetRotation() const {return fRotation;}
const real&GetUniformScaling() const {return fUniformScaling;}
const TVector3& GetHotPoint() const {return fHotPoint;}
const TVector3& GetXYZScaling() const {return fXYZScaling;}
void   GetPhyThetaPsy(real& phy, real& theta, real& psy, boolean& isDirect)const;
void    GetTransform3D(TTransform3D &aT) const;
void    GetTransform3D(TMatrix33& RR, TVector3& TT) const;
void    GetCombinedTranslation(TVector3 &TT) const;
void    GetPlane(TPlane& aplane) const;
```

Translation

```
void   Translate(const TVector3& vector);
void   TranslateHP(const TVector3& vector);
void   TranslateHPL(const TVector3& vector);
```

Rotation

```
void   RotateAxis(const TVector3& center, const real sin, const real cos, const
       TVector3& AXIS, const real& constrainAngle = kRealZero);
void   Rotate(const TVector3& center, const TVector3& PP,  const TVector3& QQ,
       const TVector3& AXIS,  const real& constrainAngle = kRealZero);
void   RotateL(const TMatrix33& a);
void   RotateR(const TMatrix33& b);
```

Composition

```
void   ComposeL(const TTreeTransform& transform);
void   ComposeR(const TTreeTransform& transform);
```

Transformations

```
void   TransformCS  (const TTreeTransform& a);
void   InverseTransformCS(const TTreeTransform& a);

TVector3& Transform    (const TVector3& point, TVector3& result) const;
TVector3& InverseTransform(const TVector3& point, TVector3& result) const;

TVector3& TransformV   (const TVector3& point, TVector3& result) const;
TVector3& InverseTransformV(const TVector3& point, TVector3& result) const;

TPlane&   ransformPlane(const TPlane& plane, TPlane& result) const;
TPlane&   InverseTransformPlane(const TPlane& plane, TPlane& result) const;

TBBox3D   TransformBBox(const TBBox3D& box) const;
TBBox3D   InverseTransformBBox(const TBBox3D& box) const;

TVector3  ScaleV (const TVector3&) const;// scaling only
```

Variables

```
TMatrix33fRotation;
```
Rotation matrix around HotPoint.

```
TVector3fOffset;
```
Vector from parent 0,0,0 to HotPoint.

```
real   fUniformScaling;
```
Uniform scaling. This scaling is applied in all directions.

```
TVector3fHotPoint;
```

©2001-2003 Eovia. All rights reserved.

Vector from local 0,0,0 to HotPoint.

```
TVector3fXYZScaling;
```
XYZ (non-uniform) scaling.

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.

TMCCOLORRGB

This is the floating point color used for most calculations since it can do interpolation without converting int-to-float.

Variables

```
real kMaxIntensity = 1;
real kInvMaxIntensity = 1.0 / kMaxIntensity;

TMCColorRGB kBlack = (0,0,0);
TMCColorRGB kWhite = (kMaxIntensity,kMaxIntensity,kMaxIntensity);
```

Constructors

```
inline TMCColorRGB(void);
inline TMCColorRGB(const real rr,const real gg, const real bb);
inline TMCColorRGB(const TMCRGBColor inColor);
```

Methods

```
inline void Set  (const real rr, const real gg, const real bb);
```
Sets the Red, Green, and Blue values.

```
boolean IsEqual  (const TMCColorRGB& rhs, const real epsilon = 0.0) const;
```
Returns True if the color "this" is equal to rhs, or within the tolerance specified. (epsilon by default is set to 0.)

Magnitudes

```
inline real GetMagnitude(void);
```
Returns the magnitude.

```
inline real GetMagnitudeSquared(void);
```
Returns the magnitude squared.

```
inline real GetMinOfRGB(void) const;
```
Finds the minium intensity.

```
inline real GetMaxOfRGB(void) const;
```
Finds the maximum intensity.

```
boolean IsClamped(void) const;
```
Use this method to see if the color falls in between [0, 1]; mainly for debugging.

©2001-2003 Eovia. All rights reserved.

```
void   Clamp  (void);
```
Clamp the color value to between [0, 1].

```
void   Interpolate  (const TMCColorRGB&from, const TMCColorRGB& to, const real
       interpolant);
void   Interpolate  (const TVector2& interpolants, const TMCColorRGB& lowerLeft,
       const TMCColorRGB& lowerRight, const TMCColorRGB& upperLeft, const TMCCo-
       lorRGB& upperRight);
```
Linear/bilinear interpolation, interpolants must be betwen 0 and 1. X values are left/right. Y values are lower/upper.

```
inline void AccumRGB(const TMCColorRGB&rhs, const realscalar );
```
Color accumulation is used so often and we want to take advantage of fmul-add. This is equiv to *this += rhs * scalar; no clamping is done. In TMCColorRBB, the alpha information is ignored.

Operators

These are operators on all the components:

```
inline booleanoperator==(const TMCColorRGB& rhs ) const;
inline booleanoperator!=(const TMCColorRGB& rhs ) const;
inline booleanoperator<(const real t) { return red<t || green<t || blue<t; }
inline booleanoperator>(const real t) { return red>t || green>t || blue>t; }

inline TMCColorRGB& operator+=(const TMCColorRGB& rhs);
inline TMCColorRGB& operator-=(const TMCColorRGB& rhs);
inline TMCColorRGB& operator*=(const real s);
inline TMCColorRGB& operator/=(const real s);

inline TMCColorRGBoperator+(const TMCColorRGB& rhs) const;
inline TMCColorRGBoperator-(const TMCColorRGB& rhs) const;
inline TMCColorRGBoperator*(const real s) const;
inline TMCColorRGBoperator/(const real s) const;
```

Dot, term-by-term product:

```
inline realoperator*(const TMCColorRGB& rhs) const;
inline TMCColorRGB  operator%(const TMCColorRGB& rhs) const;
```

Conversion operations to the other colors:

```
TMCColorRGBA8&ToRGBA8(TMCColorRGBA8&c, const uint8 alphaValue) const;
TMCColorRGBA8&ToRGBA8Unclamped(TMCColorRGBA8& c, const uint8 alphaValue) const;

TMCColorRGBA&ToRGBA(TMCColorRGBA& c, const float alphaValue) const;
```

Conversion from other colors:

```
inline TMCColorRGB& operator=(class TMCColorRGBA &c);

inline const real* GetAddrFirstComponent(void) const;
```
Returns the address of first component in the structure.

To facilitate a simpler move away from old colors, do not use R,G,B,A in new code. Use the full names instead.

```
union { real red; real R; };
union { real green;real G; };
union { real blue;real B; };
```

©2001-2003 Eovia. All rights reserved.

Fields

Red

The Red value in RGB mode, or the Cyan value in CMYK. From 0.0 to 1.0.

Green

The Green value in RGB mode, or the Magenta value in CMYK. From 0.0 to 1.0.

Blue

The Blue value in RGB mode, or the Yellow 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)

TMCGradient

A gradient contains an array of TMCColorRGBAIndex and for a value between 0 and 1 can return an interpolation between the 2 colors whose index framed the value. It can be used as a pMap parameter and be animated.

Constructors

```
inline TMCGradient (void);
inline TMCGradient (boolean isEmpty);
inline TMCGradient (const TMCClassArray<TMCColorRGBAIndex>& colorArray);
```

Methods

```
void GetColor (real32 pos, TMCColorRGBA& color ) const
```
Returns a color in the gradient. Position must be between 0 and 1.

```
int32 GetColorKeyCount () const
```
Returns the number of elements in fColorIndexArray

```
void GetColorKey (const int32 elem, TMCColorRGBAIndex& colorKey ) const
```
Access an element in the colorKey array

```
int32 FindColorKey (const TMCColorRGBAIndex& colorKey, const real32 epsilon)
      const
```
Looks for a specific colorKey in the array of TMCColorRGBAIndex. If several are found in the tolerance (epsilon), the nearest one is choosen. Returns the position of the elem in array (-1 if not found).

```
int32 FindColorKeyByIndex (const real32 index, const real32 epsilon ) const
```
Looks for a colorKey with the same index in the array of TMCColorRGBAIndex. If several are found in the tolerance (epsilon), the nearest one is choosen. Returns the position of the colorKey in the array (-1 if none is found).

```
void SetColorKeyColor (const int32 elem, TMCColorRGBA& color)
```
Sets the color for a colorKey.

```
void SetColorKeyIndex (const int32 elem, const real32 index)
```
Set the index for a colorKey.

©2001-2003 Eovia. All rights reserved.

```
int32 AddColorKey( TMCColorRGBAIndex& colorIndex )
int32 AddColorKey( TMCColorRGBA& color, real32 index )
int32 AddColorKey( real32 red, real32 green, real32 blue, real32 alpha, real32
      index )
```
Adds a new colorKey to the array of TMCColorRGBAIndex. Returns the position in the array, or -1 if the colorKey couldn't be added.

```
boolean RemoveColorKey( const int32 elem )
```
Removes a colorKey from the array. Return false if out of range

```
boolean IsEqual(const TMCGradient& grad, const real32 epsilon=0.0f) const
```
Equality test between this gradient and another one. Returns true if equal (same number of colorKey, and each colorKey with same color and same index)

```
void Interpolate(const TMCGradient& from, const TMCGradient& to, const real32
      interpolant)
```
Linear interpolation: set this gradient to a value beetween from and to. interpolant should be between 0 and 1.

```
void Interpolate(const TVector2& interpolants,  const TMCGradient& lowerLeft,
      const TMCGradient& lowerRight,  const TMCGradient& upperLeft,  const
      TMCGradient& upperRight);
```
Bilinear interpolation: sets this gradient to a value beetween the 4 gradients. interpolants: interpolation values (x [left/right], y [lower/upper])

Operators

```
inline boolean operator==(const TMCGradient& rhs) const
```
-Description: same as IsEqual

```
inline boolean operator!= (const TMCGradient& rhs) const
```
-Description: same as !IsEqual

```
inline TMCGradient&operator=(const TMCGradient& grad)
```
Copies the content of grad to the left side gradient

ETreeDeformBBoxModes

ETreeBoundingBoxModes is used to indicate which kind of bounding box is to be given by GetBoundingBox. It is a method of I3DShGroup, I3DShInstance, and I3DShTreeElement.

©2001-2003 Eovia. All rights reserved.

Variables

kApplyNoDeformer	0	Gives the original box (without applying any deformer).
kApplyDeformersLocaly	1	At each tree (group) level accumulates the bboxes of the children (recursively computed with kApplyDeformersLocaly) and then applies the tree deformers on the resulting box.
kApplyAllDeformersAtInstanceLevel	2	Valid only for instances, deforms the original box of the instance by all the deformers of the tree the instance is on.
kUnusedApplyDeformerConstant	0xFFFFFFFF	To ensure "castable" as uint32.

©2001-2003 Eovia. All rights reserved.

Chapter 4 - Shading Data Structures Reference

ShadingIn

Description

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 *"ShadingFlags"* were set earlier by the Shader. 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"*.

Field Index

<Section>Appendix , "TVector3 fPoint,<Section>"
<Section>Appendix , "TVector3 fGNormal,<Section>"
<Section>Appendix , "TVector3 fIsoU,<Section>"
<Section>Appendix , "TVector3 fIsoV,<Section>"
<Section>Appendix , "TVector2 fUV,<Section>"
<Section>Appendix , "uint32 fUVSpaceID,<Section>"
<Section>Appendix , "TVector3 fPointLoc,<Section>"
<Section>Appendix , "TVector3 fNormalLoc,<Section>"
<Section>Appendix , "boolean fBumpOn,<Section>"
<Section>Appendix , "uint32 fCurrentCompletionMask,<Section>"
<Section>Appendix , "I3DShInstance* fInstance,<Section>"
<Section>Appendix , "UVSpaceInfo* fUVInfo,<Section>"
"TVector3 fPointx and fPointy"
"TVector3 fNormalx and fNormaly"
"TVector3 fPointLocx and fPointLocy"
"TVector3 fNormalLocx and fNormalLocy"
"TVector2 fUVx,fUVy"

<zot>Please check the descriptions for all of these fields.</zot>

TVector3 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.

©2001-2003 Eovia. All rights reserved.

TVector3 fGNormal

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).

TVector3 fIsoU

The U 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. *fIsoU* and *fIsoV* provide that information.

TVector3 fIsoV

The 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. *fIsoU* and *fIsoV* provide that information.

TVector2 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.

uint32 fUVSpaceID

The UV Space ID of the UV Space to which the *fUV* values belong. It can use UVSpaceID enum.

TVector3 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.

TVector3 fNormalLoc

The surface Normal in Local Coordinates. See the Rainbow shader example in the toolkit. A shader can use *fNormalLoc* to perform shading that does not depend on the orientation of the object in the Scene.

boolean fBumpOn

The global bump setting.

©2001-2003 Eovia. All rights reserved.

uint32 fCurrentCompletionMask

Internal flags for shaders. Must be set to 0 before calling **DoShade()**.

I3DShInstance* fInstance

Used by ShadingMode to calculate the bounding Box.

UVSpaceInfo* fUVInfo

UV Space information will be filled by I3DShInstance::**DoShade()**.

TVector3 fPointx and fPointy

Surface point derivatives in Global Coordinates.

TVector3 fNormalx and fNormaly

Surface Normal derivatives in Global Coordinates.

TVector3 fPointLocx and fPointLocy

Surface point derivatives in Local Coordinates.

TVector3 fNormalLocx and fNormalLocy

Surface Normal derivatives in Local Coordinates.

TVector2 fUVx,fUVy

Texture derivatives.

ShadingFlags

Description

Shading flags are used by the 3D Shell to determine which fields of the *ShadingIn* structure the Shader requires. This allows the 3D Shell to use the minimum data and to convert between different types.

See the *ShadingIn* data structure for complete details on the meaning of each field.

Field Index

```
Zot didn't find fCallOnce.Please confirm it should be removed.
```

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.

©2001-2003 Eovia. All rights reserved.

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**).

fNeedsIsoUV

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.

fChangesNormal

The Shader will change the Normal. This allows the Shader to tell the 3D Shell that a new Normal (**ShadingOut::fChangedNormal**) will have to be read after calling the Shader. This is intended for Shaders that changes the Normal like Bump mapping Shaders.

fUVSpaceShaders

Returns true if there are different shaders in UVSpaceId.

fConstantChannelsMask

zot needs description, here is the comment from the code: // use these enum: kColorChannel,kSpecularityChannel,kShininessChannel,kNormalChannel...

Comments See the **I3DExShader** Interface documentation for further reference. Also read the description of the other Shading data structures, especially **ShadingIn** and **ShadingOut**.

©2001-2003 Eovia. All rights reserved.

ShadingOut

Description

The **ShadingOut** data structure is filled out by Shaders that implement the **I3DExShader::DoShade** 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 **ShadingOut** data structure is included in a **ShadingInOut** data structure.

Field Index

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).

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::fChangesNormal**).

fOpacity
Opacity index. Used when mixing shaders together such as a Layers List shader.

©2001-2003 Eovia. All rights reserved.

fFullAreaDone

True if the shader was evaluated on the area.

Comments See the description of the Illumination Model in the “Writing a Renderer” chapter in the Cookbook.

UVSpaceInfo

Description

The **UVSpaceInfo** record is filled out by **I3DExGeometricPrimitive::GetUVSpace**, **I3DShObject::GetUVSpace**, and **I3DShInstance::GetUVSpace** to give all the necessary information on the UV Space(s) of the Objects and Instances.

Field Index

Fields

fID

The UV Space ID identifying this UV Space.

fName

The UV Space ID name.

fMin

The minimum u and v values of the UV Space

fMax

The maximum u and v values of the UV Space

fWraparound

This array of two boolean values indicate if the UV Space is closed in the u or v direction. 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.

ShadedArea

zot I only found one reference to this in I3DShInstance. should it be removed?

©2001-2003 Eovia. All rights reserved.

Chapter 5 - Geometry Data Structures Reference

TVERTEX3D

Used with a facet mesh.

```
TVector3 fVertex;
```
X, y, z vertex coordinates.

```
TVector3 fNormal;
```
Nx, Ny, Nz normal values at that vertex.

```
TVector2 fUV;
```
Texture u,v values at that vertex.

TFACET3D

The structure for a facet mesh.

```
TVertex3D fVertices[3];
```
The facet's three vertices. This includes it's vertex coordinates, normal values, and texture uv values.

```
    int16  fUVSpace;
```
The UV Space ID to which this facet belongs.

```
    int16  fReserved;
```
Reserved - 0. **zot** does this mean set it to zero?

TPATCH3D

<zot>need description here

```
TVector3fVertices[4][4];
```
The patches' 16 vertices.

```
    real       fu[2];
```
u values at the patch boundaries.

```
    real       fv[2];
```
v values at the patch boundaries.

```
    int16      fUVSpace;
```
UV Space ID to which this patch belongs.

©2001-2003 Eovia. All rights reserved.

```
    int16     fReserved;
```
Reserved - 0.

```
};
```

FacetMesh

A FacetMesh is a representation for triangular polygon data using corresponding arrays of data.

There are several more routines in PublicUtilities.h for managing facet meshes. Be careful about maintaining the polygon eqn and edge lists.

<zot> what is a "eqn"?

Methods

```
static voidCreate(FacetMesh** outMesh)
```
Creates a facet mesh.

```
void Clone (FacetMesh** clone);
```
Clone's a facet mesh.

```
inline boolean IsEdgeListInvalid() const
    { return fEdgeList.GetElemCount() == 0; }
```
Call this routine anytime the facets change, the edge generators rely on this to generate the edge lists.

```
inline void MarkEdgeListInvalid()
       { fEdgeList.SetElemCount( 0 ); }
```
Call this routine anytime the facets change, the plane generators rely on it to generate the polygon equations. This method doesn't release the memory.

```
inline boolean IsPolygonEqnListInvalid() const
       { return fPolygonEqns.GetElemCount() == 0; }
```
zot needs description.

```
inline void MarkPolygonEqnListInvalid()
    { fPolygonEqns.SetElemCount( 0 ); }
```
This method doesn't release memory.

```
inline void CalcPolygonEqnList()
```
Call this method to calculate the list of plane equations.

These are the plane equations for the polygons (a combination of unit normal and plane coefficients.) MetaCreations uses the plane eqn of the form for compute the plane coefficient::

$$A x + B y + C z + D = 0$$

zot add ECloneMode here? (APITypes.h), or to the basic data structs list?

©2001-2003 Eovia. All rights reserved.

Chapter 6 - Ray Tracing Data Structures Reference

Ray3D

The Ray3D structure is used to describe a light ray. This structure is mostly used by ray-tracing algorithms.

Fields

fOrigin
Origin of the ray in Global Coordinates.

fDirection
Direction of the ray in Global Coordinates. It must be a unit vector.

fOx
Partial derivative of the origin.

fOy
Partial derivative of the origin.

fDx
Partial derivative of the direction.

fDy
Partial derivative of the direction.

zot didn't find these: Remove?

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:

$$Size(t) = fWidthAtFocal + (t - fFocalPoint) * fAngle$$

©2001-2003 Eovia. All rights reserved.

Where ‘t’ is the “ray parameter”: any point on the ray can be written as:

fOrigin + t * *fDirection* with t as a real value.

RayHit3D

The RayHit3D structure is used by **I3DExGeometricPrimitive::RayHit** and **I3DExGeometricPrimitive::RayAllHits**. Its purpose it to return ray-tracing hit information when a hit occurs between a ray and a Primitive’s surface.

Constructor

```
RayHit3D()
```

Fields

```
fShouldSetUV=true;
fShouldSetIsoUV=true;
fRenderInfinitePrimitives=true;
fShouldSetNormalDerivative=false;
```

Methods

```
void CalcInfo(ShadingFlags &shadingFlags,boolean calcDerivatives,Ray3D &ray);
```

Variables

```
real      ft;
```
t parameter along the ray: fPosition = RayOrigin + t * RayDirection. Ray parameter of *fPosition*. *fPosition* = RayOrigin + *ft* * RayDirection.

```
TVector3 fReflectDir;
```
Direction of reflection of the ray.

```
TTransform3DfT;
```
zot Needs description.

```
TTransform3DfInvT;
```
zot Needs description.

```
TMatrix33fNormalDerivative;
```
Derivative of the normal depending on the position in local coordinates.

```
TVector3fUVDerivative[2];
```
Derivative of the UV depending on the position in local coordinates

```
FacetInfo*fFacetInfo;
```
Private data of the raytracer (used in fCalcInfo).

```
real   fBaryCoord[3];
```

©2001-2003 Eovia. All rights reserved.

Barycentric coordinates of the hit point on a hot facet.

```
int32  fFacetIndex;
```
Facet hit (index in facetMesh with best Level Of Detail).

```
FacetMesh*fFacetMesh;
```
Pointer to the facet mesh

```
void (*fCalcInfo)(RayHit3D &me);
```
zot needs description.

```
boolean    fShouldSetUV;
```
Returns True if fUV should be set.

```
boolean    fShouldSetIsoUV;
```
Returns True if fIsoU and fIsoV should be set.

```
boolean    fRenderInfinitePrimitives;
```
Returns True if infinite primitives are rendered.

```
boolean    fShouldSetNormalDerivative;
```
Returns True if fNormalDerivate should be set (it is only needed for reflection and transparency.)

```
boolean    fNormalFlipped;
```
Returns True if the Normal is flipped.

```
boolean    fIsShadowCatcher;
```
True if the object is a shadowcatcher.

```
real       fShadowCatchingCoef;
```
Returns the percentage of the light that is not intercepted by an object (computed only for shadow catcher.)

Comments See the sphere example in the geometric primitive cookbook for more information on this structure.

RayHitParameters

The RayHitParameters structure is used by **I3DShInstance::RayHit** and **I3DExGeometricPrimitive::RayHit**. to calculate the hit of a ray on an object.

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.

**ray*

zot needs description.

**hit*

zot needs description.

Comments See **Ray3D** structure for more details.

©2001-2003 Eovia. All rights reserved.

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::GetRayHitDetails** is called.

Constructor

```
HitInfo()
```

Variables

```
fInstance=nil;
fNeedsColor=false;
fNeedsDistance=false;
fNeedsPosition=false;
fNeedsNormal=false;
fNeedsAlpha=false;
fNeedsIndex=false;
fNeedsUV=false;
fZBShading=false;
fFilterInstanceProc=nil;
```

Output data

```
TInstance* fInstance;
```
Instance that was hit by the ray.

```
VECTOR2D fUV;
```
(u,v) texture values at the intersection point. To be filled only if *fNeedsUV* is TRUE.

```
uint32  fUVSpaceID;
```
UV Space ID at the intersection point.

```
VECTOR3D fPosition;
```
Intersection point in Global Coordinates.To be filled only if *fNeedsPosition* is TRUE.

```
VECTOR3D fNormal;
```
Normal at the intersection point in Global Coordinates. To be filled only if *fNeedsNormal* is TRUE.

```
COLOR3D fColor;
```
Rendered final color (all lighting and shading included). To be filled only if *fNeedsColor* is TRUE.

```
NUM3D  fAlpha;
```
Alpha-channel value. To be filled only if *fNeedsAlpha* is TRUE.

```
uint32 fIndex;
```
Instance unique index. To be filled only if *fNeedsIndex* is TRUE.

```
NUM3D  fDistance;
```
Distance from the point of view to the intersection point. To be filled only if *fNeedsDistance* is TRUE.

Input flags

```
boolean fNeedsColor;
```

©2001-2003 Eovia. All rights reserved.

Returns True if Needs fColor.

```
boolean fNeedsDistance;
```
Returns True if Needs fDistance.

```
boolean fNeedsPosition;
```
Returns True if Needs fPosition.

```
boolean fNeedsNormal;
```
Returns True if Needs Normal.

```
boolean fNeedsAlpha;
```
Returns True if Needs Alpha.

```
boolean fNeedsIndex;
```
Returns True if Needs Instance index.

```
boolean fNeedsUV;//
```
Returns True if Needs (u,v) values. These flags control the calculation of the returned data.

```
boolean fZBShading;
```
zot, this was comment: Ask for Z-Buffer-like shading: no transparencies, no reflections and no shadows

```
FilterHitProc fFilterInstanceProc;
```
Filterproc for instances. Used only in fast hit mode when fNeedsColor is false.

Comments See the sphere example in the geometric primitive cookbook chapter for more information on this structure.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

Chapter 7 - 3D IEx* Reference

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.1 I3DExAmbientLight Interface

Family ID: ‘ambi’

Interface ID: IID_I3DExAmbientLight

Interface file: I3DExPrimitive.h

Basic Implementation file: BasicEnvironment.h, BasicEnvironment.cpp

Description

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

I3DExAmbientLight::GetColor

MCCOMErr I3DExAmbientLight::GetColor(*TMCColorRGB* resultColor***);**

Gets the color of the ambient light.

Parameters **TMCColorRGB*** *resultColor*

The color of the ambient light to return. This is global for the scene.

Return Value MC_E_NOTIMPL

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.2 I3DExAtmosphericShader Interface

Family ID: ‘atmo’

Interface ID: IID_I3DExAtmosphericShader

Interface file: I3DExPrimitive.h

Basic Implementation file: BasicEnvironment.h, BasicEnvironment.cpp

Description

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

The following two functions are implemented:

- **SegmentFilter(TVector3& beg, TVector3& end, TMCColorRGB& inOutFilter)**
- **DirectionFilter(TVector3& origin, TVector3& direction, TMCColorRGB& inOutFilter)**

Methods

“I3DExAtmosphericShader::SegmentFilter”
“I3DExAtmosphericShader::DirectionFilter”

I3DExAtmosphericShader::SegmentFilter

MCCOMErr I3DExAtmosphericShader::SegmentFilter(*TVector3& beg, TVector3& end, TMCColorRGB& inOutFilter***)**

SegmentFilter is called to filter the light beam color between two points.

Parameters

TVector3& *beg*
This point is the beginning point of the light beam in Global Coordinates.

TVector3& *end*
This point is the ending point of the light beam in Global Coordinates.

TMCColorRGB& *inOutFilter*
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 False is MC_E_NOTIMPL
True is S_OK

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

MCCOMErr I3DExAtmosphericShader::DirectionFilter(*TVector3& origin, TVector3& direction, TMCColorRGB& inOutFilter***)**

©2001-2003 Eovia. All rights reserved.

DirectionFilter works like **SegmentFilter**, 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

False is MC_E_NOTIMPL
True is S_OK

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.

©2001-2003 Eovia. All rights reserved.

7.3 I3DExBackground Interface

Family ID: ‘back’

Interface ID: IID_I3DExBackground

Interface file: I3DExPrimitive.h

Basic Implementation file: BasicEnvironment.h, BasicEnvironment.cpp

Description

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

I3DExBackground::GetBackgroundColor

MCCOMErr I3DExBackground::GetBackgroundColor(*TMCColorRGB& color,boolean& fullAreaDone,const TVector3& direction***);**

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

Parameters

TMCColorRGB& *color*
Return the calculated color here.

boolean& ***fullAreaDone***
Return True if the full pixel area has been shaded (not used, return false for the moment).

TVECTOR3D& *direction*
The unit vector defining the direction of the light beam.

Return Value MC_E_NOTIMPL

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:

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.4 I3DExBackdrop Interface

Family ID: ‘drop’

Interface ID: IID_I3DExBackdrop

Interface file: I3DExPrimitive.h

Basic Implementation file: BasicEnvironment.h, BasicEnvironment.cpp

Description

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

The following function is implemented:

- **GetBackdropColor(TMCColorRGB& color,boolean& fullAreaDone,const TBBox2D& screenMinMax,const TVector2& uv)**

I3DExBackdrop::GetBackdropColor

MCCOMErr I3DExBackdrop::GetBackdropColor(*TMCColorRGB& color,boolean& fullAreaDone,const TBBox2D& screenMinMax,const TVector2& uv)***;**

GetBackdropColor is called to return the backdrop color information.

Parameters

TMCColorRGB& *color*
Return the calculated color here.

boolean& ***fullAreaDone***
Return True if the full area of the pixel has been shaded.

TBBOX2D& ***screenMinMax***
Box corresponding to the production frame in screen coordinates.

const TVector2& *uv*
Position on the screen.

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.

You need to use the screenMinMax coordinates because the uv coordinates that are given to you might be outside the production frame (for example when the marquee rendering tools are used in the interactive preview).

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.5 I3DExCamera Interface

Family ID : ‘came’

Interface ID : IID_I3DExCamera

Interface file : I3DExCamera.h

Basic Implementation files: BasicCameraLightGel.h, BasicCameraLightGel.cpp

Description

Cameras define the projection from the 3D world to the 2D screen on which the image is rendered. There are two common forms of projection: orthographic and perspective.

An orthographic camera takes rays parallel to the z-axis and projects them on the x-y plane by removing z.

A perspective camera takes all rays coming to a point and intersects those rays with a plane some distance from the point. The distance to this plane is the focal length. As with a real camera, it is possible to implement various lenses, focal values, 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 three basic methods. **Project3DTo2D()** is used to create the projection from 3D space onto the screen. This call is used by the hybrid raytracer. The two **CreateRay()** function calls create a 3D ray from a screen point. These calls are used for the ray-tracing. (You need to implement both **CreateRay()** calls.)

For your camera to be used by the interactive renderer, you must implement **GetStandardCameraInfo()**. For performance reasons, the interactive renderer requires a standard isometric or canonical camera. The parameters for this standard camera are supplied by **GetStandardCameraInfo()**.

Method Index

“I3DExCamera::SetTransform(constTTransform3D&)”
“I3DExCamera::CreateRay(const TVector2&, TVector3&, TVector3&)”
“I3DExCamera::CreateRay(const TVector2&, const TVector2&, Ray3D&)”
“I3DExCamera::IsProjectionAffine(void)”
“I3DExCamera::PreProject3DTo2D(const TVector3&, TVector4&)”
“I3DExCamera::Project3DTo2D(const TVector3&, TVector3&)”
“I3DExCamera::GetPrimitiveID()”
“I3DExCamera::GetStandardCameraInfo(TStandardCameraInfo&)”
“I3DExCamera::SetHalfWidth(real)”

I3DExCamera::SetTransform(constTTransform3D&)

MCCOMErr I3DExCamera::SetTransform(const TTransform3D& transform)

I3DExCamera::SetTransform() is used to store the camera transformation data.

Parameter **const TTransform3D&** *transform*
transform is a reference to the transformation data. Copy its contents to an internal variable.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

Comments Pre-processing calculations for the camera extension can occur in **SetTransform()**. You should cache this transform in your camera and use it to implement **CreateRay()** .

I3DExCamera::CreateRay(const TVector2&, TVector3&, TVector3&)

boolean I3DExCamera::CreateRay(const TVector2& screenPosition, TVector3& worldOrigin, TVector3& worldDirection)

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

Parameters

const TVector2& *screenPosition*
Position on the screen in screen coordinates. (This can be any floating point value as the camera screen is considered infinite.) Values between -1 and 1 corresponds to the Production Frame.

TVector3& *worldOrigin*
worldOrigin must return the screen point, in 3D, in the Global Coordinates System. You need to use the transformation provided by SetTransform() to perform those calculations.

TVector3& *worldDirection*
worldDirection must return the direction vector of the straight line. This must be a unit vector.

Return Value MC_S_OK

I3DExCamera::CreateRay(const TVector2&, const TVector2&, Ray3D&)

boolean CreateRay(const TVector2& screenPosition, const TVector2& screenDerivate, Ray3D& ray)

I3DExCamera::CreateRay() is called by the renderer to get a ray and its derivatives.

Parameters

const TVector2& *screenPosition*
Position on the screen (can be any floating point value as the camera screen is considered infinite). Values between -1 and 1 corresponds to the Production Frame.

TVector2& *screenDerivate*
screenDerivate specifies the size of a pixel on the screen in screen coordinates.

Ray3D& *ray*
ray specifies the specific ray being created. You must fill fOx,fOy with the variation of the origin of the ray over a pixel and fDx,fDy with the variation of the direction over a pixel. You should use the value given in screeDerivate for that.

Return Value MC_S_OK

I3DExCamera::IsProjectionAffine(void)

boolean IsProjectionAffine(void)

I3DExCamera::IsProjectionAffine returns true if the w component from the projection is always 1.0.

Return Value

TRUE	w is always 1.0
FALSE	w is not always 1.0

©2001-2003 Eovia. All rights reserved.

I3DExCamera::PreProject3DTo2D(const TVector3&, TVector4&)

boolean PreProject3DTo2D(const TVector3& cameraPosition, TVector4& projectedPosition)

I3DExCamera::PreProject3DTo2D is called to project a 3D point on the screen.

Parameters

const TVector3& *cameraPosition*
The 3D position of the camera.

TVector4& ***projectedPosition***
Projection on the screen.

Return Value

TRUE	The point is visible.
FALSE	The point is not visible.

I3DExCamera::Project3DTo2D(const TVector3&, TVector3&)

boolean Project3DTo2D(const TVector3& cameraPosition, TVector3& screenPosition)

I3DExCamera::Project3DTo2D is called to project a 3D point on the screen.

Parameters

const TVector3& *cameraPosition*
The 3D position in camera space.

TVector3& ***screenPosition***
Projection on the screen in screen coordinates.

Return Value

TRUE	The point is visible.
FALSE	The point is not visible.

I3DExCamera::GetPrimitiveID()

uint32 GetPrimitiveID();

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

Return Value The primitive ID.

I3DExCamera::GetStandardCameraInfo(TStandardCameraInfo&)

void GetStandardCameraInfo(TStandardCameraInfo& cameraInfo)

I3DExCamera::GetStandardCameraInfo is used to return standard camera info used by the interactive renderer.

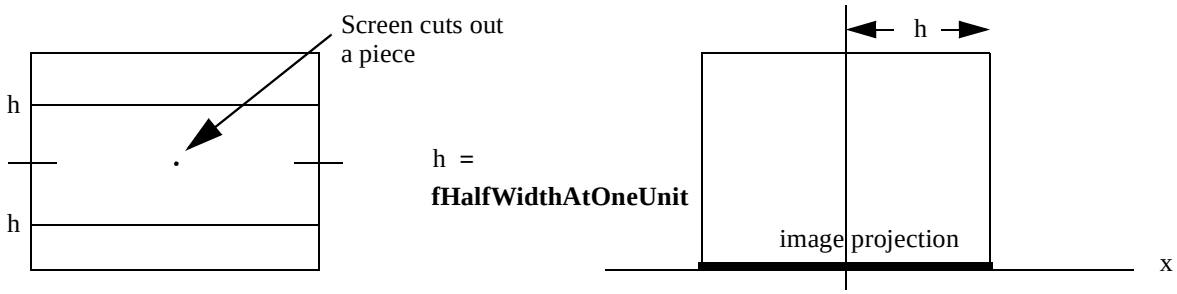
TStandardCameraInfo has the following fields:

- **EProjectionType fProjectionType** - pick the type of camera that you are most closely approximating between conical (perspective) and isometric (orthographic).
- **ECanonicalCameraType fCanonicalCameraType** - for internal use only (ignore).

©2001-2003 Eovia. All rights reserved.

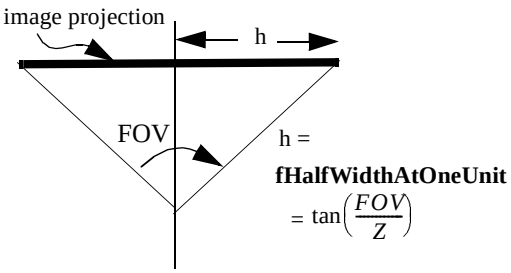
- **real fHalfWidthAtOneUnit** - the amount of camera space units at one unit from the camera location. In the case of an orthographic camera, this represents the zoom factor. For the perspective camera, this represents the field of view (FOV), or the angle of the most extreme rays that are in the image. Any rays outside of the FOV are not represented in the image. (See diagrams below for graphical description.)
- **boolean fIsMatching** - indicates whether the standard camera info is an exact or approximate conversion. For example, a spherical lens may only approximate itself with one of the types.

As you can see from the diagram below, for an orthographic camera, a large **fHalfWidthAtOneUnit** means zoom out.



Orthographic Camera

As shown below, with a perspective camera, a large **fHalfWidthAtOneUnit** means the FOV is large, like a wide angle lens, and the image is zoomed out.



h	θ	Zθ=FOV
1	45	90
0.5	26	52
2	63	126

Perspective Camera

$$FOV = Z\tan^{-1}(h)$$

Parameters **TStandardCameraInfo&** *cameraInfo*
The camera info.

I3DExCamera::SetHalfWidth(real)

MCCOMErr SetHalfWidth(real value)

I3DExCamera::SetHalfWidth is used to explicitly specify the viewable area. It is typically used for isometric (orthographic) cameras.

Parameters **real** *value*
value is the HalfWidth of the camera screen. For an isometric camera, *value* is 20.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.6 I3DExConstraint

Interface ID: IID_I3DExConstraint

Interface File: I3DExConstraint.h

Description

I3DExConstraint provides constraints between Tree Elements. A constraint, known as a link in RDS, defines motion links between Tree Elements. It allows you to define “mechanical” constrains (degrees of freedom) between one Tree Element and its father.

Method Index

“I3DExConstraint::GetNbrFreedom()”
“I3DExConstraint::IncrementFreedomValue(int16 , real&)”
“I3DExConstraint::GetMotionLimit (int16, ELimitType&)”
“I3DExConstraint::GetFreedomRange(int16, real&, real&)”
“I3DExConstraint::GetTransform(TTransform3D&)”
“I3DExConstraint::GetTransformPartialDerivate(int16, TTransform3D&)”
“I3DExConstraint::GetTransformSecondPartialDerivative (int16, int16, TTransform3D&)”
“I3DExConstraint::SetTransform(TTransform3D&)”
“I3DExConstraint::IsTransformResetRequired()”
“I3DExConstraint::ResetConstraint()”

I3DExConstraint::GetNbrFreedom()

int16 MCCOMAPI GetNbrFreedom()

GetNbrFreedom() lets the 3D Shell know the number of degrees of freedom. Because the freedom value can be limited, you have to give the range of values. This range is defined as a range of increment/decrement around the value.

I3DExConstraint::IncrementFreedomValue(int16 , real&)

MCCOMErr MCCOMAPI IncrementFreedomValue(int16 index, real& increment)

<zot>Need method description</zot> The Shell can’t directly modify the freedom value, so you need to implement **IncrementFreedomValue()**.

Parameters

int16 *index*
<zot>Need Description</zot>

real& *increment*
<zot>Need Description</zot>

Return Value MC_S_OK

I3DExConstraint::GetMotionLimit (int16, ELimitType&)

MCCOMErr MCCOMAPI GetMotionLimit (int16 index, ELimitType& limit)

©2001-2003 Eovia. All rights reserved.

<zot>Need method description</zot>

Parameters

int16 *index*
<zot>Need Description</zot>

ELimitType *limit*
<zot>Need Description</zot>

I3DExConstraint::GetFreedomRange(int16, real&, real&)

MCCOMErr MCCOMAPI GetFreedomRange(int16 index, real& min, real& max)

GetFreedomRange() returns the return the maximum range of increment and decrement.

Parameters

uint32 *index*
<zot>Need Description</zot>

real *min*
min is the minimum increment value.

real *max*
max is the maximum increment value.

I3DExConstraint::GetTransform(TTransform3D&)

MCCOMErr MCCOMAPI GetTransform(TTransform3D& transform)

GetTransform() gives the rotation matrix and translation vector due to the freedom values.

Note: When the freedom values are initialized the function **GetTransform** must return the identity matrix for the rotation and a null vector for the translation.

Parameters

TTransform3D& *transform*
Reference to the rotation matrix and translation vector. See TTransform in the data structures chapter.

I3DExConstraint::GetTransformPartialDerivate(int16, TTransform3D&)

MCCOMErr MCCOMAPI GetTransformPartialDerivate(int16 index, TTransform3D& transform)

To allow the inverse kinematics mechanism to work with a constraint, implement the **GetTransfromPartialDerivate()** function. A partial derivative transformation is a transformation where each element is derived by a freedom degree.

Parameters

int16 *index*
<zot>Need Description</zot>

TTransform3D& *transform*
Reference to the rotation matrix and translation vector. See TTransform in the data structures chapter.

©2001-2003 Eovia. All rights reserved.

I3DExConstraint::GetTransformSecondPartialDerivative (int16, int16, TTransform3D&)

MCCOMErr MCCOMAPI GetTransformSecondPartialDerivative (int16 index1, int16 index2, TTransform3D& transform)

<zot>Need method description</zot>

Parameters

int16 *index1*
<zot>Need Description</zot>

int16 *index2*
<zot>Need Description</zot>

TTransform3D& *tranform*
Reference to the rotation matrix and translation vector. See TTransform in the data structures chapter.

I3DExConstraint::SetTransform(TTransform3D&)

MCCOMErr MCCOMAPI SetTransform(TTransform3D& transform)

SetTransform() sets the rotation matrix and translation vector .

Parameters

TTransform3D& *tranform*
Reference to the rotation matrix and translation vector. See TTransform in the data structures chapter.

I3DExConstraint::IsTransformResetRequired()

virtual boolean MCCOMAPI IsTransformResetRequired()

Call **IsTransformResetRequired()** after an event in the constraint chooser. A reset is required when the constraint has changed in nature or in axis. That is when the transform will no longer match. The default is return true if you did not handle the change.

I3DExConstraint::ResetConstraint()

MCCOMErr MCCOMAPI ResetConstraint()

Call **ResetConstraint()** to reset the constraint after **IsTransformResetRequired()** returns True.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.7 I3DExDataComponent

Interface ID: IID_I3DExDataComponent

Interface File: I3DRenderFeature.h

Description

I3DExDataComponent allows you to check whether a data component has an effect on an object. You can also get a list of the postrenderers that are enabled by a data component. See the chapter in the cookbook **zot** Creating Your Own Component for more information on data components.

Method Index

"I3DExDataComponent:: GetPostRenderList(TMCArray<IDType> &)"
"I3DExDataComponent::IsActive(I3DShTreeElement*)"

I3DExDataComponent:: GetPostRenderList(TMCArray<IDType> &)

MCCOMErr MCCOMAPI GetPostRenderList(TMCArray<IDType> &idArray)

GetPostRenderList() fills the array with the class IDs of the postrenderers that are enabled by this dataComponent.

Parameters **TMCArray<IDType> &** *idArray*
<zot>need description</zot> Array containing a list of postrenderer class IDs.

Returns MC_E_NOTIMPL

I3DExDataComponent::IsActive(I3DShTreeElement*)

boolean MCCOMAPI IsActive(I3DShTreeElement* tree)

IsActive() returns false if the dataComponent has no effect on this object. By default data components are always active.

Parameters **I3DShTreeElement*** *tree*
The tree of the object.

Returns True

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.8 I3DExExportFilter Interface

Family ID: ‘3Dou’

Interface ID: IID_I3DExExportFilter

Interface file: I3DExImportExport.h

Basic Implementation files: Basic3DImportExport.h

Description

An export filter is a plugin that can save a scene or part of a scene to a file format different from the Carrara file format.

Example: the Obj Export is implemented as an export filter.

Method Index

“I3DExExportFilter::Prepare”
“I3DExExportFilter::WantsOptionDialog”
“I3DExExportFilter::NeededDiskSpace”
“I3DExExportFilter::DoExport”
“I3DExExportFilter::WantsPreview”
“I3DExExportFilter::GetPreviewSize”
“I3DExExportFilter::SetPreview”
“I3DExExportFilter::WantsTempFile”

I3DExExportFilter::Prepare

virtual boolean MCCOMAPI Prepare(I3DShScene* scene, I3DShTreeElement* fatherTree) = 0;

Prepare is called before the actual export starts. Usually you do not need to implement this method.

Parameters

I3DShScene* *scene*
scene is the scene that contains the geometry that will be exported.

I3DShTreeElement* *fatherTree*
fatherTree is the root of the sub tree that needs to be exported. (It should never be NULL)

I3DExExportFilter::WantsOptionDialog

virtual boolean MCCOMAPI WantsOptionDialog() = 0;

WantsOptionDialog returns true if an option dialog should be opened.

I3DExExportFilter::NeededDiskSpace

virtual MCCOMErr MCCOMAPI NeededDiskSpace(IMCFile * file, I3DShScene* scene, I3DShTreeElement* father-Tree, uint32* diskSpace) = 0;

©2001-2003 Eovia. All rights reserved.

NeededDiskSpace returns the amount of disk space that is necessary to perform the export. (usually you do not need to implement this method)

Parameters

IMCFile* *file*
file is a pointer to the file where you should write your data.

I3DShScene* *scene*
scene is the scene that contains the geometry that will be exported.

I3DShTreeElement* *fatherTree*
fatherTree is the root of the sub tree that needs to be exported. (It should never be NULL)

uint32* *diskSpace*
diskSpace is the amount of disk space in bytes necessary to save the file.

I3DExExportFilter::DoExport

virtual MCCOMErr MCCOMAPI DoExport(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree);

DoExport is where the actual export takes place. It must save the sub tree defined by fatherTree into the file.

Parameters

IMCFile* *file*
file is the file where the data must be exported.

I3DShScene* *scene*
scene is the scene that contains the geometry that will be exported.

I3DShTreeElement* *fatherTree*
fatherTree is the root of the sub tree that needs to be exported. (It should never be NULL)

I3DExExportFilter::WantsPreview

virtual boolean MCCOMAPI WantsPreview() = 0;

Not documented yet.

I3DExExportFilter::GetPreviewSize

virtual MCCOMErr MCCOMAPI GetPreviewSize(int16* sizeh, int16* sizev) = 0;

Not documented yet.

I3DExExportFilter::SetPreview

virtual MCCOMErr MCCOMAPI SetPreview(TMCPixelBucket* preview) = 0;

Not documented yet.

©2001-2003 Eovia. All rights reserved.

I3DExExportFilter::WantsTempFile

virtual boolean MCCOMAPI WantsTempFile() = 0;

WantsTempFile returns true if the exporter uses the standard save method (will be given a tmp file in doexport), false if the exporter needs to change the file name (eg is writing a sequence)

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.9 I3DExFinalRenderer Interface

Family ID: ‘frnd’

Interface ID:IID_I3DExFinalRenderer

Interface file: I3DExFinalRenderer.h

Basic Implementation files: BasicFinalRenderer.h, BasicFinalRenderer.cpp

Description

Rendering means building a 2D image from a 3D scene. Once the image is rendered, the post render filters can be applied.

Method Index

“I3DExFinalRenderer::SetTreeTop”
“I3DExFinalRenderer::SetCamera”
“I3DExFinalRenderer::SetEnvironment”
“I3DExFinalRenderer::SetAmbientLight”
“I3DExFinalRenderer::SetFieldRenderingData”
“I3DExFinalRenderer::GetRenderStatistics”
“I3DExFinalRenderer::PrepareDraw”
“I3DExFinalRenderer::FinishDraw”
“I3DExFinalRenderer::IsSMPAware”
“I3DExFinalRenderer::Abort”
“I3DExFinalRenderer::RenderVolumetrics”
“I3DExFinalRenderer::RenderBackGround”
“I3DExFinalRenderer::OptimizeTileSize”
“I3DExFinalRenderer::GetTileRenderer”

I3DExFinalRenderer::SetTreeTop

virtual MCCOMErr MCCOMAPI SetTreeTop(I3DShGroup* treeTop)

The **SetTreeTop()** function specifies which tree needs to be rendered (generally the universe).

Parameter **I3DShGroup*** *treeTop*
treeTop is the top of the hierarchy tree.

Return Value MC_S_OK

I3DExFinalRenderer::SetCamera

virtual MCCOMErr MCCOMAPI SetCamera(I3DShCamera* camera)

The **SetCamera()** function specifies the rendering camera.

Parameter **I3DShCamera*** *camera*
camera is the specified camera.

©2001-2003 Eovia. All rights reserved.

Return Value MC_S_OK

I3DExFinalRenderer::SetEnvironment

virtual MCCOMErr MCCOMAPI SetEnvironment(I3DShEnvironment* environment)

The **SetEnvironment()** function sets up the background, backdrop, and atmospheric shader.

Parameter **I3DShEnvironment*** *environment*
environment is the specified environment.

Return Value MC_S_OK

I3DExFinalRenderer::SetAmbientLight

virtual MCCOMErr MCCOMAPI SetAmbientLight(const TMCColorRGB& ambiantColor)

The **SetAmbientLight()** function sets the ambient light color.

Parameter **TMCColorRGB&** *ambiantColor*
ambiantColor is the color of the ambient light.

Return Value MC_S_OK
Comments

I3DExFinalRenderer::SetFieldRenderingData

virtual MCCOMErr MCCOMAPI SetFieldRenderingData(boolean useFieldRendering,int16 firstFrame)

The **SetFieldRenderingData()** function is used to optimized the rendering of FieldRendering for video output. This function is called once before the rendering starts. If FieldRendering is on, the rendering Module will render twice as many frames as it will output, then it will mix the even and odd lines from two frames to build one frame.

Parameter **boolean** *useFieldRendering*
useFieldRendering is true if fieldRendering is on.

int16 *firstFrame*
firstFrame specifies the number of the first frame to be rendered.

Return Value MC_S_OK

I3DExFinalRenderer::GetRenderStatistics

virtual void MCCOMAPI GetRenderStatistics(I3DRenderStatistics renderstats)**

The **GetRenderStatistics()** function returns a pointer to the current renderStatistics. It is called several time during the rendering to read the progress of the rendering and update the rendering statistic panel in the Render room. If you did not implement any rendering statistics you can return NULL.

If you want to add render statistics, your renderer must create the renderStatistics object which implements

©2001-2003 Eovia. All rights reserved.

I3DRenderStatistics (for more information on renderStatistics see I3DRenderStatitics.).

Parameter **I3DRenderStatistics**** *renderstats*
renderstats is a pointer to the renderStatistics of the renderer.

I3DExFinalRenderer::PrepareDraw

virtual MCCOMErr MCCOMAPI PrepareDraw(const TMCPoint&size, const TBBox2D&uvBox, const TBBox2D &productionFrame, boolean(*callback) (int16 vv, void* priv), void*priv)

The **PrepareDraw()** function is called at the beginning of the rendering of each frame. You can use it to precalculate and store some information that will be needed for each tile of the frame. For example you can get the list of the instances and some information about them.

Note: this function is called only once per frame even in SMP (multi processor mode).

If you want to use the shadow features of the lights you want to initialize them here by calling **I3DExShadowFeature::BeginRenderFrame** on each of them.

Parameter **TMCPoint&** *size*
size defines the size in pixels of the image being rendered.

TBBox2D& *uvBox*
uvBox is the is the box in screen coordinates (camera uv) corresponding to the image being rendered. Note that it is not always within the Production Frame, for example, when you do a Test Render in the Assemble room.

TBBox2D& *productionFrame*
productionFrame is the box in screen coordinates (camera uv) corresponding to the Production Frame (the image being rendered in final rendering). You can use this parameter to make sure that the size of some effects does not change when you use the Test Render tool. (For example, when you do a Test Render of the backdrop which expects screen coordinates between 0 and 1 matching the Production Frame.)

boolean (* *callback) (***int16** *vv,* **void*** *priv)*
callback is used for updating the progress bar. You need to call it when you want the progress bar information to be updated. The second parameter, *priv,* is passed to **PrepareDraw**.

void* *priv*
priv is the parameter of the progress bar callback.

Return ValueMC_S_OK

I3DExFinalRenderer::FinishDraw

virtual MCCOMErr MCCOMAPI FinishDraw()

The **FinishDraw()** function is called at the end of the rendering of each frame. You can use this function to delete the temporary rendering caches.

Note, that it is called only once even in SMP.

Return Value **MC_S_OK**

©2001-2003 Eovia. All rights reserved.

I3DExFinalRenderer::IsSMPAware

virtual boolean MCCOMAPI IsSMPAware()

The **IsSMPAware()** function returns True if the renderer supports SMP (multiprocessors) which means that *drawRect* is reentrant, returns False otherwise.

Return Value True if SMP is supported, otherwise, False.

I3DExFinalRenderer::Abort

virtual MCCOMErr MCCOMAPI Abort()=0;

The **Abort** function is called by the shell to stop the rendering. Usually the actal rendering is taking place in a different thread so you should set a boolean to tell the rendering thread to abort. The rendering thread should test this boolean and stop rendering as soon as it is set.

Return Value Should always return MC_S_OK

I3DExFinalRenderer::RenderVolumetrics

virtual void MCCOMAPI RenderVolumetrics(boolean renderEffects)=0;

The **RenderVolumetrics** function is called by the shell to enable/disable the rendering of volumetric effects (such as lightcones).

Parameter **boolean** *renderEffects*
renderEffects is true if the volumetric effects should be rendered.

Return Value Should always return MC_S_OK

I3DExFinalRenderer::RenderBackGround

virtual void MCCOMAPI RenderBackGround(boolean enable)=0;

The **RenderBackGround** function turns on/off the rendering of the background of the image (including the backdrop) . When the background rendering is turned off it should be black). This is used for alpha channel rendering/compositing

Parameter **boolean** *enable*
enable is true if the background should be rendered.

Return Value Should always return MC_S_OK

I3DExFinalRenderer::OptimizeTileSize

virtual void MCCOMAPI OptimizeTileSize(TMCPoint& inOutTileSize) = 0;

The **OptimizeTileSize()** function optimizes the tile size for the rendering. The shell calls this functions with its prefered size. The renderer can only increase the dimensions of the tile. You might want for example to get a power of 2 for the size or you might want to create very big tiles. If you support SMP, you might want to have small tiles so that the processor time is evenly shared between the processors.

Note that this is only a preferred size which means that some tiles will be smaller or bigger. This means that

©2001-2003 Eovia. All rights reserved.

you must support tiles of any size.

Parameter **const TMCPoint& inOutTileSize**
inOutTileSize is the preferred size of the tile (in pixels).

Return Value No return value

I3DExFinalRenderer::GetTileRenderer

virtual void MCCOMAPI GetTileRenderer(I3DExTileRenderer tileRenderer)=0;**

The **GetTileRenderer()** function returns a pointer to a new tile renderer. If your renderer supports multi-threading (ie returns true in **IsSMPAware**), you should create a new object for every call to this function. You can initialiase the tile renderer with the data that were passed to you in **PrepareDraw**. For more information on how to implement a tile renderer see **I3DExTileRenderer.**

This function is called once per frame for each rendering thread. If you need to build some caches shared by all tiles you can do that in **PrepareDraw** which is called only once per frame.

Note that a given tile renderer will usually render more than one tile.

Parameter **I3DExTileRenderer**** *tileRenderer*
tileRenderer is a pointer the a new tile renderer.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.10 I3DExImportFilter Interface

Family ID: '3Din'

Interface ID: IID_I3DExImportFilter

Interface file: I3DExImportExport.h

Basic Implementation files: Basic3DImportExport.h

Description

An import filter is a plugin that can open a file (with Open) or import a file into a scene (with Import).

Example: the Obj Import is implemented as an import filter.

Method Index

"I3DExImportFilter::Prepare"
"I3DExImportFilter::WantsOptionDialog"
"I3DExImportFilter::DoImport"
"I3DExImportFilter::WantsTopScene"

I3DExImportFilter::Prepare

virtual boolean MCCOMAPI Prepare(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree) = 0;

Prepare is called before the actual import starts. Usually you do not need to implement this method.

Parameters

IMCFile* *file*
file is the file that is going to be imported.

I3DShScene* *scene*
scene is the scene that contains the geometry that will be exported.

I3DShTreeElement* *fatherTree*
fatherTree is the root of the imported scene. You should add all the imported instances/groups as child of it. If fatherTree is NULL, then you should import everything in the Universe of the scene.

I3DExImportFilter::WantsOptionDialog

virtual boolean MCCOMAPI WantsOptionDialog() = 0;

WantsOptionDialog returns true if the shell should display the option dialog.

©2001-2003 Eovia. All rights reserved.

I3DExImportFilter::DoImport

virtual boolean MCCOMAPI DoImport(IMCFile * file, I3DShScene* scene, I3DShTreeElement* fatherTree) = 0;

DoImport is where the actual import should take place. This method read the file and create the instances and primitives and adds them to the scene tree under the fatherTree node.

Parameters

IMCFile* *file*
file is the file that is going to be imported.

I3DShScene* *scene*
scene is the scene that contains the geometry that will be exported.

I3DShTreeElement* *fatherTree*
fatherTree is the root of the imported scene. You should add all the imported instances/groups as child of it. If fatherTree is NULL, then you should import everything in the Universe of the scene.

I3DExImportFilter::WantsTopScene

virtual boolean MCCOMAPI WantsTopScene() = 0;

WantsTopScene is called by the shell to determine if objects should placed in the main scene or in a scene instance (=master group)

Note: This call is only used for "import"and not for "open".

Return Value true if the import should be placed in the main scene, false if they should be placed in a new Master Group.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.11 I3DExInvertibleTweener Interface

Interface ID: IID_I3DExInvertibleTweener

Interface file: I3DExTweener.h

Description

The **I3DExInvertibleTweener** Interface defines invertible Animation Tweeners. Tweeners are 3D Components that describe the interpolation between key frames.

I3DExTweener::Inverse()

MCCOMErr MCCOMAPI Inverse()

Inverse() is called by the 3D Shell to perform an inverse of a tweener.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.12 I3DExLightsource Interface

Family ID: ‘lite’

Interface ID: IID_I3DExLightsource

Interface file: I3DExLight.h

Basic Implementation file: BasicCameraLightGel.h, BasicCameraLightGel.cpp

Description

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 of 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 **I3DExLightsourceGel** interface).

Method Index

“I3DExLightsource::SetTransform(TTransform3D*)”
“I3DExLightsource::GetDirection(TVector3&, TVector3&, real&)”
“I3DExLightsource::GetColor(TVector3&, TVector3&, real, TMCColorRGB&, real&)”
“I3DExLightsource::IsVisibleInPerspective()”
“I3DExLightsource::GetPrimitiveID()”
“I3DExLightsource::ForEachShadowBuffer(ForEachShadowBufferCallback, void*)”
“I3DExLightsource::GetLightParameter(int32, void*)”
“I3DExLightsource::GetLightInfo(boolean&, boolean&, real&, TTransform3D&)”
“I3DExLightsource::GetUVfromDirection(TVector2&, const TVector3&)”
“I3DExLightsource::GetNewTrace(int16*)”
“I3DExLightsource::GetStandardLight(TStandardLight&)”

I3DExLightsource::SetTransform(TTransform3D*)

MCCOMErr MCCOMAPI SetTransform(TTransform3D* transform)

SetTransform is called to tell a light source its position and orientation in space. This function is used to cache transform information to speed up **GetDirection()** and **GetColor().**

Parameters **TTransform3D*** *transform*
Points to the transformation of the light source.

I3DExLightsource::GetDirection(TVector3&, TVector3&, real&)

MCCOMErr MCCOMAPI GetDirection(const TVector3&point, TVector3&resultDirection, real&resultDistance)

GetDirection() is called prior to **GetColor()** to perform preliminary calculations and return the distance between the surface point and light source. The surface point is the point on the object where the light is about to be calculated.

©2001-2003 Eovia. All rights reserved.

Parameters

TVector3& *point*
The position of the point on the object's surface in Global Coordinates.

TVector3& *resultDirection*
Returns 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. This vector should be normalized (norm equal to 1).

real& *resultDistance*
Returns the distance between the point and the light source.

Comment

A light bulb typically calculates *resultDirection* by subtracting the postion point to the light source origin, then normalizes the results. A simulated sun light returns its light direction in *resultDirection*, and the maximum possible value in *resultDistance*.

I3DExLightsource::GetColor(TVector3&, TVector3&, real, TMCColorRGB&, real&)

boolean MCCOMAPI GetColor(const TVector3&point, const TVector3&direction, const realdistance,/ TMCColorRGB&result, real&ShadowIntensity)

GetColor is called to calculate the color of light seen through a given point. The component has total control over the color to return.

Parameters

TVector3& *point*
The 3D position of the point to be lit, in Global coordinates.

TVector3& *direction*
The direction from the point to the light, in Global coordinates, as calculated by **GetDirection()**.

const real *distance*
The distance from the light to the point, as calculated by **GetDirection()**.

TMCColorRGB& *result*
The resulting calculated color. (Note that the value can be more than 1.0f.)

real& *ShadowIntensity*
Returns how clearly this light cast shadows. Between 0.0f (no shadows) and 1.0f (no light in shadow).

Comment

When using a spot, use the *position* parameter and the light source position and orientation to determine whether a point is in the light beam or not.
When using a sun, use the *direction* parameter to calculate the intensity.
When lighting is based on distance, (such as distance fall-off), use the *distance* parameter.

I3DExLightsource::IsVisibleInPerspective()

boolean MCCOMAPI IsVisibleInPerspective()

IsVisibleInPerspective() is called by the shell to determine if this light source can be displayed in a 3D view or not.

I3DExLightsource::GetPrimitiveID()

uint32 MCCOMAPI GetPrimitiveID()

GetPrimitiveID() is called to get the primitive ID of the light source.

©2001-2003 Eovia. All rights reserved.

I3DExLightsource::ForEachShadowBuffer(ForEachShadowBufferCallback, void*)

MCCOMErr MCCOMAPI ForEachShadowBuffer(ForEachShadowBufferCallback proc, void* priv)

ForEachShadowBuffer() is called when the shell needs to get a list of the shadow buffer planes from the light source to calculate shadows. This call is used by the finalrenderer.

Parameters

ForEachShadowBufferCallback *proc*
The callback procedure you should call for each shadow buffer plane of your light source.

void* *priv*
Give this parameter to the callback procedure.

I3DExLightsource::GetLightParameter(int32, void*)

MCCOMErr MCCOMAPI GetLightParameter(int32 keyword, void* parameter)

GetLightParameter is called to get the light parameters from the PMap.

Parameters

int32 *keyword*
The ID of the parameter in the PMap.

void* *parameter*
Points to where to store the light parameters.

I3DExLightsource::GetLightInfo(boolean&, boolean&, real&, TTransform3D&)

MCCOMErr MCCOMAPI GetLightInfo(boolean&hasLightCone, boolean&hasLightSphere, real&halfAngle, TTransform3D&transform)

GetLightInfo() is called to get some specific information about a light.

Parameters

boolean& *hasLightCone*
Returns True if the light has a light cone.

boolean& *hasLightSphere*
Returns True if the light has a light sphere.

real& *halfAngle*
Returns the half angle of the ligth cone if there is any.

TTransform3D& *transform*
Returns the transformation information for the light.

I3DExLightsource::GetUVfromDirection(TVector2&, const TVector3&)

boolean MCCOMAPI GetUVfromDirection(TVector2 &uv, const TVector3 &direction)

GetUVfromDirection() is called to get uv information for a gel, for the ray direction.

Parameters

TVector2& *uv*
Returns the uv values for the given direction.

const TVector3& *direction*
Specifies the direction of the ray.

©2001-2003 Eovia. All rights reserved.

I3DExLightsource::GetNewTrace(int16*)

LightTraceElement* MCCOMAPI GetNewTrace(int16* nb)

GetNewTrace() zot needs description. (RDS 5 docs say it was unused...)

Parameters **int16*** *nb*
zot needs description.

I3DExLightsource::GetStandardLight(TStandardLight&)

MCCOMErr MCCOMAPI GetStandardLight(TStandardLight& light)

GetStandardLight() retrieves a standard description of the light for use in the interactive renderer.

Parameters **TStandardLight&** *light*
Returns the standard description of a light.

©2001-2003 Eovia. All rights reserved.

7.13 I3DExLightsourceGel Interface

Family ID: ‘gel’

Interface ID: IID_I3DExLightsourceGel

Interface file: I3DExGel.h

Basic Implementation file: BasicCameraLightGel.h, BasicCameraLightGel.cpp

Description

The user can combine lights with gels to make additional lighting effects (see the **I3DExLightsource** interface to learn more about lights). A Gel can be thought of as a colored slide placed in front of a light source to modify the color and intensity of the light beam.

I3DExLightsourceGel::GetGelValues(constTVector2&, TMCColor-RGB&)

boolean MCCOMAPI GetGelValues(const TVector2& gelScreenPosition, TMCColorRGB& result)

GetGelValues is called to change the color of a light beam.

Parameters

TVector2& *gelScreenPosition*
2D position 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.

TMCColorRGB& *result*
Returns the factors by which the light beam color is going to be multiplied.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.14 I3DExModifier Interface

Family ID: ‘modi’

Interface ID: IID_I3DExModifier

Interface file: I3DExModifier.h

Basic Implementation file: BasicModifier.h, BasicModifier.cpp

Description

The **I3DExModifier** Interface is used to apply modifiers to any object or group of objects of the scene, that is any tree except the universe. They can affect the transform of the tree (position and scaling) and the geometry of the objects that belong to the tree.

Methods Index

“I3DExModifier::Apply(I3DShTreeElement)”
“I3DExModifier::GetMaxTimeWithoutComputation (I3DShTreeElement*)”
“I3DExModifier::SetBoundingBox(const TBBox3D&)”
“I3DExModifier::DeformPoint(TVector3&, TVector3&)”
“I3DExModifier::DeformVertex3D(TVertex3D&, TVertex3D&)”
“I3DExModifier::DeformFacetMesh(real, FacetMesh*, FacetMesh**)”
“I3DExModifier::DeformBBox(const TBBox3D&, TBox3D&)”
“I3DExModifier::GetNeededLOD(const real, real&)”
“I3DExModifier::GetDeformerInfo(real, TDeformInfo&)”

I3DExModifier::Apply(I3DShTreeElement)

MCCOMErr MCCOMAPI Apply(I3DShTreeElement* tree)

Apply gives access to all the parameters of the tree the modifier is on. To implement a behavior, just change the transforms.

Parameter **I3DShTreeElement*** *tree*
The tree of the element to be modified.

I3DExModifier::GetMaxTimeWithoutComputation (I3DShTreeElement*)

MicroTick MCCOMAPI GetMaxTimeWithoutComputation (I3DShTreeElement* tree)

GetMaxTimeWithoutComputation tells the shell how much cached data the behavior has (starting at zero).

Parameter **I3DShTreeElement*** *tree*
The tree of the element to be modified.

I3DExModifier::SetBoundingBox(const TBBox3D&)

MCCOMErr I3DExModifier::SetBoundingBox(*const TBBox3D& bbox***);**

©2001-2003 Eovia. All rights reserved.

SetBoundingBox is called by the 3D Shell to tell the 3D deformation modifier in which boundaries the 3D data to deform is going to be.

Parameter **TBBox3D&** *bbox*
The 3D bounding box of the 3D data to be deformed.

I3DExModifier::DeformPoint(TVector3&, TVector3&)

MCCOMErr I3DExModifier::DeformPoint(*TVector3& point, TVector3& result***);**

Computes result of the deformation of a given point (in local coordinates). **DeformPoint** is called by the shell if you do not implement **DeformFacetMesh** (the shell will automatically subdivide the facet mesh to achieve a smooth deformation).

Parameter ***TVector3&*** *point*
The 3D point to be deformed, in the coordinate relative to the bounding box.

TVector3& *result*
The 3D point after the deformation.

Return Value MC_E_NOTIMPL

I3DExModifier::DeformVertex3D(TVertex3D&, TVertex3D&)

MCCOMErr MCCOMAPI DeformVertex3D(const TVertex3D& vertex, TVertex3D& result) ;

DeformVertex3D is called by the 3D Shell and can be used to deform using a vertex.

Parameter ***TVertex3D&*** *vertex*
The 3D vertex to be deformed, in the coordinate relative to the bounding box.

TVector3* *result*
The 3D vertex after the deformation.

Return Value MC_E_NOTIMPL

I3DExModifier::DeformFacetMesh(real, FacetMesh*, FacetMesh**)

MCCOMErr I3DExModifier::DeformFacetMesh(*real lod,FacetMesh* in, FacetMesh** outMesh***);**

DeformFacetMesh is called to deform or create 3D facets. Given the original facet mesh of a primitive, it produces a (new) deformed mesh. You can modify whatever you want on the mesh: you can add or remove vertices, normals, facets, or change their position.
But you should NOT modify the *in* mesh. You need to create a new mesh and return it in *outMesh* (yet you can clone *in* to *outMesh* and then change *outMesh*).
If the parameters of your deformer won't deform the mesh, you can have *outMesh* reference *in* thus:

```
TMCCountedGetHelper<FacetMesh> result(outMesh); result= in;
```

Parameters **real** *lod*
The level of detail.

FacetMesh* *in*
A pointer to the input representation of triangular polygon data using corresponding arrays of data.

FacetMesh** *outMesh*
The output mesh after the deformation occurs.

Returns MC_E_NOTIMPL

I3DExModifier::DeformBBox(const TBBox3D&, TBox3D&)

MCCOMErr MCCOMAPI DeformBBox(const TBBox3D& in,TBBox3D& out)

DeformBBox is called called to get an approximation of how the modifier affects a given zone of space con-

©2001-2003 Eovia. All rights reserved.

tained in the box *in*, and returns the box that encompass the deformed space in *out*.

Parameters

TBBox& *in*
This is the reference to the input bounding box.

TBBox& *out*
This is the reference to the output bounding box.

Returns MC_S_OK

I3DExModifier::GetNeededLOD(const real, real&)

MCCOMErr MCCOMAPI GetNeededLOD(const real outputLOD, real& neededLod)

GetNeededLOD is used by the shell to know what level of detail the modifier needs to produce a result at the precision *outputLOD*. The default implementation does:

```
neededLOD= outputLOD;
```

Parameters

const real *outputLOD*
This is the output level of detail.

real& *neededLod*
This is the reference to the needed level of detail.

Returns MC_S_OK

I3DExModifier::GetDeformerInfo(real, TDeformInfo&)

MCCOMErr MCCOMAPI GetDeformerInfo(real lod, TDeformInfo& deformInfo)

GetDeformerInfo is called to get information about the modifier. If you implement only **DeformPoint** or **DeformVertex3D** and not **DeformFacetMesh**, use this function to give the shell some directions to guide the facet mesh subdivision process (see I3dExModifier.h).

Parameters

real *lod*
This is the level of detail.

TDeformInfo& *deformInfo*
This is the reference deformation information.

Returns MC_S_OK

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.15 I3DExModule Interface

Family ID: ‘modu’

Interface ID: IID_I3DExModule

Interface file: I3DExModule.h

Basic Implementation file: BasicModule.h, BasicModule.cpp

Description

The **I3DExModule** Interface is used to create module of any kind, and to associate as many views as you want. The cookbook example <zot xref later>shows how to create a modeler. A module has Preferences, which can be set by the user through the File menu> Preferences command just like any other module. A module also has its own tool set and menu bar.

There are two entry points to a module. First, a module can edit data of some known type. Second, if the type of data is not known, the module may import the data into a familiar type. Refer to **I3DExModule::Edit()** for details.

Resources:

In addition to the usual 'Comp' and 'PMap' resources, you need to provide a 'Modu' resource of the same ID to specify parameters such as the 'MBAR' (Menu Bar) ID, the 'TBAR' (Tool Bar) ID, etc. See ExternalAPI.r.

Preferences:

The Shell can read and write your Module Preferences data automatically both from and to the Preferences file. You can give this data to your Module using the **I3DExModule::SetPrefs()** call just before calling **I3DExModule::Edit()**.

1. In the 'Modu' resource, choose a Preferences View ID and a 'Prfs' ID (the 'Prfs' resource contains the default values if a Preferences file has to be created from scratch by the Shell. To make a 'Prfs resource, copy/paste the part of the Preferences File related to your Module into the 'Prfs' resource.

2. Add to the Module's 'PMap'. You will note that the same PMap is used for the Module's View and its Preferences. Thisis not a real problem as you use distinct IDs for the UI elements (buttons, check boxes, etc.). If you do not wish to have Preferences, put 0 in the Preferences View ID of the 'Modu' resource.

Drag and drop from Tool Bar: **zot** this was in the header file, does someone want to mention something about it here? If not, I’ll remove it.

Actions:

zot The RDS text discusses actions and refers to a group of interfaces that are no longer used (IShActionManager, IShAction, and IShMouseAction). What should be said about how to post undoable actions to enter the undo/redo process.

Methods Index

“I3DExModule::Initialize(IMCUnknown*)”
“I3DExModule::Destroy()”
“I3DExModule::Hydrate()”

©2001-2003 Eovia. All rights reserved.

"I3DExModule::Dehydrate()"
"I3DExModule::Activate()"
"I3DExModule::Deactivate()"
"I3DExModule::Import(IMCUnknown*, void*)"
"I3DExModule::BuildMenuBar()"
"I3DExModule::GetPreferredRoom(IDType&)"
"I3DExModule::WantsToBeInCurrentRoom()"
"I3DExModule::SubModuleClosedMainWindow(I3DShModule*)"
"I3DExModule::SubModuleAboutToBeDestroyed(I3DShModule*)"
"I3DExModule:: SubModuleAboutToBeHydrated(I3DShModule*)"
"I3DExModule::AboutToCloseMainWindow()"
"I3DExModule::SavesInNativeFormat()"

I3DExModule::Initialize(IMCUnknown*)

MCCOMErr MCCOMAPI Initialize(IMCUnknown* inElement)

Initialize() is called during creation of the module to initialize UI elements, and set the editing element. inElement is the element that the module edits. Override this function to create UI elements such as windows and initialize any data for your module.

Use QueryInterface on the inElement object to get the specific interface you need. Scene Document Modules will get an I3DShGroup object. Modelers get two calls to Initialize. First, they get passed the universe group as all Scene Document Modules do. Next, they get passed the master object in the scene selection which they are to edit. In addition, if the user leaves the modeling room and re-enters with a different object selected, **Initialize()** is called again with the new object to edit.

Global Modules do not get any object, the have a NULL inElement.

I3DExModule::Destroy()

MCCOMErr MCCOMAPI Destroy()

Destroy() is called once when the module is to be destroyed (e.g., Scene Document Modules are called when the document closes, Global Modules are called when the application is quit). Override this function to destroy UI elements and cleanup any data. However, you need destroy your slave modules, since you are in charge of them.

I3DExModule::Hydrate()

MCCOMErr MCCOMAPI Hydrate()

Hydrate() is is called when the module is to be hydrated; that is, the room that it occupies is entered.

If this module is a main module (i.e., it defines a room), override this function to show any main windows and call **Hydrate()** on slave modules.

If this module is a slave module, override this function to prepare to be activated and call **Hydrate()** on any subslave modules. This may involve showing windows, but should not install parts in shared windows since **Hydrate()** does not mean the module is active (see **Activate()** below). Also note that your window may have been closed by someone else (in this case you had been told through **AboutToCloseMainWindow()**), so you will have to recreate your window.

©2001-2003 Eovia. All rights reserved.

I3DExModule::Dehydrate()

MCCOMErr MCCOMAPI Dehydrate()

Dehydrate() is called when the module is to be dehydrated; that is, the room that it occupies is exited. If this module is a main module (i.e., it defines a room), override this function to hide any main windows and call **Dehydrate()** on slave modules.

If this module is a slave module, override this function to destroy large caches and parts and call **Dehydrate()** on any subslave modules.

NOTE: **Deactive()** is called before **Dehydrate()** on an active module.

I3DExModule::Activate()

MCCOMErr MCCOMAPI Activate()

Activate() is called when the module is to be activated. If this module is a main module (i.e., it defines a room), one of its main windows has been activated; override this function to call **Activate()** on slave modules. If this module is a slave module, override this function to activate and call **Activate()** on any subslave modules.

This may involve showing windows, and may install parts in shared windows.

I3DExModule::Deactivate()

MCCOMErr MCCOMAPI Deactivate()

Deactivate() is called when the module is to be deactivated. If this module is a main module (i.e., it defines a room), all of its main windows has been deactivated; override this function to call Deactivate() on slave modules. If this module is a slave module, override this function to deactivate and call Deactivate() on any subslave modules.

This may involve hiding windows, and may uninstall parts out of shared windows.

I3DExModule::Import(IMCUnknown*, void*)

MCCOMErr MCCOMAPI Import(IMCUnknown* inElement, void* inImportData)

Import() is called when the module is asked to import some foreign data. For example, a modeler could be asked to import the facets of another primitive.

Parameters

IMCUnknown* *inElement*
It is up to the module to do something or not with *inElement*. The module should rely on **inElement->QueryInterface** to decide if it can do anything with it of not. In any case, *inElement* should be left intact.

void* *inImportData*
inImportData points to some data block depending on the type import. For example, for modelers, *inImportData* will be a real32* containing the Fidelity.

I3DExModule::BuildMenuBar()

MCCOMErr MCCOMAPI BuildMenuBar()

BuildMenuBar() gives you opportunity to add menus dynamically (instead of using a MBAR resource).

©2001-2003 Eovia. All rights reserved.

Use *gShMenuUtilities* to add menu items.

I3DExModule::GetPreferredRoom(IDType&)

MCCOMErr MCCOMAPI GetPreferredRoom(IDType& roomId)

GetPreferredRoom() is called to determine in which room the module wants to be put in, when the module does not define a room (eg the rendering module).

By default the module will be added in the current room.

Parameters **IDType&** *roomId*
The ID of the preferred room.

I3DExModule::WantsToBeInCurrentRoom()

boolean MCCOMAPI WantsToBeInCurrentRoom()

WantsToBeInCurrentRoom() is called to determine if the module can be inserted in the current room.

I3DExModule::SubModuleClosedMainWindow(I3DShModule*)

boolean MCCOMAPI SubModuleClosedMainWindow(I3DShModule* subModule)

SubModuleClosedMainWindow() will be called by any sub module while its Main Window is being closed, so you can react and tell it that it is OK to commit suicide (return true).

Note: You tell a sub module that you are its "master" by calling **I3DShModule::SetMasterModule()**.

Parameters **I3DShModule*** *subModule*
A pointer to the sub module that is having its Main Window closed.

I3DExModule::SubModuleAboutToBeDestroyed(I3DShModule*)

MCCOMErr MCCOMAPI SubModuleAboutToBeDestroyed(I3DShModule* subModule)

SubModuleAboutToBeDestroyed() will be called by any sub module that is about to be destroyed, so you can react.

Note: You tell a sub module that you are its "master" by calling **I3DShModule::SetMasterModule()**.

Parameters **I3DShModule*** *subModule*
A pointer to the sub module that is about to be destroyed.

I3DExModule:: SubModuleAboutToBeHydrated(I3DShModule*)

MCCOMErr MCCOMAPI SubModuleAboutToBeHydrated(I3DShModule* subModule)

SubModuleAboutToBeHydrated() will be called by any sub module that is about to be hydrated, so you can react.

Note: You tell a sub module that you are its "master" by calling **I3DShModule::SetMasterModule()**.

©2001-2003 Eovia. All rights reserved.

Parameters **I3DShModule*** *subModule*
A pointer to the sub module that is about to be hydrated.

I3DExModule::AboutToCloseMainWindow()

MCCOMErr MCCOMAPI AboutToCloseMainWindow()

AboutToCloseMainWindow() is called when your main window is about to be closed by someone else.

It is very important to release any references that your module may have on your window. You do not need to close it, since this "someone" is going to do it.

I3DExModule::SavesInNativeFormat()

booleanMCCOMAPI SavesInNativeFormat()

SavesInNativeFormat() is called to determine whether the module wants to save data in the nativefile format.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.16 I3DExPostRenderer Interface

Family ID: 'post'

Interface ID: IID_I3DExPostRenderer

Interface file: I3DExPostRenderer.h

Basic Implementation File: BasicPostRenderer.h, BasicPostRenderer.cpp

Description

The **I3DExPostRenderer** Interface is used to apply additional effects (filters) to the image after the rendering.

Method Index

"I3DExPostRenderer::GetBufferNeeds(uint32&, void*)"
"I3DExPostRenderer::PrepareDraw(IShRasterLayer*, const TMCRect&, IShChannel*, I3DShScene*, I3DShCamera*, TBBox2D&, TBBox2D&)"
"I3DExPostRenderer::DrawRect(const TMCRect&, const TChannelDataBucket*,TBBox2D&)"
"I3DExPostRenderer::FinishDraw()"
"I3DExPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean)"
"I3DExPostRenderer::EndRender()"

I3DExPostRenderer::GetBufferNeeds(uint32&, void*)

void GetBufferNeeds(uint32& needs, void* renderer)

GetBufferNeeds gets information needed when you initialize the postrenderer. This call is called before the rendering starts.

Parameter **uint32&** *needs*

Returns a set of flags to determine which GBuffers should be filled during the rendering. The following constants are defined to help you (they define the bit number to be set so to get the buffer of the NormalX and the distance you should return 2^k32Distance + 2^k32NormalX:

Name	Value	Description
k32Distance	0	Distance from the camera to the object visible at a point in the image.
k32PositionX	1	X position of a point if not the background
k32PositionY	2	Y position of a point if not the background
k32PositionZ	3	Z position of a point if not the background
k32NormalX	4	X direction of an object's surface Normal if not the background
k32NormalY	5	Y direction of an object's surface Normal if not the background
k32NormalZ	6	Z direction of an object's surface Normal if not the background

©2001-2003 Eovia. All rights reserved.

Name	Value	Description
k32Alpha	7	The alpha channel with black (0) as the background
k32Index	8	UV space index
k32SurfaceU	9	U value of the UV space of an object at a point
k32SurfaceV	10	V value of the UV space of an object at a point
k32DiffuseColor	11	Corresponds to the Color channel of the object's shader
k32SpecularColor	12	Corresponds to the Highlight channel of the object's shader
k32SpecularSizeValue	13	Corresponds to the Shininess channel of the object's shader
k32AmbientValue	14	
k32LambertValue	15	
k32ReflectionColor	16	Corresponds to the Reflection channel of the object's shader
k32TransparencyColor	17	Corresponds to the Transparency channel of the object's shader
k32RefractionValue	18	Corresponds to the Refraction channel of the object's shader
k32GlowColor	19	Corresponds to the Glow channel of the object's shader

void* ***renderer***
This element is not available yet.

I3DExPostRenderer::PrepareDraw(IShRasterLayer*, const TMCRect&, IShChannel*, I3DShScene*, I3DShCamera*, TBBox2D&, TBBox2D&)

MCCOMErr PrepareDraw(IShRasterLayer* input, const TMCRect& bounds, IShChannel* buffers[],I3DShScene* scene, I3DShCamera* renderingCamera, TBBox2D& uvBox,TBBox2D& productionFrame)

PrepareDraw() is called prior to the rendering. You should store most of the info provided.

Parameters

IShRasterLayer* *input*
Is a pointer to the buffer of the input image. (The input image is composed of 3 planes of 16 bits each for the 3 colors.)

const TMCRect& *bounds*
Is a rectangle corresponding to the image.

IShChannel* *buffers[]*
This is a table of pointers to the buffers requested by **GetBufferNeeds()**. Each element in the buffers array corresponds to the value of each buffer (buffer[0] for distance through buffer[19] for glow color). These pointers are null when the buffer is not allocated.

See the table in "I3DExPostRenderer::GetBufferNeeds(uint32&, void*)" on page 115 for the complete list.

©2001-2003 Eovia. All rights reserved.

I3DShScene* *scene*
This is a pointer to the scene structure.

I3DShCamera* *renderingCamera*
This is a pointer to the rendering camera.

TBBox2D& *uvBox*
uvBox of the image being rendered in screen coordinates.

TBBox2D& *Production Frame*
uv Box of the Production Frame in screen coordinates (this might be different from the uvBox when doing marquee rendering in the intereactive preview).

Return Value MC_S_OK

Comments You need to use the Production Frame and uvBox to make sure Area Rendering in the interactive preview will work.

I3DExPostRenderer::DrawRect(const TMCRect&, const TChannel-DataBucket*,TBBox2D&)

MCCOMErr DrawRect(const TMCRect& outputRect, const TChannelDataBucket* output[],TBBox2D &uvBox)

DrawRect() is called for each tile of the image.

Parameters

TMCRect& *outputRect*
Coordinates for the tile to be rendered in pixels. Should be inside the bounds rectangle given in **PrepareDraw()**. It matches an actual tile in the input and output buffers.

TChannelDataBucket* *output[]*
PixelBucket corresponding to the tile to be rendered in the output image. You can be sure it is an actual tile of the output buffer.

I3DExPostRenderer::FinishDraw()

MCCOMErr FinishDraw()

FinishDraw() is called for cleanup at the end of the post-rendering.

I3DExPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean)

MCCOMAPI BeginRender(MicroTick beginTime,MicroTick endTime, uint32 framePerSeconds,int32nbrFrames, boolean isRenderMovie)

BeginRender() is called for set up at the beginning of the post-rendering.

I3DExPostRenderer::EndRender()

MCCOMAPI EndRender()

EndRender() is called for cleanup at the end of the post-rendering.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.17 I3DExGeometricPrimitive Interface

Family ID: ‘prim’

Interface ID: IID_I3DExGeometricPrimitive

Interface file: I3DExPrimitive.h

Basic Implementation file: BasicPrimitive.h, BasicPrimitive.cpp

Description

Geometric primitives define a geometric object. They return a list of facets or bicubic patches and can give information on an object’s UV Space. The shell will check for patches first by calling **EnumPatches()** and then call **GetFacetMesh()** if **EnumPatches()** fails. At least one or both of these two methods must be implemented.

In addition to describing geometry, a geometric primitive can define a custom ray-tracing routine rather than relying only on facets or patches. This is done by inheriting from the **“IRaytracablePrimitive Interface”** in addition to **I3DExGeometricPrimitive** (or TBasicPrimitive).

A primitive can also support skinning (skinning is the deformation of the primitive geometry using a skeleton composed of bones). This is done by inheriting from the “I3dExSkinable Interface” in addition to **I3DExGeometricPrimitive** (or TBasicPrimitive).

Rather than defining geometry, a primitive can also describe a volume. This is achieved by inheriting from the **“IVolumePrimitive Interface”** as well as **I3DExGeometricPrimitive** (or TBasicPrimitive).

Method Index

“I3DExGeometricPrimitive::GetBoundingBox(TBox3D*)”
“I3DExGeometricPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean&)”
“I3DExGeometricPrimitive::GetNbrLOD(int16&)”
“I3DExGeometricPrimitive::GetLOD(int16, real&)”
“I3DExGeometricPrimitive::GetFacetMesh(uint32, FacetMesh**)”
“I3DExGeometricPrimitive::GetFacetMesh(real, FacetMesh**)”
“I3DExGeometricPrimitive::CanBeSplit()”
“I3DExGeometricPrimitive::SplitPrimitive(TMCCountedPtrArray<I3DExGeometricPrimitive>&, TMCArray<TTransform3d>&)”
“I3DExGeometricPrimitive::GetUVSpaceCount()”
“I3DExGeometricPrimitive::GetUVSpace(uint32, UVSpaceInfo*)”
“I3DExGeometricPrimitive::UV2XYZ(TVector2*, uint32, TVEctor3*, boolean*)”
“I3DExGeometricPrimitive::GetUVSpaceRDS5(uint32, UVSpaceInfoRDS5*)”
“I3DExGeometricPrimitive::AppendToRenderables(const TTransform3D&, TRenderableAndTfmArray&)”
“I3DExGeometricPrimitive::AutoSwitchToModeler()”
“I3DExGeometricPrimitive::IsBBoxPickable()”
“I3DExGeometricPrimitive::GetDefaultShader”
“I3DExGeometricPrimitive::ChangedData”

I3DExGeometricPrimitive::GetBoundingBox(TBox3D*)

MCCOMErr I3DExGeometricPrimitive::GetBoundingBox(TBox3D* bbox)

GetBoundingBox() is called to pass the bounding box of the primitive to the 3D Shell. This call is optional.

©2001-2003 Eovia. All rights reserved.

If this call is not implemented the bounding box will be calculated automatically from the patches or facets list.

Parameter **TBox3D*** *bbox*
The 3D bounding box of the object.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

I3DExGeometricPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean&)

MCCOMErr MCCOMAPI EnumPatches(EnumPatchesCallback callback, void* privData, boolean& closed)

I3DExGeometricPrimitive::EnumPatches is called by the 3D Shell to get the primitive as a list of bicubic patches. Try not to use it. Use **GetFacetMesh()** instead.

Parameters **EnumPatchesCallback** *callback*
The callback procedure should be called for each Patch generated by the Primitive. Its interface is as follows:

```
typedef void (*EnumPatchesCallback) (TPatch3D* patch, void* privData);
```

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

void* ***privData***
Points to private data from the Shell. Pass this pointer to *callback.*

boolean& *closed*
Returns True when the surface is closed. This helps set the backface mask correctly.

Return **MC_E_NOTIMPL;**

I3DExGeometricPrimitive::GetNbrLOD(int16&)

MCCOMErr GetNbrLOD(int16& nbrLod)

GetNbrLOD() is called to get the number of levels of detail of the primitive. This value is used as the *lodIndex* in **GetLOD()**.

Parameters **int16&** *nbrLod*
This is the number of levels of detail. A level of detail is the threshold used to display your primitive.

Return value

MC_S_OK	
MC_E_NOTIMPL	Function is not implemented.

I3DExGeometricPrimitive::GetLOD(int16, real&)

MCCOMErr GetLOD(int16 lodIndex, real& lod)

©2001-2003 Eovia. All rights reserved.

GetLOD() is called to get the level of detail of the primitive. Refer to the Cookbook chapter on creating a primitive for a detailed discussion on level of detail.

Parameters

int16 *lodIndex*
The level of detail index.

real& *lod*
The level of detail.

Return value

MC_S_OK	
MC_E_NOTIMPL	Function is not implemented.

I3DExGeometricPrimitive::GetFacetMesh(uint32, FacetMesh**)

MCCOMErr MCCOMAPI GetFacetMesh(uint32 lodIndex, FacetMesh outMesh)**

GetFacetMesh() is an alternate method for creating a facet mesh. It is more efficient than using **EnumFacets()** as it allows for calculating less facets in conditions that require less detail. Implement this version.

Parameters

uint32 *index*
The index of the level of detail.

FacetMesh** *outMesh*
A pointer to the output facet mesh.

Return value A FacetMesh* defining the geometric primitive. Refer to "FacetMesh" on page 50 to learn more.

I3DExGeometricPrimitive::GetFacetMesh(real, FacetMesh**)

MCCOMErr MCCOMAPI GetFacetMesh(real lod, FacetMesh outMesh)**

Never implement this version of GetFacetMesh. It is reserved for future use (with primitives with an infinite number of resolutions.)

Parameters

real *lod*
The level of detail.

FacetMesh** *outMesh*
A pointer to the output facet mesh.

Return value A FacetMesh* defining the geometric primitive. Refer to "FacetMesh" on page 50 to learn more.

I3DExGeometricPrimitive::CanBeSplit()

boolean CanBeSplit()

CanBeSplit() returns true if the primitive can split itself into sub parts of the same type.

Return value

True	**SplitPrimitive()** will be called.

©2001-2003 Eovia. All rights reserved.

False	**SplitPrimitive()** will not be called.

I3DExGeometricPrimitive::SplitPrimitive(TMCCountedPtrArray<I3DExGeometricPrimitive>&, TMCArray<TTransform3d>&)

MCCOMErr SplitPrimitive(TMCCountedPtrArray<I3DExGeometricPrimitive>& subParts, TMCArray<TTransform3D>& subPartPositions)

SplitPrimitive() builds a list of sub-parts of itself. Both arrays need to be of same size. The original primitive remains intact.

Parameters

TMCCountedPtrArray<I3DExGeometricPrimitive>& *subParts*
The split parts of the primitive.<zot>Could use better description here</zot>

TMCArray<TTransform3d>& *subPartPositions*
subPartPosition gives the relative position of each sub part in the object's local coordinates.

I3DExGeometricPrimitive::GetUVSpaceCount()

uint32 GetUVSpaceCount()

GetUVSpaceCount() is called to get the number of UV Spaces as defined in "UVSpaceInfo" on page 48.

Return value Number of UV Spaces for this object. If it is not possible to do direct parametric mapping and there are no UV values then return 0.

I3DExGeometricPrimitive::GetUVSpace(uint32, UVSpaceInfo*)

MCCOMErr GetUVSpace(uint32 uvSpaceID, UVSpaceInfo* uvSpaceInfo)

GetUVSpace() is called to define the UV Space(s) of the primitive.

Parameters

uint32 *uvSpaceID*
The UV Space ID on which the Component needs to return information, indexed from 0 to **GetUVSpaceCount**()-1.

UVSpaceInfo* *uvSpaceInfo*

The UVSpace info pointer. Refer to "UVSpaceInfo" on page 48 to learn more.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

I3DExGeometricPrimitive::UV2XYZ(TVector2*, uint32, TVEctor3*, boolean*)

MCCOMErr UV2XYZ(TVector2* uv, uint32 uvSpaceID, TVector3* resultPosition, boolean* inUVSpace)

UV2XYZ() is called when a point on the surface defined only by its uv coordinates needs to be converted in x, y and z values. This call is optional. If this call is not implement then the default rules will apply by using the patches or the facets of the Primitive. When possible implement **I3DExGeometricPrimitive::UV2XYZ** for optimal performance and accuracy.

©2001-2003 Eovia. All rights reserved.

Parameters

TVector2* *uv*
The given uv values.

uint32 *uvSpaceID*
The identifier of the UV space.

TVector3* ***resultPosition***
The resulting 3D coordinates in the object Local Coordinates System.

boolean* ***inUVSpace***
Return TRUE if the uv values are within the UV Space boundaries.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

I3DExGeometricPrimitive::GetUVSpaceRDS5(uint32, UVSpaceInfoRDS5*)

MCCOMErr GetUVSpaceRDS5(uint32 uvSpaceID, UVSpaceInfoRDS5* uvSpaceInfoRDS5)

GetUVSpaceRDS5 returns the info about the uvSpace in the format of Raydream Studio 5 (RDS5). UVSpaces in Raydream were not constrained between 0 and 1 as they are in Carrara.

Parameters

uint32 *uvSpaceID*
The identifier of the UV space

UVSpaceInfoRDS5 * ***uvSpaceInoRDS5***
A pointer to the structure that this function should fill.

Return Value **MC_E_NOTIMPL** if you do not have old style UVSpaces

I3DExGeometricPrimitive::AppendToRenderables(const TTransform3D&, TRenderableAndTfmArray&)

MCCOMErr AppendToRenderables(const TTransform3D& worldFromModelTfm,TRenderableAndTfmArray& renderableAndTfm)

This called is used to add extra renderables for a primitive. By default a primitive is displayed using its facet mesh. However you might want to add extra information in the 3D view (for example display the emitter wireframe for a particle emitter). This function is called for each instance of the primitive.

Parameters

TTransform3D& *worldFromModelTfm*
worldFromModelTfm is the transform of the object.

TRenderableAndTfmArray& *renderableAndTfm*
renderableAndTfm is the array of all the renderables to which you can append extra renderables for your primitive.

I3DExGeometricPrimitive::AutoSwitchToModeler()

virtual boolean AutoSwitchToModeler()

AutoSwitchToModeler() automatically switches to the modeler after the primitive is inserted.

©2001-2003 Eovia. All rights reserved.

Return value

True	True if the primitive should be open in its modeler immediately after its insertion
False	False if the primitive should not automatically open its modeler.

I3DExGeometricPrimitive::IsBBoxPickable()

booleanMCCOMAPI IsBBoxPickable()

Return true if the primitive can be picked in the 3D View when clicking on its bounding box. Usually, you want to reply no, since the picking is done thru a ray-tracing hit test. However, some primitives such as volumic primitives or particles systems may want to answer yes.

Default implementation in TBasicPrimitive returns false.

I3DExGeometricPrimitive::GetDefaultShader

virtual void MCCOMAPI GetDefaultShader(I3DShShader shader) = 0;**

GetDefaultShader returns the default shader for this primitive. This is called each time a primitive is inserted in a scene without a shader. If no shader is returned (ie *shader=NULL), then the default shader of the scene is used instead.

Parameters **I3DShShader**** *shader*

shader returns a pointer the default shader for this primitive. (You need to create a new shader each time this function is called)

I3DExGeometricPrimitive::ChangedData

virtual void MCCOMAPI ChangedData() = 0;

Notifies the primitive that something has changed so that it can update its caches.

©2001-2003 Eovia. All rights reserved.

7.18 I3DExReflectionFeature

Interface ID: IID_I3DExReflectionFeature

Interface File: I3DRenderFeature.h

Description

I3DExReflectionFeature allows you to identify the reflection needs and calculate the reflections.

Method Index

"I3DExReflectionFeature::GetReflectionNeeds(uint32 &)"
"I3DExReflectionFeature::CalcReflection(RenderFeatureIO&)"

I3DExReflectionFeature::GetReflectionNeeds(uint32 &)

MCCOMErr MCCOMAPI GetReflectionNeedFlags(uint32& needFlags)

GetReflectionNeeds() counts the number of needed reflections based on *needFlags*.

Parameters **uint32 &** *needFlags*
<zot>need description</zot> .

I3DExReflectionFeature::CalcReflection(RenderFeatureIO&)

MCCOMErr MCCOMAPI CalcReflection(RenderFeatureIO& theFIOParms)

CalcReflection() calculates the reflections.

Parameters **RenderFeatureIO&** *theFIOParms*
<zot>need description</zot>.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.19 I3DExRefractionFeature

Interface ID: IID_I3DExRefractionFeature

Interface File: I3DRenderFeature.h

Description

I3DExRefractionFeature allows you to identify the refraction needs and calculate the refractions.

Method Index

"I3DExRefractionFeature::GetRefractionNeeds(uint32 &)"
"I3DExRefractionFeature::CalcRefraction(RenderFeatureIO&)"

I3DExRefractionFeature::GetRefractionNeeds(uint32 &)

MCCOMErr MCCOMAPI GetRefractionNeedFlags(uint32& needFlags)

GetRefractionNeeds() counts the number of needed refractions based on *needFlags*.

Parameters **uint32 &** *needFlags*
<zot>need description</zot> .

I3DExRefractionFeature::CalcRefraction(RenderFeatureIO&)

MCCOMErr MCCOMAPI CalcRefraction(RenderFeatureIO& theFIOParms)

CalcRefraction() calculates the refractions.

Parameters **RenderFeatureIO&** *theFIOParms*
<zot>need description</zot>.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.20 I3DExRendererBox Interface

Family ID: ‘RndB’

Interface ID: IID_I3DExRendererBox

Interface file: I3DExRendererBox.h

Basic Implementation files: BasicRendererBox.h

Description

A Renderer Box is a real time renderer used to display objects in the 3D view window. (For instance OpenGL is implemented as a renderer box.

See the comments in I3DExRendererBox.h for more information on the method of I3DExRendererBox

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.21 I3DExShadowFeatureFeature

Family ID: ‘shdF’

Interface ID: IID_I3DExShadowFeature

Interface File: I3DRenderFeature.h

Description

I3DExShadowFeature allows you to create new types for shadows (for instance you can use shadow buffers or raytraced shadows in Carrara).

Method Index

“I3DExShadowFeature::BeginRenderFrame”
“I3DExShadowFeature::EndRenderFrame”

I3DExShadowFeature::BeginRenderFrame

virtual MCCOMErr MCCOMAPI BeginRenderFrame(I3DShLightsource *light)=0;

BeginRenderFrame() is called at the beginning of each frame with the light to which the shadow feature is applied. This is a good place to precalculate any cache you would like to use for your shadows.

Parameters **I3DShLightsource*** *light*
The light source to which the shadow feature is applied.

I3DExShadowFeature::EndRenderFrame

virtual MCCOMErr MCCOMAPI EndRenderFrame()=0;

EndRenderFrame() is called at the end of each frame so that you can delete any caches you might have built.

I3DExShadowFeature::GetShadowNeedFlags

virtual MCCOMErr MCCOMAPI GetShadowNeedFlags(uint32& needFlags) = 0;

GetShadowNeedFlags() is obsolete and should not be implemented.

I3DExShadowFeature::CalcShadow

virtual boolean MCCOMAPI CalcShadow(RenderFeatureIO& theFIOParms) = 0;

CalcShadow() calculates the light reaching a point from a light source.

Note: this function must be SMP compliant !

Parameters **RenderFeatureIO& theFIOParms**
A structure describing the current point to be shadowed.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.22 I3DExSceneCommand Interface

Family ID: ‘scmd’

Interface ID: IID_I3DExSceneCommand

Interface file: I3DExSceneCommand.h

Basic Implementation files: Basic3DCOMImplementations.h

Description

A scene command is a plugin that implements an extra command on the scene. The scene commands appear in the edit menu in the assemble room of Carrara. They can be enable/disabled depending on the current selection context.

Method Index

“I3DExSceneCommand::GetMenuCallBack”
“I3DExSceneCommand::Init”
“I3DExSceneCommand::Prepare”
“I3DExSceneCommand::CanUndo”
“I3DExSceneCommand::Do”
“I3DExSceneCommand::Undo”
“I3DExSceneCommand::Redo”
“I3DExSceneCommand::PostPrepare”

I3DExSceneCommand::GetMenuCallBack

virtual void MCCOMAPI GetMenuCallBack(ISelfPrepareMenuCallBack callBack) = 0;**

GetMenuCallBack() returns a callBack that is called to check if this scene command is enabled for a given document. Note that the returned object should never keep a reference to the scene document or anything in the scene since it will only be deleted when the application shutsdown

Parameter **ISelfPrepareMenuCallBack**** *callBack*

callBack returns contain a pointer to the new callBack (use TMCCountedCreateHelper to make sure the reference count is accurate).

I3DExSceneCommand::Init

virtual MCCOMErr MCCOMAPI Init(ISceneDocument* sceneDocument) = 0;

Init() is called after the Scene Command is instanciated. Use sceneDocument to get the selection or any other Scene data you would need to perform your action

Parameter **ISceneDocument*** *sceneDocument*

sceneDocument is a pointer to the current scene document.

©2001-2003 Eovia. All rights reserved.

I3DExSceneCommand::Prepare

virtual MCCOMErr MCCOMAPI Prepare() = 0;

Prepare() allows you to perform any preparation work needed (even posting a dialog). Make sure you return MC_S_OK if it is OK to proceed with the Scene Command. If your Scene Command has a PMap resource and the corresponding 'Node' UI resources, then the application will create a modal dialog for you just after Prepare().

I3DExSceneCommand::CanUndo

virtual boolean MCCOMAPI CanUndo() = 0;

CanUndo() return true if the Scene Command is undoable

I3DExSceneCommand::Do

virtual boolean MCCOMAPI Do() = 0;

Do() performs the action associated with the scene command. Return true if you actually changed something in the Scene.

I3DExSceneCommand::Undo

virtual boolean MCCOMAPI Undo() = 0;

Undo() undoes the action of the scene command. Return true if you actually changed something in the Scene.

I3DExSceneCommand::Redo

virtual boolean MCCOMAPI Redo() = 0;

Redo() performs the action again (after a undo). Return true if you actually changed something in the scene.

I3DExSceneCommand::PostPrepare

virtual boolean MCCOMAPI PostPrepare() = 0;

PostPrepare() is called after Prepare and the options dialog for the pmap so that you can create data caches needed for do/undo/redo operations

Return value true if the scene command can go on

false if the scene command must be stopped.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.23 I3DExShader Interface

Family ID: ‘shdr’

Interface ID: IID_I3DExShader

Interface file: I3DExShader.h

Basic Implementation file: BasicShader.h, BasicShader.cpp

Description

Shaders are used to give photorealistic appearance to an object’s surface. They define a value for each point in the UV space of an object. They are used by the renderer to calculate the appearance of an object in the rendered image. A shader is comprised of a group of channels (color, transparency, reflections, etc.) which are combined to form a complete shader.

Shaders are designed to be re-entrant such that the output from one shader can be used as the input for another shader. This is accomplished by implementing methods to return a value, color, or complete shading structure and then using **GetShadingFlags()** and **GetImplementedOutput()** to define which implementation was used. Methods that return simply a value, color, or vector use the *“ShadingIn”* struct, filling in the fields as defined by **GetShadingFlags()** independent of the channel in which the shader is placed. A shader that implements an entire shading structure uses a *“ShadingOut”* structure that sets values for a ShadingIn structure for each channel of the shader.

Besides implementing methods to define a value for each point, which uses a *ShadingIn* or *ShadingOut*, you can also implement an additional version of each function that uses a *ShadedArea*. A *ShadedArea* is defined by four *ShadingIn* points. If the *ShadedArea* version of the function is implemented, the *ShadingIn* version must also be implemented.

By using a *ShadedArea* you can find the average of a pixel. In implementing this method you can control the sampling level and greatly increase the speed at which your shader is rendered. This is particularly easy to implement for shaders which produce the same or similar result over an extended area.

Method Index

“I3DExShader::IsEqualTo(I3DExShader*)”
“I3DExShader::GetShadingFlags(ShadingFlags&)”
“I3DExShader::GetImplementedOutput()”
“I3DExShader::DoShade(ShadingOut&, ShadingIn&)”
“I3DExShader::GetValue(real&, boolean, ShadingIn&)”
“I3DExShader::GetColor(TMCColorRGB&, boolean, ShadingIn&)”
“I3DExShader::GetVector(TVector3&, ShadingIn&)”
“I3DExShader::ConvertToPoly(TMCArray<TVector2>*)”
“I3DExShader::OutlineInPreviewMode()”
“I3DExShader::Contains(ShadingIn&, TVector2&)”

I3DExShader::IsEqualTo(I3DExShader*)

boolean IsEqualTo(I3DExShader* aShader)

IsEqualTo() is used to compare the values of two shaders. **IsEqualTo()** must first get a pointer on the same interface as itself before comparing the values.

©2001-2003 Eovia. All rights reserved.

Parameter **I3DExShader*** *aShader*
Pointer to the comparison shader.

Return Value

Value	Meaning
TRUE	Shaders have identical values
FALSE	Shaders are different

I3DExShader::GetShadingFlags(ShadingFlags&)

MCCOMErr GetShadingFlags(ShadingFlags& theFlags)

GetShadingFlags() specifies which parameters the shader will need to perform shading calculations. If the shader is a *constant shader* which values does not depend on any of the *ShadingIn* parameters then it should set the *fCallOnce* flag in *theFlags* to true. See the description of the *"ShadingFlags"* structure for more details.

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

Return Value MC_S_OK

I3DExShader::GetImplementedOutput()

uint32 GetImplementedOutput()

GetPreferredOutput() is called by the 3D Shell to determine which methods the Shader actually implements. Return only one of the following values if you implement only one of the methods. If more than one is implemented, such as the *ShadedArea* implementation, then return the result as the OR of the two values (i.e. return kUsesGetValueArea | kUsesGetValue).

Return Value

Shading Defines

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
kUsesGetValueArea	32	**I3DExShader::GetValue** with *ShadedArea* is implemented
kUsesGetColorArea	64	**I3DExShader::GetColor** with *ShadedArea* is implemented
kUsesGetVectorArea	128	**I3DExShader::GetVector** with *ShadedArea* is implemented
kUsesDoShadeArea	256	**I3DExShader::DoShade** with *ShadedArea* is implemented

©2001-2003 Eovia. All rights reserved.

I3DExShader::DoShade(ShadingOut&, ShadingIn&)

MCCOMErr DoShade(ShadingOut& result, ShadingIn& shadingIn)

DoShade() has total control of the input and output parameters. See the description of *"ShadingOut"* and *"ShadingIn"* for more details.

Parameters

ShadingOut& *result*
Reference to the complete input/output shading data structure.

ShadingIn& *shadingIn*
The Shading element associated with the shader.

Return Value

MC_S_OK	**DoShade()** is implemented by this shader.
MC_E_NOTIMPL	**DoShade()** is not implemented by this shader.

Comments

I3DExShader::GetValue(real&, boolean, ShadingIn&)

MCCOMErr GetValue(real& result, boolean fullArea, ShadingIn& shadingIn)

GetValue() returns a value between 0.0 and 1.0. Typical examples are noise functions, fractal functions, gradients, etc. See the description of *"ShadingIn"* for more details.

Parameters

real& *result*
Returns a result between 0.0 and 1.0.

boolean *fullArea*
<zot> needs description </end zot>

ShadingIn& *shadingIn*
Reference to the input shading data structure. The shader uses the parameters requested by **"I3DExShader::GetShadingFlags(ShadingFlags&)"**.

Return Value

MC_S_OK	**GetValue()** is implemented by this shader.
MC_E_NOTIMPL	**GetValue()** is not implemented by this shader.

I3DExShader::GetColor(TMCColorRGB&, boolean, ShadingIn&)

MCCOMErr GetColor(TMCColorRGB& result, boolean fullArea, ShadingIn& shadingIn)

GetColor() returns a color. Typical examples are constant colors or texture maps. See the description of *"ShadingIn"* for more details.

Parameters

TMCColorRGB *& result*
Return the color values.

©2001-2003 Eovia. All rights reserved.

boolean *fullArea*
<zot> needs description </end zot>

ShadingIn *& in*
Points to the input shading data structure. The shader uses the parameters requested by **"I3DExShader::GetShadingFlags(ShadingFlags&)"**.

Return Value

MC_S_OK	**GetColor()** is implemented by this shader.
MC_E_NOTIMPL	**GetColor()** is not implemented by this shader.

Comments

I3DExShader::GetVector(TVector3&, ShadingIn&)

MCCOMErr GetVector(TVector3& result, ShadingIn& shadingIn)

GetVector is reserved for a future use.

I3DExShader::ConvertToPoly(TMCArray<TVector2>*)

void ConvertToPoly(TMCArray<TVector2>* vertices)

ConvertToPoly() must be implemented to allow a shader to display an outline. This is typically used by layers such as 'rectangle', 'ellipse', and 'polygon'.

Parameters **TMCArray<TVector2>*** *vertices*
An array of 2D vertices used to draw a layout around the layer. Those vertices' coordinates are in the UVSpace of the layer.

I3DExShader::OutlineInPreviewMode()

boolean OutlineInPreviewMode()

Return Value Return True if the shader will draw an outline. **ConvertToPoly()** must then be implemented.

I3DExShader::Contains(ShadingIn&, TVector2&)

boolean Contains(ShadingIn& shadingIn, TVector2& localUV)

Contains() is used for selecting the layer in the Preview window of the Texture Room. It must return true if the layer is selected and false otherwise.

Parameters **ShadingIn&** *shadingIn*
Reference to the *ShadingIn* structure.

TVector2& *localUV*
localUV returns the uv coordinates found in the uv coordinates of the layer.

©2001-2003 Eovia. All rights reserved.

7.24 I3dExSkinable Interface

Interface ID: IID_I3dExSkinable

Interface file: I3dExSkinable.h

Description

I3DExSkinable is an interface for a primitive that accepts bones. To create a primitive to which bones can be applied, you just need to derive it from this interface and implement its methods.

At this point only the vertex primitives support this interface.

If you implement the I3DExSkinable interface, it is recommended that your class offers a way to access weights data. For example, you can use the I3DExVertex interface to access the weights on a vertex of a vertex primitive (using SetSkiningData, GetSkiningData).

Method Index

"I3dExSkinable::AttachSkeleton"
"I3dExSkinable::DettachAllJoints"
"I3dExSkinable::IsAttached"
"I3dExSkinable::GetJoints"
"I3dExSkinable::InvalidateSkining"
"I3dExSkinable::GetInstance"
"I3dExSkinable::GetInstanceGlobalReferenceTransform"
"I3dExSkinable::SetInstanceGlobalReferenceTransform"

I3dExSkinable::AttachSkeleton

virtual void MCCOMAPI AttachSkeleton(TMCPtrArray<I3DShJoint>& joints,I3DShInstance* instance,boolean computeWeights=true)=0;

AttachSkeleton attaches a skeleton to the instance of an object for skinning. Note that the object should have only one instance.

Note: do NOT call it twice without first dettaching the Skeleton

Parameters

TMCPtrArray<I3DShJoint>& *joints*
joints is the list of the joints forming the skeleton. They MUST all have a common root.

I3DShInstance* *instance*
instance is a pointer to the instance of the primitive to skin.

boolean *computeWeights*
computeWeights is true to force the computing of the weights of the joints for each vertex. (only pass false if the weights are already there, for instance when loading a file).

I3dExSkinable::DettachAllJoints

virtual void MCCOMAPI DettachAllJoints() = 0;

DettachAllJoints detaches the skeleton from the primitive.

©2001-2003 Eovia. All rights reserved.

I3dExSkinable::IsAttached

virtual boolean MCCOMAPI IsAttached() = 0;

IsAttached returns true if the primitive is skinned.

Return Value true if the primitive is skinned

I3dExSkinable::GetJoints

virtual void MCCOMAPI GetJoints(TMCPtrArray<I3DShJoint>& joints) = 0;

GetJoints returns the joints attached to this primitive.

Parameters **TMCPtrArray<I3DShJoint>&** *joints*

joints is the list of the joints that apply to this primitive. (The array must be empty when you call this function).

I3dExSkinable::InvalidateSkining

virtual void MCCOMAPI InvalidateSkining() = 0;

InvalidateSkining is called when the transform of a joint changed so that the primitive can invalidate its caches. This is not a good place to recalculate the skinning since it can be called many times per frame !

I3dExSkinable::GetInstance

virtual voidMCCOMAPI GetInstance(I3DShInstance instance) = 0;**

GetInstance returns the instance of the primitive that is skinned.

Parameters **I3DShInstance**** *instance*

instance contains a pointer to the instance that is skinned when the function returns.

I3dExSkinable::GetInstanceGlobalReferenceTransform

virtual void MCCOMAPI GetInstanceGlobalReferenceTransform(TTransform3D& outTransform) = 0;

GetInstanceGlobalReferenceTransform returns the global transform of the instance (when the skining was created)

Parameters **TTransform3D&** *outTransform*

outTransform is the reference global transform of the instance.

I3dExSkinable::SetInstanceGlobalReferenceTransform

virtual void MCCOMAPI SetInstanceGlobalReferenceTransform(const TTransform3D& inTransform) = 0;

SetInstanceGlobalReferenceTransform sets the global transform of the instance (when loading a file)..

Parameters **const TTransform3D&** *inTransform*

inTransform is the reference global transform.

©2001-2003 Eovia. All rights reserved.

7.25 I3DExTileRenderer Interface

Interface ID:IID_I3DExTileRenderer

Interface file: I3DExFinalRenderer.h

Description

Rendering means building a 2D image from a 3D scene. Once the image is rendered, the post render filters can be applied. The tile renderer is a sub component of the renderer that is created for each frame. There is one tile renderer per rendering thread.

Method Index

"I3DExTileRenderer::Init"
"I3DExTileRenderer::GetNextSubTile"
"I3DExTileRenderer::RenderSubTile"
"II3DExTileRenderer::FinishRender"

I3DExTileRenderer::Init

virtual void MCCOMAPI Init(const TMCRect& rect,const TBBox2D& uvBox) = 0;

The **Init()** function is called at the beginning of the rendering of a tile.

Parameter

const TMCRect& *rect*
rect is the rectangle of the tile in the image (in pixel).

const TBBox2D& *uvBox*
uvBox is the rectangle of the tile in the image in screen coordinates.

I3DExTileRenderer::GetNextSubTile

virtual boolean MCCOMAPI GetNextSubTile(TMCRect& rect) = 0;

The **GetNextSubTile()** function is called by the shell to get the next sub tile that needs to render. If it returns false, the shell will consider that there is no more sub tiles to render and call **FinishRender**.

Parameter

TMCRect& *rect*
rect returns the rectangle of the sub tile in pixels.

Return Value true if there is one more sub tile to render

I3DExTileRenderer::RenderSubTile

virtual void MCCOMAPI RenderSubTile(RTData& pixels) = 0;

The **RenderSubTile** function is where the actual rendering should take place (if you do not support sub tiles, you can do your rendering in **FinishRender**).

Note: You need to call **IShUtilities::YieldProcesses** from time to time during the rendering if you want the user interface to be responsive while you render.

©2001-2003 Eovia. All rights reserved.

This call must be reentrant if you want your renderer to support SMP (if not return false in **IsSmpAware**). That means that you cannot use global variables in the call, and that you must protect with critical section any code that cannot be executed by two processors at the same time. Note that one tile renderer is created for each rendering thread so you can store information in the tile renderer that are specific to a particular thread without worrying about conflicts. If you need to build some caches shared by all tiles you can do that in **PrepareDraw** which is called only once per frame.

Parameter **RTData&** *pixels*

pixels is a structure that describes the image buffer as well as a the GBuffers for the sub tile (it points to the top left corner of the sub tile specified in **GetNextSubTile**). You can only update the sub tile, the rest of the tile is not accessible.

```
struct RTData
{
   uint8* red;// pointers to the pixels color
   uint8* green;
   uint8* blue;
   uint8* alpha;// null if we do need the alpha channel

   uint32  **buffers;// array of pointers to the GBuffers data
   uint32rowBytes;// offset to the next line
};
```

Return Value MC_S_OK

II3DExTileRenderer::FinishRender

virtual void MCCOMAPI FinishRender(const RTData& pixels) = 0;

The **FinishRender()** function is called at the end of the rendering of each tile. If you do not support sub tiles, you can perform the rendering of the tile here.

Parameter **RTData&** *pixels*

pixels is a structure that describes the image buffer as well as a the GBuffers for the tile (it points to the top left corner of the tile). You can only update the tile, the rest of the image is not accessible.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.26 I3DExTweener Interface

Family ID: ‘twee’

Interface ID: IID_I3DExTweener

Interface file: I3DExTweener.h

Description

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

Method Index

“I3DExTweener::SimpleTween(real&, int32, int32, int32)”
“I3DExTweener::Tween(I3DShKeyFrame*,int32, I3DShTweenerChainLink*)”

I3DExTweener::SimpleTween(real&, int32, int32, int32)

virtual MCCOMErr MCCOMAPI SimpleTween(real& result, int32 time, int32 time1, int32 time2)

SimpleTween() is called by the 3D Shell to perform interpolation at a given time. Replaces I3DExTweener::DoTweening from RDS.

Parameter

real& *result*
A value that tells the shell where to put the frame by interpolating linearly between the two extreme key frames.

int32 *time*
The time when to calculate the animation. Its unit is the ticks.

int32 *time1*
The time of the first key frame.

int32 *time2*
The time of the second key frame.

Comments

When *time* is equal to *time1*, return results=0.0; when *time* is equal to *time2*, return results=1.0.
For a linear tweener, the formula is:
*result=FixedToReal(time-time1)/FixedToReal(time2-time1);

I3DExTweener::Tween(I3DShKeyFrame*,int32, I3DShTweenerChainLink*)

virtual MCCOMErr MCCOMAPI Tween(I3DShKeyFrame* res,int32 time, I3DShTweenerChainLink* alink)

Tween() is more powerful than **SimpleTween()**. **Tween()** has access to the entire chain, current key frame, and the time. Anything from the timeline can be used, such as previous key frames and tweeners.

©2001-2003 Eovia. All rights reserved.

Parameter

I3DShKeyFrame* *res*
A pointer to the current key frame.

int32 *time*
The time when to calculate the animation. Its unit is the ticks.

I3DShTweenerChainLink* *alink*
A pointer to the entire chain.

©2001-2003 Eovia. All rights reserved.

7.27 I3DExTweenerEdition Interface

Family ID: ‘twee’

Interface ID: IID_I3DExTweenerEdition

Interface file: I3DExTweener.h

Description

The **I3DExTweenerEdition** Interface does **zot**. Tweeners are 3D Components that describe the interpolation between key frames.

I3DExTweener::PrepareEdit(I3DShTweenerChainLink*)

MCCOMErr MCCOMAPI PrepareEdit(I3DShTweenerChainLink* alink)

PrepareEdit() zot needs description .

Parameter **I3DShTweenerChainLink*** *alink*
A pointer to which link to edit.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.28 I3DExVertexPrimitive Interface

Interface ID: IID_I3DExVertexPrimitive

Interface file: I3DExVertexPrimitive.h

Description

I3DExVertexPrimitive is the main interface to access a vertex primitive. It can be useful if you want to export directly a vertex primitive (this allow to access the polygons data without having to go through triangulation).

You can query this interface (using QueryInterface) to know if a given primitive is a vertex primitive. If the primitive supports this interface it means that it is a vertex primitive.

Note that only a small part of the methods of this interface are described here. For more details please see the header file (I3DExVertexPrimitive.h)

Method Index

"I3dExVertexPrimitive::GetNbPolymeshes"
"I3dExVertexPrimitive::GetNbEdges"
"I3dExVertexPrimitive::GetNbPolygons"
"I3dExVertexPrimitive::GetPolymesh"
"I3dExVertexPrimitive::GetBoundingBox"
"I3dExVertexPrimitive::Pick"
"I3dExVertexPrimitive::DeleteSelectedEdges"
see header file for the rest of the methods...

I3dExVertexPrimitive::GetNbPolymeshes

virtual long MCCOMAPI GetNbPolymeshes()=0;

GetNbPolymeshes returns the number of polymeshes in the primitive.

I3dExVertexPrimitive::GetNbEdges

virtual long MCCOMAPI GetNbEdges()=0;

GetNbEdges returns the number of edges in this primitive

I3dExVertexPrimitive::GetNbPolygons

virtual long MCCOMAPI GetNbPolygons()=0;

GetNbPolygons returns the number of polygons in this primitive.

©2001-2003 Eovia. All rights reserved.

I3dExVertexPrimitive::GetPolymesh

virtual void MCCOMAPI GetPolymesh(IPolymesh polymesh, long index) =0;**

GetPolymesh returns a pointer to a polymesh.

Parameters

IPolymesh** *polymesh*
polymesh returns a pointer to the polymesh. (Use TMCCountedGetHelper to be sure that the reference count is correct)

long *index*
index is the index of the polymesh in the primitive must be inferior to the result of **GetNbPolymeshes**.

I3dExVertexPrimitive::GetBoundingBox

virtual void MCCOMAPI GetBoundingBox(TBBox3D& bbox, boolean ignoreSkining, boolean selection=false) =0;

GetBoundingBox Return the bounding box of the vertex primitive. To get the bounding box of the selection, set selection to true.

Parameters

TBBox3D& *bbox*
bbox returns the bounding box of the primitive

boolean *ignoreSkining*
ignoreSkining should be set to true in order to get the bounding box before the skinning is applied.

boolean *selection*
selection should be set to true to get the bounding box of the current selection. If it is set to false, the bounding box of the whole object will be returned.

I3dExVertexPrimitive::Pick

virtual long MCCOMAPI Pick(void obj,long flags, FatRay& fatRay, TVector3& hitPosition) =0;**

Not documented.

I3dExVertexPrimitive::DeleteSelectedEdges

virtual void MCCOMAPI DeleteSelectedEdges()=0;

DeleteSelectedEdges deletes all the selected edges.

©2001-2003 Eovia. All rights reserved.

7.29 I3DExVolumetricEffect Interface

Family ID: 'Volu'

Interface ID: IID_I3DExVolumetricEffect

Interface file: I3DExVolumetricEffect.h

Basic Implementation file: BasicVolumetricEffect.h

Description

A volumetric effect is an effect that is applied to simulate volume effects such as light cones, light spheres, volumetric aura that are attached to an object, a light or a camera...

At the beginning of the rendering of a frame, the shell go through all the data components (see I3DExDataComponent) of the lights, cameras and primitives and ask them if a volumetric effect should be created by calling **GetVolumetricList().**

It then creates one volumetric effect of each kind. The volumetric effects are then called for each ray to determine its effects on the light going through the ray.

Method Index

"I3DExVolumetricEffect::Initialise"
"I3DExVolumetricEffect::DirectionFilter"

I3DExVolumetricEffect::Initialise

virtual void MCCOMAPI Initialise(I3DShScene* scene)=0;

Initialise is called at the beginning of the rendering of a frame. It allows you to create caches for your effect. For example if you are writing a light cone, this is the place to go though the scene and retrieve all the spot lights.

Parameters **I3DShScene*** *scene*
scene is a pointer to the scene that is being rendered.

I3DExVolumetricEffect::DirectionFilter

virtual void MCCOMAPI DirectionFilter(const TVector3& origin,const TVector3& direction,real maxt,TMCColor-RGBA& inOutFilter,boolean isShadowCasting) = 0;

DirectionFilter should filter a color with the volumetricEffect. When the function is called, inOutFilter contains the color of an object at the end point. When it returns inOutFilter should have been modified to include the filtering effect.

Parameters **const TVector3&** *origin*
origin is the point of the ray from which we see.

©2001-2003 Eovia. All rights reserved.

const TVector3& *direction*
direction is the direction of the ray (direction in which we look).

TMCColorRGBA& *inOutFilter*
inOutFilter is color of the end point (when called), color after filtering (when returning).

boolean *isShadowCasting*
isShadowCasting is true if this is a shadow ray.

©2001-2003 Eovia. All rights reserved.

7.30 I3DExWireFrame Interface

Interface ID: IID_I3DExWireFrame

Interface file: I3DWireFrame.h

Description

The **I3DExWireFrame** Interface allows you to activate, deactivate, and position wire frames. Wire frames are the lines projected by an object that display on the planes of the working box.

Method Index

"I3DExWireFrame::TrackWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*, int16, const TRACKINFO&, const TRACKINFO&, const TRACKINFO&)"
"I3DExWireFrame::DataToWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*)"
"I3DExWireFrame::WireFrameToData(I3DShWireFrame*, int16, I3DShTreeElement*)"
"I3DExWireFrame::GetWireFrameBBox(I3DShTweenerChainLink*)"
"I3DExWireFrame::Activate()"
"I3DExWireFrame::Deactivate()"
"I3DExWireFrame::HandlesTool(int16)"

I3DExWireFrame::TrackWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*, int16, const TRACKINFO&, const TRACKINFO&, const TRACKINFO&)

MCCOMErr MCCOMAPI TrackWireFrame(I3DShWireFrame* wireFrame, int16 proj, I3DShTreeElement* tree, int16 handle, const TRACKINFO&startinfo, const TRACKINFO& previnfo, const TRACKINFO& nextinfo)

TrackWireFrame() tracks the location of the specfied wireframe, on the specified projection of the working box, for the specified tree element.

Parameter

I3DShWireFrame* *wireFrame*
A pointer to the specified wire frame to track.

int16 *proj*
Specifies the projection of the working box.

I3DShTreeElement* *tree*
A pointer to tree of the object.

int16 *handle*
zot needs description.

const TRACKINFO& *startinfo*
zot needs description.

const TRACKINFO& *previnfo*
zot needs description.

©2001-2003 Eovia. All rights reserved.

const TRACKINFO& *nextinfo*
zot needs description.

I3DExWireFrame::DataToWireFrame(I3DShWireFrame*, int16, I3DShTreeElement*)

MCCOMErr MCCOMAPI DataToWireFrame(I3DShWireFrame* wireFrame, int16 proj, I3DShTreeElement* tree)

DataToWireFrame() orients the working box to the location of the wire frame.

Parameter

I3DShWireFrame* *alink*
A pointer to the specified wire frame.

int16 *proj*
Specifies the projection of the working box.

I3DShTreeElement* *tree*
A pointer to tree of the object.

I3DExWireFrame::WireFrameToData(I3DShWireFrame*, int16, I3DShTreeElement*)

MCCOMErr MCCOMAPI WireFrameToData(I3DShWireFrame* wireFrame, int16 proj, I3DShTreeElement* tree)

WireFrameToData() orients the wire frame to the location of the working box.

Parameter

I3DShWireFrame* *alink*
A pointer to the specified wire frame.

int16 *proj*
Specifies the projection of the working box.

I3DShTreeElement* *tree*
A pointer to tree of the object.

I3DExWireFrame::GetWireFrameBBox(I3DShTweenerChainLink*)

MCCOMErr MCCOMAPI GetWireFrameBBox(TBBox3D* outBBox, int16proj, I3DShTreeElement*tree)

GetWireFrameBBox() gets the bounding box for the wire frame.

Parameter

TBBox3D* *outBBox*
A pointer to the resulting bounding box.

int16 *proj*
Specifies the projection of the working box.

I3DShTreeElement* *tree*
A pointer to tree of the object.

©2001-2003 Eovia. All rights reserved.

I3DExWireFrame::Activate()

MCCOMErr MCCOMAPI Activate()

Activate() activates the wire frame.

I3DExWireFrame::Deactivate()

MCCOMErr MCCOMAPI Deactivate()

Deactivate() deactivates the wire frame.

I3DExWireFrame::HandlesTool(int16)

boolean MCCOMAPI HandlesTool(int16 inTool)

HandlesTool() zot needs description .

Parameter **int16** *inTool*
zot needs description.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.31 IRaytracablePrimitive Interface

Interface ID: IID_IRaytracablePrimitive

Interface file: I3DExPrimitive.h

Description

In addition to describing geometry a geometric primitive can define a custom ray-tracing routine rather than relying only on facets or patches. This is done by inheriting from the **"IRaytracablePrimitive Interface"** in addition to the **"I3DExGeometricPrimitive Interface"** (or TBasicPrimitive).

Methods Index

"IRaytracablePrimitive::RayHit(boolean*, Ray3D*, RayHitParameters*, RayHit3D*)"
"IRaytracablePrimitive::GetRayHitDetails(RayHit3D*)"
"IRaytracablePrimitive::IsInfiniteBB(boolean&)"

IRaytracablePrimitive::RayHit(boolean*, Ray3D*, RayHitParameters*, RayHit3D*)

MCCOMErr RayHit(boolean* didHit, Ray3D* ray, RayHitParameters* hitParams, RayHit3D* hit)

RayHit() 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.

Call **IRaytracablePrimitive::GetRayHitDetails** if there is a hit and that you want more details on it.

If **IRaytracablePrimitive::RayHit** is implemented, then **IRaytracablePrimitive::GetRayHitDetails** must be implemented as well.

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

Parameters

boolean* *didHit*
Return TRUE in this parameter if a hit occurred.

Ray3D* *ray*
This is the ray provided by the 3D Shell.

RayHitParameters* ***hitParams***
These parameter give the minimum and the maximum 't' Ray parameter above which the hit is allowed to occur. See the Data Structure reference for "RayHitParameters" for more information.

RayHit3D* ***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 in *hit*.

©2001-2003 Eovia. All rights reserved.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

IRaytracablePrimitive::GetRayHitDetails(RayHit3D*)

MCCOMErr GetRayHitDetails(RayHit3D* hit)

GetRayHitDetails() is called after a ray-tracing hit occurred. If **IRaytracablePrimitive::GetRayHitDetails** is implemented, then **IRaytracablePrimitive::RayHit** must be implemented as well.

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

Parameter **RayHit3D*** *hit*

Points to a record that must be filled-in. See the Data Structure on "RayHit3D".

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

IRaytracablePrimitive::IsInfiniteBB(boolean&)

MCCOMErr IsInfiniteBB(boolean& isInfinite)

If the primitive has an infinite bounding box (i.e. the infinite plane) this method must return TRUE.

Parameter **boolean&** *isInfinite*

TRUE if the primitive has an infinite bounding box; FALSE if not.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

©2001-2003 Eovia. All rights reserved.

7.32 ISceneDocument Interface

Interface ID: IID_ISceneDocument

Interface file: ISceneDocument.h

Description

zot needs description. A scene is a collection of objects stored in a file and accessed through IDShScene. A scene also includes documents which are accessed through ISceneDocument. You can access the document windows, specify which room (module) you want to use, specify how you want the scene to be rendered in the window, and start or stop animation previews.

Method Index

"ISceneDocument::GetScene(I3DShScene**)"
"ISceneDocument::GetSceneSelection(ISceneSelection**)"
"ISceneDocument::GetSceneSelectionChannel(IChangeChannel**)"
"ISceneDocument::GetTreeHierarchyChannel(IChangeChannel**)"
"ISceneDocument::GetDocumentWindowByID(IMFPart**, IDType)"
"ISceneDocument::GetModuleByID(I3DShModule**, IDType)"
"ISceneDocument::GetSceneRenderingModule(I3DRenderingModule**)"
"ISceneDocument::StopAnimPreview()"
"ISceneDocument::StartAnimPreview()"

ISceneDocument::GetScene(I3DShScene**)

voidMCCOMAPI GetScene(I3DShScene)**

GetScene() is called to get a scene. **zot** needs more description.

Parameters **I3DShScene****
Specifies the scene.

ISceneDocument::GetSceneSelection(ISceneSelection**)

void MCCOMAPI GetSceneSelection(ISceneSelection)**

GetSceneSelection() is called to get a scene selection. **zot** needs more description.

Parameters **ISceneSelection****
Specifies the scene selection.

ISceneDocument::GetSceneSelectionChannel(IChangeChannel**)

void MCCOMAPI GetSceneSelectionChannel(IChangeChannel)**

GetSceneSelectionChannel() is called to get a scene selection channel. **zot** needs more description.

Parameters **IChangeChannel****
Specifies the selection channel.

©2001-2003 Eovia. All rights reserved.

ISceneDocument::GetTreeHierarchyChannel(IChangeChannel**)

void MCCOMAPI GetTreeHierarchyChannel(IChangeChannel)**

GetTreeHierarchyChannel() is called to get a tree hierarchy channel. **zot** needs more description.

Parameters **IChangeChannel****
Specifies the selection channel.

ISceneDocument::GetDocumentWindowByID(IMFPart**, IDType)

voidMCCOMAPI GetDocumentWindowByID(IMFPart, IDType inWindowID)**

GetDocumentWindowByID() is called to get a document window by specifying the window ID. **zot** needs more description.

Parameters **IMFPart****
zot needs description.

IDType *inWindowID*
zot needs description.

ISceneDocument::GetModuleByID(I3DShModule**, IDType)

voidMCCOMAPI GetModuleByID(I3DShModule, IDType inModuleID)**

ModuleByID() is called to get a module by specifying the ID. **zot** needs more description.

Parameters **I3DShModule****
zot needs description.

IDType *inModuleID*
zot needs description.

ISceneDocument::GetSceneRenderingModule(I3DRenderingModule**)

voidMCCOMAPI GetSceneRenderingModule(I3DRenderingModule renderingModule)**

GetSceneRenderingModule() is called to get a rendering module. **zot** needs more description.

Parameters **I3DRenderingModule**** *renderingModule*
zot needs description.

ISceneDocument::StopAnimPreview()

booleanMCCOMAPI StopAnimPreview()

StopAnimPreview() is called to stop an animation preview. **zot** needs more description.

ISceneDocument::StartAnimPreview()

voidMCCOMAPI StartAnimPreview()

StartAnimPreview() is called to start an animation preview. **zot** needs more description.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

7.33 IVolumePrimitive Interface

Interface ID: IID_IVolumePrimitive

Interface file: I3DExPrimitive.h

Description

Rather than defining geometry, a primitive can also describe a volume. This is achieved by inheriting from the **"IVolumePrimitive Interface"** as well as the **"I3DExGeometricPrimitive Interface"** (or TBasicPrimitive).

There are two methods for implementing a volume primitive. **IVolumePrimitive::GetVolumedensity** provides a filter color and a glow color for each point of the volume primitive. **IVolumePrimitive::GetVolumeAttenuation** calculates the color from a ray.

Method Index

"IVolumePrimitive::GetVolumeDensity(const TVector3&, TMCColorRGB&, TMCColorRGB&)"
"IVolumePrimitive::GetVolumeAttenuation(const TVector3&, constTVector3&, TMCColorRGB&)"

IVolumePrimitive::GetVolumeDensity(const TVector3&, TMCColorRGB&, TMCColorRGB&)

MCCOMErr GetVolumeDensity(const TVector3& point, TMCColorRGB& filter, TMCColorRGB& glow)

GetVolumeDensity() is called during rendering to calculate a color from a ray going thru the volume primitive. The shell calls this method to get a filtered color and a glow color for many points on the ray. It begins with a base color from an object or background color behind the volumetric primitive to calculate the color within the volume:

Volume Color = (base color * filtered color) + glow

Parameters

const TVector3& *point*
The point within the volume in local coordinates.

TMCColorRGB& ***filter***
The resulting filtered color.

TMCColorRGB& ***glow***
The resulting glow value.

Return Value

MC_S_OK	
MC_E_NOTIMPL	Call not implemented.

©2001-2003 Eovia. All rights reserved.

IVolumePrimitive::GetVolumeAttenuation(const TVector3&, constTVector3&, TMCColorRGB&)

MCCOMErr GetVolumeAttenuation(const TVector3& from, const TVector3& to, TMCColorRGB& attenuation)

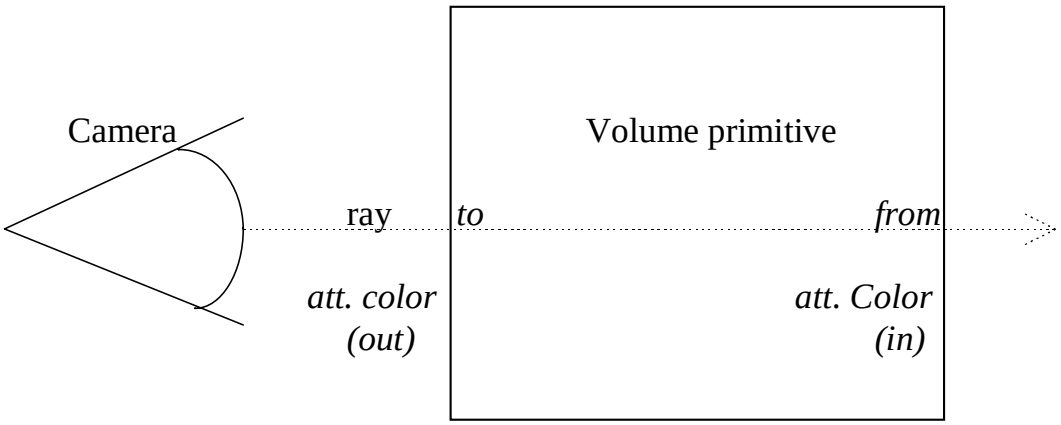
GetVolumeAttenuation() is called by the shell to get a color from a ray going thru a volume primitive.

Parameters

const TVector3& *from*
The intersection of the ray with the back of the volume primitive. This is where the calculation begins.

const TVector3& ***to***
The intersection of the ray with the front of the volume primitive. This is where the calculation ends.

TMCColorRGB& ***attenuation***
The color behind the object is given. The resulting color is returned.



Return Value

MC_S_OK	
MC_E_NOTIMPL	This method is not supported. **IVolumePrimitive::GetVolumeDensity** will be used**.**

©2001-2003 Eovia. All rights reserved.

Chapter 8 - MCCommon ISh* Reference

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

8.1 IShAction Interface

Interface ID: IID_IShAction

Interface file: IShUtilities.h

Description

The **IShAction** Interface contains methods to do and undo changes. <zot needs more description>

Methods Index

"IShAction::Do()"
"IShAction::Undo()"
"IShAction::Redo()"
"IShAction::WillCauseChange()"
"IShAction::CanUndo()"
"IShAction::GetPartToRedraw(IMFPart**, int32)"
"IShAction::GetName(TMCString&)"
"IShAction::NeedsToPrepareMenus(void)"

IShAction::Do()

MCCOMErr MCCOMAPI Do()

Do() is called to do the action. Since the constructor of the action is not protected by the exception mecanism, no memory allocations (like 'pointer = new ...') should be made in the constructor. Make only initializations (like 'pointer = NULL;').

IShAction::Undo()

MCCOMErr MCCOMAPI Undo()

Undo() is called to undo the action. When undone, an action must restore the database in the EXACT SAME state than before Do is called. This means that if a pointer is cloned, the action must store the original pointer in Do(), and apply its modifications to the clone. And therefore when Undo() is called, the action simply replaces the cloned and modified pointer by the orginal and unmodified one.

If CanUndo() returns False, just Undo() should return MC_E_NOTIMPL.

IShAction::Redo()

MCCOMErr MCCOMAPI Redo()

Redo() is called to redo the action.

If CanUndo() returns False, just Redo() should return MC_E_NOTIMPL.

IShAction::WillCauseChange()

boolean MCCOMAPI WillCauseChange()

©2001-2003 Eovia. All rights reserved.

WillCauseChange() returns True if the action will modify the database.

IShAction::CanUndo()

boolean MCCOMAPI CanUndo()

CanUndo() returns True if the action is undoable. If CanUndo() returns False, just Undo() and Redo() should return MC_E_NOTIMPL.

IShAction::GetPartToRedraw(IMFPart**, int32)

void MCCOMAPI GetPartToRedraw(IMFPart outPart, int32 stage)**

GetPartToRedraw() provides an opportunity to the action to tell the Shell that the returned view should be redrawn after Do/Undo/Redo. Return NULL if no redraw is needed.

Parameter

IMFPart** *outPart*
The resulting part that is redrawn.

int32 *stage*
The area to be redrawn.

IShAction::GetName(TMCString&)

MCCOMErr MCCOMAPI GetName(TMCString& name)

GetName() returns the name to display in the undo/redo list. To avoid localization problems, don't use hardcoded strings such as char name[] ="My Tool"; Use strings from resources instead.

Parameter

TMCString& *name*
The name to be displayed in the undo/redo list.

IShAction::NeedsToPrepareMenus(void)

boolean MCCOMAPI NeedsToPrepareMenus(void)

NeedsToPrepareMenus() returns True if execution (Do()/Undo()) of the action should cause the menus to be prepared. For example, the contents of the menus need to be enabled or diasabled.

Parameter

void
<zot needs description>.

©2001-2003 Eovia. All rights reserved.

8.2 IShActionManager Interface

Interface ID: IID_IShActionManager

Interface file: IShUtilities.h

Description

The **IShActionManager** Interface contains methods to launch and measure mouse actions and keyboard clicks. <zot needs more description>

Methods Index

"IShActionManager::PostAction(IShAction*, int32, IMFResponder*)"
"IShActionManager::PostMouseAction(IShMouseAction*, int32, IMFResponder* IMFPart*, const TMCPoint&)"
"IShActionManager::IsOptionDown()"
"IShActionManager::IsShiftDown()"
"IShActionManager::IsCommandDown()"
"IShActionManager::IsControlDown()"
"IShActionManager::IsLongStaticClick()"
"IShActionManager::IsShortStaticClick()"

IShActionManager::PostAction(IShAction*, int32, IMFResponder*)

MCCOMErr MCCOMAPI PostAction(IShAction* action,int32 actionNbr,IMFResponder *context)

PostAction() launches an action.

Parameter

IShAction* *action*
A pointer to the action.

int32 *actionNbr*
The action number.

IMFResponder* *context*
The context should be the IMFResponder interface of the document.

IShActionManager::PostMouseAction(IShMouseAction*, int32, IMFResponder* IMFPart*, const TMCPoint&)

MCCOMErr MCCOMAPI PostMouseAction(IShMouseAction* action,int32 actionNbr,IMFResponder *context, IMFPart *part,const TMCPoint &mouse)

PostMouseAction() launches a mouse action.

Parameter

IShAction* *action*
A pointer to the action.

int32 *actionNbr*
The action number.

©2001-2003 Eovia. All rights reserved.

IMFResponder* *context*
The context should be the IMFResponder interface of the document.

IMFPart* *part*
A pointer to the part.

const TMCPoint& *mouse*
A reference to the mouse location.

IShActionManager::IsOptionDown()

boolean MCCOMAPI IsOptionDown()

IsOptionDown() returns True if the Option key is depressed.

IShActionManager::IsShiftDown()

boolean MCCOMAPI IsShiftDown()

IsShiftDown() returns True if the Shift key is depressed.

IShActionManager::IsCommandDown()

boolean MCCOMAPI IsCommandDown()

IsCommandDown() returns True if the Command key is depressed.

IShActionManager::IsControlDown()

boolean MCCOMAPI IsControlDown()

IsControlDown() returns True if the Control key is depressed on a Macintosh.

IShActionManager::IsLongStaticClick()

boolean MCCOMAPI IsLongStaticClick()

IsLongStaticClick() returns True if click was short and the mouse did not move too much. Call this method if the mouse is already down.

IShActionManager::IsShortStaticClick()

boolean MCCOMAPI IsShortStaticClick()

IsShortStaticClick() returns True if click was long and the mouse did not move too much. Call this method if the mouse is already down.

©2001-2003 Eovia. All rights reserved.

8.3 IShComponent Interface

Interface ID: IID_IShComponent

Interface file: IShComponent.h

Description

The **IShComponent** Interface contains methods for identifying a component. <zot needs more description>

Methods Index

"IShComponent::GetFamilySignature()"
"IShComponent::GetClasSignature()"
"IShComponent::GetClassName(TMCString&)"
"IShComponent::IsDefaultComponent()"
"IShComponent::InvalidatePreview()"

IShComponent::GetFamilySignature()

IDTypeMCCOMAPI GetFamilySignature()

GetFamilySignature() gets the family signature. <zot needs description.>

<zot: is the signature the same as the Family ID in the COMP resource? If so, I can add more to this.>

IShComponent::GetClasSignature()

IDTypeMCCOMAPI GetClassSignature()

GetClassSignature() gets the signature of the class. <zot needs description>.

<zot: is signature the same as the Class ID in the COMP resource? If so, I can add more to this.>

IShComponent::GetClassName(TMCString&)

MCCOMErrMCCOMAPI GetClassName(TMCString& name)

GetClassName() gets the name of the class.

Parameter **TMCString&** *name*
A reference to the name of the class .

IShComponent::IsDefaultComponent()

booleanMCCOMAPI IsDefaultComponent()

IsDefaultComponent() usually returns False, returns True for components who use a default instance (like

©2001-2003 Eovia. All rights reserved.

dataComponents).

IShComponent::InvalidatePreview()

voidMCCOMAPI InvalidatePreview()

InvalidatePreview() invalidates the component preview (if there is any).

©2001-2003 Eovia. All rights reserved.

8.4 IShComponentUtilities Interface

Interface ID: IID_IShComponentUtilities

Interface file: IShUtilities.h

Description

The **IShComponentUtilities** Interface contains methods to create, read, and write components. <zot needs more description>

Methods Index

"IShComponentUtilities::CoCreateInstance(const MCCLSID&, IMCUnknown*, uint32, const MCIID&, void**)"
"IShComponentUtilities::CreateComponent(IDType, IDType, IShComponent**)"
"IShComponentUtilities::CreateIUnknown(IDType, IDType, IMCUnknown*, IMCUnknown**)"
"IShComponentUtilities::CreateDefaultComponent(IDType, IShComponent**)"
"IShComponentUtilities::GetComponentClassInfoArrayByFamilyID(IDType, TMCArray<TComponent-ClassInfo>&)"
"IShComponentUtilities::WriteComponent(IShTokenStream*, IShParameterComponent*)"
"IShComponentUtilities::WriteComponent(IShTokenStream*, IDType, IShParameterComponent*)"
"IShComponentUtilities::ReadComponent(IShTokenStream*, IShParameterComponent**)"
"IShComponentUtilities::ReadComponent(IShTokenStream*, IShParameterComponent**, IDType)"
"IShComponentUtilities::ReadComponentData(TMCiostream*, IShParameterComponent*)"
"IShComponentUtilities::AskForParams(IShParameterComponent*)"

IShComponentUtilities::CoCreateInstance(const MCCLSID&, IMCUnknown*, uint32, const MCIID&, void**)

MCCOMErr MCCOMAPI CoCreateInstance(const MCCLSID& rclsid, IMCUnknown* pUnkOuter, uint32 dwClsContext, const MCIID& riid, void ppv)**

CoCreateInstance() creates an instance of a component.

Parameter

const MCCLSID& *relsid*
A reference to the MetaCreations class id.

IMCUnknown* *pUnkOuter*
A pointer to the <zot needs definition>.

uint32 *dwClsContext*
The class context value.

const MCIID& *riid*
The class IID_ value.

void** *ppv*
The <zot needs defintion>.

©2001-2003 Eovia. All rights reserved.

IShComponentUtilities::CreateComponent(IDType, IDType, IShComponent**)

MCCOMErr MCCOMAPI CreateComponent(IDType familyID, IDType classID, IShComponent component)**

CreateComponent() creates a component, given the family id, the class id, and returns a pointer to the component. For example, CreateInstanceByID('gel ', 'blnd', &gel) creates a "blinds" gel.

Parameter

IDType *familyID*
The family id of the component being created.

IDType *classID*
The class id of the component being created.

IShComponent** *component*
A pointer to the component created using the first two arguments.

IShComponentUtilities::CreateIUnknown(IDType, IDType, IMCUnknown*, IMCUnknown**)

MCCOMErr MCCOMAPI CreateIUnknown(IDType familyID, IDType classID, IMCUnknown* pUnkOuter, IMCUnknown component)**

CreateIUnknown() creates a component, given the family id, the class id, and returns a pointer to the component.

Parameter

IDType *familyID*
The family id of the component being created.

IDType *classID*
The class id of the component being created.

IMCUnknown* *pUnkOuter*
A pointer to the <zot needs definition>.

IMCUnknown** *component*
A pointer to the component created using the first two arguments.

IShComponentUtilities::CreateDefaultComponent(IDType, IShComponent**)

MCCOMErr MCCOMAPI CreateDefaultComponent(IDType familyID, IShComponent component)**

CreateDefaultComponent() creates a component, given the family id, and returns a pointer to the component.

Parameter

IDType *familyID*
The family id of the component being created.

IShComponent** *component*
A pointer to the component created using the first argument.

©2001-2003 Eovia. All rights reserved.

IShComponentUtilities::GetComponentClassInfoArrayBy-FamilyID(IDType, TMCArray<TComponentClassInfo>&)

MCCOMErr MCCOMAPI GetComponentClassInfoArrayByFamilyID(IDType familyID, TMCArray<TComponent-ClassInfo>& componentClassInfoArray)

GetComponentClassInfoArrayByFamilyID() gets the component class information array, given the family id, and returns a reference to the array.

Parameter

IDType *familyID*
The family id of the component.

TMCArray<TComponentClassInfo>& *componentClassInfoArray*
A reference to the array of the component indicated using the first argument.

IShComponentUtilities::WriteComponent(IShTokenStream*, IShParam-eterComponent*)

MCCOMErr MCCOMAPI WriteComponent(IShTokenStream* stream, IShParameterComponent* component)

WriteComponent() writes a specified component to the stream.

Parameter

IShTokenStream* *stream*
A pointer to the stream.

IShParameterComponent* *component*
A pointer to the component.

IShComponentUtilities::WriteComponent(IShTokenStream*, IDType, IShParameterComponent*)

MCCOMErr MCCOMAPI WriteComponent(IShTokenStream* stream, IDType keyword, IShParameterCompo-nent* component)

WriteComponent() writes a specified component with a specified keyword to the stream.

Parameter

IShTokenStream* *stream*
A pointer to the stream.

IDType *keyword*
The keyword specfies the id type.

IShParameterComponent* *component*
A pointer to the component.

IShComponentUtilities::ReadComponent(IShTokenStream*, IShParam-eterComponent**)

MCCOMErr MCCOMAPI ReadComponent(IShTokenStream* stream, IShParameterComponent component)**

ReadComponent() reads a specified component from the stream.

©2001-2003 Eovia. All rights reserved.

Parameter

IShTokenStream* *stream*
A pointer to the stream.

IShParameterComponent** *component*
A pointer to the component.

IShComponentUtilities::ReadComponent(IShTokenStream*, IShParameterComponent**, IDType)

MCCOMErr MCCOMAPI ReadComponent(IShTokenStream* stream, IShParameterComponent component, IDType familyID)**

ReadComponent() reads a specified component from the stream. Use this method when the family token has already been read (the begin token is still in the stream).

Parameter

IShTokenStream* *stream*
A pointer to the stream.

IShParameterComponent** *component*
A pointer to the component.

IDType *familyID*
The family id of the component being read.

IShComponentUtilities::ReadComponentData(TMCiostream*, IShParameterComponent*)

MCCOMErr MCCOMAPI ReadComponentData(TMCiostream* stream,IShParameterComponent* component)

ReadComponentData() reads the data from a specified component from the stream.

Parameter

IShTokenStream* *stream*
A pointer to the stream.

IShParameterComponent* *component*
A pointer to the component.

IShComponentUtilities::AskForParams(IShParameterComponent*)

boolean MCCOMAPI AskForParams(IShParameterComponent *component)

AskForParams() returns True if there are parameters for the component.

Parameter

IShParameterComponent* *component*
A pointer to the component.

©2001-2003 Eovia. All rights reserved.

8.5 IShParameterComponent Interface

Interface ID: IID_IShParameterComponent

Interface file: IShComponent.h

Description

The **IShParameterComponent** Interface contains methods for identifying and accessing a parameter component. <zot needs more description>

Methods Index

"IShParameterComponent::GetFamilySignature()"
"IShParameterComponent::GetClasSignature()"
"IShParameterComponent::GetClassName(TMCString&)"
"IShParameterComponent::GetParameter(IDType, void*)"
"IShParameterComponent::SetParameter(IDType, void*, boolean)"
"IShParameterComponent::GetParameterType(IDType, IDType&)"
"IShParameterComponent::GetParameterCount()"
"IShParameterComponent::GetParameterInfo(uint32, TPMapElement&)"
"IShParameterComponent::Inval(boolean, boolean)"

IShParameterComponent::GetFamilySignature()

IDTypeMCCOMAPI GetFamilySignature()

GetFamilySignature() gets the family signature. <zot needs description.>

<zot: is the signature the same as the Family ID in the COMP resource? If so, I can add more to this.>

IShParameterComponent::GetClasSignature()

IDTypeMCCOMAPI GetClassSignature()

GetClassSignature() gets the signature of the class. <zot needs description>.

<zot: is signature the same as the Class ID in the COMP resource? If so, I can add more to this.>

IShParameterComponent::GetClassName(TMCString&)

MCCOMErrMCCOMAPI GetClassName(TMCString& name)

GetClassName() gets the name of the class.

Parameter **TMCString&** *name*
A reference to the name of the class .

©2001-2003 Eovia. All rights reserved.

IShParameterComponent::GetParameter(IDType, void*)

MCCOMErr MCCOMAPI GetParameter(IDType keyword, void* parameter)

GetParameter() gets the parameter, given the IDType.

Parameter

IDType *keyword*
<zot needs description>.

void* *parameter*
A pointer to the parameter.

IShParameterComponent::SetParameter(IDType, void*, boolean)

MCCOMErr MCCOMAPI SetParameter(IDType keyword, void* parameter, boolean inNotify=true)

SetParameter() sets the parameter, given the IDType, and returns True unless <zot needs description>.

Parameter

IDType *keyword*
<zot needs description>.

void* *parameter*
A pointer to the parameter.

boolean *inNotify*
Set to True, <zot needs description>. Returns False if <zot needs description>

IShParameterComponent::GetParameterType(IDType, IDType&)

MCCOMErr MCCOMAPI GetParameterType(IDType keyword, IDType& parameterType)

GetParameterType() gets the parameter type, given the IDType, updates the reference to that IDType <zot needs description>.

Parameter

IDType *keyword*
<zot needs description>.

IDType& *parameterType*
A reference to the parameter type.

IShParameterComponent::GetParameterCount()

uint32 MCCOMAPI GetParameterCount()

GetParameterCount() returns the count of the parameters.

IShParameterComponent::GetParameterInfo(uint32, TPMapElement&)

MCCOMErr MCCOMAPI GetParameterInfo(uint32 index, TPMapElement& info)

GetParameterInfo() gets the parameter information given an index, and returns a reference to the information. <zot needs description>.

©2001-2003 Eovia. All rights reserved.

Parameter **uint32** *index*
<zot needs description>.

TPMapElement& *info*
A reference to <zot needs description>

IShParameterComponent::Inval(boolean, boolean)

void MCCOMAPI Inval(boolean inNotifyExternal = true, boolean inNotifyUI = false)

Inval() <zot needs description>.

Parameter **boolean** *inNotifyExternal*
Returns True when <zot needs description>. Returns False when <zot needs description>.

boolean *inNotifyUI*
Returns False when <zot needs description>. Returns True when <zot needs description>.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

8.6 IShUtilities Interface

Interface ID: IID_IShUtilities

Interface file: IShUtilities.h

Description

The **IShUtilities** Interface contains methods for accessing a progress bar, tracking and reporting information about the progress. It also manages resources related to the process. <zot needs more description>

Methods Index

"IShUtilities::BeginProgress(TMCString&, IMCUnknown**, real32)"
"IShUtilities::EndProgress(IMCUnknown*)"
"IShUtilities::SetProgressPrompt(TMCString&, IMCUnknown*)"
"IShUtilities::IncrementProgress(real32, IMCUnknown*)"
"IShUtilities::SetProgressValue(real32, IMCUnknown*)"
"IShUtilities::SetProgressValue0to1(real32, IMCUnknown*)"
"IShUtilities::YieldProcesses(uint32)"
"IShUtilities::TickCount()"
"IShUtilities::Random()"
"IShUtilities::SetRandomSeed(uint32)"
"IShUtilities::RandomStep(int32)"
"IShUtilities::GetAPINumber()"
"IShUtilities::Gestalt(void*, uint32)"
"IShUtilities::DoAlert(EAlerts, TMCString&, TMCString&, TMCString&, TMCString&, int32, TMC-String&, TMCString&, TMCString&)"
"IShUtilities::ErrorToString(MCCOMErr, TMCString&)"
"IShUtilities::BuildAlertMessage(MCCOMErr, TMCString&, TMCString&)"
"IShUtilities::GetResource(int32, int32, int32, void**)"
"IShUtilities::SetupResources(int32, int32, void**)"
"IShUtilities::RestoreResources(void*)"
"IShUtilities::BlitUIImage(IMCGraphicContext*, TMCRect&, TMCRect&, uint32)"
"IShUtilities::GetUIImageInfo(uint32, int32&, int32&, int32)"
"IShUtilities::GetUIImageBucket(uint32, TMCPixelBucket*)"
"IShUtilities::GetUIImage(uint32, TMCUIImage**)"
"IShUtilities::CreateCompatiblePixelBucket(int32, int32, int32, TMCCompatiblePixelBucket**)"
"IShUtilities::DestroyCompatiblePixelBucket(TMCCompatiblePixelBucket*)"
"IShUtilities::GetAsciiTranslationTable()"
"IShUtilities::GetOpenDocumentsCount()"
"IShUtilities::GetOpenDocumentByIndex(IMFDocument**, index)"
"IShUtilities::GetLastOpenDocPath(TMCString&)"
"IShUtilities::GetLastSaveDocPath(TMCString&)"
"IShUtilities::SetLastOpenDocPath(TMCString&)"
"IShUtilities::SetLastSaveDocPath(TMCString&)"
"IShUtilities::UseMMX()"
"IShUtilities::IsPentiumProcessor()"
"IShUtilities::IsPentiumIIProcessor()"
"IShUtilities::IsPentiumIIIProcessor()"
"IShUtilities::StringWidth(TMCTextStyle&, TMCString&)"
"IShUtilities::GetTextMetrics(TMCTextStyle&, TMCTextMetrics&)"
"IShUtilities::GetCreator()"
"IShUtilities::SetCursor(int32)"

©2001-2003 Eovia. All rights reserved.

"IShUtilities::GetFontList(IMCFontList**)"
"IShUtilities::CheckForUserCancel()"
"IShUtilities::GetCurrentModifiers()"
"IShUtilities::CheckKeyDown(int16)"

IShUtilities::BeginProgress(TMCString&, IMCUnknown**, real32)

void MCCOMAPI BeginProgress(TMCString& prompt, IMCUnknown outProgressKey, real32 inMaxValue = 100.0f)**

BeginProgress() begins a progress bar. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress.

Parameter

TMCString& *prompt*
A reference to the string that displays with the progress bar.

IMCUnknown** *outProgressKey*
A pointer to the progress key used to specify which progress the bar is tracking, and tracks the progress of the process.

real32 *inMaxValue*
The maximum value of a progressKey is set to 100.0f by default. <zot what is the range or typical range?>

IShUtilities::EndProgress(IMCUnknown*)

void MCCOMAPI EndProgress(IMCUnknown* inProgressKey)

EndProgress() ends the progress bar for the progress specified by the progress key. <zot needs description.>

Parameter

IMCUnknown** *inProgressKey*
Identifies which progress is being ended. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress.

IShUtilities::SetProgressPrompt(TMCString&, IMCUnknown*)

void MCCOMAPI SetProgressPrompt(TMCString& inPrompt, IMCUnknown* inProgressKey = NULL)

SetProgressPrompt() sets up the string that displays while process is occuring.

Parameter

TMCString& *inPrompt*
A reference to the string that displays while the progress bar is progressing.

IMCUnknown* *inProgressKey*
Identifies which progress is being set. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress. Progress values are between 0.0f and maxValue.

IShUtilities::IncrementProgress(real32, IMCUnknown*)

voidMCCOMAPI IncrementProgress(real32 inIncrement, IMCUnknown* inProgressKey = NULL)

©2001-2003 Eovia. All rights reserved.

IncrementProgress() changes the value of the progress bar.

Parameter

TMCString& *inPrompt*
A reference to the string that displays during the process.

IMCUnknown* *inProgressKey*
Identifies which progress is being set. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress. Progress values are between 0.0f and maxValue.

IShUtilities::SetProgressValue(real32, IMCUnknown*)

voidMCCOMAPI SetProgressValue(real32 inValue, IMCUnknown* inProgressKey = NULL)

SetProgressValue() sets the value for the progress bar.

Parameter

real32 *inValue*
The initial progress value. Progress values are between 0.0f and maxValue.

IMCUnknown* *inProgressKey*
Identifies which progress is being set. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress.

IShUtilities::SetProgressValue0to1(real32, IMCUnknown*)

voidMCCOMAPI SetProgressValue0to1(real32 inValue, IMCUnknown* inProgressKey = NULL)

SetProgressValue() sets the value for the progress bar to a range of 0 to 1.

Parameter

real32 *inValue*
The initial progress value.

IMCUnknown* *inProgressKey*
Identifies which progress is being set. BeginProgress returns a progressKey which is to be passed into the other methods---if you pass NULL as a key, it is assumed to be the main/first progress.

IShUtilities::YieldProcesses(uint32)

MCCOMErr MCCOMAPI YieldProcesses(uint32 ticks)

YieldProcesses() allows background processes to receive processing time. This method is often used by renderers to allow background rendering.

Parameter

uint32 *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. 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.

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 Overview <zot xref here> for more information on the Failure Handling system.

©2001-2003 Eovia. All rights reserved.

IShUtilities::TickCount()

uint32 MCCOMAPI TickCount()

TickCount() gets 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()

uint32 MCCOMAPI Random()

Random() returns a random integer.

IShUtilities::SetRandomSeed(uint32)

MCCOMErr MCCOMAPI SetRandomSeed(uint32 newSeed)

SetRandomSeed() changes the seed value of a random function.

Parameter **uint32** *newSeed*

The new seed value.

IShUtilities::RandomStep(int32)

int32 MCCOMAPI RandomStep(int32 before)

RandomStep() changes the step value in a random function < zot, BIG guess on my part...>. RandomStep uses the initial step value to calculate and return a new step value.

Parameter **int32** *before*

The initial step value.

IShUtilities::GetAPINumber()

uint32 MCCOMAPI GetAPINumber()

GetAPINumber() gets the version of the API. This is used to determine component version compatibility.

IShUtilities::Gestalt(void*, uint32)

MCCOMErr MCCOMAPI Gestalt(void* data,uint32 id)

Gestalt() gets information on the system <zot needs description>.

©2001-2003 Eovia. All rights reserved.

Parameter

void* *data*
A pointer to where to store the information.

uint32 *id*
The type of information.

ID	Type of data	Version	Meaning
Vers	long	1	The actual version of Gestalt.
FN83	Boolean	1	TRUE if the 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.

IShUtilities::DoAlert(EAlerts, TMCString&, TMCString&, TMCString&, TMCString&, int32, TMCString&, TMCString&, TMCString&)

int16 MCCOMAPI DoAlert(EAlerts alertSeverity, const TMCString& mainMessage, const TMCString& arg0, const TMCString& arg1, const TMCString& arg2, int32 buttonsCount, const TMCString& button0, const TMCString& button1, const TMCString& button2)

DoAlert() opens a standard alert. Use DoAlert to post easily an alert and get some response from the user. The returned value is the number of the hit button (0, 1 or 2). If you give empty strings for buttons, you get the following default buttons:

```
//  buttonsCount == 1   => OK
//  buttonsCount == 2   => OK Cancel
//  buttonsCount == 3   => Yes No Cancel
```

Parameter

IDType *alertSeverity*
Alert severity icon. See kShNotificationAlert, kShCautionAlert and kShStopAlert constants.

TMCString& *mainMessage*
A reference to the Main message to display.

TMCString& *arg0, arg1, arg2*
A reference to the Substitution strings : any "^0", "^1" and "^2" token in 'mainMessage' will be replaced by 'arg0', 'arg1', and 'arg2'.

int32 *buttonsCount*
A reference to the Number of buttons (1, 2 or 3).

TMCString& *button0*
A reference to the text of the default button.

TMCString& *button1*
A reference to the text of the second button.

©2001-2003 Eovia. All rights reserved.

TMCString& *button2*
A reference to the text of the third button.

IShUtilities::ErrorToString(MCCOMErr, TMCString&)

MCCOMErr MCCOMAPI ErrorToString(MCCOMErr inErr, TMCString& outErrorString)

ErrorToString() writes the error message to a string, so that it can be used as text for an alert.

Parameter

MCCOMErr *inErr*
The error to convert to a string.

TMCString& *outErrorString*
A reference to the string containing the error message text.

IShUtilities::BuildAlertMessage(MCCOMErr, TMCString&, TMCString&)

void MCCOMAPI BuildAlertMessage(MCCOMErr inErr, const TMCString& inContext, TMCString& outMessage)

BuildAlertMessage() builds the alert, using the error message string, as text for an alert.

Parameter

MCCOMErr *inErr*
The error that generates the need for an alert.

TMCString& *inContext*
A reference to the string containing the error message text.

TMCString& *outMessage*
A reference to the string containing the error text as it appears in the alert.

IShUtilities::GetResource(int32, int32, int32, void**)

MCCOMErr MCCOMAPI GetResource(int32 family, int32 classID, int32 resType, void data)**

GetResource() gets a resource with the specified family, class ID, resource type, and provides a pointer to its data.

Parameter

int32 *family*
The family id.

int32 *classID*
The class ID.

int32 *resType*
The resource type.

void** *data*
A pointer to data for that resource.

©2001-2003 Eovia. All rights reserved.

IShUtilities::SetupResources(int32, int32, void**)

MCCOMErr MCCOMAPI SetupResources(int32 family, int32 classID, void oldResources)**

SetupResource() forces the Shell to use a given resource file. Just ask for the family and class of the extension your currently consider, and **SetupResources** sets the correct resource file.

Parameter

int32 *family*
The family id.

int32 *classID*
The class ID.

void** *oldResources*
This is the previous resource file information. Store this value, and give it back when restoring the previous resource file via **IShUtilities::RestoreResources**.

IShUtilities::RestoreResources(void*)

MCCOMErr MCCOMAPI RestoreResources(void* oldResources)

RestoreResources() restores a previous resource file.

Parameter

void* *oldResources*.
This is the previous resource file information. Give this value back when restoring the previous resource file.

IShUtilities::BlitUIImage(IMCGraphicContext*, TMCRect&, TMCRect&, uint32)

MCCOMErr MCCOMAPI BlitUIImage(IMCGraphicContext* inGraphicContext, TMCRect& inRect, const TMCRect& inUpdateBBox, uint32 inResourceID)

BlitUIImage() forces the Shell to update a specific portion of the screen specified by the update bounding box. <zot needs description>

Parameter

IMCGraphicContext* *inGraphicContext*
A pointer to the context for the <zot needs description>.

TMCRect& *inRect*
A reference to the <zot needs description>.

TMCRect& *inUpdateBox*
A reference to the <zot needs description>.

uint32 *inResourceID*
The resource ID of the image being updated? <zot needs description>.

IShUtilities::GetUIImageInfo(uint32, int32&, int32&, int32)

MCCOMErr MCCOMAPI GetUIImageInfo(uint32 inResourceID, int32& width, int32& height, int32& planes)

GetUIImageInfo() allocates the UI image. It must be allocated before you can call GetUIImageBucket.

©2001-2003 Eovia. All rights reserved.

<zot needs description>

Parameter

uint32 *inResourceID*
The resource ID. <zot needs description>.

int32& *width*
A reference to the width of the image <zot needs description>.

int32& *height*
A reference to the height of the image <zot needs description>.

int32& *planes*
A reference to the planes of the image <zot needs description>.

IShUtilities::GetUIImageBucket(uint32, TMCPixelBucket*)

MCCOMErr MCCOMAPI GetUIImageBucket(uint32 inResourceID, TMCPixelBucket* bucket)

GetUIImageBucket() blits the UI uimage into a pixel bucket - it must be allocated before calling (that's what the previous call is for).

Parameter

uint32 *inResourceID*
The resource ID of the object. <zot needs description>.

TMCPixelBucket* *bucket*
A pointer to the destination pixel bucket.<zot needs description>.

IShUtilities::GetUIImage(uint32, TMCUIImage**)

MCCOMErr MCCOMAPI GetUIImage(uint32 inResourceID, TMCUIImage outUiImage)**

GetUIImage() directly gets the UI uimage.

Parameter

uint32 *inResourceID*
The resource ID of the object. <zot needs description>.

TMCUIImage** *outUiImage*
Apointer to the resulting UI Image. <zot needs description>.

IShUtilities::CreateCompatiblePixelBucket(int32, int32, int32, TMC-CompatiblePixelBucket**)

MCCOMErr MCCOMAPI CreateCompatiblePixelBucket(int32 widthParam, int32 heightParam, int32 depthParam, TMCCompatiblePixelBucket outBucket)**

CreateCompatiblePixelBucket() creates a TMCCompatiblePixelBucket with the specified dimensions. <zot needs description>

Parameter

int32 *widthParam*
The width value for the resulting pixel bucket. <zot needs description>.

int32& *heightParam*
The heigth value for the resulting pixel bucket. <zot needs description>.

©2001-2003 Eovia. All rights reserved.

int32& *depthParam*
The depth value for the resulting pixel bucket. <zot needs description>.

TMCCompatiblePixelBucket** *outBucket*
A pointer to the resulting pixel bucket. <zot needs description>.

IShUtilities::DestroyCompatiblePixelBucket(TMCCompatiblePixelBucket*)

MCCOMErr MCCOMAPI DestroyCompatiblePixelBucket(TMCCompatiblePixelBucket* outBucket)

DestroyCompatiblePixelBucket() destroys the specified a TMCCompatiblePixelBucket. <zot needs description>

Parameter **TMCCompatiblePixelBucket*** *outBucket*
A pointer to the specified pixel bucket. <zot needs description>.

IShUtilities::GetAsciiTranslationTable()

TMCAsciiTranslationTable* MCCOMAPI GetAsciiTranslationTable()

GetAsciiTranslationTable() returns a pointer to a TMCAsciiTranslationTable. <zot needs description>

IShUtilities::GetOpenDocumentsCount()

uint32 MCCOMAPI GetOpenDocumentsCount()

GetOpenDocumentsCount() returns the number of open documents. <zot needs description>

IShUtilities::GetOpenDocumentByIndex(IMFDocument**, index)

MCCOMErr MCCOMAPI GetOpenDocumentByIndex(IMFDocument document, uint32 index)**

GetOpenDocumentByIndex() gets the document specified by the index. <zot needs description>

Parameter **IMFDocument**** *document*
A pointer to the specified document. <zot needs description>.

uint32 *index*
The index to a specific open document. <zot needs description>.

IShUtilities::GetLastOpenDocPath(TMCString&)

void MCCOMAPI GetLastOpenDocPath(TMCString& path)

GetLastOpenDocPath() gets the last open document path stored in path. <zot needs description>

Parameter **TMCString&** *path*
A reference to the string containing the path information. <zot needs description>.

IShUtilities::GetLastSaveDocPath(TMCString&)

void MCCOMAPI GetLastSaveDocPath(TMCString& path)

©2001-2003 Eovia. All rights reserved.

GetLastSaveDocPath() gets the last save document path stored in path. <zot needs description>

Parameter **TMCString&** *path*
A reference to the string containing the path information. <zot needs description>.

IShUtilities::SetLastOpenDocPath(TMCString&)

void MCCOMAPI SetLastOpenDocPath(const TMCString& path)

SetLastOpenDocPath() sets the last open document path stored in path. <zot needs description>

Parameter **const TMCString&** *path*
A reference to the string containing the path information. <zot needs description>.

IShUtilities::SetLastSaveDocPath(TMCString&)

void MCCOMAPI SetLastSaveDocPath(TMCString& path)

SetLastSaveDocPath() sets the last save document path stored in path. <zot needs description>

Parameter **TMCString&** *path*
A reference to the string containing the path information. <zot needs description>.

IShUtilities::UseMMX()

boolean MCCOMAPIUseMMX()

UseMMX() returns True if the processor supports MMX and if we want to use it.

IShUtilities::IsPentiumProcessor()

boolean MCCOMAPIIsPentiumProcessor()

IsPentiumProcessor() returns True if the processor is a pentium family (P,Ppro,PII,PIII...).

IShUtilities::IsPentiumIIProcessor()

boolean MCCOMAPIIsPentiumIIProcessor()

IsPentiumIIProcessor() returns True if the processor is a PII.

IShUtilities::IsPentiumIIIProcessor()

boolean MCCOMAPIIsPentiumIIIProcessor()

IsPentiumIIIProcessor() returns True if the processor is a PIII.

IShUtilities::StringWidth(TMCTextStyle&, TMCString&)

int16 MCCOMAPI StringWidth(const TMCTextStyle& inStyle, const TMCString& inString)

StringWidth() returns the width needed for the specified string, using the specified text style.

©2001-2003 Eovia. All rights reserved.

Parameter **const TMCTextStyle&** *inStyle*
A reference to the style to use.

const TMCString& *inString*
A reference to the string.

IShUtilities::GetTextMetrics(TMCTextStyle&, TMCTextMetrics&)

void MCCOMAPI GetTextMetrics(const TMCTextStyle& inStyle, TMCTextMetrics& outMetrics)

GetTextMetrics() updates the text metrics for the given text style. <zot needs more information. Please provide more details about the metrics or an example.>

Parameter **const TMCTextStyle&** *inStyle*
A reference to the style to use to calculate the text metrics.

TMCTextMetrics& *outMetrics*
A reference to the metrics output by the function.

IShUtilities::GetCreator()

IDType MCCOMAPI GetCreator()

GetCreator() returns the ID type.

IShUtilities::SetCursor(int32)

void MCCOMAPI SetCursor(int32 cursorId)

SetCursor() changes the cursor to the specified cursor.

Parameter **int32** *cursorID*
The id of the specified cursor.

IShUtilities::GetFontList(IMCFontList**)

void MCCOMAPI GetFontList(IMCFontList outFontList)**

GetFontList() updates the font list.

Parameter **IMCFontList**** *outFontList*
A pointer to the resulting FontList.

IShUtilities::CheckForUserCancel()

boolean MCCOMAPI CheckForUserCancel()

CheckForUserCancel() returns True if the user has canceled the operation.

IShUtilities::GetCurrentModifiers()

const TMCModifiers& MCCOMAPI GetCurrentModifiers()

GetCurrentModifiers() returns a reference to the current modifier.

©2001-2003 Eovia. All rights reserved.

IShUtilities::CheckKeyDown(int16)

boolean MCCOMAPI CheckKeyDown(int16 inVirtualKey)

CheckKeyDown() returns True if the key specified by inVirtualKey is down.

Parameter **int16** *inVirtualKey*

The value of the key being checked.

©2001-2003 Eovia. All rights reserved.

Chapter 9 - 3D ISh* Reference

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.1 I3DShAtmosphericShader Interface

Interface ID: IID_I3DShAtmosphericShader

Interface file: I3DShEnvironment.h

Description

The **I3DShAtmosphericShader** interface lets you get some information about the atmospheric shader of the scene.

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

I3DShAtmosphericShader::GetScene(I3DShScene**)

MCCOMAPI GetScene(I3DShScene scene)**

Call **GetScene()** to return the scene of the object.

Parameters

I3DShScene** *scene*
Points to the scene of the object.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.2 I3DShBackdrop Interface

Interface ID: IID_I3DShBackdrop

Interface file: I3DShEnvironment.h

Description

The **I3DShBackdrop** interface lets you get some information on the backdrop of the scene. <zot>Beef up description </zot>

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

I3DShBackdrop::GetScene(I3DShScene**)

MCCOMAPI GetScene(I3DShScene scene)**

Call **GetScene()** to return the scene of the object.

Parameters

I3DShScene** *scene*
Points to the scene of the object.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.3 I3DShBackground Interface

Interface ID: IID_I3DShBackground

Interface file: I3DShEnvironment.h

Description

The **I3DShBackground** interface lets you get some information on the background of the scene. <zot>Beef up description </zot>

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

I3DShBackground::GetScene(I3DShScene**)

MCCOMAPI GetScene(I3DShScene scene)**

Call **GetScene()** to return the scene of the object.

Parameters



I3DShScene** *scene*
Points to the scene of the object.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.4 I3DShCamera Interface

Interface ID: IID_I3DShCamera

Interface file: I3DShCamera.h

Description

The **I3DShCamera** interface is the shell interface for cameras.

I3DShCamera defines a camera as a tree element. It can return a I3DShTreeElement interface on demand.

Methods Index

"I3DShCamera::GetGlobalToCameraTransform(TTransford3d*)"
"I3DShCamera::Preset(uint32, const TVector3*, real)"
"I3DShCamera::Project3DTo2D(const TVector3*, TVector2*, real*)"
"I3DShCamera::CreateRay(const TVector2*, TVector3*, TVector3*)"
"I3DShCamera::GetRenderUVBox(TBBox2d*, uint32, uint32, real)"
"I3DShCamera::SetCameraComponent(int32)"
"I3DShCamera::SetCameraComponent(IShComponent*, ECloneMode)"
"I3DShCamera::GetCameraComponent(IShComponent**)"
"I3DShCamera::PreProject3DTo2D(const TVector3&, TVector4&)"
"I3DShCamera::IsProjectionAffine(void)"
"I3DShCamera::GetStandardCameraInfo(TStandardCameraInfo&)"
"I3DShCamera::SetCanonicalCameraType(ECanonicalCameraType)"
"I3DShCamera::GetCanonicalCameraType()"
"I3DShCamera::SetProductionOffset(const TVector2&)"
"I3DShCamera::GetProductionOffset()"
"I3DShCamera::SetProductionZoom(real)"
"I3DShCamera::GetProductionZoom()"

I3DShCamera::GetGlobalToCameraTransform(TTransford3d*)

MCCOMErr GetGlobalToCameraTransform(TTransform3D* transform)

Gives the transformation from the Global Coordinate System to the Camera Coordinate System.

Parameters **TTRANSFORM3D*** *transform*

transform is a reference to the transformation data. Copy its contents to an internal variable.

Return Value MC_S_OK

I3DShCamera::Preset(uint32, const TVector3*, real)

MCCOMErr Preset(uint32 preset, const TVector3* aimedPoint, real distance)

Sets up preset values for the camera.

Parameters **uint32** *preset*

Specifies which preset is being set in ECameraPreset: kReferencePreset, kTopPreset, kBottomPreset, kLeftPreset, bkRightPreset, or kFrontPreset.

©2001-2003 Eovia. All rights reserved.

TVector3* *aimedPoint*
The 3D location of the point aimed by the camera.

real *distance*
distance between the Camera and the *aimedPoint*.

Return Value MC_S_OK

I3DShCamera::Project3DTo2D(const TVector3*, TVector2*, real*)

boolean Project3DTo2D(const TVector3* position, TVector2* resultScreenPosition,real* resultDistanceToScreen)

Project3DTo2D() is called to project a 3D point on the screen.

Parameters

const TVector3* *position*
The 3D position in camera space.

TVector2* *resultscreenPosition*
Projection on the screen.

real* *resultDistanceToScreen*
The Z value.

Return Value

TRUE	The point is visible.
FALSE	The point is not visible.

I3DShCamera::CreateRay(const TVector2*, TVector3*, TVector3*)

boolean CreateRay(const TVector2* screenPosition, TVector3* resultOrigin, TVector3* resultDirection)

CreateRay() is called when a ray (i.e. a straight line) from a screen position needs to be created.

Parameters

const TVector2* *screenPosition*
Position on the screen; a relative parameter between -1 and 1.

TVector3* *resultOrigin*

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

TVector3* *resultDirection*

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

Return Value

TRUE	A ray could be calculated.
FALSE	Calculation was not possible.

You must always return true to be able to ray-trace an entire image.

©2001-2003 Eovia. All rights reserved.

I3DShCamera::GetRenderUVBox(TBBox2d*, uint32, uint32, real)

MCCOMErr GetRenderUVBox(TBox2D* uvBox, uint32 width, uint32 height, real pixelAspectRatio)

Gives the uvBox of the productionFrame given the width and height, and pixel aspect ratio of the screen.

Parameters

TBBox2D* *uvBox*
uvBox of the productionFrame.

Uint32 *width*
The screen width.

Uint32 *height*
The screen height.

real *pixelAspectRatio*
The screen pixel aspect ratio.

Return Value MC_S_OK

I3DShCamera::SetCameraComponent(int32)

MCCOMErr SetCameraComponent(int32 kindID)

<zot>Description here</zot>

Parameters

int32 kindID
<zot>Description here</zot>

Return Value MC_S_OK

I3DShCamera::SetCameraComponent(IShComponent*, ECloneMode)

MCCOMErr SetCameraComponent(IShComponent * cameraComponent, ECloneMode cloneMode)

<zot>Description here</zot>

Parameters

IShComponent* *cameraComponent*
<zot>Description here</zot>

ECloneMode *cloneMode*
<zot>Description here</zot>

Return Value MC_S_OK

I3DShCamera::GetCameraComponent(IShComponent**)

MCCOMErr GetCameraComponent(IShComponent ** cameraComponent)

<zot>Description here</zot>

©2001-2003 Eovia. All rights reserved.

Parameters **IShComponent**** *cameraComponent*
<zot>Description here</zot>

Return Value MC_S_OK

I3DShCamera::PreProject3DTo2D(const TVector3&, TVector4&)

boolean PreProject3DTo2D(const TVector3& cameraPosition, TVector4& projectedPosition)

PreProject3DTo2D() is called to project a 3D point on the screen.

Parameters **const TVector3&** *cameraPosition*
The 3D position of the camera.

TVector4& *projectedPosition*
Projection on the screen.

Return Value

TRUE	The point is visible.
FALSE	The point is not visible. It may be behind the camera.

I3DShCamera::IsProjectionAffine(void)

boolean IsProjectionAffine(void)

IsProjectionAffine() returns true if the w component from the projection is always 1.0.

Return Value

TRUE	w is always 1.0
FALSE	w is not always 1.0

I3DShCamera::GetStandardCameraInfo(TStandardCameraInfo&)

void GetStandardCameraInfo(TStandardCameraInfo& cameraInfo)

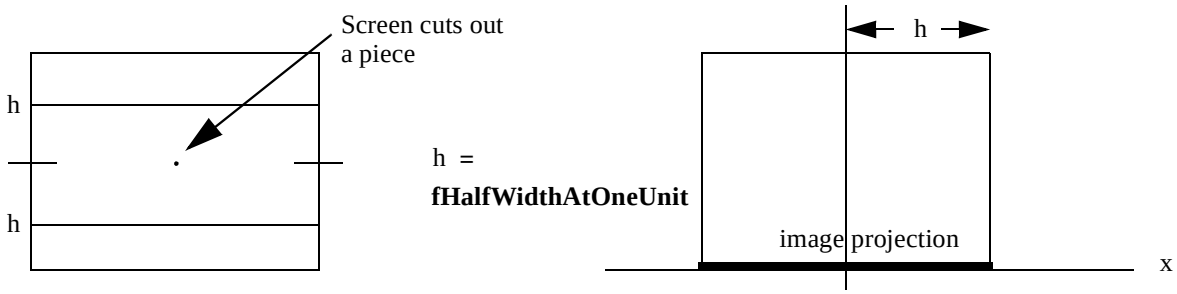
GetStandardCameraInfo() is used to return standard camera info used by the interactive renderer.

TStandardCameraInfo has the following fields:

- **EProjectionType fProjectionType -** pick the type of camera that you are most closely approximating between conical (perspective) and isometric (orthographic)
- **ECanonicalCameraType fCanonicalCameraType** - internal use only (ignore)
- **real fHalfWidthAtOneUnit** - the amount of camera space units at one unit from the camera location. In the case of an orthographic camera, this represent the zoom factor. For the perspective camera, this represents the field of view (FOV), or angle of the most extreme rays that are in the image. Any rays outside of the FOV are not represented in the image. (See diagrams below for graphical description.)
  boolean fIsMatching - indicates whether the standard camera info is an exact or approximate conversion. For example, a spherical lens may only approximate itself with one of the types.

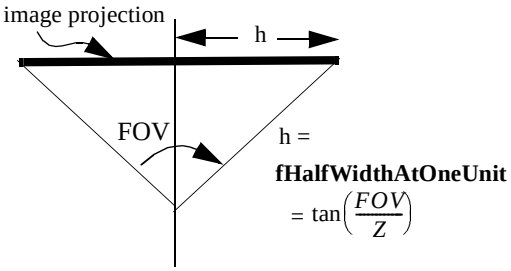
As you can see from the diagram below, for an orthographic camera, a large **fHalfWidthAtOneUnit** means a zoom out.

©2001-2003 Eovia. All rights reserved.



Orthographic Camera

As shown below, with a perspective camera, a large **fHalfWidthAtOneUnit** means the FOV is large, like a wide angle lens, and the image is zoomed out.



h	θ	Zθ=FOV
1	45	90
0.5	26	52
2	63	126

Perspective Camera

$$FOV = Z\tan^{-1}(h)$$

Parameters **TStandardCameraInfo&** *cameraInfo*
The camera info.

I3DShCamera::SetCanonicalCameraType(ECanonicalCameraType)

void SetCanonicalCameraType(ECanonicalCameraType canonicalCameraType)

Chooses between canonical (standard) cameras (default cameras).

Parameters **ECanonicalCameraType** *canonicalCameraType*
kCanonicalCameraType_NotCanonical
kCanonicalCameraType_Left
kCanonicalCameraType_Right
kCanonicalCameraType_Top
kCanonicalCameraType_Bottom
kCanonicalCameraType_Front
kCanonicalCameraType_Back

I3DShCamera::GetCanonicalCameraType()

ECanonicalCameraType GetCanonicalCameraType()

Returns the canonical camera type or kCanonicalCameraType_NotCanonical if this camera is not canonical.

©2001-2003 Eovia. All rights reserved.

Return Value The Canonical Camera Type.

I3DShCamera::SetProductionOffset(const TVector2&)

void SetProductionOffset(const TVector2& inOffset)

Sets the camera screen offset

Parameters **TVECTOR2&** *inOffset*
The offset value.

I3DShCamera::GetProductionOffset()

TVector2 GetProductionOffset()

Gets the camera screen offset.

Return Value The production Offset.

I3DShCamera::SetProductionZoom(real)

void SetProductionZoom(real zoom)

Sets the camera screen zoom value.

Parameters **real** *zoom*
The zoom value.

I3DShCamera::GetProductionZoom()

real GetProductionZoom()

Gets the camera screen zoom value.

Return Value The Zoom value.

©2001-2003 Eovia. All rights reserved.

9.5 I3DShConstraint

Interface ID: IID_I3DShConstraint

Interface File: I3DShConstraint.h

Description

I3DShConstraint provides constraints between Tree Elements. A constraint, known as a link in RDS, defines motion links between Tree Elements. It allows you to define “mechanical” constrains (degrees of freedom) between one Tree Element and its father.

zot there are no methods to document in I3DShConstraint.h as of 11/22/99.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.6 I3DShEnvironment Interface

Interface ID: IID_I3DShEnvironment

Interface file: I3DShEnvironment.h

Description

The **I3DShEnvironment** interface lets you get some information on the environment of the scene.
<zot>Beef up description </zot>

Methods Index

"I3DShEnvironment::SegmentFilter(const TVector3& ,const TVector3&, TMCColorRGB&, boolean)"
"I3DShEnvironment::SegmentFilter(const TVector3&,const TVector3&,TMCColorRGB&)"
"I3DShEnvironment::DirectionFilter(const TVector3 ,const TVector3&, TMCColorRGB&)"
"I3DShEnvironment::GetBackdropColor(const TVector2&,boolean&,const TBBox2D&, TMCColor-RGB&)"
"I3DShEnvironment::GetEnvironmentColor(const TVector3&, boolean&, TMCColorRGB&)"
"I3DShEnvironment::HasAtmosphere()"
"I3DShEnvironment::HasBackdrop()"
"I3DShEnvironment::HasBackground()"

I3DShEnvironment::SegmentFilter(const TVector3& ,const TVector3&, TMCColorRGB&, boolean)

MCCOMErr MCCOMAPI SegmentFilter(const TVector3 &beg,const TVector3 &end,TMCColorRGB &inOutFilter, boolean isShadowCasting)

Call **SegmentFilter()** to get the color of 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 hits and you want to get the color seen because the ray may go thru a fog or a volume primitive.

zot need to say **more** about the differences between the two SegmentFilter functions.

Parameters

const TVector3& *beg*
The point where you want to get the color.

const TVector3& *end*
The point where you know the color.

TMCColorRGB& *inOutFilter*
Gives the color of the ray at point *end*, you get back the color at point *beg* of the ray.

boolean *isShadowCasting*
Returns True if the object casts shadows.

©2001-2003 Eovia. All rights reserved.

I3DShEnvironment::SegmentFilter(const TVector3&,const TVector3&,TMCColorRGB&)

MCCOMErr MCCOMAPI SegmentFilter(const TVector3 &beg,const TVector3 &end,TMCColorRGB &inOutFilter)

Call **SegmentFilter()** to get the color of 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 hits and you want to get the color seen because the ray may go thru a fog or a volume primitive.

Parameters

const TVector3& *beg*
The point where you want to get the color.

const TVector3& *end*
The point where you know the color.

TMCColorRGB& *inOutFilter*
Gives the color of the ray at point *end*, you get back the color at point *beg* of the ray.

I3DShEnvironment::DirectionFilter(const TVector3 ,const TVector3&, TMCColorRGB&)

MCCOMErr MCCOMAPI DirectionFilter(const TVector3 &origin,const TVector3 &direction, TMCColorRGB &inOutFilter)

Call **I3DShEnvironment::DirectionFilter** to get the color of a ray that reaches no object.

Parameters

const TVector3& *origin*
The point where the ray starts.

const TVector3& *direction*
The direction of the ray.

TMCColorRGB& *inOutFilter*
Gives the color of the background, and you get back the resulting color.

I3DShEnvironment::GetBackdropColor(const TVector2&,boolean&,const TBBox2D&, TMCColorRGB&)

MCCOMErr MCCOMAPI GetBackdropColor(const TVector2 &screenPoint,boolean &fullAreaDone,const TBBox2D &screenMinMax, TMCColorRGB &resultColor)

GetBackdropColor() is called to get the backdrop color at a given point on the screen.

Parameters

const TVector2& *screenPoint*
The screen point in **zot Local coordinates? Global coordinates?**

boolean& *fullAreaDone*
Return True if the full area is covered.

TBBOX2D& *screenMinMax*
The boundaries (minimum and maximum values) of the *fullAreaDone* range of values.

©2001-2003 Eovia. All rights reserved.

TMCColorRGB& *color*
Return the calculated color here.

I3DShEnvironment::GetEnvironmentColor(const TVector3&, boolean&, TMCColorRGB&)

MCCOMErr MCCOMAPI GetEnvironmentColor(const TVector3 &direction,boolean &fullAreaDone,TMCColorRGB &resultColor)

Call **I3DShEnvironment::GetEnvironmentColor** to get the color of the background.

Parameters

const TVector3& *direction*
The direction of the ray.

boolean& *fullAreaDone*
Return True if the full area is covered.

TMCColorRGB& *color*
Return the calculated color here.

I3DShEnvironment::HasAtmosphere()

boolean MCCOMAPI HasAtmosphere()

Call **HasAtmosphere()** to find out if the current Environment has an atmospheric effect or not. Returns True if it has an atmospheric effect.

I3DShEnvironment::HasBackdrop()

boolean MCCOMAPI HasBackdrop()

Call **HasBackdrop()** to find out if the current Environment has a backdrop effect or not. Returns True if it has a backdrop effect.

I3DShEnvironment::HasBackground()

boolean MCCOMAPI HasBackground()

Call **HasBackground()** to find out if the current Environment has a background effect or not. Returns True if it has a background effect.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.7 I3DShExternalPrimitive Interface

Interface ID: IID_I3DShExternalPrimitive

Interface file: I3DShObject.h

Description

I3DShExternalPrimitive is a host for external primitives. It points to an **"I3DShPrimitiveComponent Interface"** that points to the External primitive itself. To get to the external primitive itself call **GetPrimitiveComponent()**. To get the component part, query the **I3DShPrimitiveComponent** interface, and call **I3DShPrimitiveComponent::GetExternalPrimitive**.

Method Index

"I3DShExternalPrimitive::SetPrimitiveComponent(IShComponent*, ECloneMode)"
"I3DShExternalPrimitive::GetPrimitiveComponent(IShComponent**)"

I3DShExternalPrimitive::SetPrimitiveComponent(IShComponent*, ECloneMode)

MCCOMErr SetPrimitiveComponent(IShComponent * primitiveComponent, ECloneMode cloneMode)

<zot>Need method description</zot>

Parameter

I3DShComponent* *primitiveComponent*
primitiveComponent **zot needs description.**

ECloneMode *cloneMode*
cloneMode specifies whether or not animation information is included? **zot needs description.**

I3DShExternalPrimitive::GetPrimitiveComponent(IShComponent**)

MCCOMErr GetPrimitiveComponent(IShComponent ** primitiveComponent)

<zot>Need method description</zot> **GetPrimitiveComponent()** gets the component part of an external primitive.

Parameter

I3DShComponent* *primitiveComponent*
primitiveComponent **zot needs description.**

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.8 I3DShObject Interface

Interface ID: IID_I3DShObject

Interface file: I3DShObject.h

Description

I3DShObject contains the geometry and shading information of objects. There are two types of Objects: Primitives and Scenes. They are kept in a list within a scene called the objects list. An object can be instanciated as many times as needed in a scene.

Method Index

"I3DShObject::SetName(const TMCString&)"
"I3DShObject::GetName(TMCString&)"
"I3DShObject::GetInstancesCount()"
"I3DShObject::GetInstanceArray(TMCCountedPtrArray<I3DShInstance>&)"
"I3DShObject::GetUVSpaceCount()"
"I3DShObject::GetUVSpace(uint32, UVSpaceInfo*)"
"I3DShObject::GetScene(I3DShScene**)"
"I3DShObject::IsInfiniteObject()"
"I3DShObject::GetBoundingBox(TBBox3D&)"
"I3DShObject::GetFacetBoundingBox(TBBox3D&)"
"I3DShObject::Clone(I3DShObject**, ECloneMode) const"
"I3DShObject::GetEditor()"
"I3DShObject::GetObjectID()"

I3DShObject::SetName(const TMCString&)

MCCOMErr SetName(const TMCString& name)

SetName() sets the name of the object. This is the same call as **I3DShMaster::SetName**<zot>Need Cross-ref</zot>.

Parameters **TMCString&** *name*
The name of the object.

Return Value MC_S_OK

I3DShObject::GetName(TMCString&)

MCCOMErr GetName(TMCString& name)

GetName() gets the name of the object. This is the same call as **I3DShMaster::GetName**<zot>Need Cross-Ref</zot>.

Parameters **TMCString&** *name*
The name of the object.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

I3DShObject::GetInstancesCount()

uint32 GetInstancesCount()

GetInstancesCount() returns the number of instances of the object. This is the same call as **I3DShMaster::GetInstancesCount**<zot>Need Cross-Ref</zot>.

Return Value The number of instances of the object.

I3DShObject::GetInstanceArray(TMCCountedPtrArray<I3DShInstance>&)

void GetInstanceArray(TMCCountedPtrArray<I3DShInstance>& instances)

GetInstanceArray() builds an array of all the instances of an object. This is the same call as **I3DShMaster::GetInstanceArray**<zot>Need Cross-Ref</zot>.

Parameters **TMCCountedPtrArray<I3DShInstance>&** *instances*
An array of all instances of the object.

I3DShObject::GetUVSpaceCount()

uint32 GetUVSpaceCount()

GetUVSpaceCount() returns the number of UV spaces for an object. If 0 is returned only projection mapping can be performed on the object.

Return Value The number of UV Spaces for an object.

I3DShObject::GetUVSpace(uint32, UVSpaceInfo*)

virtual MCCOMErrMCCOMAPI GetUVSpace(uint32 uvSpaceID, UVSpaceInfo* uvSpaceInfo)

Get information on a specific UV Space

Parameters **uint32** *uvSpaceId*
UV Space ID, starting at 0, up to GetUVSpaceCount()-1.

UVSpaceInfo *uvSpaceInfo*
Returned data of this UVSpace (you provide a pointer on the storage struct).

I3DShObject::GetScene(I3DShScene**)

virtual voidMCCOMAPI GetScene(I3DShScene scene)**

Returns the scene of the object.

Parameters **I3DShScene** *scene*
Returns the scene.

©2001-2003 Eovia. All rights reserved.

I3DShObject::IsInfiniteObject()

virtual booleanMCCOMAPI IsInfiniteObject()

IsInfiniteObject() returns TRUE if the object is infinite (like the infinite plane).

I3DShObject::GetBoundingBox(TBBox3D&)

virtual voidMCCOMAPI GetBoundingBox(TBBox3D& bbox)

Gets the bounding box of the object (in local coordinates).

Parameters

TBBox3D *bbox*
Returns the coordinates for the bounding box.

I3DShObject::GetFacetBoundingBox(TBBox3D&)

virtual void MCCOMAPI GetFacetBoundingBox(TBBox3D &bbox, TMatrix33 &mappingTransform, void *listener, uint32 matrixChangedPhase)

|The bounding box calculated here exactly fits the facets of the object in a given direction. The FacetBounding boxes are cached inside the Object to reduce calculation time. You don't have to compare mappingTransform (TMatrix33) in the look-up table. A last hit is provide inside the GetFacetBoundingBox method to make the Get on same listener very fast. Each time the BoundingBox of the object changed , the list is flushed.

Parameters

TBBox3D& *bbox*
Returns the coordinates for the bounding box.

TMatrix33& *mappingTransform*
zot needs description

void* *listener*
Accelerates the Get.

uint32 *matrixChangedPhase*
zot needs description.

I3DShObject::Clone(I3DShObject**, ECloneMode) const

virtual MCCOMErrMCCOMAPI Clone(I3DShObject clone, ECloneMode cloneMode) const**

<zot>Clones an object.

Parameters

I3DShObject** *clone*
Returns the cloned object.

ECloneMode *cloneMode*
Specifies the clone mode (with or without the animation data).

I3DShObject::GetEditor()

virtual int32MCCOMAPI GetEditor()

©2001-2003 Eovia. All rights reserved.

GetEditor() returns the ID of the Editor (i.e. the module) capable of editing this object in the Modeling room.

I3DShObject::GetObjectID()

virtual int32MCCOMAPI GetObjectID()

GetObjectID() returns the id of an object. In the case of a primitive, it returns the class id. For a Scene, it returns 'sc '

©2001-2003 Eovia. All rights reserved.

9.9 I3DShPatchArray

Interface ID: IID_I3DShPatchArray

Interface File: I3DShObject.h

Description

I3DShPatchArray is an array of cubic 4 x 4 Bezier patches. 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 !

Method Index

"I3DShPatchArray::Init(uint32, uint32)"
"I3DShPatchArray::SetPatch(uint32, uint32, TPatch3D*)"
"I3DShPatchArray::GetPatch(uint32, uint32, TPatch3D*)"
"I3DShPatchArray::MeshSetU(uint32, real)"
"I3DShPatchArray::MeshSetV(uint32, real)"
"I3DShPatchArray::MeshGetU(uint32, real*)"
"I3DShPatchArray::MeshGetV(uint32, real*)"
"I3DShPatchArray::MeshCalcUV()"
"I3DShPatchArray::GetSize(uint32&, uint32&)"
"I3DShPatchArray::BuildFacetMesh(real, FacetMesh**)"

I3DShPatchArray::Init(uint32, uint32)

MCCOMErr Init(uint32 uCount, uint32 vCount)

Initializes a patch array.

Parameters

uint32 *uCount*
Number of patches in the U direction.

uint32 *vCount*
Number of patches in the V direction.

Return Value MC_S_OK

I3DShPatchArray::SetPatch(uint32, uint32, TPatch3D*)

MCCOMErr SetPatch(uint32 uIndex, uint32 vIndex, TPatch3D* patch)

©2001-2003 Eovia. All rights reserved.

Use **I3DShPatchArray::SetPatch** to set Patch values in the Patch Array.

Parameters

uint32 *uIndex*
Patch index in the U direction, between 0 and uCount-1.

uint32 *vIndex*
Patch index in the V direction, between 0 and vCount-1.

TPatch3D* *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 Value MC_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.

I3DShPatchArray::GetPatch(uint32, uint32, TPatch3D*)

MCCOMErr GetPatch(uint32 uIndex, uint32 vIndex, TPatch3D* patch)

Parameters

uint32 *uIndex*
Patch index in the U direction, between 0 and uCount-1

uint32 *vIndex*
Patch index in the V direction, between 0 and vCount-1

TPatch3D* *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 MC_S_OK

I3DShPatchArray::MeshSetU(uint32, real)

MCCOMErr MeshSetU(uint32 uIndex, real uValue)

Use I3DShPatchArray::MeshSetU to set the U coordinate of all points at uIndex.

Parameters

uint32 *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.

real *uValue*
The U value for all points at uIndex. This value is between 0 and 1.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

I3DShPatchArray::MeshSetV(uint32, real)

MCCOMErr MeshSetV(uint32 vIndex, real vValue)

Use **I3DShPatchArray::MeshSetV** to set the V coordinate of all points at vIndex.

Parameters

uint32 *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.

real *vValue*
The V value for all points at vIndex. This value is between 0 and 1.

Return Value MC_S_OK

I3DShPatchArray::MeshGetU(uint32, real*)

MCCOMErr MeshGetU(uint32 uIndex, real* uValue)

Use **I3DShPatchArray::MeshGetU** to get the U coordinate of all points at uIndex.

Parameters

uint32 *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.

real* *uValue*
Where to copy the U value for all points at uIndex. This value is between 0 and 1.

Return Value MC_S_OK

I3DShPatchArray::MeshGetV(uint32, real*)

MCCOMErr MeshGetV(uint32 vIndex, real* vValue)

Use **I3DShPatchArray::MeshGetV** to get the V coordinate of all points at vIndex.

Parameters

uint32 *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.

real* *vValue*
Where to copy the V value for all points at vIndex. This value is between 0 and 1.

Return Value MC_S_OK

I3DShPatchArray::MeshCalcUV()

MCCOMErr MeshCalcUV()

Call **I3DShPatchArray::CalcUV** if you want to have all the UV values calculated for you.You must call this procedure if you did not set any UV values using

©2001-2003 Eovia. All rights reserved.

I3DShPatchArray::MeshSetU and **I3DShPatchArray::MeshSetV**.

Return Value MC_S_OK

I3DShPatchArray::GetSize(uint32&, uint32&)

MCCOMErr GetSize(uint32& uCount, uint32& vCount)

Gets the size of the patch array in the U and V directions.

Parameters

uint32 *uCount*
Where to copy the number of patches in the U direction.

uint32 *vCount*
Where to copy the number of patches in the Vdirection.

Return Value MC_S_OK

I3DShPatchArray::BuildFacetMesh(real, FacetMesh**)

virtual MCCOMErr MCCOMAPI BuildFacetMesh(real lod, FacetMesh outMesh)**

This is a low-level procedure that always returns the facet mesh at the requested LOD (different than when using the **I3DShPrimitive::GetFacetMesh()** system of sampling LODs values.)

Parameters

real *lod*
The level of detail.

FacetMesh** *outMesh*
The created mesh that represents the same object than the patch array but with facets.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

9.10 I3DShPatchList

Interface ID: IID_I3DShPatchList

Interface File: I3DShObject.h

Description

A Patch List is simply a bag of bicubic patches in unsorted order. Of course, it will not return patches, only facets: use **I3DShPrimitive::EnumPatches**.

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.

Method Index

"I3DShPatchList::Init(boolean)"
"I3DShPatchList::AddPatch(TPatch3D*)"
"I3DShPatchList::GetIndPatches(uint32)"
"I3DShPatchList::CountPatches()"
"I3DShPatchList::SetHasUVSpace(boolean)"
"I3DShPatchList::SetUVSpace(UVSpaceInfo*)"
"I3DShPatchList::GetUVSpace(uint32, UVSpaceInfo*)"
"I3DShPatchList::GetUVSpaceCount()"
"I3DShPatchList::BuildFacetMesh(real, FacetMesh**)"

I3DShPatchList::Init(boolean)

MCCOMErr Init(boolean hasUV)

I3DShPatchList::Init must be right after creating a Patch List object.

Parameters **boolean** *hasUV*
TRUE if this object supports a UV Space

Return Value MC_S_OK

I3DShPatchList::AddPatch(TPatch3D*)

MCCOMErr AddPatch(TPatch3D* patch)

Use **I3DShPatchList::AddPatch** to add patches to a Patch List object one after the other.

©2001-2003 Eovia. All rights reserved.

Parameters **TPatch3D*** *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 **MC_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::AddPatch fails if running out of RAM.

I3DShPatchList::GetIndPatches(uint32)

TPatch3D*MCCOMAPI GetIndPatches(uint32 index)

Provides acces to the patches by index.

Parameters **uint32** *index*
The index of the patch you want to acces, *index* is 0-based.

Return Value **The indexth patch of the list.**

I3DShPatchList::CountPatches()

uint32 CountPatches()

Counts the number of patches in the patch list.

Return Value The number of patches in the patchlist as an uint32.

I3DShPatchList::SetHasUVSpace(boolean)

MCCOMErr SetHasUVSpace(boolean hasUV)

If false, will remove all previous UV Space definitions.

Parameters **boolean** *hasUV*
the flag for HasUVSpace.

Return Value **MC_S_OK**

I3DShPatchList::SetUVSpace(UVSpaceInfo*)

MCCOMErr SetUVSpace(UVSpaceInfo* uvSpaceInfo)

Set a UV Space. The UV space will be added if it was not previously defined. See the Data Structure reference section for more information on the *UVSpaceInfo* structure.

Parameters **UVSpaceInfo*** *uvSpaceInfo*
Pointer to the UVSpaceInfo structure for defining the UV space.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

I3DShPatchList::GetUVSpace(uint32, UVSpaceInfo*)

MCCOMErr GetUVSpace(uint32 uvSpaceID, UVSpaceInfo* uvSpaceInfo)

Returns the *UVSpaceInfo* structure for the corresponding *uvSpaceID*. See the Data Structure reference section for more information on the *UVSpaceInfo* structure.

Parameters

uint32 *uvSpaceID*
The ID of the UV Space to retrieve.

UVSpaceInfo* *uvSpaceInfo*
The *UVSpaceInfo* structure.

Return Value MC_S_OK

I3DShPatchList::GetUVSpaceCount()

uint32MCCOMAPI GetUVSpaceCount()

Gets the count of UV spaces.

Return Value The count of UV spaces as an uint32.

I3DShPatchList::BuildFacetMesh(real, FacetMesh**)

MCCOMErrMCCOMAPI BuildFacetMesh(real lod, FacetMesh outMesh)**

This is a low-level procedure that always returns the facet mesh at the requested LOD (different than when using the **I3DShPrimitive::GetFacetMesh()** system of sampling LODs values.)

Parameters

real *lod*
The level of detail.

FacetMesh** *outMesh*
The created mesh that represents the same object than the patch array but with facets.

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.11 I3DShPolygonArray

Interface ID: IID_I3DShPolygonArray

Interface File: I3DShObject.h

Description

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 !

Method Index

"I3DShPolygonArray::Init(uint32, uint32, boolean, boolean)"
"I3DShPolygonArray::MeshSetVertex(uint32, uint32, TVector3*, TVector3*)"
"I3DShPolygonArray::MeshGetVertex(uint32, uint32, TVector3*, TVector3*)"
"I3DShPolygonArray::MeshSetU(uint32, real)"
"I3DShPolygonArray::MeshSetV(uint32, real)"
"I3DShPolygonArray::MeshGetU(uint32, real*)"
"I3DShPolygonArray::MeshGetV(uint32, real*)"
"I3DShPolygonArray::MeshCalcUV()"
"I3DShPolygonArray::MeshCalcNormals(boolean)"
"I3DShPolygonArray::GetSize(uint32&, uint32&, boolean&, boolean&)"

I3DShPolygonArray::Init(uint32, uint32, boolean, boolean)

MCCOMErr Init(uint32 uCount, uint32 vCount, boolean uClosed, boolean vClosed)

Call **I3DShPolygonArray::Init** just after creating a **I3DShPolygonArray** object to initialize all internal data structures.

Parameters

uint32 *uCount*

Number of vertices in the U direction.

uint32 *vCount*

Number of vertices in the V direction.

©2001-2003 Eovia. All rights reserved.

boolean *uClosed*
TRUE if the Array is closed in the U direction.

boolean *vClosed*
TRUE if the Array is closed in the V direction.

Return Value MC_S_OK

Comments **I3DShPolygonArray::Init** allocates the Polygon Array in RAM so you can access its points through I3DShPolygonArray::SetVertex and **I3DShPolygonArray::GetVertex**.

I3DShPolygonArray::MeshSetVertex(uint32, uint32, TVector3*, TVector3*)

MCCOMErr MeshSetVertex(uint32 uIndex, uint32 vIndex, TVector3* vertex, TVector3* normal)

Use **I3DShPolygonArray::SetVertex** to set the Vertex and the Normal at a specific point of the Array. Normal can be NULL.

Parameters

uint32 *uIndex*
Index in the U direction. Index in between 0 and uCount-1.

uint32 *vIndex*
Index in the V direction. Index in between 0 and vCount-1

TVector3* *vertex*
Points to the Vertex coordinates. The values will be copied in the internal Polygon Array data.

TVector3* *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::CalcNormals**.

Return Value MC_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::CalcNormals(***FALSE***)**.

I3DShPolygonArray::MeshGetVertex(uint32, uint32, TVector3*, TVector3*)

MCCOMErr MeshGetVertex(uint32 uIndex, uint32 vIndex, TVector3* vertex, TVector3* normal)

Use **I3DShPolygonArray::GetVertex** to get the Vertex and the Normal at a specific point of the Array.

©2001-2003 Eovia. All rights reserved.

Parameters

uint32 *uIndex*
Index in the U direction. Index in between 0 and uCount-1.

uint32 *vIndex*
Index in the V direction. Index in between 0 and vCount-1.

TVector3* *vertex*
Where to return the Vertex coordinates. The values will be copied from the internal Polygon Array data into this place.

TVector3* *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 MC_S_OK

I3DShPolygonArray::MeshSetU(uint32, real)

MCCOMErr MeshSetU(uint32 uIndex, real uValue)

Use **I3DShPolygonArray::MeshSetU** to set the U coordinate of all points at uIndex.

Parameters

uint32 *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.

real *uValue*
The U value for all points at uIndex. This value is between 0 and the U value of the last vertex.

Return Value MC_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(uint32, real)

MCCOMErr MeshSetV(uint32 vIndex, real vValue)

Use **I3DShPolygonArray::MeshSetV** to set the V coordinate of all points at vIndex.

Parameters

uint32 *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.

real *vValue*
The V value for all points at vIndex. This value is between 0 and the V value of the last vertex.

©2001-2003 Eovia. All rights reserved.

Return Value MC_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(uint32, real*)

MCCOMErr MeshGetU(uint32 uIndex, real* uValue)

Use **I3DShPolygonArray::MeshGetU** to get the U coordinate of all points at uIndex.

Parameters

uint32 *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.

real* *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 MC_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(uint32, real*)

MCCOMErr MeshGetV(uint32 vIndex, real* vValue)

Use **I3DShPolygonArray::MeshGetU** to get the V coordinate of all points at vIndex.

Parameters

uint32 *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.

real* *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 MC_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::MeshCalcUV()

MCCOMErr MeshCalcUV()

Call **I3DShPolygonArray::CalcUV** if you want to have all the UV values calculated for you.

©2001-2003 Eovia. All rights reserved.

Return Value MC_S_OK

Comments You must call this procedure if you did not set any UV values using **I3DShPolygonArray::MeshSetU** and **I3DShPolygonArray::MeshSetV**.

I3DShPolygonArray::MeshCalcNormals(boolean)

MCCOMErr MeshCalcNormals(boolean smooth)

Use **I3DShPolygonArray::CalcNormals** to have all Normals calculated for you.

Return Value MC_S_OK

Comments You must call this procedure if you did not set any Normals using **I3DShPolygonArray::SetVertex**.

I3DShPolygonArray::GetSize(uint32&, uint32&, boolean&, boolean&)

MCCOMErr GetSize(uint32& uCount, uint32& vCount, boolean& uClosed, boolean& vClosed)

Retrives the information passed to I3DShPolygonArray::Init.

Parameters

- **uint32&** *uCount*
- **uint32&** *vCount*
- **boolean&** *uClosed*
- **boolean&** *vClosed*

Return Value MC_S_OK

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.12 I3DShPolygonList Interface

Interface ID: IID_I3DShPolygonList

Interface File: I3DShObject.h

Description

A Polygon List is simply a bag of triangular facets in unsorted order.
For better efficiency you should use the **FacetMeshPrimitive**.

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::CalcNormals** 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.

Method Index

"I3DShPolygonList::Init(boolean, boolean)"
"I3DShPolygonList::PreAllocateFacets(uint32)"
"I3DShPolygonList::AddFacet(TFacet3D*)"
"I3DShPolygonList::CountFacets()"
"I3DShPolygonList::CalcNormals(real)"
"I3DShPolygonList::GetIndFacets(uint32)"
"I3DShPolygonList::SetHasUVSpace(boolean)"
"I3DShPolygonList::SetUVSpace(UVSpaceInfo*)"
"I3DShPolygonList::GetUVSpace(uint32, UVSpaceInfo*)"
"I3DShPolygonList::GetUVSpaceCount()"

I3DShPolygonList::Init(boolean, boolean)

MCCOMErr Init(boolean hasNormals, boolean hasUV)

I3DShPolygonList::Init must be called right after creating a Polygon List object.

Parameters

boolean *hasNormals*
Indicates if the polygon list has surface normals.

©2001-2003 Eovia. All rights reserved.

boolean *hasUV*
Indicates if the polygon list has UV values.

Return Value MC_S_OK

I3DShPolygonList::PreAllocateFacets(uint32)

MCCOMErr PreAllocateFacets(uint32 nbFacets)

Call **I3DShPolygonList::PreAllocateFacets** to pre-allocate the memory space needed for *nbFacets* facets.

Parameters **uint32** *nbFacets*
Number of facets that will be added later with I3DShPolygonList::AddFacet.

Return Value MC_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(TFacet3D*)

MCCOMErr AddFacet(TFacet3D* facet)

Use **I3DShPolygonList::AddFacet** to add facets to a Polygon List object one after the other.

Parameters **TFacet3D* 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 MC_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::AddFacet fails if running out of RAM (see I3DShPolygonList::PreAllocateFacets about memory management).

I3DShPolygonList::CountFacets()

uint32 CountFacets()

Returns the number of facets in the Polygon List.

Return Value The number of facets as an uint32.

©2001-2003 Eovia. All rights reserved.

I3DShPolygonList::CalcNormals(real)

MCCOMErr CalcNormals(real smoothAngle)

Calculates the surface normals with the given smooth angle.

Parameters **real** *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 MC_S_OK

Comments **I3DShPolygonList::CalcNormals** must be called if hasNormals was FALSE when calling **I3DShPolygonList::Init**.

I3DShPolygonList::GetIndFacets(uint32)

TFacet3D* GetIndFacets(uint32 index)

Use **I3DShPolygonList::GetIndFacets** to get a facet in the list.

Parameters **uint32** *index*

Index of the facet, must be between 0 and I3DShPolygonList::CountFacets - 1

Return Value A TFacet3D* referenced by the index.

I3DShPolygonList::SetHasUVSpace(boolean)

MCCOMErr SetHasUVSpace(boolean hasUV)

If *hasUV* is false **SetHasUVSpace()** removes all previous UV Space definitions.

I3DShPolygonList::SetUVSpace(UVSpaceInfo*)

MCCOMErr SetUVSpace(UVSpaceInfo* uvSpaceInfo)

Set a UV Space. The UV space will be added if it was not previously defined. See the Data Structure reference section for more information on the *UVSpaceInfo* structure.

Parameters **UVSpaceInfo*** *uvSpaceInfo*

Pointer to the UVSpaceInfo structure for defining the UV space.

Return Value MC_S_OK

I3DShPolygonList::GetUVSpace(uint32, UVSpaceInfo*)

MCCOMErr GetUVSpace(uint32 uvSpaceID, UVSpaceInfo* uvSpaceInfo)

Returns the *UVSpaceInfo* structure for the corresponding *uvSpaceID*. See the Data Structure reference section for more information on the *UVSpaceInfo* structure.

Parameters **uint32** *uvSpaceID*

The ID of the UV Space to retrieve.

©2001-2003 Eovia. All rights reserved.

UVSpaceInfo* *uvSpaceInfo*
The *UVSpaceInfo* structure.

Return Value MC_S_OK

I3DShPolygonList::GetUVSpaceCount()

uint32MCCOMAPI GetUVSpaceCount()

Gets the count of UV spaces.

Return Value The count of UV spaces as an uint32.

©2001-2003 Eovia. All rights reserved.

9.13 I3DShPostRenderer Interface

Interface ID : IID_I3DShPostRenderer

Interface file : I3DShPostRenderer.h

Description

A Post Render Filter operates on the pixels of an image after rendering. In addition to the pixel color data, G-Buffers are provided with information about the distance, XYZ position, XYZ normals, Surface UV, Alpha Channel, Diffuse color values, and more.

Method Index

"I3DShPostRenderer::SetScene(I3DShScene*)"
"I3DShPostRenderer::GetBufferNeeds(uint32&)"
"I3DShPostRenderer::PrepareDraw(IShRasterLayer*, const TMCRect&, IShChannel*, I3DShScene*, TBox2D&, TBox2D&)"
"I3DShPostRenderer::DrawRect(const TMCRect&, const TChannelDataBucket*, TBBox2D)"
"I3DShPostRenderer::FinishDraw()"
"I3DShPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean)"
"I3DShPostRenderer::EndRender()"

I3DShPostRenderer::SetScene(I3DShScene*)

MCCOMErr SetScene(I3DShScene* scene)

<zot>Need description here</zot> Sets a scene as the current scene.

Parameter **I3DShScene*** *scene*
A pointer to the scene.

Return Value MC_S_OK

I3DShPostRenderer::GetBufferNeeds(uint32&)

void GetBufferNeeds(uint32& needs)

GetBufferNeeds() is used to indicate which G-Buffers are needed.

©2001-2003 Eovia. All rights reserved.

Parameter **uint32&** *needs*

Return a set of flags for the informations you need as found in the table below:

Name	Value	Description
k32Distance	0	Distance from the camera to the object visible at a point in the image.
k32PositionX	1	X position of a point if not the background
k32PositionY	2	Y position of a point if not the background
k32PositionZ	3	Z position of a point if not the background
k32NormalX	4	X direction of an object's surface Normal if not the background
k32NormalY	5	Y direction of an object's surface Normal if not the background
k32NormalZ	6	Z direction of an object's surface Normal if not the background
k32Alpha	7	The alpha channel with black (0) as the background
k32Index	8	UV space index
k32SurfaceU	9	U value of the UV space of an object at a point
k32SurfaceV	10	V value of the UV space of an object at a point
k32DiffuseColor	11	Corresponds to the Color channel of the object's shader
k32SpecularColor	12	Corresponds to the Highlight channel of the object's shader
k32SpecularSizeValue	13	Corresponds to the Shininess channel of the object's shader
k32AmbientValue	14	<zot></zot>
k32LambertValue	15	Corresponds to the<zot></zot> channel of the object's shader
k32ReflectionColor	16	Corresponds to the Reflection channel of the object's shader
k32TransparencyColor	17	Corresponds to the Transparency channel of the object's shader
k32RefractionValue	18	Corresponds to the Refraction channel of the object's shader
k32GlowColor	19	Corresponds to the Glow channel of the object's shader

I3DShPostRenderer::PrepareDraw(IShRasterLayer*, const TMCRect&, IShChannel*, I3DShScene*, TBox2D&, TBox2D&)

MCCOMErr PrepareDraw(IShRasterLayer* input, const TMCRect& bounds, IShChannel* buffers[],I3DShScene* scene, I3DShCamera* renderingCamera, TBBox2D& uvBox, TBBox2D& productionFrame)

PrepareDraw() is called prior to rendering each block of the image.

©2001-2003 Eovia. All rights reserved.

Parameters

IShRasterLayer* *input*
<zot>desc.</zot>

const TMCRect& *bounds*
<zot>desc.</zot>

IShChannel* *buffers[]*
This is a table of pointers to the buffers requested by **GetBufferNeeds()**. Each element in the buffers array corresponds to the value of each buffer (buffer[0] for distance through buffer[19] for glow color). See the table in **GetBufferNeeds()** for the complete list.

I3DShScene* *scene*
This is a pointer to the scene structure.

I3DShCamera* *renderingCamera*
This is a pointer to the rendering camera.

TBBox2D& *uvBox*
Screen coordinates of the image.

TBBox2D& *productionFrame*
creen coordinates of the production frame.

Return Value MC_S_OK

I3DShPostRenderer::DrawRect(const TMCRect&, const TChannelDataBucket*, TBBox2D)

MCCOMErr DrawRect(const TMCRect& outputRect, const TChannelDataBucket* output[], TBBox2D& uvBox)

DrawRect() is called for each tile of the image.

Parameters

TMCRect& *outputRect*
<zot>desc.</zot>screen coordinates for the output

TChannelDataBucket* *output[]*
<zot>desc.</zot>

TBBox2D& *uvBox*
Screen coordinates of the image.

I3DShPostRenderer::FinishDraw()

MCCOMErr FinishDraw()

FinishDraw() is called for cleanup at the end of the post-rendering.

©2001-2003 Eovia. All rights reserved.

I3DShPostRenderer::BeginRender(MicroTick, MicroTick, unit32, uint32, boolean)

MCCOMErrMCCOMAPI BeginRender(MicroTick beginTime,MicroTick endTime, uint32 framePerSeconds,int32 nbrFrames, boolean isRenderMovie)

BeginRender() is called to start the post-rendering.

Parameters

MicroTick *beginTime*
Specifies the starting time of the movie for the render.

MicroTick *endTime*
Specifies the ending time of the movie for the render.

uint32 *framePerSeconds*
Specifies the frames per second.

uint32 *nbrFrames*
Specifies the frame rate.

boolean *isRenderMovie*
Is True when rendering a movie.

I3DShPostRenderer::EndRender()

MCCOMErr EndRender()

EndRender() is called for cleanup at the end of the post-rendering.

©2001-2003 Eovia. All rights reserved.

9.14 I3DShMasterShader Interface

Interface ID : IID_I3DShMasterShader

Interface file : I3DShShader.h

Description

zot add description

Method Index

"I3DShMasterShader::SetName(const TMCString&)"
"I3DShMasterShader::GetName(TMCString&)"
"I3DShMasterShader::SetShader(I3DShShader*, ECloneMode)"
"I3DShMasterShader::GetShader(I3DShShader**)"
"I3DShMasterShader::Clone(I3DShMasterShader**, ECloneMode)"
"I3DShMasterShader::GetScene(I3DShScene**)"
"I3DShMasterShader::IsDefaultMaster()"

I3DShMasterShader::SetName(const TMCString&)

MCCOMErr MCCOMAPI SetName(const TMCString& name)

<zot>Sets the name of a master shader.

Parameters

TMCString& *name*
The name of the master shader.

I3DShMasterShader::GetName(TMCString&)

MCCOMErr MCCOMAPI GetName(TMCString& name)

<zot>Gets the name of a master shader.

Parameters

TMCString& *name*
The name of the master shader.

I3DShMasterShader::SetShader(I3DShShader*, ECloneMode)

MCCOMErr MCCOMAPI SetShader(I3DShShader* shader, ECloneMode cloneMode)

SetShader() specifies which shader is set.

Parameter

I3DShShader *shader*
Specifies which shader to set.

ECloneMode *cloneMode*
Specifies whether to clone with or without animation information.

©2001-2003 Eovia. All rights reserved.

I3DShMasterShader::GetShader(I3DShShader**)

MCCOMErr MCCOMAPI GetShader(I3DShShader shader)**

GetShader() gets a shader.

Parameters **I3DShShader**** *shader*

Specifies which shader to get.

I3DShMasterShader::Clone(I3DShMasterShader**, ECloneMode)

MCCOMErr MCCOMAPI Clone(I3DShMasterShader clone, ECloneMode cloneMode)**

Clone() clones a master shader.

Parameters **I3DShMasterShader**** *clone*

Specifies which shader to get.

ECloneMode *cloneMode*

Specifies whether to clone with or without animation information.

I3DShMasterShader::GetScene(I3DShScene**)

MCCOMErr MCCOMAPI GetScene(I3DShScene scene)**

GetScene() returns a scene.

Parameters **I3DShScene**** *scene*

Specifies the scene.

I3DShMasterShader::IsDefaultMaster()

boolean MCCOMAPI IsDefaultMaster()

IsDefaultMaster() specifies whether a master shader is the default master shader.

©2001-2003 Eovia. All rights reserved.

9.15 I3DShPrimitive Interface

Interface ID: IID_I3DShPrimitive

Interface file: I3DShObject.h

Description

A Primitive is a simple geometric object. **I3DShPrimitive** contains the methods needed to create the geometry.

Method Index

"I3DShPrimitive::IsPatchBased()"
"I3DShPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean)"
"I3DShPrimitive::ChangedData()"
"I3DShPrimitive::GetFMesh(real, FacetMesh**)"
"I3DShPrimitive::IsVolumic()"
"I3DShPrimitive::UV2XYZ(const TVector2&, int16, TVector3&, TVector3*)"

I3DShPrimitive::IsPatchBased()

boolean IsPatchBased()

This method simply indicates if an object has patches or is comprised only of facets.

Return Value

True	The Primitive is Patch based.
False	The Primitive is not Patch based.

I3DShPrimitive::EnumPatches(EnumPatchesCallback, void*, boolean)

MCCOMErr EnumPatches(EnumPatchesCallback callback, void* privData, boolean surfaceClosed)

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

Parameters

EnumPatchesCallback *callback*

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

```
typedef void (*EnumPatchesCallback) (TPatch3D* patch, void* privData);
```

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

void* ***privData***

Points to private data from the Shell. Pass this pointer to *callback.*

boolean *surfaceClosed*

Returns True when the surface is closed . This helps set the backface mask correctly.

©2001-2003 Eovia. All rights reserved.

Return Value

MC_S_OK	Patches were generated correctly.
MC_E_NOTIMPL	Patches can not be generated.

I3DShPrimitive::ChangedData()

MCCOMErr ChangedData()

<zot>below is the comment from the code - needs clarification</zot>

Call this if you changed some data of the Primitive so the Shell hears about it.

I3DShPrimitive::GetFMesh(real, FacetMesh**)

MCCOMErr GetFMesh(real lod, FacetMesh mesh)**

<zot>Needs description here</zot>

GetFMesh() is an alternate method for creating a facet mesh. It is more efficient than using **EnumFacets()** as it allows for calculating less facets in conditions that require less detail.

Parameters

real *lod*
The level of detail.

FacetMesh** *mesh*
A FacetMesh defining the primitive.

I3DShPrimitive::IsVolumic()

boolean IsVolumic()

<zot>below is the comment from the code - needs clarification</zot>

"Volumic" primitives are special primitives such as particle systems, fog, fountain, etc.

I3DShPrimitive::UV2XYZ(const TVector2&, int16, TVector3&, TVector3*)

boolean UV2XYZ(const TVector2& uv, int16 uvSpaceID, TVector3& thePos3D, TVector3* normal)

<zot>Needs descirption of how this method differs from I3DExGeometricPrimitive::UV2XZ</zot>

I3DShPrimitive::UV2XYZ is called when a point on the surface defined only by its uv coordinates needs to be converted in x, y and z values. This call is optional. If this call is not implement then the default rules will apply by using the patches or the facets of the Primitive. When possible implement **I3DShPrimitive::UV2XYZ** for optimal performance and accuracy.

Parameters

const TVector2& *uv*
The given uv values.

int16 ***uvSpaceID***
The identifier of the UV space.

©2001-2003 Eovia. All rights reserved.

TVector3& *thePos3D*
The resulting 3D coordinates in the object Local Coordinates System.

TVector3* *normal*
<zot>Need description.</zot>

Return Value

True	<zot>Need description.</zot>
False	<zot>Need description.</zot>

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.16 I3DShPrimitiveComponent Interface

Interface ID: IID_I3DShPrimitiveComponent

Interface File: I3DShObject.h

Description

I3DShPrimitiveComponent points to the External primitive itself. See the **"I3DShExternalPrimitive Interface"**.

I3DShPrimitiveComponent provides access to the owner primitive from the component. This primitive interface then permits access to the scene and other elements through **QueryInterface()**. It also gives a way to get to the External Primitive attached to it such as I3DExGeometricPrimitive. IShComponent is also supported.

Method Index

"I3DShPrimitiveComponent::GetPrimitive(I3DShPrimitive**)"
"I3DShPrimitiveComponent::GetExternalPrimitive(IMCUnknown**)"

I3DShPrimitiveComponent::GetPrimitive(I3DShPrimitive**)

void GetPrimitive(I3DShPrimitive primitive)**

This provides access to the parent **I3DShExternalPrimitive**.

<zot>Need method description</zot>

Parameters **I3DshPrimitive**** *primitive*
<zot>Need Description</zot>

I3DShPrimitiveComponent::GetExternalPrimitive(IMCUnknown**)

MCCOMErr GetExternalPrimitive(IMCUnknown extPrimitive)**

Provides access to the external primitive itself (**I3DExGeometricPrimitive**, etc.).

<zot>Need method description</zot>

Parameters **IMCUnknown**** *extPrimitive*
<zot>Need Description</zot>

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

9.17 I3DShShader Interface

Interface ID : IID_I3DShShader

Interface file : I3DShShader.h

Description

With the exceptions of **IsConstant()** and **AssignsChannels()** the methods in **I3DShShader** call the **I3DShShader** method with the passed arguments.

Method Index

"I3DShShader::Clone(I3DShShader**, ECloneMode) const"
"I3DShShader::IsConstant()"
"I3DShShader::AssignsChannels()"
"I3DShShader::GetShadingFlags(ShadingFlags&)"
"I3DShShader::DoShade(ShadingOut&, ShadingIn&)"
"I3DShShader::GetValue(real&, boolean, ShadingIn&)"
"I3DShShader::GetColor(TMCColorRGB&, boolean, ShadingIn&)"
"I3DShShader::GetVector(TVector3&, ShadingIn&)"
"I3DShShader::GetBump(TVector3&, ShadingIn&)"
"I3DShShader::GetImplementedOutput()"
"I3DShShader::GetOriginalParametricTextureMap(uint32&,uint32&, IShTextureMap**)"

I3DShShader::Clone(I3DShShader**, ECloneMode) const

voidMCCOMAPI Clone(I3DShShader clone, ECloneMode cloneMode) const**

<zot>Clones a shader.

Parameters

I3DShShader** *clone*
Returns the cloned shader.

ECloneMode *cloneMode*
Specifies the clone mode (with or without the animation data).

I3DShShader::IsConstant()

boolean IsConstant()

If a shader is constant a single value is applied to the entire surface. This is used for basic components such as the color shader.

Return Value

True	The shader value is constant across the surface.
False	The shader value is not constant across the surface.

©2001-2003 Eovia. All rights reserved.

I3DShShader::AssignsChannels()

boolean AssignsChannels()

This function indicates if the **DoShade()** method is implemented by the shader.

Return Value

True	**DoShade()** is implemented.
False	**DoShade()** is not implemented.

I3DShShader::GetShadingFlags(ShadingFlags&)

MCCOMErr GetShadingFlags(ShadingFlags& theFlags)

GetShadingFlags specifies which parameters the shader will need to perform shading calculations. If the shader is a *constant shader* which values does not depend on any of the *ShadingIn* parameters then it should set the *fCallOnce* flag in *theFlags* to true. See the description of the "ShadingFlags" structure for more details.

Parameter **ShadingFlags&** *theFlags*

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

Return Value **MC_S_OK**

I3DShShader::DoShade(ShadingOut&, ShadingIn&)

MCCOMErr DoShade(ShadingOut& result, ShadingIn& shadingIn)

I3DShShader::DoShade has total control of the input and output parameters. See the description of "ShadingFlags" and "ShadingIn" for more details.

Parameters **ShadingOut&** *result*

Reference to the complete input/output shading data structure.

ShadingIn& *shadingIn*

The Shading element associated with the shader.

Return Value

MC_S_OK	**DoShade()** is implemented by this shader.
MC_E_NOTIMPL	**DoShade()** is not implemented by this shader.

Comments

I3DShShader::GetValue(real&, boolean, ShadingIn&)

MCCOMErr GetValue(real& result, boolean fullArea, ShadingIn& shadingIn)

GetValue returns a value between 0.0 and 1.0. Typical examples are noise functions, fractal functions, gradients, etc. See the description of "ShadingIn" for more details.

©2001-2003 Eovia. All rights reserved.

Parameters

real& *result*
Returns a result between 0.0 and 1.0.

boolean ***fullArea***
<zot> needs description </end zot> guess: Returns True if the full area uses the same value.

ShadingIn& ***shadingIn***
Reference to the input shading data structure. The shader uses the parameters requested by **"I3DShShader::GetShadingFlags(ShadingFlags&)"**.

Return Value

MC_S_OK	**GetValue()** is implemented by this shader.
MC_E_NOTIMPL	**GetValue()** is not implemented by this shader.

I3DShShader::GetColor(TMCColorRGB&, boolean, ShadingIn&)

MCCOMErr GetColor(TMCColorRGB& result, boolean fullArea, ShadingIn& shadingIn)

GetColor returns a color. Typical examples are constant colors or texture maps. See the description of "ShadingIn" for more details.

Parameters

TMCColorRGB& *result*
Return the color values.

boolean *fullArea*
<zot> needs description </end zot>

ShadingIn *& in*

Points to the input shading data structure. The shader uses the parameters requested by **"I3DShShader::GetShadingFlags(ShadingFlags&)"**.

Return Value

MC_S_OK	**GetColor()** is implemented by this shader.
MC_E_NOTIMPL	**GetColor()** is not implemented by this shader.

Comments

I3DShShader::GetVector(TVector3&, ShadingIn&)

MCCOMErr GetVector(TVector3& result, ShadingIn& shadingIn)

I3DShShader::GetVector is reserved for a future use.

I3DShShader::GetBump(TVector3&, ShadingIn&)

MCCOMErr GetBump(TVector3& result, ShadingIn& shadingIn)

Returns the bump channel only (usually implemented with DoShade).

©2001-2003 Eovia. All rights reserved.

Parameters **TVector3&** *result*
Return the bump values.

ShadingIn *&shadingIn*
Points to the input shading data structure. The shader uses the parameters requested by "I3DShShader::GetShadingFlags(ShadingFlags&)".

I3DShShader::GetImplementedOutput()

uint32 GetImplementedOutput()

GetImplementedOutput is called by the 3D Shell to determine which methods the Shader actually implements. Return only one of the following values if you implement only one of the methods. If more than one is implemented, such as the *ShadedArea* implementation, then return the result as the OR of the two values (i.e. return kUsesGetValueArea | kUsesGetValue).

Return Value

Shading Defines

Constant	Value	Meaning
kUsesGetValue	1	**I3DShShader::GetValue** is implemented
kUsesGetColor	2	**I3DShShader::GetColor** is implemented
kUsesGetVector	4	**I3DShShader::GetVector** is implemented
kUsesDoShade	16	**I3DShShader::DoShade** is implemented
kUsesGetValueArea	32	**I3DShShader::GetValue** with *ShadedArea* is implemented
kUsesGetColorArea	64	**I3DShShader::GetColor** with *ShadedArea* is implemented
kUsesGetVectorArea	128	**I3DShShader::GetVector** with *ShadedArea* is implemented
kUsesDoShadeArea	256	**I3DShShader::DoShade** with *ShadedArea* is implemented

I3DShShader::GetOriginalParametricTextureMap(uint32&,uint32&, IShTextureMap**)

MCCOMErrMCCOMAPI GetOriginalParametricTextureMap(const uint32& uvSpaceID, const uint32& channelIndex, IShTextureMap outMap)**

Gets the original parametric texture map.

Parameters **uint32&** *uvSpaceID*
Returns the UV Space ID .

uint32& *channelIndex*
Return the index to bump values.

IShTextureMap** *outMap*
Points to the output shading data structure.

©2001-2003 Eovia. All rights reserved.

9.18 I3DShTreeElement Interface

Interface ID : IID_I3DShTreeElement

Interface file : I3DShTreeElement.h

Description

I3DShTreeElement is the abstract class of all elements in a tree (called "tree elements"). These elements can be groups, object instances, lights sources, cameras, and so on. You can always get this interface from a **I3DShGroup**, **I3DShInstance**, etc.

Method Index

"I3DShTreeElement::SetScene(I3DShScene*)"
"I3DShTreeElement::GetScene(I3DShScene**)"
"I3DShTreeElement::GetSceneNoAddRef()"
"I3DShTreeElement::SetName(const TMCString&)"
"I3DShTreeElement::GetName(TMCString&)"
"I3DShTreeElement::GetFirst(I3DShTreeElement**)"
"I3DShTreeElement::GetLast(I3DShTreeElement**)"
"I3DShTreeElement::GetLeft(I3DShTreeElement**)"
"I3DShTreeElement::GetRight(I3DShTreeElement**)"
"I3DShTreeElement::GetTop(I3DShTreeElement**)"
"I3DShTreeElement::Unlink(I3DShTreeElement**, RelinkTreeElementInfo*)"
"I3DShTreeElement::Relink(RelinkTreeElementInfo*)"
"I3DShTreeElement::InsertFirst(I3DShTreeElement*)"
"I3DShTreeElement::InsertLast(I3DShTreeElement*)"
"I3DShTreeElement::InsertLeft(I3DShTreeElement*)"
"I3DShTreeElement::InsertRight(I3DShTreeElement*)"
"I3DShTreeElement::GetHotPoint(TVector3&)"
"I3DShTreeElement::SetHotPoint(const TVector3&)"
"I3DShTreeElement::CenterElementOnHotPoint()"
"I3DShTreeElement::CenterHotPointOnElement()"
"I3DShTreeElement::TranslateHotPoint(TVector3&)"
"I3DShTreeElement::ComClone(I3DShTreeElement**, ECloneMode, boolean)"
"I3DShTreeElement:: InvalidateBoundingBox()"
"I3DShTreeElement::GetBoundingBoxWithoutLightsAndCameras(TBBox3D&)"
"I3DShTreeElement::GetBoundingBox(TBBox3D&, ETreeDeformBBoxModes)"
"I3DShTreeElement::CanBeDistorted()"
"I3DShTreeElement::BeginMove()"
"I3DShTreeElement::PostMoveChange(boolean)"
"I3DShTreeElement::EndMove(boolean)"
"I3DShTreeElement::GetXYZScaling(TVector3&)"
"I3DShTreeElement::SetXYZScaling(TVector3&)"
"I3DShTreeElement::GetNbrFreedom()"
"I3DShTreeElement::GetFreedomRange(int16, real&, real&)"
"I3DShTreeElement::GetFreedomTransform(TMatrix33&, TVector3&)"
"I3DShTreeElement::GetFreedomDerivative(int16, TMatrix33&, TVector3&)"
"I3DShTreeElement::GetFreedomSecondDerivative(int16, int16, TMatrix33&, TVector3&)"
"I3DShTreeElement::IncrementFreedomValue(int16 , real*)"
"I3DShTreeElement::GetFullName(TMCString&)"
"I3DShTreeElement::InsertModifier(I3DShModifier* , uint32, ECloneMode)"
"I3DShTreeElement::RemoveModifier(uint32)"

©2001-2003 Eovia. All rights reserved.

"I3DShTreeElement::GetModifiersCount()"
"I3DShTreeElement::GetModifierByIndex(I3DShModifier**, uint32)"
"I3DShTreeElement::SetModifiers(const TMCCountedPtrArray<I3DShModifier>*, ECloneMode)"
"I3DShTreeElement::LinksChanged()"
"I3DShTreeElement::BeginGetRenderables(const TRenderableAndTfmArray*&, boolean)"
"I3DShTreeElement::EndGetRenderables()"
"I3DShTreeElement::GetConstraint(I3DShConstraint**)"
"I3DShTreeElement::SetConstraint(I3DShConstraint*, ECloneMode)"
"I3DShTreeElement::SetConstraintByID(int32)"
"I3DShTreeElement::GetDataComponentsCount()"
"I3DShTreeElement::GetDataComponentByIndex(I3DShDataComponent**, uint32)"
"I3DShTreeElement::SetDataComponentByID(I3DShDataComponent*, ECloneMode)"
"I3DShTreeElement::GetDataComponentByID(I3DShDataComponent**, IDType)"
"I3DShTreeElement::InsertDataComponent(I3DShDataComponent*)"
"I3DShTreeElement::InsertDataComponentByID(int32, int32)"
"I3DShTreeElement::RemoveDataComponent(uint32)"
"I3DShTreeElement::SetData(TDataKey, IMCUnknown*)"
"I3DShTreeElement::GetData(TDataKey, IMCUnknown**)"
"I3DShTreeElement::ReleaseData(TDataKey)"
"I3DShTreeElement::IsOpen()"
"I3DShTreeElement::GetEditor()"
"I3DShTreeElement::GetAnimationMethod(I3DShAnimationMethod**)"
"I3DShTreeElement::SetAnimationMethod(I3DShAnimationMethod*, ECloneMode, real)"
"I3DShTreeElement::GetLocalTreeTransform(TTreeTransform&)"
"I3DShTreeElement::SetLocalTreeTransform(const TTreeTransform&)"
"I3DShTreeElement::GetGlobalTreeTransform(TTreeTransform&)"
"I3DShTreeElement::SetGlobalTreeTransform(const TTreeTransform&, int16, boolean)"
"I3DShTreeElement::GetGlobalTreeTransform3D(TTreeTransform3D&)"
"I3DShTreeElement::SetGlobalTransform3D(const TTransform3D&, int16)"
"I3DShTreeElement::GetFatherGlobalTransform3D(TTransform3D&)"
"I3DShTreeElement::CloneSubTree(ECloneMode, I3DShTreeElement**)"
"I3DShTreeElement::GetUndoInfo(TTreeUndoInfo&)"
"I3DShTreeElement::SetFromUndoInfo(const TTreeUndoInfo&, const TTreeUndoInfo*)"
"I3DShTreeElement::GetTopIfConstrained(I3DShTreeElement**)"
"I3DShTreeElement::AddWireFrame(I3DExWireFrame*)"
"I3DShTreeElement::RemoveWireFrame(I3DExWireFrame*)"
"I3DShTreeElement::WireFrameArrayFree()"
"I3DShTreeElement::GetWireFrameArray(TMCClassArray<TWireFrameInfo>&)"
"I3DShTreeElement:: IsPickable()"
"I3DShTreeElement:: IsBBoxPickable()"
"I3DShTreeElement::RemoveConstraint(I3DShConstraint**)"
"I3DShTreeElement::RestoreConstraint(I3DShConstraint*)"
"I3DShTreeElement::GetExists()"
"I3DShTreeElement::SetExists(boolean)"
"I3DShTreeElement::GetIsShown()"
"I3DShTreeElement::SetIsShown(boolean)"
"I3DShTreeElement::SetShadowCasting(boolean)"
"I3DShTreeElement::SetShadowCatching(boolean)"
"I3DShTreeElement::GetShadowCasting()"
"I3DShTreeElement::GetShadowCatching()"
"I3DShTreeElement::GetAnimateTree()"
"I3DShTreeElement::SetAnimateTree(boolean)"

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::SetScene(I3DShScene*)

MCCOMErrMCCOMAPI SetScene(I3DShScene* scene)

SetScene() must be called after creating a Tree Element to set which scene it belongs to.

Parameters **I3DShScene*** *scene*
The scene to which the tree element belongs.

I3DShTreeElement::GetScene(I3DShScene**)

MCCOMErrMCCOMAPI GetScene(I3DShScene scene)**

GetScene() gets the scene a tree element belongs to.

Parameters **I3DShScene**** *scene*
Address of a counted pointer that is set to the scene to which the tree element belongs (the scene is addrefed).

I3DShTreeElement::GetSceneNoAddRef()

I3DShScene* MCCOMAPI GetSceneNoAddRef()

Returns a pointer to the scene to which the tree element belongs (without addrefing it).

I3DShTreeElement::SetName(const TMCString&)

MCCOMErr MCCOMAPI SetName(const TMCString& name)

Use **SetName()** to change the name of the tree element.

Parameters **const TMCString&** *name*
The new the name (31 characters maximum.)

Comments The pointer is not kept; the string content it points to is copied into the I3DShTreeElement internal space.

I3DShTreeElement::GetName(TMCString&)

MCCOMErr MCCOMAPI GetName(TMCString& name)

Gets the name of a tree element.

Parameters **TMCString&** *name*
Points to where to store the name (31 characters maximum.)

I3DShTreeElement::GetFirst(I3DShTreeElement**)

MCCOMAPI GetFirst(I3DShTreeElement)**

Used for tree navigation. Gets the first son of a tree element.

Parameter **I3dShTreeElement****
Returns the first son, NULL if there is none.

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::GetLast(I3DShTreeElement**)

MCCOMAPI GetLast(I3DShTreeElement)**

Used for tree navigation. Gets the last son of a tree element

Parameter **I3dShTreeElement****
Returns the last son, NULL if there is none.

I3DShTreeElement::GetLeft(I3DShTreeElement**)

MCCOMAPI GetLeft(I3DShTreeElement)**

Used for tree navigation. Gets the left brother of a tree element.

Parameter **I3dShTreeElement****
Returns the left brother, NULL if there is none.

I3DShTreeElement::GetRight(I3DShTreeElement**)

MCCOMAPI GetRight(I3DShTreeElement)**

Used for tree navigation. Gets the right brother of a tree element.

Parameter **I3dShTreeElement****
Returns the right brother, NULL if there is none.

I3DShTreeElement::GetTop(I3DShTreeElement**)

MCCOMAPI GetTop(I3DShTreeElement)**

GetTop() is used for tree navigation. Gets the father of a tree element.

Parameter **I3dShTreeElement****
Returns the father, NULL if there is none.

I3DShTreeElement::Unlink(I3DShTreeElement**, RelinkTreeElementInfo*)

MCCOMErr MCCOMAPI Unlink(I3DShTreeElement treeElement, RelinkTreeElementInfo* outRelink=NULL)**

Unlink() unhooks a tree element from its tree. The whole sub-tree is unhooked along with it.

Parameters **I3DShTreeElement**** *treeElement*
Specifies the element to unhook within the tree.

RelinkTreeElementInfo *outRelink*
Returns the old father.

I3DShTreeElement::Relink(RelinkTreeElementInfo*)

MCCOMErr MCCOMAPI Relink(RelinkTreeElementInfo* inRelink)

Relink() rehooks a tree element based upon relink.

©2001-2003 Eovia. All rights reserved.

Parameters **RelinkTreeElementInfo*** *inRelink*
Returns a pointer to the old father.

I3DShTreeElement::InsertFirst(I3DShTreeElement*)

MCCOMErr MCCOMAPI InsertFirst(I3DShTreeElement* newSon)

InsertFirst() inserts an element as the first son of the tree element.

Parameters **I3DShTreeElement*** *newSon*
Points to the element to insert.

I3DShTreeElement::InsertLast(I3DShTreeElement*)

MCCOMErr MCCOMAPI InsertLast(I3DShTreeElement* newSon)

InsertLast() inserts an element as the last son of a tree element.

Parameters **I3DShTreeElement*** *newSon*
Points to the element to insert.

I3DShTreeElement::InsertLeft(I3DShTreeElement*)

MCCOMErr MCCOMAPI InsertLeft(I3DShTreeElement* newBrother)

InsertLeft() inserts an element as the first left brother of a tree element.

Parameters **I3DShTreeElement*** *newBrother*
Points to the element to insert.

I3DShTreeElement::InsertRight(I3DShTreeElement*)

MCCOMErr MCCOMAPI InsertRight(I3DShTreeElement* newBrother)

InsertLeft() inserts an element as the first right brother of a tree element.

Parameters **I3DShTreeElement*** *newBrother*
Points to the element to insert.

I3DShTreeElement::GetHotPoint(TVector3&)

MCCOMErr MCCOMAPI GetHotPoint(TVector3& hotPoint)

GetHotPoint returns a hot point in local coordinates. The hot point is the point around which rotation and scaling occur.

Parameters **TVector3&** *hotPoint*
Points to where to store the coordinates.

I3DShTreeElement::SetHotPoint(const TVector3&)

MCCOMErr MCCOMAPI SetHotPoint(const TVector3& hotPoint)

SetHotPoint sets a hot point in local coordinates. The hot point is the point around which rotation and

©2001-2003 Eovia. All rights reserved.

scaling occur.

Parameters **const TVector3&** *hotPoint*
Points to where to store the coordinates.

Comments Changing the hot point position of a tree element changes the object global coordinates, and not the hot point position in global coordinates.

I3DShTreeElement::CenterElementOnHotPoint()

MCCOMErr MCCOMAPI CenterElementOnHotPoint()

CenterElementOnHotPoint() centers an element on a hot point, by moving the TreeElement so that the bounding box is centered on the hot point. The hot point is the point around which rotation and scaling occur.

I3DShTreeElement::CenterHotPointOnElement()

MCCOMErr MCCOMAPI CenterHotPointOnElement()

CenterHotPointOnElement() centers a hot point on an element, by moving the hot point to the center of the bounding box. The hot point is the point around which rotation and scaling occur.

I3DShTreeElement::TranslateHotPoint(TVector3&)

MCCOMErr MCCOMAPI TranslateHotPoint(TVector3& delta)

Translates the hot point of *delta*, in local coordinates. The hot point is the point around which rotation and scaling occur.

Parameters **TVector3&** *delta*
Points to where to store the coordinates of the point *delta*.

I3DShTreeElement::ComClone(I3DShTreeElement**, ECloneMode, boolean)

MCCOMErr MCCOMAPI ComClone(I3DShTreeElement treeElement, ECloneMode cloneMode, boolean cloneAll)**

ComClone() . Duplicates a tree element. The new tree element is unhooked from the tree the original tree element is a member of, and becomes an independent branch.

Parameters **I3DShTreeElement**** *treeElement*
Where to put the cloned tree element**.**

ECloneMode *cloneMode*
Specifies whether to clone with or without animation information.

boolean *cloneAll*
zot needs description.

I3DShTreeElement:: InvalidateBoundingBox()

MCCOMAPI InvalidateBoundingBox()

©2001-2003 Eovia. All rights reserved.

Invalidates the current bounding box. **zot** need more description

I3DShTreeElement::GetBoundingBoxWithoutLight-sAndCameras(TBBox3D&)

MCCOMAPI GetBoundingBoxWithoutLightsAndCameras(TBBox3D& bbox)

GetBoundingBoxWithoutLightsAndCameras() gets the specified bounding box without lights and camera information. This is used to get a bounding box on only the geometric information in a tree hierarchy.

Parameters **TBBox3D&** *bbox*
The coordinates of the bounding box.

I3DShTreeElement::GetBoundingBox(TBBox3D&, ETreeDeformBBox-Modes)

MCCOMAPI GetBoundingBox(TBBox3D& bbox, ETreeDeformBBoxModes deformMode)

GetBoundingBox() gets the bounding box of a tree element.

Parameters **TBBox3D&** *bbox*
The coordinates of the bounding box.

ETreeDeformBBoxModes *deformMode*
Indicates how to compute the bbox when deformers are applied on the tree or its parents. Refer to the data structures chapter (zot xref) for more information.

I3DShTreeElement::CanBeDistorted()

boolean MCCOMAPI CanBeDistorted()

CanBeDistorted() returns True if the bounding box can be distorted.

I3DShTreeElement::BeginMove()

MCCOMErr MCCOMAPI BeginMove()

BeginMove() 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.

I3DShTreeElement::PostMoveChange(boolean)

void MCCOMAPI PostMoveChange(boolean invalidateDeformedGeometry = false)

PostMoveChange() tells the listeners that the tree moved. It is called by I3DShTreeElement::EndMove().

Parameters **boolean** *invalidateDeformedGeometry*
Should the instances with deformers invalidate their deformed meshes (false by default).

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::EndMove(boolean)

void MCCOMAPI EndMove(boolean invalidateDeformedGeometry = false)

EndMove() 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 change the parameters of the tree element.

Parameters **boolean** *invalidateDeformedGeometry*
See I3DShTreeElement::PostMoveChange().

I3DShTreeElement::GetXYZScaling(TVector3&)

MCCOMErr MCCOMAPI GetXYZScaling(TVector3& xyzScaling)

GetXYZScaling() is called to get the scaling in the X, Y, and Z directions of a tree element.

Parameters **TVector3&** *xyzScaling*
References the coordinates for the X, Y, and Z scaling.

I3DShTreeElement::SetXYZScaling(TVector3&)

MCCOMErr MCCOMAPI SetXYZScaling(TVector3& xyzScaling)

Call **SetXYZScaling()** to set the scaling in the X, Y, and Z directions of a tree element.

Parameters **TVector3&** *xyzScaling*
References the coordinates for the X, Y, and Z scaling.

I3DShTreeElement::GetNbrFreedom()

int16 MCCOMAPI GetNbrFreedom()

Call **GetNbrFreedom()** to get the number of degrees of freedom in a tree element. This is the total number of possible translations and rotations. A totally free tree element has 6 degrees of freedom. Inverse Kinematics uses this method to calculate movement.

I3DShTreeElement::GetFreedomRange(int16, real&, real&)

MCCOMErr MCCOMAPI GetFreedomRange(int16 index, real& min, real& max)

GetFreedomRange() returns the return the range of variation fo a given freedom. Inverse Kinematics uses this method to calculate movement.

Parameters **uint32** *index*
Select the freedom.

real *min*
min is the minimum increment value.

real *max*
max is the maximum increment value.

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::GetFreedomTransform(TMatrix33&, TVector3&)

MCCOMErr MCCOMAPI GetFreedomTransform(TMatrix33& RR,TVector3& TT)

GetFreedomTransform() gets the transformation of a tree element given by a method. Inverse Kinematics uses this method to calculate movement.

Parameters

TMatrix33& *RR*
Reference to the rotation matrix. See TMaxtrix33 in the data structures chapter.

TVector3& *TT*
Indicates a three dimensional vector for transformation **zot**. Refer to the data structures chapter (zot xref) for more information.

I3DShTreeElement::GetFreedomDerivative(int16, TMatrix33&, TVector3&)

MCCOMErr MCCOMAPI GetFreedomDerivative(int16 index,TMatrix33& RR,TVector3& TT)

GetFreedomDerivative() gets the derivative of a tree element given by a modifier. Inverse Kinematics uses this method to calculate movement. The derviative corresponds to the difference of freedom transformation when the given freedom value increases by 1. That information is used by the 3DShell to know which freedom value must be changed when the user drags the tree element.

Parameters

int16 *index*
Selects the freedom.

TMatrix33& *RR*
Reference to the rotation matrix. See TMaxtrix33 in the data structures chapter.

TVector3& *TT*
Reference to a three dimensional vector for transformation **zot**. Refer to the data structures chapter (zot xref) for more information.

I3DShTreeElement::GetFreedomSecondDerivative(int16, int16, TMatrix33&, TVector3&)

MCCOMErr MCCOMAPI GetFreedomSecondDerivative(int16 index1, int16 index2, TMatrix33& RR,TVector3& TT)

GetFreedomSecondDerivative() **zot** needs more information.

Parameters

int16 *index1*
zot needs description: Selects the first freedom?

int16 *index2*
zot needs description: Selects the second freedom?

TMatrix33& *RR*
Reference to the rotation matrix. See TMaxtrix33 in the data structures chapter.

TVector3& *TT*
Indicates a three dimensional vector for transformation **zot**. Refer to the data structures chapter (zot

©2001-2003 Eovia. All rights reserved.

xref) for more information.

I3DShTreeElement::IncrementFreedomValue(int16 , real*)

MCCOMErr MCCOMAPI IncrementFreedomValue(int16 index,real* value)

IncrementFreedomValue() increases the freedome value of a given freedom.

Parameters

int16 *index*
Selects the freedom.

real* *value*
Points to the value to add to the freedom.

I3DShTreeElement::GetFullName(TMCString&)

MCCOMErr MCCOMAPI GetFullName(TMCString& fullName)

GetFullName() returns the path name of the tree element (nodes separated by kPathNameSeparator). This is the name of the tree element and all of the fathers starting from the universe.

Parameters

TMCString& *fullName*
The full path name of the tree element.

I3DShTreeElement::InsertModifier(I3DShModifier* , uint32, ECloneMode)

MCCOMErr MCCOMAPI InsertModifier(I3DShModifier* modifier, uint32 index, ECloneMode cloneMode)

Inserts a modifier into the tree's modifier list at the given index.

Parameters

I3DShModifier* *modifier*
The modifier to insert.

uint32 *index*
Where to insert the modifier.

ECloneMode *cloneMode*
Currently unused.

I3DShTreeElement::RemoveModifier(uint32)

MCCOMErr MCCOMAPI RemoveModifier(uint32 index)

Removes the modifier at the given index from the tree's modifier list.

Parameters

uint32 *index*
The modifier to remove.

I3DShTreeElement::GetModifiersCount()

uint32 MCCOMAPI GetModifiersCount()

©2001-2003 Eovia. All rights reserved.

Return value The number of modifiers applied to the tree element.

I3DShTreeElement::GetModifierByIndex(I3DShModifier**, uint32)

MCCOMErr MCCOMAPI GetModifierByIndex(I3DShModifier modifier, uint32 index)**

Gets the indexth modifier applied to the tree.

Parameters

I3DShModifier** *modifier*
Returns the modifier.

uint32 *index*
Index of the modifier (zero based).

I3DShTreeElement::SetModifiers(const TMCCountedPtrArray<I3DShModifier>*, ECloneMode)

MCCOMErr MCCOMAPI SetModifiers(const TMCCountedPtrArray<I3DShModifier>* modifiers, ECloneMode cloneMode)

Sets the whole list of modifiers applied to the tree.

Parameters

const TMCCountedPtrArray<I3DShModifier>* *modifiers*
The new modifier list.

ECloneMode *cloneMode*
Specifies whether to clone with or without animation information.

I3DShTreeElement::LinksChanged()

MCCOMErr MCCOMAPI LinksChanged()

Call **LinksChanged()** to tell the Shell that a link of the tree element has changed.

I3DShTreeElement::BeginGetRenderables(const TRenderableAndTfmArray*&, boolean)

MCCOMErr MCCOMAPI BeginGetRenderables(const TRenderableAndTfmArray*& instances, boolean inOnlyShown = false)

Gets the renderables of the tree.

Parameters

const TRenderableAndTfmArray*& *instances*
The list of renderables to fill.

boolean *inOnlyShown*
Set *inOnlyShown* to true to filter out elements that have been hidden by the user.

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::EndGetRenderables()

MCCOMErr MCCOMAPI EndGetRenderables()

<zot>Need method description</zot> Call after BeginGetRenderables.

I3DShTreeElement::GetConstraint(I3DShConstraint**)

MCCOMErr MCCOMAPI GetConstraint(I3DShConstraint link)**

Gets the constraint applied to the tree.

Parameters **I3DShConstraint**** *link*

I3DShTreeElement::SetConstraint(I3DShConstraint*, ECloneMode)

MCCOMErr MCCOMAPI SetConstraint(I3DShConstraint* link, ECloneMode cloneMode)

Sets a constraint on the tree element.

Parameters **I3DShConstraint*** *link*

The constraint to be applied.

ECloneMode *cloneMode*

Specifies whether to clone with or without animation information.

I3DShTreeElement::SetConstraintByID(int32)

MCCOMErr MCCOMAPI SetConstraintByID(int32 kindID)

Sets a constraint on a tree element by specifying the constraint's ID.

Parameters **int32** *kindID*

The constraint's ID

I3DShTreeElement::GetDataComponentsCount()

uint32 MCCOMAPI GetDataComponentsCount()

Gets the number of the data components attached to the tree element.

I3DShTreeElement::GetDataComponentByIndex(I3DShDataComponent**, uint32)

MCCOMErr MCCOMAPI GetDataComponentByIndex(I3DShDataComponent dataComp, uint32 index)**

Gets the indexth data component of the tree.

Parameters **I3DShDataComponent**** *dataComp*

©2001-2003 Eovia. All rights reserved.

uint32 *index*
Index of the data component (zero based).

I3DShTreeElement::SetDataComponentByID(I3DShDataComponent*, ECloneMode)

MCCOMErr MCCOMAPI SetDataComponentByID(I3DShDataComponent* dataComp, ECloneMode cloneMode)

Adds a data component to the tree. If a data component with the same ID is found in the data components list, it is replaced by the new one.

Parameters

I3DShDataComponent* *dataComp*
The new data component to add to the tree.

ECloneMode *cloneMode*
Specifies whether to clone with or without animation information.

I3DShTreeElement::GetDataComponentByID(I3DShDataComponent**, IDType)

MCCOMErr MCCOMAPI GetDataComponentByID(I3DShDataComponent dataComp, IDType kindID)**

Gets the data component which class signature is equal to kindID.

Parameters

I3DShDataComponent** *dataComp*
Returns the data component.

IDType *kindID*
The class signature of the data component to return.

I3DShTreeElement::InsertDataComponent(I3DShDataComponent*)

MCCOMErr MCCOMAPI InsertDataComponent(I3DShDataComponent* dataComp)

Inserts a data component into the tree's data components list.

Parameters

I3DShDataComponent* *dataComp*
The data component to be inserted.

I3DShTreeElement::InsertDataComponentByID(int32, int32)

MCCOMErr MCCOMAPI InsertDataComponentByID(int32 kindID, int32 index)

Inserts a data component of the specified ID into the tree's data components list at the given index.

Parameters

int32 *kindID*
The ID of the data component to be inserted.

int32 *index*
Where to insert the data component.

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::RemoveDataComponent(uint32)

MCCOMErr MCCOMAPI RemoveDataComponent(uint32 index)

Removes the data component at the given index from the tree's data components list.

Parameters **uint32** *index*

The index of the data component to remove.

I3DShTreeElement::SetData(TDataKey, IMCUnknown*)

void MCCOMAPI SetData(TDataKey dataKey, IMCUnknown* data)

<zot>Need method description</zot> comment in file says: // DataByKey

Parameters **TDataKey** *dataKey*

<zot>Need Description</zot>

IMCUnknown* *data*

<zot>Need Description</zot>

I3DShTreeElement::GetData(TDataKey, IMCUnknown**)

void MCCOMAPI GetData(TDataKey dataKey, IMCUnknown data)**

<zot>Need method description</zot> comment in file says: // DataByKey

Parameters **TDataKey** *dataKey*

<zot>Need Description</zot>

IMCUnknown** *data*

<zot>Need Description</zot>

I3DShTreeElement::ReleaseData(TDataKey)

void MCCOMAPI ReleaseData(TDataKey dataKey)

<zot>Need method description</zot> comment in file says: // DataByKey

Parameters **TDataKey** *dataKey*

<zot>Need Description</zot>

I3DShTreeElement::IsOpen()

boolean MCCOMAPI IsOpen()

Tells if the tree (group) is opened or closed in the hierarchy.

I3DShTreeElement::GetEditor()

int32 MCCOMAPI GetEditor()

©2001-2003 Eovia. All rights reserved.

Return the ID of the Editor (i.e. the module) capable of editing this tree in the Modeling room. Returns zero when no editor is available

I3DShTreeElement::GetAnimationMethod(I3DShAnimationMethod**)

MCCOMErr MCCOMAPI GetAnimationMethod(I3DShAnimationMethod animationMethod)**

Gets the animation method of the tree.

Parameters

I3DShAnimationMethod** *animationMethod*
Returns the animation method of the tree.

I3DShTreeElement::SetAnimationMethod(I3DShAnimationMethod*, ECloneMode, real)

void MCCOMAPI SetAnimationMethod(I3DShAnimationMethod* animationMethod, ECloneMode cloneMode, real conversionQuality=0.0f)

<zot>Need method description</zot> Sets the animation method.

Parameters

I3DShAnimationMethod* *animationMethod*
<zot>Need Description</zot>

ECloneMode *cloneMode*
Specifies whether to clone with or without animation information.

real *conversionQuality*
By default, set to 0.0f.

I3DShTreeElement::GetLocalTreeTransform(TTreeTransform&)

void MCCOMAPI GetLocalTreeTransform(TTreeTransform& mat)

Gets the local transformation of the tree.

Parameters

TTreeTransform& *mat*
Returns the local transformation of the tree.

I3DShTreeElement::SetLocalTreeTransform(const TTreeTransform&)

void MCCOMAPI SetLocalTreeTransform(const TTreeTransform& mat)

Sets the local transformation of the tree.

Parameters

TTreeTransform& *mat*
The new local transformation of the tree.

I3DShTreeElement::GetGlobalTreeTransform(TTreeTransform&)

void MCCOMAPI GetGlobalTreeTransform(TTreeTransform& mat)

Gets the global transformation of the tree.

©2001-2003 Eovia. All rights reserved.

Parameters **TTreeTransform& *mat***
Returns the global transformation of the tree.

I3DShTreeElement::SetGlobalTreeTransform(const TTreeTransform&, int16, boolean)

void MCCOMAPI SetGlobalTreeTransform(const TTreeTransform& mat, int16 mode = kXTreeBehaviorDefault, boolean withXYZScaling = true)

Sets the global transformation of the tree.

Parameters **TTreeTransform& *mat***
The new global transformation of the tree.

int16 *mode*
<zot>Need Description</zot>*mode* is set to kXTreeBehaviorDefault.

boolean *withXYZScaling*
Default to true, unused.

I3DShTreeElement::GetGlobalTreeTransform3D(TTreeTransform3D&)

void MCCOMAPI GetGlobalTransform3D(TTransform3D& mat)

Gets the global transformation of the tree.

Parameters **TTreeTransform3D& *mat***
Returns the global transformation of the tree.

I3DShTreeElement::SetGlobalTransform3D(const TTransform3D&, int16)

void MCCOMAPI SetGlobalTransform3D(const TTransform3D& mat,int16 mode = kXTreeBehaviorDefault)

Sets the global transformation of the tree.

Parameters **const TTreeTransform& *mat***
The new global transformation of the tree.

int16 *mode*
<zot>Need Description</zot>*mode* is set to kXTreeBehaviorDefault.

I3DShTreeElement::GetFatherGlobalTransform3D(TTransform3D&)

void MCCOMAPI GetFatherGlobalTransform3D(TTransform3D& mat)

<zot>Need method description</zot>.

Parameters **TTreeTransform3D& *mat***
<zot>Need Description</zot>

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::CloneSubTree(ECloneMode, I3DShTreeElement**)

MCCOMErr MCCOMAPI CloneSubTree(ECloneMode cloneMode, I3DShTreeElement clone)**

<zot>Need method description</zot>Clones part of a tree element.

Parameters **ECloneMode** *cloneMode*
Specifies whether to clone with or without animation information.

I3DShTreeElement** *clone*
<zot>Need Description</zot>

I3DShTreeElement::GetUndoInfo(TTreeUndoInfo&)

void MCCOMAPI GetUndoInfo(TTreeUndoInfo& undoInfo)

<zot>Need method description</zot>Gets information to undo changes.

Parameters **TTreeUndoInfo&** *undoInfo*
<zot>Need Description</zot>

I3DShTreeElement::SetFromUndoInfo(const TTreeUndoInfo&, const TTreeUndoInfo*)

void MCCOMAPI SetFromUndoInfo(const TTreeUndoInfo& toInfo, const TTreeUndoInfo* fromInfo)

<zot>Need method description</zot>.

Parameters **TTreeUndoInfo&** *undoInfo*
<zot>Need Description</zot>

const TTreeUndoInfo& *fromInfo*
<zot>Need Description</zot>

I3DShTreeElement::GetTopIfConstrained(I3DShTreeElement**)

MCCOMErr MCCOMAPI GetTopIfConstrained(I3DShTreeElement treeElement)**

<zot>Need method description</zot>If a constraint is applied, returns the top of the tree element.

Parameters **I3DShTreeElement**** *treeElement*
<zot>Need Description</zot>

I3DShTreeElement::AddWireFrame(I3DExWireFrame*)

MCCOMErr MCCOMAPI AddWireFrame(I3DExWireFrame* wireFrame)

<zot>Need method description</zot>.

Parameters **I3DExWireFrame*** *wireFrame*
<zot>Need Description</zot>

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::RemoveWireFrame(I3DExWireFrame*)

MCCOMErr MCCOMAPI RemoveWireFrame(I3DExWireFrame* wireFrame)

<zot>Need method description</zot>.

Parameters **I3DExWireFrame*** *wireFrame*
<zot>Need Description</zot>

I3DShTreeElement::WireFrameArrayFree()

MCCOMErr MCCOMAPI WireFrameArrayFree()

<zot>Need method description</zot>Frees the memory used in the array.

I3DShTreeElement::GetWireFrameArray(TMCClassArray<TWire-FrameInfo>&)

MCCOMErr MCCOMAPI GetWireFrameArray(TMCClassArray<TWireFrameInfo>& wireFrameArray)

<zot>Need method description</zot>.

Parameters **TMCClassArray<TWireFrameInfo>&** *wireFrameArray*
<zot>Need Description</zot>

I3DShTreeElement:: IsPickable()

boolean MCCOMAPI IsPickable()

<zot>Need method description</zot>.

I3DShTreeElement:: IsBBoxPickable()

boolean MCCOMAPI IsBBoxPickable()

Return true if the tree element can be selected by its bounding box (primitive instances usually reply no to allow accurate picking).

I3DShTreeElement::RemoveConstraint(I3DShConstraint**)

void MCCOMAPI RemoveConstraint(I3DShConstraint outLink)**

<zot>Need method description</zot>Removes a constraint from a tree element.

Parameters **I3DShConstraint**** *outLink*
<zot>Need Description</zot>

I3DShTreeElement::RestoreConstraint(I3DShConstraint*)

void MCCOMAPI RestoreConstraint(I3DShConstraint* inLink)

©2001-2003 Eovia. All rights reserved.

<zot>Need method description</zot>Restores a constraint to a tree element.

Parameters **I3DShConstraint*** *inLink*
<zot>Need Description</zot>

I3DShTreeElement::GetExists()

boolean MCCOMAPI GetExists()

<zot>Need method description</zot>Gets the animated visibility/existence parameter. Returns True when the parameter exists.

I3DShTreeElement::SetExists(boolean)

void MCCOMAPI SetExists(boolean inExists)

<zot>Need method description</zot>Returns True if it sets the animated visibility/existence parameter.

Parameters **boolean** *inExists*
<zot>Need Description</zot>

I3DShTreeElement::GetIsShown()

boolean MCCOMAPI GetIsShown()

<zot>Need method description</zot>Gets the non-animated visibility parameter in the 3D views.

I3DShTreeElement::SetIsShown(boolean)

void MCCOMAPI SetIsShown(boolean inIsShown)

<zot>Need method description</zot>Sets the non-animated visibility parameter in the 3D views.

Parameters **boolean** *inIsShown*
<zot>Need Description</zot>

I3DShTreeElement::SetShadowCasting(boolean)

MCCOMErr MCCOMAPI SetShadowCasting(boolean shadowCasting)

<zot>Need method description</zot>Returns True if shadows are cast.

Parameters **boolean** *shadowCasting*
<zot>Need Description</zot>

I3DShTreeElement::SetShadowCatching(boolean)

MCCOMErr MCCOMAPI SetShadowCatching(boolean shadowCatching)

<zot>Need method description</zot>Returns True if shadows are caught.

Parameters **boolean** *shadowCatching*
<zot>Need Description</zot>

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement::GetShadowCasting()

boolean MCCOMAPI GetShadowCasting()

<zot>Need method description</zot>Gets whether shadows are cast.

I3DShTreeElement::GetShadowCatching()

boolean MCCOMAPI GetShadowCatching()

<zot>Need method description</zot> Gets whether shadows are caught.

I3DShTreeElement::GetAnimateTree()

boolean MCCOMAPI GetAnimateTree()

<zot>Need method description</zot>

I3DShTreeElement::SetAnimateTree(boolean)

void MCCOMAPI SetAnimateTree(boolean animateTree)

<zot>Need method description</zot>

Parameters **boolean** *animateTree*

<zot>Need Description</zot>

I3DShTreeElement::ClearInstanceCache()

void MCCOMAPI ClearInstanceCache()

Clears the renderable cache that is stored at the Tree Element level.

©2001-2003 Eovia. All rights reserved.

9.19 I3DShTweener Interface

Family ID: ‘twee’

Interface ID: IID_I3DShTweener

Interface file: I3DShTweener.h

Description

The **I3DShTweener** interface is the wrapper for an I3DExTweener. Tweeners are 3D Components that describe the interpolation between key frames.

Method Index

“I3DShTweener::Tween(I3DShKeyFrame*, int32, I3DShTweenerChainLink)”
“I3DShTweener::Edit(I3DShTweener**,I3DShTweenerChainLink*)”
“I3DShTweener::SetLink(I3DShTweenerChainLink*)”
“I3DShTweener::GetLinkNoAddRef()”

I3DShTweener::Tween(I3DShKeyFrame*, int32, I3DShTweenerChainLink)

MCCOMErr MCCOMAPI Tween(I3DShKeyFrame* res, int32 time, I3DShTweenerChainLink* alink)

Tween() fills a key frame with interpolated data given the time and the chain link to use. The given chain link contains the needed time line information, the previous key even, the tweener, as well as the next key event (contained in the chain link).

Parameter

I3DShKeyFrame* *res*
The specified key frame that will be filled with interpolated data.

int32 *time*
The time line information.

I3DShTweenerChainLink *alink*
The given chain link.

I3DShTweener::Edit(I3DShTweener**,I3DShTweenerChainLink*)

void MCCOMAPI Edit(I3DShTweener tweener, I3DShTweenerChainLink* chainLink)**

Edit() is zot needs description. Check with Greg

Parameter

I3DShTweener** *tweener*
zot needs definition.

I3DShTweenerChainLink* *alink*
zot needs definition.

©2001-2003 Eovia. All rights reserved.

I3DShTweener::SetLink(I3DShTweenerChainLink*)

voidMCCOMAPISetLink(I3DShTweenerChainLink* inLink)

SetLink() is used to access a chain link with a tweener.

Parameter **I3DShTweenerChainLink*** *inlink*
zot needs definition.

I3DShTweener::GetLinkNoAddRef()

I3DShTweenerChainLink* MCCOMAPIGetLinkNoAddRef()

GetLinkNoAddRef() is used to access a chain link with a tweener. Use this function with caution because GetLink can return a NULL pointer if the tweener is not the one on the time line.

©2001-2003 Eovia. All rights reserved.

9.20 I3DShTweenerChainLink Interface

Interface ID: IID_I3DShTweenerChainLink

Interface file: I3DShTweener.h

Description

The **I3DShTweenerChainLink** interface is the key data structure for animation. Tweeners are 3D Components that describe the interpolation between key frames. Each track in the time line is made of a two-way linked list of !3DShTweenerChainLink. I3DShTweeenerChainLink corresponds to a key event in the time line. It also contains with the tweeener immediately following, and a link to the next I3DShTweenerChainLink.

Method Index

"I3DShTweenerChainLink::Clone(I3DShTweenerChainLink**, I3DShParamTimeLine*)"
"I3DShTweenerChainLink::CopyData(I3DShTweenerChainLink*)"
"I3DShTweenerChainLink::GetTweenerNoAddRef(boolean, boolean)"
"I3DShTweenerChainLink::GetTweener(I3DShTweener**, boolean, boolean)"
"I3DShTweenerChainLink::SetTweener(I3DShTweener*)"
"I3DShTweenerChainLink::GetKeyFrameNoAddRef()"
"I3DShTweenerChainLink::GetKeyFrame(I3DShKeyFrame**)"
"I3DShTweenerChainLink::SetKeyFrame(I3DShKeyFrame**)"
"I3DShTweenerChainLink::GetTime()"
"I3DShTweenerChainLink::SetTime(MicroTick)"
"I3DShTweenerChainLink::GetNextNoAddRef()"
"I3DShTweenerChainLink::GetNext(I3DShTweenerChainLink**)"
"I3DShTweenerChainLink::SetNext(I3DShTweenerChainLink*)"
"I3DShTweenerChainLink::GetPrevNoAddRef()"
"I3DShTweenerChainLink::GetPrev(I3DShTweenerChainLink**)"
"I3DShTweenerChainLink::SetPrev(I3DShTweenerChainLink*)"
"I3DShTweenerChainLink::GetFatherNoAddRef()"
"I3DShTweenerChainLink::RemoveFromChain()"
"I3DShTweenerChainLink::InsertAfter(I3DShTweenerChainLink*)"
"I3DShTweenerChainLink::OffsetLink(MicroTick)"
"I3DShTweenerChainLink::GetTimeInfo(TTimeInfo&)"
"I3DShTweenerChainLink::IsTweenerSelected()"
"I3DShTweenerChainLink::IsKeyFrameSelected()"
"I3DShTweenerChainLink::SelectKeyFrame(boolean)"
"I3DShTweenerChainLink::SelectTweener(boolean)"

I3DShTweenerChainLink::Clone(I3DShTweenerChainLink**, I3DShParamTimeLine*)

MCCOMErr MCCOMAPI Clone(I3DShTweenerChainLink outLink, I3DShParamTimeLine*inFatherTimeLine)**

Call **Clone()** to get a clone of a tweener chain link.

Parameter **I3DShTweenerChainLink**** *outLink*
Contains a brand new clone of *inFatherTimeLine*.

©2001-2003 Eovia. All rights reserved.

I3DShParamTimeLine* *inFatherTimeLine*
Contains the object to clone.

I3DShTweenerChainLink::CopyData(I3DShTweenerChainLink*)

MCCOMErrMCCOMAPI CopyData(I3DShTweenerChainLink* inDest)

CopyData() copies a tweener chain link. Overwrites the data in the already existing *inDest*.

Parameter **I3DShTweenerChainLink*** *inDest*
A pointer to a tweener chain link.

I3DShTweenerChainLink::GetTweenerNoAddRef(boolean, boolean)

I3DShTweener*MCCOMAPI GetTweenerNoAddRef(boolean inDefaultIfNone = false, boolean inCreateIfNone = false)

GetTweenerNoAddRef() is used to access a tweener. or to create one if none exists.

Use this function to avoid doing an unnecessary AddRef. Refer to Creating COM objects in the Overview to learn more.

Parameter **boolean*** *inDefaultNone*
inDefaultNone is set to false. If *inDefaultNone* is set to true, it will create a default tweener if none already exists.

boolean* *inCreateIfNone*
inCreateIfNone is set to false. <**zot**> needs more description.

I3DShTweenerChainLink::GetTweener(I3DShTweener**, boolean, boolean)

MCCOMErrMCCOMAPI GetTweener(I3DShTweener outTweener, boolean inDefaultIfNone = false,booleaninCreateIfNone = false)**

GetTweener() is used to access a tweener, or to create one if none exists.

Parameter **I3DShTweener**** *outTweener*
outTweener is a pointer to the output tweener.

boolean* *inDefaultNone*
inDefaultNone is set to false. If *inDefaultNone* is set to true, it will create a default tweener if none already exists and store it in the I3DShTweenerChainLink.

boolean* *inCreateIfNone*
inCreateIfNone is set to false. <**zot**> needs more description.

I3DShTweenerChainLink::SetTweener(I3DShTweener*)

MCCOMErrMCCOMAPI SetTweener(I3DShTweener* inTweener)

©2001-2003 Eovia. All rights reserved.

SetTweener() is used to store a tweener in the I3DShTweenerChainLink.

Parameter **I3DShTweener*** *inTweener*
inTweener is a pointer to the tweener to be stored.

I3DShTweenerChainLink::GetKeyFrameNoAddRef()

I3DShKeyFrame* MCCOMAPI GetKeyFrameNoAddRef()

GetTweenerNoAddRef() is used to access key frame data.

Use this function to avoid doing an unnecessary AddRef.

I3DShTweenerChainLink::GetKeyFrame(I3DShKeyFrame**)

MCCOMErr MCCOMAPI GetKeyFrame(I3DShKeyFrame outKeyFrame)**

GetKeyFrame() is used to access key frame data.

Parameter **I3DShKeyFrame**** *outKeyFrame*
outKeyFrame is <**zot**> needs more description.

I3DShTweenerChainLink::SetKeyFrame(I3DShKeyFrame**)

MCCOMErrMCCOMAPI SetKeyFrame(I3DShKeyFrame* inKeyFrame)

SetKeyFrame() is used to access key frame data. <**zot**> needs more description.

Parameter **I3DShKeyFrame**** *inKeyFrame*
inKeyFrame is <**zot**> needs more description.

I3DShTweenerChainLink::GetTime()

MicroTickMCCOMAPI GetTime()

GetTime() is used to access key event time.

I3DShTweenerChainLink::SetTime(MicroTick)

MCCOMErrMCCOMAPI SetTime(MicroTick inTime)

SetTime() is used to access key event time.

Parameter **MicroTick** *inTime*
inTime is <**zot**> needs more description.

I3DShTweenerChainLink::GetNextNoAddRef()

I3DShTweenerChainLink*MCCOMAPI GetNextNoAddRef()

©2001-2003 Eovia. All rights reserved.

GetNextNoAddRef() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Use this function to avoid doing an unnecessary AddRef.

I3DShTweenerChainLink::GetNext(I3DShTweenerChainLink**)

MCCOMErrMCCOMAPI GetNext(I3DShTweenerChainLink outNext)**

GetNext() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Parameter **I3DShTweenerChainLink**** *outNext*
outNext is <**zot**> needs more description.

I3DShTweenerChainLink::SetNext(I3DShTweenerChainLink*)

MCCOMErr MCCOMAPI SetNext(I3DShTweenerChainLink* inNext)

GetNext() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Parameter **I3DShTweenerChainLink*** *inNext*
inNext is <**zot**> needs more description.

I3DShTweenerChainLink::GetPrevNoAddRef()

I3DShTweenerChainLink*MCCOMAPI GetPrevNoAddRef()

GetPrevNoAddRef() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Use this function to avoid doing an unnecessary AddRef.

I3DShTweenerChainLink::GetPrev(I3DShTweenerChainLink**)

MCCOMErrMCCOMAPI GetPrev(I3DShTweenerChainLink outPrev)**

GetPrev() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Parameter **I3DShTweenerChainLink**** *outPrev*
outPrev is <**zot**> needs more description.

I3DShTweenerChainLink::SetPrev(I3DShTweenerChainLink*)

MCCOMErr MCCOMAPI SetPrev(I3DShTweenerChainLink* inPrev)

GetPrev() is used to navigate the chain links. A track of time line is composed of a 2-way linked list of I3DShTweenerChainLink.

Parameter **I3DShTweenerChainLink*** *inPrev*
inPrev is <**zot**> needs more description.

©2001-2003 Eovia. All rights reserved.

I3DShTweenerChainLink::GetFatherNoAddRef()

I3DShParamTimeLine*MCCOMAPI GetFatherNoAddRef()

GetFatherNoAddRef() is used to retrieve the time line track to whom the chain link belongs.

Use this function to avoid doing an unnecessary AddRef.

I3DShTweenerChainLink::RemoveFromChain()

MCCOMErrMCCOMAPI RemoveFromChain()

RemoveFromChain() is used to remove a link from the chain. It also links the previous and next links together.

I3DShTweenerChainLink::InsertAfter(I3DShTweenerChainLink*)

MCCOMErrMCCOMAPI InsertAfter(I3DShTweenerChainLink* inAfter)

InsertAfter() is used to insert a link after a specific link (inAfter).

Parameter **I3DShTweenerChainLink*** *inAfter*
inAfter is a pointer to the specific link.

I3DShTweenerChainLink::OffsetLink(MicroTick)

MCCOMErrMCCOMAPI OffsetLink(MicroTick inDeltaTime)

OffsetLink() is used to increment/decrement the time of a chain link.

Parameter **MicroTick** *inDeltaTime*
inDeltaTime is a pointer to delta of time for the offset.

I3DShTweenerChainLink::GetTimeInfo(TTimeInfo&)

MCCOMErrMCCOMAPI GetTimeInfo(TTimeInfo& outTimeInfo)

GetTimeInfo() is used to differenciate key frames that completely overlap and have the same time information.

Parameter **TTimeInfo&** *outTimeInfo*
outTimeInfo is a pointer to the time and index of a key event.

I3DShTweenerChainLink::IsTweenerSelected()

boolean MCCOMAPI IsTweenerSelected()

IsTweenerSelected() returns True if the tweener is selected.

I3DShTweenerChainLink::IsKeyFrameSelected()

boolean MCCOMAPI IsKeyFrameSelected()

IsKeyFrameSelected() returns True if the key frame is selected.

©2001-2003 Eovia. All rights reserved.

I3DShTweenerChainLink::SelectKeyFrame(boolean)

MCCOMErr MCCOMAPI SelectKeyFrame(boolean select)

SelectKeyFrame() is used to select a key frame.

Parameter **boolean** *select*
select <**zot**> needs more description.

I3DShTweenerChainLink::SelectTweener(boolean)

MCCOMErr MCCOMAPI SelectTweener(boolean select)

SelectTweener() is used to select a tweener.

Parameter **boolean** *select*
select <**zot**> needs more description.

©2001-2003 Eovia. All rights reserved.

9.21 I3DShWireFrame Interface

Interface ID: IID_I3DShWireFrame

Interface file: I3DWireFrame.h

Description

The **I3DShWireFrame** interface is used for accessing wire frames.

Method Index

"I3DShWireFrame::AllocateFMesh(int32, int32, int32)"
"I3DShWireFrame::GetFMesh(FacetMesh**)"
"I3DShWireFrame::AllocateWireFrame(int16, int16)"
"I3DShWireFrame::GetWireFrame(WFData&)"
"I3DShWireFrame::Connect(int16, int16)"
"I3DShWireFrame::SetPoint(int16, EWFHandleType, const TVector3&)"
"I3DShWireFrame::Transform(const TTreeProjectionTransform&)"
"I3DShWireFrame::TestHit(TVector2&, real)"
"I3DShWireFrame::AddRenderables(TMCPtrArray<I3DShRenderable>&)"
"I3DShWireFrame::GetColor()"
"I3DShWireFrame::SetColor(const TMCColorRGB&)"
"I3DShWireFrame::ExcludeFromZBuffer(boolean, boolean)"
"I3DShWireFrame::SetSelectedState(const boolean)"
"I3DShWireFrame::ExcludeFromProjection(boolean)"

I3DShWireFrame::AllocateFMesh(int32, int32, int32)

MCCOMErrMCCOMAPI AllocateFMesh(int32 numFacets, int32 numVertices, int32 facetStreamSize)

Call **AllocateFMesh()** to allocate memory for a facet mesh.

Parameter

int32 *numFacets*
Specifies the number of facets in the mesh.

int32 *numVertices*
Specifies the number of vertices in the mesh.

int32 *facetStreamSize*
Specifies the <Zot>.

I3DShWireFrame::GetFMesh(FacetMesh**)

MCCOMErrMCCOMAPI GetFMesh(FacetMesh aMesh)**

GetFMesh() gets a facet mesh.

Parameter

FacetMesh** *aMesh*
A pointer to a facet mesh.

©2001-2003 Eovia. All rights reserved.

I3DShWireFrame::AllocateWireFrame(int16, int16)

MCCOMErrMCCOMAPI AllocateWireFrame(int16 numPoints, int16 numConnections)

AllocateWireFrame() is used to allocate memory for a wire frame

Parameter

int16 *numPoints*
Specifies the number of points in the wire frame.

int16 *numConnections*
Specifies the number of connections in the wire frame.

I3DShWireFrame::GetWireFrame(WFData&)

MCCOMErrMCCOMAPI GetWireFrame(WFData& aWireFrame)

GetWireFrame() is used to access a wire frame

Parameter

WFData& *aWireFrame*
The data of the specfied wire frame.

I3DShWireFrame::Connect(int16, int16)

MCCOMErrMCCOMAPI Connect(int16 point, int16 from, int16 to)

Connect() is used to connect two points. <zot needs more>

Parameter

int16 *from*
The starting point to connect.

int16 *to*
The ending point to connect.

I3DShWireFrame::SetPoint(int16, EWFHandleType, const TVector3&)

MCCOMErrMCCOMAPI SetPoint(int16 point, EWFHandleType type, const TVector3& loc)

SetPoint() is used to set a point in the wire frame in a specified location.

Parameter

int16 *point*
The specified point.

EWFHandleType *type*
The type of point. <zot verify>

const TVector3& *loc*
The location of the point.

©2001-2003 Eovia. All rights reserved.

I3DShWireFrame::Transform(const TTreeProjectionTransform&)

MCCOMErrMCCOMAPI Transform(const TTreeProjectionTransform& trans)

Transform() is used to <zot do what?>.

Parameter **const TTreeProjectionTransform&** *trans*
trans is <**zot**> needs more description.

I3DShWireFrame::TestHit(TVector2&, real)

shortMCCOMAPI TestHit(const TVector2& point, real error)

TestHit() is used to <**zot**> needs more description.

Parameter **TVector2&** *point*
point is <**zot**> needs more description.

real *error*
error is <**zot**> needs more description.

I3DShWireFrame::AddRenderables(TMCPtrArray<I3DShRenderable>&)

MCCOMErrMCCOMAPI AddRenderables(TMCPtrArray<I3DShRenderable>& renderables)

AddRenderables() is used to <zot needs description>.

Parameter **TMCPtrArray<I3DShRenderable>&** *renderables*
renderables is <**zot**> needs more description.

I3DShWireFrame::GetColor()

TMCColorRGBMCCOMAPI GetColor()

GetColor() returns the color of the wire frame.

I3DShWireFrame::SetColor(const TMCColorRGB&)

MCCOMErrMCCOMAPI SetColor(const TMCColorRGB& color)

SetColor() is used to set the color of the wire frame.

Parameter **const TMCColorRGB&** *color*
color is the new color value for the wire frame.

©2001-2003 Eovia. All rights reserved.

I3DShWireFrame::ExcludeFromZBuffer(boolean, boolean)

MCCOMErrMCCOMAPI ExcludeFromZBuffer(boolean excludeLines, boolean excludePoints)

ExcludeFromZBuffer() is used to exclude a specified wire frame from being rendered by the ZBuffer renderer.

Parameter

boolean *excludeLines*
Specifies whether the lines are rendered.

boolean *excludePoints*
Specifies whether the points are rendered.

I3DShWireFrame::SetSelectedState(const boolean)

MCCOMErrMCCOMAPI SetSelectedState(const boolean selected)

SetSelectedState() is used to set whether the wire frame is selected or not.

Parameter

const boolean *selected*
selected is true when the wire frame is selected.

I3DShWireFrame::ExcludeFromProjection(boolean)

MCCOMErrMCCOMAPI ExcludeFromProjection(boolean exclude)

ExcludeFromProjection() is used to exclude a specific wire from from displaying on the projection planes of the working box.

Parameter

boolean *exclude*
exclude is true when the wire frame is excluded.

©2001-2003 Eovia. All rights reserved.

Chapter 10 - DataBase Overview

Carrara stores all the data of currently open documents in a global database. Third party extensions can access this database to access or modify these data. The database contains a list of scenes. To each open document corresponds one scene.

10.1 Scene

A *Scene* is mostly comprised of 2 structures:

- An *Object List*, that contains all the *Objects* used in the scene.
- A *Scene Tree* that contains the hierarchical structure of the scene. This defines the positions and relationships between *Tree Elements*. These tree elements can be object instances, light sources, cameras and groups.

Objects

Objects contains the geometric information defining its surface regardless of its position. It also contains a reference to its shading information. The Objects used in a Scene are located in the Objects tab of the Browser window in Carrara.

An Object (I3DShObject) can be of two types: a *Primitive* (I3DShPrimitive) or a Scene (I3DShScene).

Primitives are the basic objects that are located in the toolbar as icons. There are different classes of primitives. Carrara defines a certain number of default internal primitives that can be used by extensions. Those primitives will always be there. They are the Spline Primitive, the Polygon Array, the Patch Array, the Polygon List (list of triangular facets), the Flat Primitive, the 3D Text, etc. Some other primitives are external primitives (coded in regular extensions) like the Cube, the Sphere, the Isocaedra, etc. Third party developers can define new classes of primitives.

A Scene is also considered an Object, because it can be instanciated in another Scene, like if it was a complex object. This is sometimes called “Scene Instancing”.

Scene Tree and Tree Elements

The scene tree is a hierarchical structure containing *Tree Elements* (I3DShTreeElement).

There are four types of tree elements: Object Instances (I3DShInstance), Light Sources (I3DShLightsource), Cameras (I3DShCamera) and Groups (I3DShGroup). Those elements are detailed below. All these elements have 2 things in common:

- a position in 3D space (orientation, location, scaling, etc...)
- a name

©2001-2003 Eovia. All rights reserved.

Object Instances

An object instance is a tree element that has a reference to an Object in the object list. Therefore there can be multiple instances in a scene of a same object. This object can be either a Primitive or another Scene.

Shading can be “overriden” at the instance level, thus surpassing any shading information the original object may have.

Lights

A *Light Source* is a tree element. There can be as many lights as needed in a scene. Most Light Sources are implemented as Extensions.

Cameras

A *Camera* is another type of tree element. Like lights, there can be many cameras in a scene. One of them will be the *Rendering Camera*, i.e. the camera the final image will be rendered from. Most Cameras are implemented as Extensions.

Groups

A *Group* is a tree element used to gather other tree elements together. It can be open or closed.

Scene Instance vs regular Instances

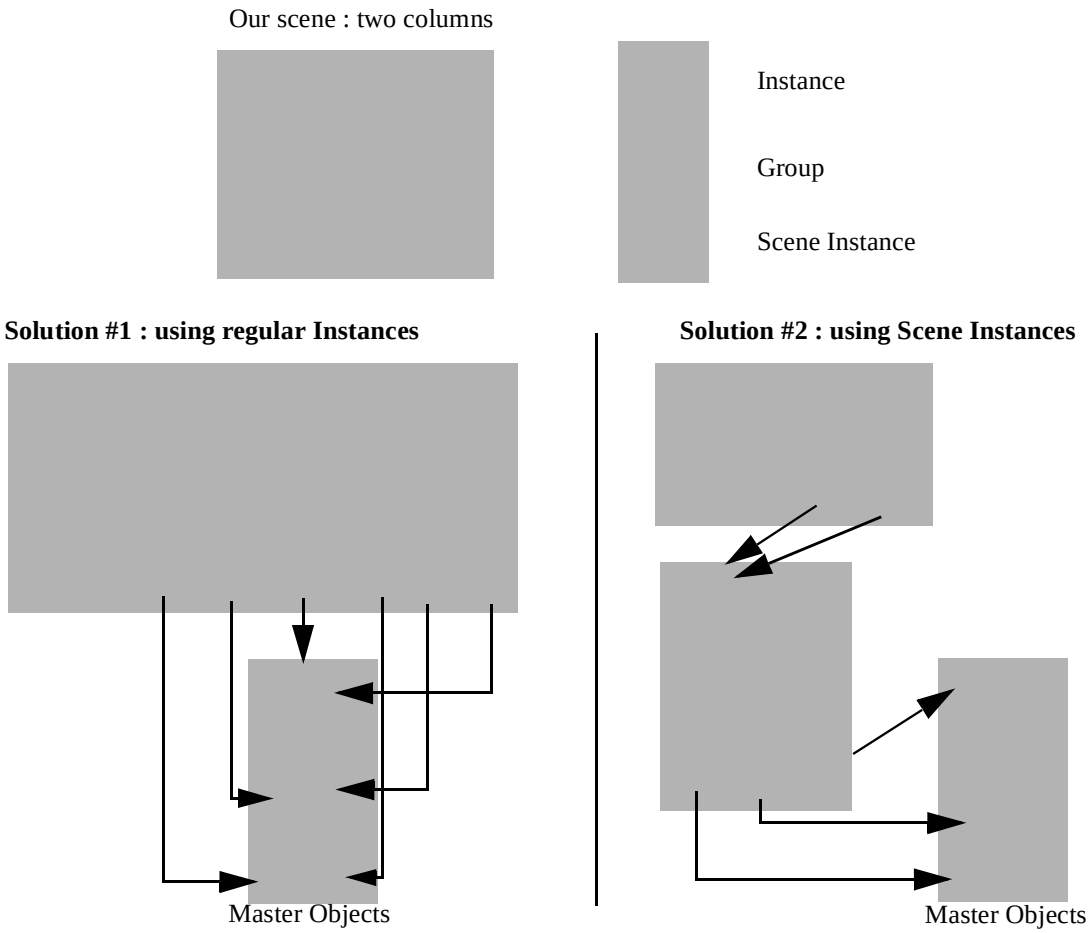
In order to make clearer these two notions, let’s consider a basic example. We are supposed to model a scene with two columns (see fig. 1).

First using regular instances, we can make the parallel with a class in C++. Basically, if we change the color or the shape of the Column Master Object, these two instances will change as well.

However, we are not able to “factorize” the position or orientation. If we want to move the base of the column, it will not move the base of the other column.

As opposed to the previous situation, by using Scene Instances, moving the base of one column results in moving the other one. Indeed, in that case, the transformations are automaticaly updated.

©2001-2003 Eovia. All rights reserved.



zot recreate graphic

10.2 Coordinate Systems

Global Coordinate System

When you look at the Assemble room in Carrara, the axis of the Global Coordinate System are organized like this:

©2001-2003 Eovia. All rights reserved.

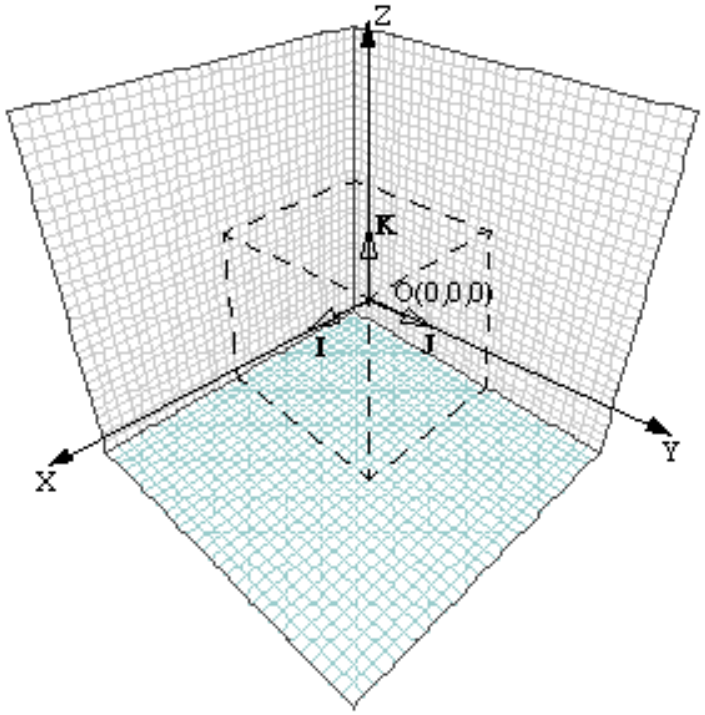


Figure 1. The Global Coordinate System

The projection of the (0, 0, 0) origin falls in the middle of each plane of the Working Box (the origin is *not* the far corner of the box). The **I**, **J**, **K** vectors are the unit vectors of the X, Y and Z axis.

Working Box Coordinate System

The Working Box Coordinate System is defined by the position of the Working Box. As the Working Box can be moved and rotated in Carrara, this coordinate system can be useful in some complex scenes. However, this system is never used when dealing with Extensions, so we will just mention it here.

Local Coordinate System (or Object Coordinate System)

The Local Coordinate System (also sometimes called the Object Coordinate System) define the system of an object or a group. Its axis, origin and scale depends on the positioning of the object in space.

©2001-2003 Eovia. All rights reserved.

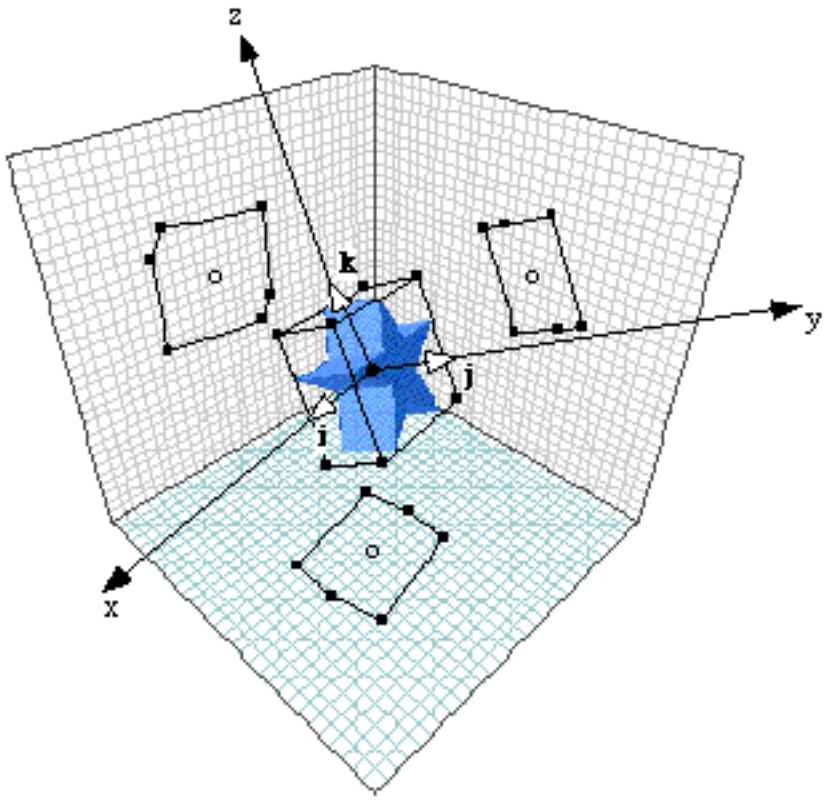


Figure 2. The Local Coordinate System of an object in a scene

The **i**, **j** and **k** vectors are the unit vectors of the x, y and z axis.

Screen Coordinate System

When a rendering occurs, the 3D data is projected onto the screen through the rendering camera. The axis of the 3D coordinate system attached to the screen is like this:

©2001-2003 Eovia. All rights reserved.

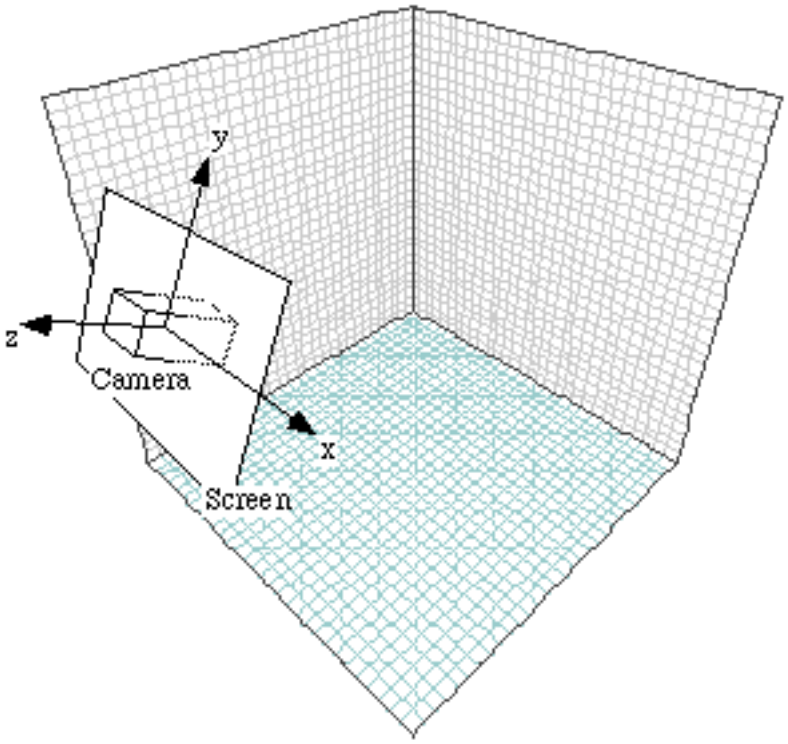


Figure 3. The Screen Coordinate System

As one can see, the objects seen by the camera have a negative z coordinate. The screen and the Screen Coordinate System are centered on the camera's center.

The Screen Coordinate System versus the Camera's Coordinate System

There is a slight difficulty here. The camera's aim is along the y axis of its transformation. As a result, here is the Local Coordinate System of the camera in the previous figure:

©2001-2003 Eovia. All rights reserved.

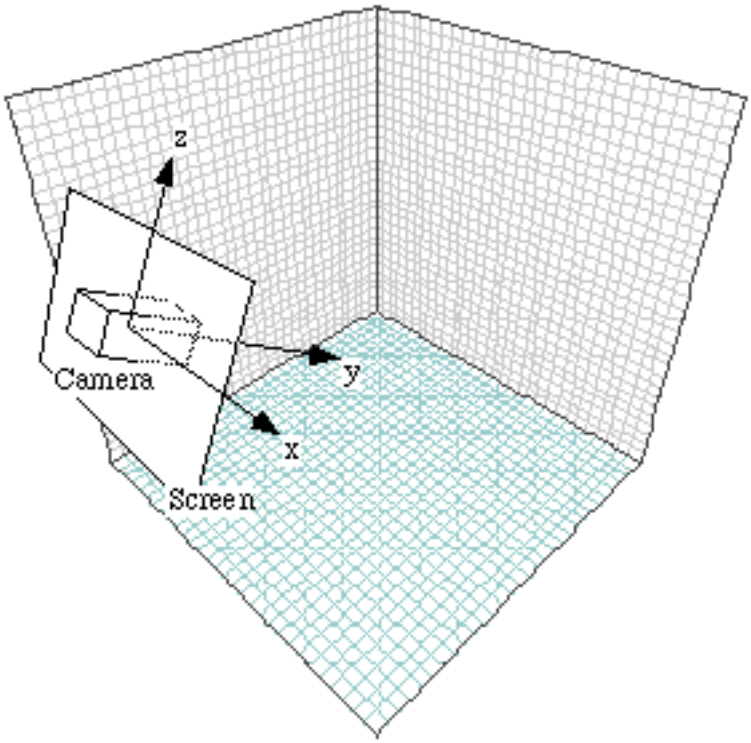


Figure 4. The camera's Local Coordinate System

You do not need to worry too much about this difference, because transformations calculated by the 3D Shell takes this into account. It is important only if you intend to position a camera in a scene.

The screen pixels space

The Screen Pixels Space is the actual Pixels coordinate system used to render the final image. Its unit system is in Points, as opposed to all other coordinate systems which are in 3D units. As a result:

1 3D unit = 288 points

because 1 inch = 72 points. More on the units business is covered later.

It is oriented with its vertical axis going down, and its (0, 0) origin in the top left corner of the image:

©2001-2003 Eovia. All rights reserved.

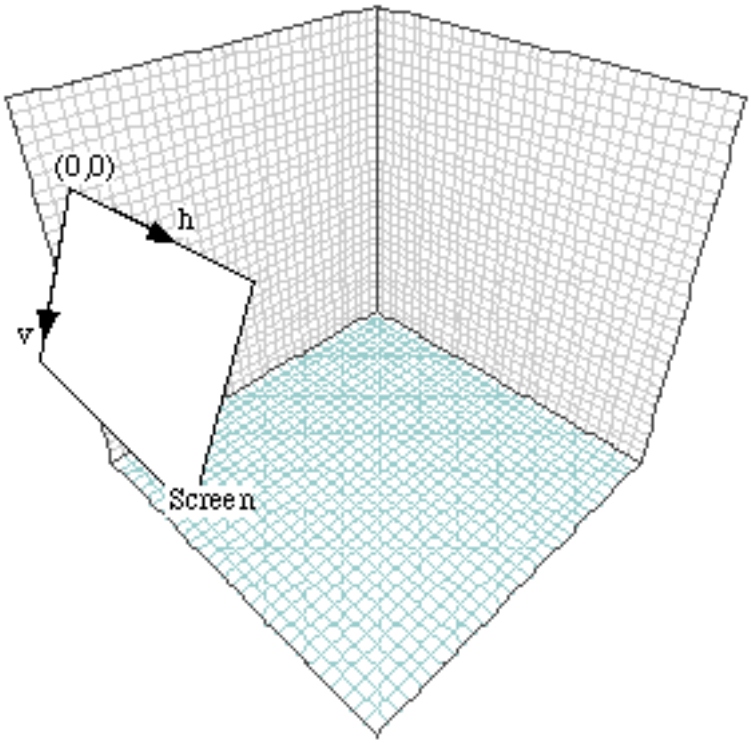


Figure 5. The Screen Pixels Space

10.3 Geometry

Geometric data type: 32-bit floating point (float)

All geometric data is in 32-bit floating point format:

```
typedef float NUM3D;
```

Units System

The units system used in Carrara is defined as follows:

1 3D unit = 4 inches

All geometric data uses the 3D units. The various units shown by Carrara in the Properties tray are just handled at the user interface level.

The advantage of using a fixed system like this is that there is no problem of data conversion into different units.

©2001-2003 Eovia. All rights reserved.

Tree Elements Transformation

Each tree element (object instance, light source, group or camera) is defined relatively to its parent in the tree. Depending of the API procedure you call, you will get the transformation parameters that define the attitude of the object in space in one format or another. Whatever callback you use, you will get the following data:

- A 3x3 rotation matrix **R** (or the 3 vectors that define it)
- A translation vector **T**
- A uniform scaling factor **s**

To transform a point from the Local Coordinate System of this tree element into the Coordinate System of its parent, the following formula is used:

$$M = s[R]m + T$$

with m the point in local coordinates, and M the same point in the parent's coordinates.

To make it easier to read, here is the same equation in expanded format for each x, y and z coordinate:

$$M_x = s(R_{ix} \times m_x + R_{jx} \times m_y + R_{kx} \times m_z) + T_x$$

$$M_y = s(R_{iy} \times m_x + R_{jy} \times m_y + R_{ky} \times m_z) + T_y$$

$$M_z = s(R_{iz} \times m_x + R_{jz} \times m_y + R_{kz} \times m_z) + T_z$$

With :

- $m(m_x, m_y, m_z)$ point in local coordinates
- $M(M_x, M_y, M_z)$ point in parent's coordinates
- $T(T_x, T_y, T_z)$ position of the tree element origin in parent's coordinates
- s uniform scaling factor
- R rotation matrix (see below)

More on the rotation matrix:

If you consider the 3 **i**(i_x, i_y, i_z), **j**(j_x, j_y, j_z) and **k**(k_x, k_y, k_z) vectors of the Local Coordinates System, you can write easily the **R** matrix by putting each vector in each column like this:

$$[R] = \begin{bmatrix} i_x & j_x & k_x \\ i_y & j_y & k_y \\ i_z & j_z & k_z \end{bmatrix}$$

For additional information on transformation matrices and Tree Transformations, see the Data Structure chapters in the Reference.

©2001-2003 Eovia. All rights reserved.

Geometry basics

When you deal with objects, there are several concepts to define that are related to the object's shape.

Surface Point

Carrara deals with surfaces, not volumes. An object is made of surfaces that can be arbitrary complex. A point P on a surface is made of its x, y and z coordinates in the Object Coordinate System.

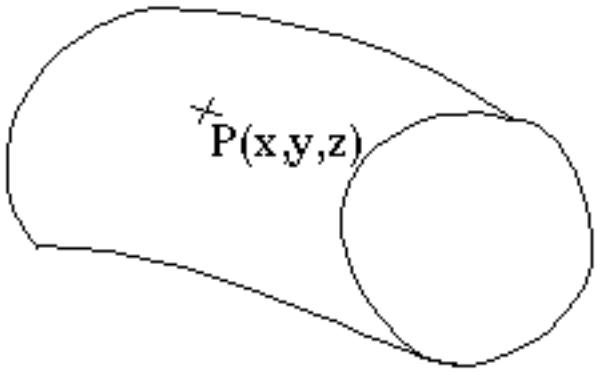


Figure 6. A point on a surface

Surface Normal

The Surface Normal at a point is the vector perpendicular to a plan that would be tangent to the surface at that point. The Normal is very important in 3D computer graphics, because it is heavily used for shading (especially in Phong and Ray Tracing shadings). Normals are usually normalized (their length is equal to 1), and always point outward.

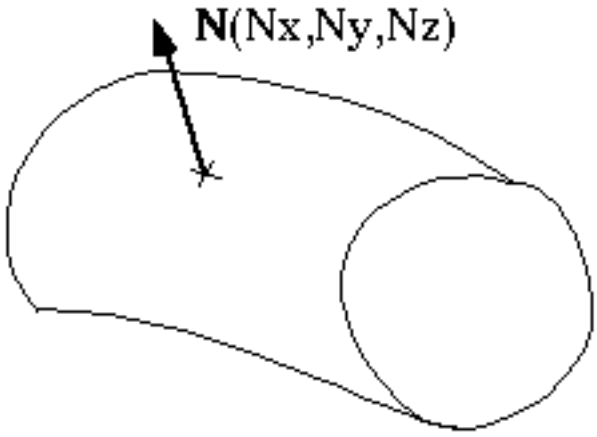


Figure 7. A Normal to the surface

Note that :

$$N_x^2 + N_y^2 + N_z^2 = 1$$

©2001-2003 Eovia. All rights reserved.

UV Space (Texture Coordinate System)

Wrapping a texture (such as a texture map) on an object surface involves finding the correspondence (the "mapping") between a 2D space (the image) and a 3D space (the object surface). The typical rendering question is: "I have a point P(x, y, z) on my surface, where should I look up in the texture map ?".

When one deals with simple shapes (sphere, cube, cylinders, etc.), it is easy to find a mapping. When one deals with complex objects (list of facets or patches), the matter becomes much more difficult. To solve this, most packages use a technique called "projection mapping": an intermediate imaginary surface surrounding the object is used, and the texture is placed on it. Then the surface color at a point is calculated by using the part of the texture that the point is facing.

The problem with this is that you can get awkward results when the object is very unlike the intermediate surface. Unwanted deformation is a typical problem: you want the texture to shrink or enlarge only where the object surface does.

Parametric mapping - a better solution

The Carrara Spline modeler is in fact capable of generating what we call "UV spaces", and the architecture of Carrara can keep this information down to the facets or bicubic patches level, in order to allow direct mapping (also called "Parametric Mapping").

The idea is this: because objects are built by combining 2D curves, it is possible to generate a 2D space that will behave topologically like the object surface.

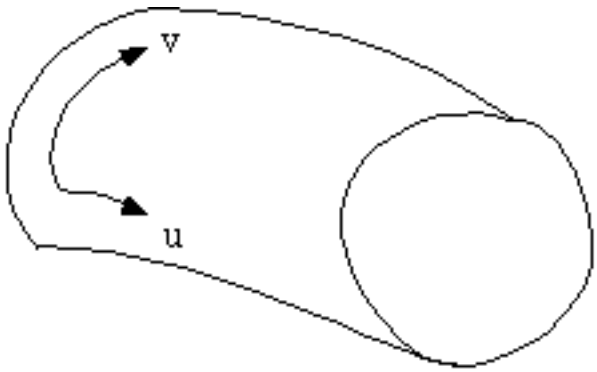


Figure 8. A typical u,v space on the side surface of an object

Every time you deal with a 3D point on the surface, you will be able to calculate the u,v coordinates for this point (provided that the object supports UV spacing). Say you need to know if the point is covered by a layer. The location of the layer is known by the u,v coordinates of its corners. The test becomes a simple 2D test: "are the u,v coordinates of the point inside a 2D rectangle ?".

Most of the time you get the u,v values from the 3D Shell. The only case where you have to generate them is when you develop a geometric primitive extension.

If the object has some surface discontinuities, then several u,v spaces are generated for the object. For example if your extrude a closed 2D curve, you will get 3 u,v spaces: one for the side surface, one for the back face and one for the front face. This is because it is not possible to have a u,v continuity across these different parts of the object (think of the problem of a napkin on a table). Each UV space has a number (0, 1, 2...), called a **UV Space ID**.

©2001-2003 Eovia. All rights reserved.

Shading

The shading can be applied on an object by giving general properties (color, reflection...) to the object or by mapping a texture on it. There is 4 ways of mapping a texture on an object.

- the parametric mapping,
- the box mapping,
- the cylindrical mapping,
- the spherical mapping.

The parametric mapping is the mapping using the UV Space information as seen in the last section. This is the better way of mapping, but because all objects don't have necessarily a UV Space, four other ways of mapping are provided.

The box mapping, the cylindrical mapping and the sphere mapping are obtained by projecting a texture on a cube, cylinder or a cube on the object.

Facets

There are two kinds of low level geometric data used for rendering, exporting, etc.: **facets** and **bicubic patches**. <zot> this section needs to be rewriten; there are only patches now.

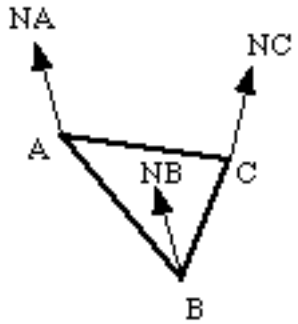


Figure 9. A 3D facet

Facets are triangles. Each vertex contains the (x, y, z) coordinates of the point, the Normal at this point, and the u,v coordinates at this point. The facet also stores the u,v Space ID to which it belongs.

```
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;

typedef struct FACET3D{
  VERTEX3D  fVertices[3];    // The facet three vertices
  short     fUVSpace;        // UV Space ID this facet belongs to
  short     fReserved;       // Reserved - 0
  } FACET3D;
```

Interpolating in a facet

Interpolating a point in a facet is a common exercise in 3D. If you call the vertices A, B and C, their Normals NA, NB, NC, and their u,v values uA, vA, uB, vB, and uC, vC, then:

©2001-2003 Eovia. All rights reserved.

from:

$$P = \alpha A + \beta B + \gamma C$$

you can interpolate:

$$N = \alpha NA + \beta NB + \gamma NC$$

Don't forget to renormalize the normal:

$$N_{nnit} = \frac{N}{\sqrt{N^2}}$$

Likewise:

$$u = \alpha uA + \beta uB + \gamma uC$$
$$v = \alpha vA + \beta vB + \gamma vC$$

Of course, this is an approximation of the real values, but facets are already approximations of the real surface. As long as facets are small enough, this works just fine.

Bicubic Patches

Bicubic patches are more interesting geometric entities. They can describe more complex surfaces than facets, and are more resolution independent. For example Extrusions created by Carrara's Spline modeler generate bicubic patches. Turn on the Wireframe display; what you see are the boundaries of the patches.

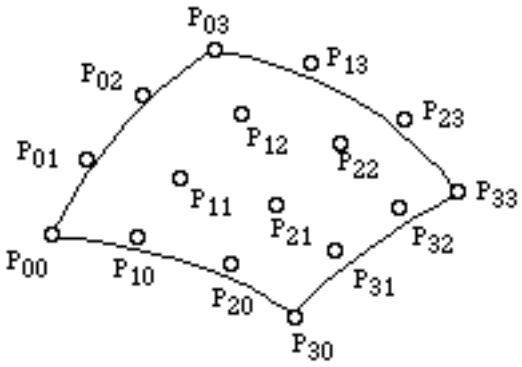


Figure 10. A bicubic patch

A patch is made of 16 3D points. You can think of a bicubic patch as a cubic Bézier curve that has its 4 vertices moving along 4 other Bézier curves.

```
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
```

©2001-2003 Eovia. All rights reserved.

```
short     fReserved;          // Reserved - 0
} PATCH3D;
```

One can think of a bicubic Bézier patch as a Bézier curve moving on 4 other perpendicular Bézier curves. Let's apply this concept to calculate a point on the surface.

The patch can be defined as a parametric surface of two normalized parameters, tu and tv.

$$\text{Bicubic Patch} = S(t_u, t_v)$$
$$0.0 \le t_u \le 1.0$$
$$0.0 \le t_v \le 1.0$$

First consider the 4 Bézier curves defined by C0=(P00, P10, P20, P30), C1=(P01, P11, P21, P31), C2=(P02, P12, P22, P32), and C3=(P03, P13, P23, P33), and calculate on each a point at t_u:

$$P_n(t_u) = P_{0n}(1-t_u)^3 + 3\,P_{1n}(1-t_u)^2 t_u + 3\,P_{2n}(1-t_u)\,t_u^{\,2} + P_{3n}t_u^{\,3}$$

Then calculate the point at tv on the Bézier curve ($P0(t_u)$, $P1(t_u)$, $P2(t_u)$, $P3(t_u)$) by re-using the same formulae as above. This is the result.

Patch normals

A nice thing about patches is that you do not have to worry about storing normals: they are automatically defined by the patch geometry. So this means that you do not have to calculate them for the 3D Shell. For the Mathematics savvy reader, remember, normals are defined as:

$$N(t_u, t_v) = \frac{\partial S}{\partial t_u} \times \frac{\partial S}{\partial t_v}$$

Patch u,v Space

The u and v values are constant along the patch boundaries.

P00, P01, P02, P03: u[0] P00, P10, P20, P30: v[0]

P30, P31, P32, P33: u[1] P03, P13, P23, P33: v[1]

(u, v) values at any point on the patch is calculated from these boundaries values by doing a Bézier interpolation.

©2001-2003 Eovia. All rights reserved.

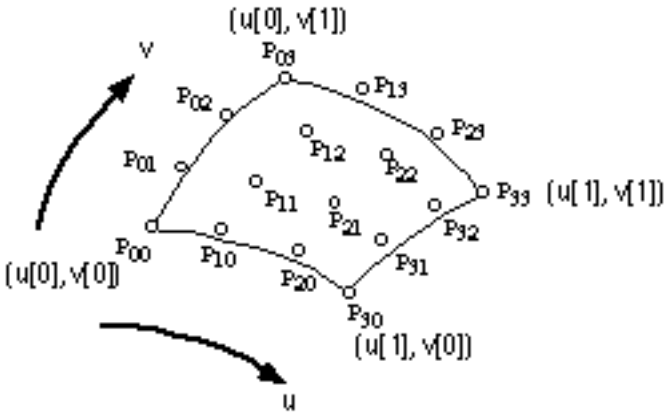


Figure 11. A patch (u,v) Space

To learn more about Bézier bicubic Patches

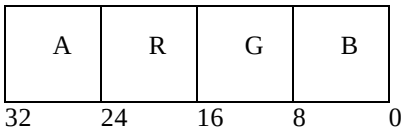
Read the excellent book « Introduction to Computer Graphics » by Folley - Van Dame - , published by Addison-Weisley.

Also take a look at source of the examples of the Carrara SDK, like the Teapot primitive or the patch deformer.

10.4 Color

In Carrara, two ways are used to stock a color : first a COLOR3D (See the Data Structures chapter in Reference Manual), then a long. <zot> TColor3D??

In a long (32 bits), we use this order ARGB, with the higher significant byte as A and the lower significante byte B.



Color in long (32-bits) format.

©2001-2003 Eovia. All rights reserved.

©2001-2003 Eovia. All rights reserved.

Chapter 11 - Index

A

Abort()
I3DExFinalRenderer 88
AboutToCloseMainWindow() 113
Activate() 111, 165
AddPatch 233
AddRenderables 293
AddWireFrame (I3DExWireFrame*) 279
AllocateFMesh 291
AllocateWireFrame 292
Apply(I3DShTreeElement) 105
AskForParams 186
AttachSkeleton()
I3DExSkinable 147
I3DExVertexPrimitive 159

B

BeginGetRenderables (const TRenderableAndTfmArray*&, boolean) 273
BeginProgress 192
BeginRender 250
bicubic patch 307
BlitUIImage 197
BOX3D 32
BuildAlertMessage 196
BuildFacetMesh 235
BuildMenuBar() 111

C

Camera 296
camera 296
Camera Coordinate System 300
CanUndo 178
CanUndo()
I3DExSceneCommand 138
ChangedData
I3DExGeometricPrimitive 124
CheckForUserCancel 201
CheckKeyDown 202
Clone 227, 259, 285
CoCreateInstance 183

©2001-2003 Eovia. All rights reserved.

COLOR3D 38
Connect 292
Coordinate System
    Camera Coordinate System 300
    Global Coordinate System 297
    Local Coordinate System 298
    Object Coordinate System 298
    Working Box Coordinate System 298
Coordinate Systems
    Screen Coordinate System 299
coordinate systems 297
    global coordinate system 297
    local (or object) coordinate system 298
    screen pixels space 301
    UV space 305
    working box coordinate system 298
CopyData (I3DShTweenerChainLink*) 286
CountPatches 234
CreateCompatiblePixelBucket 198
CreateComponent 184
CreateDefaultComponent 184
CreateIUnknown 184
CreateRay()
    I3DExCamera 68

D

DataToWireFrame 164
Deactivate() 111, 165
DeformBBox() 106
DeformFacetMesh() 106
DeformPoint() 106
Dehydrate() 111
DeleteSelectedEdges()
    I3DExVertexPrimitive 160
Destroy() 110
DestroyCompatiblePixelBucket 199
DettachAllJoints()
    I3DExSkinable 147
DirectionFilter()
    I3DExAtmosphericShader 61
    I3DExVolumetricEffect 161
DirectionFilter(const TVector3 ,const TVector3&, TMCColorRGB&) 220
Do 177
Do()
    I3DExSceneCommand 138
DoAlert 195
DoExport 22

©2001-2003 Eovia. All rights reserved.

DoExport()
I3DExExportFilter 80
DoImport 25
DoImport()
I3DExImportFilter 92
DoShade()
I3DExShader 145, 260
DrawRect 249
DrawRect()
I3DExFinalRenderer 89

E

EndGetRenderables() 274
EndProgress 192
ErrorToString 196
ExcludeFromProjection 294
ExcludeFromZBuffer 294
ExtensionDataChanged 18

F

facet 306
FACET3D 49
Filter() 116
FinishDraw 249
FinishDraw()
I3DExFinalRenderer 87
FinishRender()
I3DExTileRenderer 150

G

geometric data type 302
geometry 302
geometry basics 304
Gestalt 194
GetAnimationMethod (I3DShAnimationMethod**) 277
GetAPINumber 194
GetAsciiTranslationTable 199
GetBackdropColor()
I3DExBackdrop 65
GetBackdropColor(const TVector2&,boolean&,const TBBox2D&, TMCColorRGB&) 220
GetBackgroundColor() 63
GetBBox()
I3DExGeometricPrimitive 119
GetBoundingBox 227
GetBoundingBox()
I3DExVertexPrimitive 160

©2001-2003 Eovia. All rights reserved.

GetBufferNeeds() 115, 247
GetBump() 261
GetClasSignature 181, 187
GetClassName 181, 187
GetColor 293
GetColor()
I3DExAmbientLight 59
I3DExShader 145, 261
GetComponentClassInfoArrayByFamilyID 185
GetComponentCount 27
GetComponentData 27
GetConstraint (I3DShConstraint**) 274
GetCreator 201
GetCurrentModifiers 201
GetData (TDataKey, IMCUnknown*) 276
GetDataComponentByID (I3DShDataComponent*, IDType) 275
GetDataComponentByIndex(I3DShDataComponent**, uint32) 274
GetDataComponentsCount () 274
GetDefaultShader
I3DExGeometricPrimitive 124
GetEditor 227
GetEditor () 276
GetEnvironmentColor(const TVector3&, boolean&, TMCColorRGB&) 221
GetExtensionDataBuffer 18
GetFacetBoundingBox 227
GetFamilySignature 181, 187
GetFatherGlobalTransform3D(TTransform3D&) 278
GetFatherNoAddRef() 289
GetFMesh 291
GetFontList 201
GetFreedomRange() 74, 270
GetFullName(TMCString&) 272
GetGlobalTreeTransform3D (TTreeTransform3D&) 278
GetIndPatches 234
GetInstance()
I3DExSkinable 148
GetInstanceArray 226
GetInstanceGlobalReferenceTransform()
I3DExSkinable 148
GetInstancesCount 226
GetJoints()
I3DExSkinable 148
GetKeyFrame (I3DShKeyFrame**) 287
GetKeyFrameNoAddRef () 287
GetLastOpenDocPath 199
GetLastSaveDocPath 199
GetMaxTimeWithoutComputation (I3DShTreeElement*) 105
GetMenuCallBack()

©2001-2003 Eovia. All rights reserved.

I3DExSceneCommand 137
GetModifierByIndex (I3DShModifier**, uint32) 273
GetMotionLimit() 73
GetName 178, 225
GetNbEdges()
I3DExVertexPrimitive 159
GetNbPolygons()
I3DExVertexPrimitive 159
GetNbrFreedom() 73, 270
GetNeededLOD() 107
GetNext (I3DShTweenerChainLink**) 288
GetNextNoAddRef() 287
GetNextSubTile()
I3DExTileRenderer 149
GetObjectID 228
GetOpenDocumentByIndex 199
GetOpenDocumentsCount 199
GetOriginalParametricTextureMap() 262
GetParameter 18, 188
GetParameterCount 188
GetParameterInfo 188
GetParameterType 188
GetPartToRedraw 178
GetPolymesh()
I3DExVertexPrimitive 160
GetPreferredOutput() 144, 262
GetPreferredRoom(IDType&) 112
GetPrev (I3DShTweenerChainLink**) 288
GetPreviewSize 22
GetPreviewSize()
I3DExExportFilter 80
GetPrevNoAddRef() 288
GetPrimitiveID()
I3DExCamera 69
GetRayHitDetails() 168
GetRenderStatistics()
I3DExFinalRenderer 86
GetResID 19
GetResource 196
GetScene 226
GetScene(I3DShScene**) 205, 207, 209
GetShadingFlags()
I3DExShader 144
GetTextMetrics 201
GetTime() 287
GetTimeInfo(TTimeInfo&) 289
GetTopIfConstrained (I3DShTreeElement**) 279
GetTweener (I3DShTweener**, boolean, boolean) 286

©2001-2003 Eovia. All rights reserved.

GetTweenerNoAddRef (boolean, boolean) 286
GetUIImage 198
GetUIImageBucket 198
GetUIImageInfo 197
GetUVSpace 226
GetUVSpace()
I3DExGeometricPrimitive 122
GetUVSpaceCount 226, 235, 246
GetUVSpaceCount()
I3DExGeometricPrimitive 120, 121, 122
GetValue() 145, 260
GetVector() 146, 261
GetVolumeAttenuation() 174
GetVolumeDensity() 173
GetWireFrame 292
GetWireFrameBBox 164
Global Coordinate System 297
global coordinate system 297
Group 296
group 296

H

HandleEvent 18
HandlesTool 165
HasAtmosphere() 221
HasBackdrop() 221
HasBackground() 221
HitInfo 54
Hydrate() 110

I

I3DExAmbientLight interface 59
I3DExAtmosphericShader interface 61
I3DExBackdrop interface 65
I3DExBackground interface 63
I3DExCamera interface 67
I3DExExportFilter interface 21, 79
I3DExFinalRenderer 88
I3DExFinalRenderer interface 85
I3DExGeometricPrimitive interface 119
I3DExImportFilter interface 25, 91
I3DExLightsource

ForEachShadowBuffer(ForEachShadowBufferCallback, void*) 101
GetColor(TVector3&, TVector3&, real, TMCColorRGB&, real&) 100
GetDirection(TVector3&, TVector3&, real&) 99
GetLightInfo(boolean&, boolean&, real&, TTransform3D&) 101

©2001-2003 Eovia. All rights reserved.

GetLightParameter(int32, void*) 101
GetNewTrace(int16*) 102
GetStandardLight(TStandardLight&) 102
GetUVfromDirection(TVector2&, const TVector3&) 101
IsVisibleInPerspective() 100
SetTransform(TTransform3D*) 99
I3DExLightsource interface 99, 103
I3DExLightsourceGel

GetGelValues(constTVector2&, TMCColorRGB&) 103
I3DExModifier

Apply(I3DShTreeElement) 105, 110
GetMaxTimeWithoutComputation (I3DShTreeElement*) 105
I3DExModifier interface 105
I3DExModule

AboutToCloseMainWindow() 113
Activate() 111
BuildMenuBar() 111
Deactivate() 111
Dehydrate() 111
Destroy() 110
GetPreferredRoom(IDType&) 112
Hydrate() 110
Import(IMCUnknown*, void*) 111
Initialize(IMCUnknown*) 110
SavesInNativeFormat() 113
SubModuleAboutToBeDestroyed(I3DShModule*) 112
SubModuleAboutToBeHydrated(I3DShModule*) 112
SubModuleClosedMainWindow(I3DShModule*) 112
WantsToBeInCurrentRoom() 112
I3DExModule interface 109
I3DExPostRenderer interface 115, 117, 249
I3DExRendererBox interface 129
I3DExSceneCommand interface 137
I3DExShader interface 143
I3DExShadowFeature Interface 135
I3dExSkinable interface 147
I3DExTileRenderer interface 149
I3DExTweener

Inverse() 97
I3DExTweener interface 97, 155
I3DExTweenerEdition interface 157
I3DExVertexPrimitive interface 159
I3DExVolumePrimitive interface 169, 173
I3DExVolumetricEffect interface 161

©2001-2003 Eovia. All rights reserved.

I3DExWireFrame interface 163
I3DShAtmosphericShader

GetScene(I3DShScene**) 205
I3DShAtmosphericShader interface 205
I3DShBackdrop

GetScene(I3DShScene**) 207, 209
I3DShBackdrop interface 207, 209
I3DShCamera interface 211
I3DShEnvironment

DirectionFilter(const TVector3 ,const TVector3&, TMCColorRGB&) 220
GetBackdropColor(const TVector2&,boolean&,const TBBox2D&, TMCColorRGB &) 220
GetEnvironmentColor(const TVector3&, boolean&, TMCColorRGB&) 221
HasAtmosphere() 221
HasBackdrop() 221
HasBackground() 221
SegmentFilter(const TVector3& ,const TVector3&, TMCColorRGB&, boolean) 219
SegmentFilter(const TVector3&,const TVector3&,TMCColorRGB&) 220
I3DShEnvironment interface 219
I3DShMasterShader

Clone(I3DShMasterShader**, ECloneMode) 252
GetName(TMCString&) 251
GetScene(I3DShScene**) 252
GetShader(I3DShShader**) 252
IsDefaultMaster() 252
SetShader(I3DShShader*, ECloneMode) 251
I3DShObject 225
I3DShPostRenderer

BeginRender 250
I3DShTreeElement

BeginMove () 269
CanBeDistorted () 269
ComClone (I3DShTreeElement**, ECloneMode, boolean) 268
EndMove(boolean) 270
GetBoundingBox (TBBox3D&, ETreeDeformBBoxModes) 269
GetBoundingBoxWithoutLightsAndCameras(TBBox3D&) 269
GetScene (I3DShScene* scene) 265
GetSceneNoAddRef() 265
GetXYZScaling(TVector3&) 270
InvalidateBoundingBox () 268
IsBBoxPickable 280
PostMoveChange (boolean) 269
TranslateHotPoint (TVector3&) 268

©2001-2003 Eovia. All rights reserved.

I3DShTreeElement interface 247, 251, 253, 259, 263
I3DShTweener

Edit(I3DShTweener**,I3DShTweenerChainLink*) 283
GetLinkNoAddRef() 284
SetLink(I3DShTweenerChainLink*) 284
I3DShTweener interface 283
I3DShTweenerChainLink

Clone (I3DShTweenerChainLink**, I3DShParamTimeLine*) 285
CopyData (I3DShTweenerChainLink*) 286, 291
GetFatherNoAddRef() 289
GetKeyFrame (I3DShKeyFrame**) 287
GetKeyFrameNoAddRef () 287
GetNext (I3DShTweenerChainLink**) 288
GetNextNoAddRef() 287
GetPrev (I3DShTweenerChainLink**) 288
GetPrevNoAddRef() 288
GetTime() 287
GetTimeInfo(TTimeInfo&) 289
GetTweener (I3DShTweener**, boolean, boolean) 286
GetTweenerNoAddRef (boolean, boolean) 286
InsertAfter(I3DShTweenerChainLink*) 289
IsKeyFrameSelected() 289
IsTweenerSelected() 289
OffsetLink(MicroTick) 289
RemoveFromChain() 289
SelectKeyFrame(boolean) 290
SelectTweener(boolean) 290
SetKeyFrame (I3DShKeyFrame**) 287, 293
SetNext (I3DShTweenerChainLink*) 288
SetPrev(I3DShTweenerChainLink*) 288
SetTime (MicroTick) 287
SetTweener (I3DShTweener*) 286
I3DShTweenerChainLink interface 285, 291
IExDataExchanger interface 17
IExRegisterer interface 27
Import(IMCUnknown*, void*) 111
IncrementFreedomValue() 73, 272
IncrementProgress 192
Init 233
Init()
I3DExSceneCommand 137
I3DExTileRenderer 149
Initialise()
I3DExVolumetricEffect 161
Initialize(IMCUnknown*) 110
InsertAfter(I3DShTweenerChainLink*) 289

©2001-2003 Eovia. All rights reserved.

InsertDataComponent (I3DShDataComponent*) 275
InsertDataComponentByID (int32, int32) 275
InsertModifier(I3DShModifier* , uint32, ECloneMode) 272
instance
object instance 296
interpolating in a facet 306
Inval 189
InvalidatePreview 182
InvalidateSkining()
I3DExSkinable 148
IsAttached()
I3DExSkinable 148
IsCommandDown 180
IsControlDown 180
IsDefaultComponent 181
IsEqual 17
IsEqualTo() 143
IShAction interface 177
IShActionManager interface 179
IShComponent interface 181
IShComponentUtilities interface 183
IShParameterComponent interface 187
IShUtilities interface 191
IsInfiniteObject 227
IsKeyFrameSelected() 289
IsLongStaticClick 180
IsOpen () 276
IsOptionDown 180
IsPentiumIIIProcessor 200
IsPentiumIIProcessor 200
IsPentiumProcessor 200
IsShiftDown 180
IsShortStaticClick 180
IsSMPAware()
I3DExFinalRenderer 88
IsTweenerSelected() 289

L

light 296
Light Source 296
LinksChanged () 273
Local Coordinate System 298
local coordinate system 298

M

Master Objects 295
MATRIX3D 32

©2001-2003 Eovia. All rights reserved.

N

NeededDiskSpace 21
NeededDiskSpace()
    I3DExExportFilter 79
NeedsToPrepareMenus 178
normal
    patch normal 308
NUM3D 29, 302

O

object 295
Object Coordinate System 298
object coordinate system 298
object instance 296
Object List 295
Objects 295
OffsetLink(MicroTick) 289
OptimizeTileSize() 88

P

parametric mapping 305
patch 307
    normal 308
    u,v space 308
PATCH3D 49
Pick()
    I3DExVertexPrimitive 160
PostAction 179
PostMouseAction 179
PostPrepare()
    I3DExSceneCommand 138
Prepare 17, 21, 25
    I3DExSceneCommand 138
Prepare()
    I3DExExportFilter 79
    I3DExImportFilter 91
PrepareDraw() 248
    I3DExFinalRenderer 87
Primitive 295
Project3DTo2D()
    I3DExCamera 69

R

Random 194
RandomStep 194
Ray3D 51

©2001-2003 Eovia. All rights reserved.

RayHit()
I3DExGeometricPrimitive 167
RayHit3D 52
RayHitParameters 53
ReadComponent 185, 186
ReadComponentData 186
Redo 177
Redo()
I3DExSceneCommand 138
ReleaseData (TDataKey) 276
RemoveDataComponent (uint32) 276
RemoveFromChain() 289
RemoveModifier 272
RemoveModifier(uint32) 272
RemoveWireFrame (I3DExWireFrame*) 280
RenderBackGround()
I3DExFinalRenderer 88
RenderVolumetrics()
I3DExFinalRenderer 88
RestoreResources 197
rotation matrix 303

S

SavesInNativeFormat() 113
Scene
Object List 295
Objects 295
Scene Tree 295
Tree Elements 295
scene 295
Scene Instancing 295
Scene Tree 295
scene tree 295
Screen Coordinate System 299
screen pixels space 301
SegmentFilter()
I3DExAtmosphericShader 61
SegmentFilter(const TVector3& ,const TVector3&, TMCColorRGB&, boolean) 219
SegmentFilter(const TVector3&,const TVector3&,TMCColorRGB&) 220
SelectKeyFrame(boolean) 290
SelectTweener(boolean) 290
SetAmbientLight()
I3DExFinalRenderer 86
SetAnimationMethod (I3DShAnimationMethod*, ECloneMode, real) 277
SetBoundingBox()
I3DExModifier 105
SetCamera()

©2001-2003 Eovia. All rights reserved.

I3DExFinalRenderer 85
SetColor 293
SetConstraint (I3DShConstraint*, ECloneMode) 274
SetConstraintByID (int32) 274
SetCursor 201
SetData (TDataKey, IMCUnknown*) 276
SetEnvironment()
I3DExFinalRenderer 86, 149
SetFieldRenderingData()
I3DExFinalRenderer 86
SetFromUndoInfo (const TTreeUndoInfo&, const TTreeUndoInfo*) 279
SetGlobalTransform3D (const TTransform3D&, int16) 278
SetGlobalTreeTransform (const TTreeTransform&, int16, boolean) 278
SetHasUVSpace 234
SetInstanceGlobalReferenceTransform()
I3DExSkinable 148
SetKeyFrame (I3DShKeyFrame**) 287
SetLastOpenDocPath 200
SetLastSaveDocPath 200
SetModifiers (const TMCCountedPtrArray*, ECloneMode) 273
SetName() 225
SetNext (I3DShTweenerChainLink*) 288
SetParameter 18, 188
SetPoint 292
SetPrev(I3DShTweenerChainLink*) 288
SetPreview 22
SetPreview()
I3DExExportFilter 80
SetProgressPrompt 192
SetProgressValue 193
SetProgressValue0to1 193
SetRandomSeed 194
SetScene 247
SetScene() 211
SetSelectedState 294
SetTime (MicroTick) 287
SetTransform()
I3DExCamera 67
SetTreeTop()
I3DExFinalRenderer 85
SetTweener (I3DShTweener*) 286
SetupResources 197
SetUVSpace 234
shading 306
parametric mapping 305
ShadingFlags 45
ShadingIn 43
ShadingInOut 48

©2001-2003 Eovia. All rights reserved.

ShadingOut 47
StringWidth 200
SubModuleAboutToBeDestroyed(I3DShModule*) 112
SubModuleAboutToBeHydrated(I3DShModule*) 112
SubModuleClosedMainWindow(I3DShModule*) 112
surface normal 304
surface point 304

T

TBBOX2D 31
TestHit 293
texture coordinate system 305
TickCount 194
TrackWireFrame 163
Transform 293
TRANSFORM3D 35
tree element 295
Tree Elements 295
tree elements transformation 303
TREETRANSFORM3D 36

U

u,v space 305
    patch u,v space 308
Undo 177
Undo()
    I3DExSceneCommand 138
units system 302
UnregisterComponents 27
UseMMX 200
UV Space 305
UVShMaps 48
UVSpaceInfo 48

V

VECTOR2D 29
VECTOR3D 31
VERTEX3D 49

W

WantsOptionDialog 21, 25
WantsOptionDialog()
    I3DExExportFilter 79
    I3DExImportFilter 91
WantsPreview 22
WantsPreview()

©2001-2003 Eovia. All rights reserved.

I3DExExportFilter 80
WantsTempFile()
I3DExExportFilter 81
WantsToBeInCurrentRoom() 112
WantsTopScene()
I3DExImportFilter 92
WillCauseChange 177
WireFrameToData 164
Working Box Coordinate System 298
working box coordinate system 298
WriteComponent 185

Y

YieldProcesses 193

©2001-2003 Eovia. All rights reserved.