

Volume Library Interface

Quick Reference

<http://www.3dvolumegraphics.com>

This document lists VLI functions and parameters. **Constant** declarations are omitted from the function declarations. Full descriptions of the functions, their parameters and return values are in the *Volume Library Interface User's Guide*. Based on V1.95 of the `vli.h` header file.

To include VLI, applications must link using `vli.lib` and `vli.dll`. VLI version numbers are defined as follows:

```
#define kVLIIMajorVersion 0;kVLIIMinorVersion 9
```

VLI System Init & Config

```
VLIStatus VLIOpen();  
void VLIClose(void);
```

VLI Configuration Class

```
VLIConfiguration(void);  
virtual ~VLIConfiguration(void);
```

```
int GetBoardMajorVersion(void) const;  
int GetBoardMinorVersion(void) const;  
int GetNumberOfBoards(void) const;  
int GetMaxCutPlanes(void) const;  
int GetVLIIMajorVersion(void);  
int GetVLIIMinorVersion(void);  
int GetGradientTableLength(void) const;
```

```
VLIuint32 GetAvailableMemory(  
int inIthBoard) const;
```

```
void GetMaxLockedSize(unsigned int inFormat,  
unsigned int &outNx,&outNy,&outNz);  
void GetMaxMappedSize(unsigned int inFormat,  
unsigned int &outNx,&outNy,&outNz);
```

VLI Global Types and Constants

```
typedef char VLIint8;  
typedef short VLIint16;  
typedef long VLIint32;  
typedef unsigned char VLIuint8;  
typedef unsigned short VLIuint16;  
typedef unsigned long VLIuint32;  
typedef float VLIfloat32;  
typedef double VLIfloat64;  
typedef VLIuint32 VLIPixel;  
typedef VLIuint32 VLIRGBAPacked;  
typedef int VLIfbool;  
#define VLIfalse 0  
#define VLIftrue (!VLIfalse)
```

```
VLIStatus VLISetEpsilon(double inEpsilon);  
double VLIGetEpsilon(void);
```

VLI LookupTable Class

```
struct VLIRGBColor{  
VLIuint8 m_blue,m_green,m_red};  
struct VLIRGBAFloat{  
float m_red,m_green,m_blue,m_alpha}  
enum Size {kSize4096, kSize2048,  
kSize1024, kSize512, kSize256};  
static VLIILookupTable*  
Create(Size inTableSize = kSize4096);  
Create(Size inTableSize,  
VLIuint8 inColor[][3],VLIuint16 *inAlphas);  
Create(Size inTableSize,  
VLIRGBAPacked *inEntries);  
Create(Size inTableSize,  
VLIRGBAFloat *inEntries);
```

```
VLIStatus  
SetColorEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIuint8 inRGB[][3]);  
SetColorEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBColor *inEntries);
```

```
VLIStatus  
GetColorEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIuint8 outRGB[][3]) const;  
GetColorEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBColor *outEntries) const;
```

```
VLIStatus  
SetAlphaEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIuint16 *inAlpha);  
SetAlphaEntries(unsigned int inIndex,  
unsigned int inLength,  
float *inAlpha);
```

```
VLIStatus  
GetAlphaEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIuint16 *outAlpha) const;  
GetAlphaEntries(unsigned int inIndex,  
unsigned int inLength,  
float *outAlpha ) const;
```

```
VLIStatus  
SetRGBAEEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBAPacked *inRGBA);  
SetRGBAEEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBAFloat *inRGBA);
```

```
VLIStatus  
GetRGBAEEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBAPacked* outRGBA) const;  
GetRGBAEEntries(unsigned int inIndex,  
unsigned int inLength,  
VLIRGBAFloat* inRGBA) const;
```

```
Size GetSize(void) const;
```

```
int AddRef(void);  
int Release(void);  
void *m_appData;
```

VLI Light Class

```
const static VLIVector3D sNegativeZ;  
static VLIILight* CreateDirectional(  
VLIVector3D &inDirection = sNegativeZ,  
double inIntensity = 1.0);  
const VLIVector3D& GetDirection(void) const;  
double GetIntensity(void) const;  
VLIStatus SetDirection(  
VLIVector3D &inDirection);  
VLIStatus SetIntensity(double inIntensity);  
int AddRef(void);  
int Release(void);  
void *m_appData;
```

VLI Camera Class

```
VLICamera(void);  
VLICamera(VLIIMatrix &inMatrix);  
virtual ~VLICamera(void);  
VLIIMatrix GetViewMatrix(void);  
VLIStatus SetViewMatrix(VLIIMatrix &inMatrix);  
void GetViewport(double& outWidth,outHeight);  
VLIStatus SetViewport(double inWidth,inHeight);
```

VLI CutPlane Class

```
enum Flags {kInside, kOutside};  
static VLIICutPlane* Create(  
double inA,inB,inC,inD,  
double inOffset, inFallOff = 0.0,  
Flags inFlags = kInside);  
void GetPlane(double& outA,outB,outC,outD);  
VLIStatus SetPlane(double inA,inB,inC,inD);  
double GetThickness(void) const;  
VLIStatus SetThickness(double inThickness);  
double GetFallOff(void) const;  
VLIStatus SetFallOff(double inFallOff);  
Flags GetFlags(void) const;  
VLIStatus SetFlags(Flags inFlags);  
int AddRef(void);  
int Release(void);  
void *m_appData;
```

VLI LookupTable Class (cont'd)

```
VLIStatus  
SetColorEntry(unsigned int inIndex,  
unsigned int inRed,inGreen,inBlue);  
SetColorEntry(unsigned int inIndex,  
double inRed,inGreen,inBlue);  
SetAlphaEntry(unsigned int inIndex,inAlpha);  
SetAlphaEntry(unsigned int inIndex,  
double inAlpha);  
SetRGBAEEntry(unsigned int inIndex,  
unsigned int inRed, inGreen, inBlue,  
unsinted int inAlpha);  
SetRGBAEEntry(unsigned int inIndex,  
double inRed, inGreen,inBlue, inAlpha);  
SetRGBAEEntry(unsigned int inIndex,  
VLIRGBAPacked inPacked);
```

VLICrop Class

```
enum Flags { kDisable,
             kEnableX0,kEnableY0,kEnableZ0,kInvert0,
             kEnableX1,kEnableY1,kEnableZ1,kInvert1,
             kEnableX2,kEnableY2,kEnableZ2,kInvert2,
             kOrSelect,kInvertOutput,kSubVolume,
             k3DCross, k3DCrossInvert,k3DFence,
             kDFenceInvert};

VLICrop(void);
VLICrop(double inMinX,inMaxX,
         double inMinY,inMaxY,
         double inMinZ,inMaxZ,
         int inFlags = kSubVolume);
~VLICrop(void);

void GetSlabs(double& outMinX,outMaxX,
             double& outMinY,outMaxY,
             double& outMinZ,outMaxZ);
VLIStatus SetSlabs(double inMinX,inMaxX,
                  double inMinY,inMaxY,
                  double inMinZ,inMaxZ);

VLIStatus SetXSlab(double inMinX,inMaxX);
VLIStatus SetYSlab(double inMinY,inMaxY);
VLIStatus SetZSlab(double inMinZ,inMaxZ);

int GetFlags(void) const;
VLIStatus SetFlags(int inFlags);
```

VLICursor Class

```
enum Types {kCrossHair, kPlane};
enum Axes {kZAxis, kXAxis, kYAxis};
enum Flags {kEnableX, kEnableY, kEnableZ,
            kEnableAll, kDisableCrop};

VLICursor(void);
virtual ~VLICursor(void);

VLIStatus SetType(int inType);
int GetType(void) const;

VLIStatus SetWidth(double inWidth);
double GetWidth(void);

VLIStatus SetLength(double inLength);
double GetLength(void);

VLIStatus SetPosition(double inX,inY,inZ);
void GetPosition(double &outX,&outY,&outZ);

VLIStatus SetAttributes(int inModes);
int GetAttributes(void) const;

VLIStatus SetColor(int inAxes,
                  double inRed, inGreen, inBlue);
void GetColor(int inAxis,
              double &outRed, &outGreen, &outBlue);

VLIStatus SetOpacity(int inAxes,
                    double inAlpha);
double GetOpacity(int inAxis);

VLIStatus SetColorAndOpacity(int inAxes,
                             VLIRGBAPacked inRGBA);
VLIRGBAPacked GetColorAndOpacity(int inAxis);
```

VLIGraphicsContext Class

```
enum InfoType {kView,kLight,kCutPlane,
               kMaterial,kAll};

virtual int TransferBasePlane(int inBuffer,
                              unsigned int inNx, inNy, inIX, inIY,
                              VLI Pixel*inBasePlane)=0;

virtual int ImportAttributesToVLIContext(
    VLIContext &inContext,
    InfoType inFlags)=0;

virtual int GetTextureMemoryAddress(
    int inBuffer,
    unsigned int inNX, inNY,
    void *& outAddress,
    unsigned int outBusAddress[2])=0;

virtual int RenderHexagon( int inBuffer,
                          VLIVector3D inGeomHex[6],
                          VLIVector2D inTexHex[6])=0;

VLI PixelFormat GetPixelFormat(void) const;
int SetPixelFormat(VLI PixelFormat inFormat);
virtual VLIbool ShouldLeaveOnBoard(void)=0;

VLI OpenGLContext Class
Note: To include VLI OpenGLContext class, applications must
include vliopengl.h and vliopengl.dll.

VLI OpenGLContext(void);
virtual int TransferBasePlane(int inBuffer,
                              unsigned int inNx, inNy,
                              unsigned int inIX, inIY,
                              VLI Pixel *inBasePlane);

virtual int GetTextureMemoryAddress(
    int inBuffer,
    unsigned int inX, inY,
    unsigned int &outStride, &outFormat,
    void *&outVirtualAddress,
    unsigned int outBusAddress[2]);

virtual int RenderHexagon(
    int inBuffer,
    VLIVector3D inHex[6],
    VLIVector2D inTex[6]);

GLuint NewTextureObject(int inBuffer);
virtual int ShouldLeaveOnBoard(void);

virtual int ImportAttributesToVLIContext(
    VLIContext &inContext,
    InfoType inFlag);
```

VLI ErrorHandler Class

```
VLI ErrorHandler(void);
virtual ~VLI ErrorHandler(void);

void ErrorNotify(int inErrNum, char *inFile,
                 int inLineNum);

VLI ErrorHandler* VLI SetErrorHandler(
    VLI ErrorHandler *inHandler);
```

VLIEvents

```
enum VLIEvent { kVLIEventRenderDone,
                kVLIEventTransferDone, kVLIEventLightMap,
                kVLIEventLightMapChange,
                kVLIEventMultiPassBasePlane,
                kVLI MaxEvents};

class VLIEXPORT VLIEventBase {
public:int buffer;
    virtual VLIEvent EventType(void) = 0;
    virtual ~VLIEventBase() {}
    VLIEventBase(int buffer); };

class VLIEXPORT VLIEventRenderDone:
public VLIEventBase {
public:
    VLIEvent EventType(void);
    VLIEvent RenderDone(int buffer);
};

class VLIEXPORT VLIEventTransferDone:
public VLIEventBase {
public:
    VLIEvent EventType(void);
    VLIEvent TransferDone(int buffer);
};

typedef enum{kVLLightMapType}VLLightMapType;

class VLIEXPORT VLIEventLightMap :
public VLIEventBase {
public:
    VLLightMapType lightFormat;
    int bytesPerElement, numberOfElements;
    void *pointerToMap1, *pointerToMap2;
    VLIEvent EventType(void);
    VLIEvent LightMap( int buffer,
                      VLLightMapType format,
                      int eltSize, nElts,
                      void *map1, *map2);
};

class VLIEXPORT VLIEventLightMapChange :
public VLIEventBase {
public:
    VLLightMapType lightFormat;
    int bytesPerElement, numberOfElements;
    void *pointerToMap1, *pointerToMap2;
    VLIEvent EventType(void);
    VLIEvent LightMapChange( int buffer,
                             VLLightMapType format,
                             int eltSize, nElts,
                             void *MapChange1,*MapChange2);
};

class VLIEXPORT VLIEventMultiPassBasePlane :
public VLIEventBase {
public:
    VLIEvent EventType(void);
    VLIEvent MultiPassBasePlane (
        int buffer,multiPassBuffer);
    int multiPassBuffer;
};

typedef void (*VLIEventCallback)(
    VLIEventBase *inEvent, void*inData);
```

VLIVector3D Class

```
VLIVector3D(void);
VLIVector3D(VLIVector3D& inVec);
VLIVector3D(double inX,inY,inZ);
VLIVector3D(const double[3]);
VLIVector3D(const float[3]);
VLIVector3D&
    Assign(double x,y,z);
    Assign(double[3]);
    Assign(float[3]);
double&
    X(void);
    Y(void);
    Z(void);
    operator [] (int);
const double& operator [] (int) const;
void
    CopyTo(float *outX, *outY, *outZ);
    CopyTo(double* outX, outY, outZ) const;
    CopyTo(float outArray[3]);
    CopyTo(double outArray[3]);
double
    Length(void) const;
    LengthSquared(void) const;
friend double
    VLIDistance(const VLIVector3D& inA,inB);
    VLIDistanceSquared(
        const VLIVector3D& inA,inB);
VLIVector3D& Normalize(double to=1.0);
VLIVector3D&
    operator = (VLIVector3D& v);
    operator += (VLIVector3D&);
    operator -= (VLIVector3D&);
    operator *= (double);
    operator /= (double);
VLIVector3D
    operator + (VLIVector3D&);
    operator - (VLIVector3D&);
    operator - (void);
    operator * (double);
    operator / (double);
    operator * (double, VLIVector3D&);
VLIbool
    operator == (VLIVector3D&);
    operator != (VLIVector3D&);
double
    Dot(VLIVector3D& inV);
    VLIDot(VLIVector3D& inA,inB);
VLIVector3D VLICross(VLIVector3D& inA, inB);

ostream & operator <<(ostream&,VLIVector3D&);
```

VLIMatrix Class

```
VLIMatrix(void);
VLIMatrix(const VLIMatrix&);
VLIMatrix(double inArray[16]);
VLIMatrix(float inArray[16]);
VLIMatrix(double in00,in01,in02,in03,
            double in10,in11,in12,in13,
            double in20,in21,in22,in23,
            double in30,in31,in32,in33);
~VLIMatrix(void);
VLIMatrix&
    Assign(double inArray[16]);
    Assign(float inArray[16]);
    Assign(double in00,in01,in02,in03,
            in10,in11,in12,in13,
            in20,in21,in22,in23,
            in30,in31,in32,in33);
void
    CopyTo(double outArray[16]);
    CopyTo(float outArray[16]);
void
    TransformPoint(VLIVector3D &in,&out);
    TransformVector(VLIVector3D &in, &out);
double* operator [] (int);
const double* operator [] (int) const;
VLIMatrix&
    operator *= (VLIMatrix&);
    operator = (VLIMatrix&);
VLIMatrix
    operator * (double) const;
    operator * (VLIMatrix&) const;
    operator * (double,VLIMatrix&);
VLIbool
    operator == (VLIMatrix&) const;
    operator != (VLIMatrix&) const;
VLIMatrix
    Transpose(void) const;
    Inverse (void) const;
    Adjoint(void) const;
double Determinant(void) const;
static VLIMatrix
    Identity(void);
    Rotate(double inDegrees, VLIVector3D &inAxis);
    Scale (double inSx,inSy,inSZ);
    Translate(double inTx,inTy,inTZ);
    LookAt(VLIVector3D &inEye,inAim,inUp);
VLIbool
    Issingular(void) const;

ostream & operator <<(ostream&, VLIMatrix&);
```

Helpful Hints

```
To change view:
pContext-> GetCamera.SetViewMatrix(
    VLIMatrix::LookAt ( VLIVector3D (0.0, 0.0, 10.0),
                       VLIVector3D (0.0, 0.0, 0.0),
                       VLIVector3D (0.0, 1.0, 0.0)
    );
```

VLIVector2D Class

```
VLIVector2D();
VLIVector2D(VLIVector2D&v);
VLIVector2D(double x, y);
VLIVector2D(const double[2]);
VLIVector2D(const float[2]);
VLIVector2D&
    Assign(double x, y);
    Assign(double[2]);
    Assign(float[2]);
double&
    X(void);
    Y(void);
    operator [] (int);
const double& operator [] (int) const;
void
    CopyTo(double *x,*y) const;
    CopyTo(float[2]) const;
    CopyTo(double[2]) const;
double
    Length(void) const;
    LengthSquared(void) const;
double
    VLIDistance(VLIVector2D& inA,inB);
    VLIDistanceSquared(VLIVector2D& inA,inB);
VLIVector2D& Normalize(double to=1.0);
VLIVector2D&
    operator = (VLIVector2D& v);
    operator += (VLIVector2D&);
    operator -= (VLIVector2D&);
    operator *= (double);
    operator /= (double);
VLIVector2D
    operator + (VLIVector2D&);
    operator - (VLIVector2D&);
    operator - (void);
    operator * (double);
    operator / (double);
    operator * (double,VLIVector2D&);
VLIbool
    operator == (VLIVector2D&);
    operator != (VLIVector2D&);
double
    Dot(VLIVector2D&) const;
    VLIDot (VLIVector2D& inA,inB);

ostream & operator <<(ostream&,VLIVector2D&);
```

VLIVolume Class

```

enum
Layout {kSliced,kBlocked};
ReadMode { kReadOnly, kCopyOnWrite, kReadWrite};
VLIAccessType {kVLIAccessNoControl,
kVLIAccessReadExclusive,kVLIAccessReadShared,
kVLIAccessWrite, kVLIAccessWriteExclusive,
kVLIAccessWriteShared};
const VLIuint32
kVLIIVoxelFormatUINT8;
kVLIIVoxelFormatUINT12L;
kVLIIVoxelFormatUINT12U;
VLIIVolume*
Create(VLIuint32 inFormat,void *inVoxels,
unsigned int inSizeX,inSizeY, inSizeZ,
Layout inLayout = kSliced);
CreateFromFile (char *inFilename,
int inFileType =0,
ReadMode inReadMode =kReadOnly);
VLIbool IsReadOnly(void);
void GetSize(unsigned int& nXVox,nYVox,nZVox);
VLIuint32 GetFormat(void) const;
Layout GetLayout(void) const;
VLIIMatrix GetModelMatrix (void);
VLIStatus SetModelMatrix (VLIIMatrix& inModel);
void GetActiveSubVolumeSize(
unsigned int &inNX,&inNY,&inNZ);
VLIStatus SetActiveSubVolumeSize(
unsigned int inNX,inNY,inNZ);
void GetActiveSubVolumeOrigin(
unsigned int &inNX, &inNY, &inNZ);
VLIStatus SetActiveSubVolumeOrigin(
unsigned int inNX,inNY,inNZ);
VLIStatus
LockVolume(void);
UnlockVolume(void);
VLIbool IsLocked(void) const;
VLIStatus
MapVolume(VLIAccessType inAccess,
void *&outBaseAddress,
unsignedint &outSx,&outSy);
MapSubVolume(VLIAccessType inAccess,
unsigned int inAtX,inAtY,inAtZ,
unsigned int inNx,inNy,inNz,
void *&outBaseAddress,
unsigned int &outSx, &outSy);
UnmapVolume(void);
VLIbool IsLoaded(void) const;
VLIStatus UpdateVolume(VLIuint32 inFormat,
void*inVoxels,
unsigned int inTargX, inTargY, inTargZ,
unsigned int inNXVox, inNYVox, inNZVox,
Layout layout = kSliced);
VLIAccessType GetAccessControl(void) const;
int AddRef(void);
int Release(void);
void *m_appData;

```

VLIContext Class

```

#define VLILinearGradientRamp 0
enum VLICoordinateSpace {kVLIObjectSpace,
kVLI_CameraSpace, kVLIWorldSpace};
int kVLI_MaxRenderBuffers = 16;
static VLIContext* Create(
VLIILookupTable *inColor = 0);
VLI_Camera& GetCamera(void);
void SetCamera(VLI_Camera &inCam);
VLIILookupTable* GetLookupTable(void) const;
VLIStatus SetLookupTable(VLIILookupTable *inLUT);
void GetReflectionProperties(
double& outDiffuse, outSpecular,
double& outEmissive, outSpecularExponent)
const;
VLIStatus SetReflectionProperties (
double inDiffuse, inSpecular,
double inEmissive,inSpecularExponent);
void GetSpecularColor(
double& outRed,outGreen,outBlue) ;
VLIStatus SetSpecularColor(
double inRed,inGreen,inBlue);
VLIStatus AddLight(VLIILight *inLight);
VLIStatus RemoveLight(VLIILight *inLight);
unsigned int GetLightCount(void) const;
VLIILight* GetLight(unsigned int inI);
VLIbool GetCorrectOpacity(void) const;
VLIStatus SetCorrectOpacity(VLIbool inOn);
VLIbool GetGradientOpacityModulation();
VLIStatus SetGradientOpacityModulation (
VLIbool inOn);
VLIbool GetGradientSpecularIllumination
Modulation(void) const;
VLIStatus SetGradientSpecularIllumination
Modulation (VLIbool inGMIMSpecular);
VLIbool GetGradientDiffuseIllumination
Modulation (void) const;
VLIStatus SetGradientDiffuseIllumination
Modulation(VLIbool inGMIMDiffuse);
double* GetGradientTable(void);
VLIStatus SetGradientTable(double inTable[]);
enum VLIBlendMode{kVLIBlendMIP,kVLIBlendMINIP,
kVLIBlendFTB};
VLIBlendMode GetBlendMode(void) const;
VLIStatus SetBlendMode(VLIBlendMode inMode);
VLIStatus AddCutPlane(VLICutPlane* inPlane);
VLIStatus RemoveCutPlane(
VLICutPlane *inPlane);
VLICutPlane* GetCutPlane(unsigned int inI);
unsigned int GetCutPlaneCount(void) const;
VLI_Crop& GetCrop(void);
VLIStatus SetCrop(VLI_Crop &inCrop);
VLI_Cursor& GetCursor(void);
VLIStatus SetCursor(VLI_Cursor& inCursor);
void GetSuperSamplingFactor(
double& outX,outY,outZ) const;
VLIStatus SetSuperSamplingFactor(
double inX,inY,inZ);

```

```

VLICoordinateSpace GetSuperSamplingSpace(
void) const;
VLIStatus SetSuperSamplingSpace(
VLICoordinateSpace inSpace);
enum AccumulationMode {
GrowBasePlane,BlendBasePlane};
AccumulationMode GetBasePlaneAccumulation();
VLIStatus SetBasePlaneAccumulation(
AccumulationMode inMode);
enum VLIPixelFormat {kVLIFirstPixelFormat,
kVLIPixelFormatRGBA,kVLIPixelFormatRGBA8,
kVLIPixelFormatBGRA8,kVLIPixelFormatARGB8,
kVLIPixelFormatABGR8};
VLIPixelFormat GetBasePlaneFormat();
VLIStatus SetBasePlaneFormat(
VLIPixelFormat inFormat);
VLIStatus DescribePixelFormat(
VLIPixelFormat inFormat,
int& outRedWidth, outRedOffset,
int& outGreenWidth,outGreenOffset,
int& outBlueWidth, outBlueOffset,
int& outAlphaWidth,outAlphaOffset);
VLIStatus FetchBasePlane(int inBuffer,
int &outBaseWidth, &outBaseHeight,
int &outImageWidth, &outImageHeight,
VLIIPixel *&outBasePlane,
VLIVector3D outHexGeometry[6],
VLIVector2D outTexCoordinates[6]);
VLIStatus ReleaseBasePlane(int inBuffer);
VLIStatus RenderBasePlane(
VLIIVolume *inVolume,
int inBuffer);
VLIStatus RenderAndTransferBasePlane(
VLIIVolume *inVolume,
int inBuffer,
VLIIGraphicsContext& inGC,
VLIVector3D outHex[6],
VLIVector2D outTex[6]);
VLIStatus RenderToGraphicsContext(
VLIIVolume* inVolume,
int inBuffer,
VLIIGraphicsContext& inGC);
VLIStatus Abort(int inBuffer);
VLIStatus WaitForEvent(
long inTimeoutMilliseconds,
VLIEventBase*&outEvent,
void*&outData);
VLIStatus SetNotifyCallback(int inBuffer,
VLIEvent inEvent,
VLIEventCallback inNotify,
void *inData);
static VLIEventCallback
kVLIEventNoCallbackRoutine;
int AddRef(void);
int Release(void);
void* m_appData;

```

VLIILight, VLI_Camera, VLI_CutPlane, VLIILookupTable on A1.
VLI_Crop Class is on B1.
VLIEventHandler Class and VLIEvents on B1.