Autodesk®
Lustre® 2011
A Discreet® systems product

Sparks® API Reference Guide

Autodesk®

Autodesk® Lustre® 2011

Trademarks

Disclaimer

Published by: Autodesk, Inc.
111 McInnis Parkway
San Rafael, CA 94903, USA

Title: Autodesk Lustre 2011 Sparks API Reference Guide

Document Version: 1

Date: April 1, 2010

Contents

Sparks Plug-ins

1

Topics in this chapter:

About Developing Lustre Sparks Plug-ins

When developing an Autodesk® Lustre® Sparks® plug-in there are mandatory tasks and optional tasks. For example, some functions and naming conventions are mandatory. Calling functions of the Lustre Sparks plug-in API and building GUI elements are optional.

CPlugin Class

To create a Lustre Sparks plug-in you must implement a plug-in class derived from the base class: CPlugin. This base class is defined in the *CFront.h* and *CFront.c* files, which are accessible to Lustre users and should not be modified. Newer versions of the system may require a new plug-in API, which means new *CFront.h* and *CFront.c* files. In the next section the base CPlugin class is introduced, indicating both mandatory and optional tasks.

Variables

The CPlugin class has several public variables and functions that you can access and use, since all plug-in classes should be derived from this class. The public variables are:

- **`m_dwPos (int):`** The position of the plug-in in the order provided by the system workflow (and the UI). The first plug-in in the workflow has the position value of 0, the last has the position value of 5.

- **`m_PluginName (CStdString):`** Name of the plug-in which is almost the same as the name of the plug-in file except that it will be truncated when it does not fit into the UI button of the plug-in.
- **`m_Input (CImage):`** Input image of the plug-in. The input image buffer contains the result of the grading workflow stopped after the Secondary Correction and after the effects of the preceding plug-ins. (Example: If our plug-in is the second plug-in, its input buffer contains the result of the Secondary Correction combined with the effect of the first plug-in.)
- **`m_Output (CImage):`** Output image buffer of the plug-in.

The definition of the CImage class used above is the following:

```
class CPLUGIN_API CImage
{
public:
  RGB16BIT *m_data; // image data
  int m_width; // width of the image
  int m_height; // height of the image
  int m_pixels; // number of pixels in the image

public:
  CImage(void){};
  void InitImage(RGB16BIT *data,int width,int height,int pixels){};
  operator RGB16BIT* () {};
};
```

where the RGB16BIT means:

```
// Type of the image buffer
typedef struct
{
  uint16_t r,g,b;
  uint16_t a;  // alpha reserved; needed for proper memory alignment on certain systems


}
RGB16BIT;
```

Functions

This section explains the public functions of the base CPlugin class.

Mandatory: virtual void constructor

Can be empty but *you must* implement it, since it is called by the system.

Mandatory: virtual void destructor

Can be empty but *you must* implement it.

Mandatory: virtual void Init( )

You must implement this function, since it is called by the system. This method is called ONLY once when the DLL gets loaded. If the user deletes the plug-in or creates new instances, the Init( ) function is not called. It is NOT called after shot-change either!

Mandatory: virtual void Interact(int event_type, int data, float sx, float sy)

You must implement this function, since it is called by the system. It is called when user interaction, with the mouse, is detected. Event_type is any of BUTTONPRESS_EVENT, BUTTONRELEASE_EVENT or

MOTIONNOTIFY_EVENT (these symbols are defined in the *CFront.h* header). Data contains the button mask of the mouse, which can be any combination (i.e. the bitwise OR operation) of BUTTON_LEFT, BUTTON_MIDDLE, BUTTON_RIGHT, or 0 if no buttons were pressed. Example: When you drag the mouse with the left button pressed, you will receive MOTIONNOTIFY_EVENT in event_type and BUTTON_LEFT in data. (sx,sy) are screen coordinates; (0,0) represents the bottom left corner of the screen.

Mandatory: virtual void Overlay( )

You must implement this function, since it is called by the system. Overlay function for adding user-defined drawings (using OpenGL) on top of the image window that is actually not part of the image.

Mandatory: virtual void Process( )

You must implement this function, since it is called by the system. This is the image processing function of the plug-in. If multi-threaded computation is needed the image processing code is not in this function (discussed later).

Mandatory: virtual void PreProcess( )

You must implement this function, since it is called by the system. Called right after the process( ) and is usually empty.

NOTE The body of the functions that are required to be implemented may be empty.

Mandatory: virtual void PostProcess( )

You must implement this function, since it is called by the system. Called right after the process( ) and is usually empty.

NOTE The body of the functions that are required to be implemented may be empty.

void RefreshGui(CBaseController &ctrl)

Forces the system to redraw the GUI controller defined by ctrl.

void GetFrame(CTime Time,CImage Image)

This function is not working now, reserved for later purposes

void GetZoom(float *XZoom,float *Yzoom)

Returns the current zoom factor into XZoom and Yzoom.

void GetPicPos(float *XPicPos,float *YpicPos)

Returns the position of the current bottom left corner of the image measured in screen coordinates.

void MpCallThread(void (*func) ( ),unsigned long value_to_be_divided, int par_num,...)

This member function is responsible for the parallel execution of the user function named func. Parameters are:

- func: Pointer to the function wanted to be executed concurrently. func must be prototyped as void func (unsigned long *chunk_start,unsigned long *chunk_size,<parameters>), where <parameters> denoted the parameters wanted to be passed for func. func is allowed to have MAX_MPTHREAD_ARG (value of 6 presently) parameters at a maximum; the size of any parameters must not exceed the size of a pointer! *Limitation: 'func' must not be a member function of the plug-in!*

TIP Since 'func' can not be a member of the plug-in class, it does not have access to plug-in variables and function calls. However, there is a workaround. You can pass 'func' a pointer to the plug-in instance, thereby calling from this function a method of the plug-in.

- value_to_be_divided: The value which must be divided among the available processors. This value can be anything which is appropriate for the task of func, e.g. number of pixels, width of the image, height of the image, etc.
- par_num: Exact number of the parameters needed by func (the maximum value allowed here is MAX_MPTHREAD_ARG) excluding the parameters chunk_start and chunk_size.

int IsLogColorSpace(void)

You can check the current color space mode.

int IsAnamorphic(void)

You can check whether the system uses anamorphic squeeze.

int GetRowOffset(void)

If the image is cropped AND displaced vertically, it returns the offset of the image center.

void *GetMask(int secid,int width,int height)

You can use the mask with id **`secid`** in your calculations. It returns an **`unsigned short *`** pointer from where you can read out the mask values. The passed width and height should be the image size.

int GetActFrameNumber(void)

Returns the number of the actual frame.

void GetShotRange(int *firstframe,int *numframe)

Returns the shot start frame number and the shot length. Using this info and GetActFrameNumber(void), you can calculate the frame number in the actual shot.

void SetVKernel(int vkernel)

Available for the Autodesk® Incinerator® system only and for use only with the PreProcess function. You can tell how many extra pixel rows required calculating the plug-in. This is useful when the image processing is kernel-based. And here you must set what is the vertical kernel size of the processing algorithm to get correct results on Incinerator.

CBaseController Classes

There are six kinds of controller classes:

- Label (CLabel)
- Push button (CPushButton)
- Boolean button (CBooleanButton)
- Radio button (CRadioButton)
- Slider (CSlider)
- Data controller (CData)

These controller classes are derived from the base class CBaseController. All these classes are defined in the header *CFront.h*. The controllers for your plug-in should be created right before the implementation of the plug-in functions (see example). In the plug-in functions you can assign and read out values of the controllers. Some examples:

```
// read out boolean value
if (BooleanButton1) {...};

// read out numerical value
float myvalue = Slider1;

// assign value
Slider1 = 2.4;
```

The controller variable names MUST be created using the following format string rules:

- Labels: "Label%d"
- Push buttons: "PushButton%d"
- Boolean buttons: "BooleanButton%d"
- Radio buttons: "RadioButton%d"
- Sliders: "Slider%d"
- Data controllers: "Data%d"

where %d represents a positive integer number ranging between 1 and MAX_CONTROLLER (defined in *CFront.h*). This means essentially that all controller names must end with a unique number.

Label Controller

Labels are not interactive so they are not real controllers (just a simple label on the GUI). No callback is needed.

Parameters for instancing:

- Control Page identifier (it is recommended to be always 0, since currently it is the only control page)
- Grid number: The available horizontal space is split into sections with the number of points defined
- Start point (where the label starts, minimum is 0)
- End point (where the label ends, maximum is the grid number defined above)
- The line the label resides on (0 – 7)
- The height of the label (lines)
- Normal text
- Selected text

Sample:

```
PLUGIN_API CLabel Label1(0,8,0,1,0,1,"Some text","Some text");
```

Push Button Controller

Its callback function is called when the button is pushed. Does not have on and off states.

Parameters for instancing:

- Control Page identifier (it is recommended to be always 0, since currently it is the only control page)
- Initial value
- Callback function
- Grid number: The available horizontal space is split into sections with the number of points defined
- Start point (where the label starts, minimum is 0)
- End point (where the label ends, maximum is the grid number defined above)
- The line the label resides on (0 – 7)
- The height of the label (lines)
- Normal text
- Selected text

Sample:

```
PLUGIN_API CPushButton PushButton1(0,0,callback,10,6,7,6,1,"start","start");
```

Boolean Button Controller

Its callback function is called when the button is pushed. It has on and off states as well.

Parameters for instancing:

- Initial value (true or false)
- Callback function
- Control Page identifier (it is recommended to be always 0, since currently it is the only control page)
- Grid number: The available horizontal space is split into sections with the number of points defined
- Start point (where the label starts, minimum is 0)
- End point (where the label ends, maximum is the grid number defined above)
- The line the label resides on (0 – 7)
- The height of the label (lines)
- Normal text
- Selected text

Sample:

```
PLUGIN_API CBooleanButton BooleanButton1(FALSE,callback, 0, 10,8,10,5,1,"normaltxt","selec
tedtxt");
```

Radio Button Controller

Its callback function is called when the button is pushed. Radio buttons have on and off states as well, but only one button in the same group can be active at once.

Parameters for instancing:

- Initial value (true or false, but set true only one button)

- Callback function
- Group identifier (only one button can be true in one group)
- Control Page identifier (it is recommended to be always 0, since currently it is the only control page)
- Grid number: the available horizontal space is split into sections with the number of points defined
- Start point (where the label starts, minimum is 0)
- End point (where the label ends, maximum is the grid number defined above)
- The line the label resides on (0 – 7)
- The height of the label (lines)
- Normal text
- Selected text

Sample:

```
PLUGIN_API CRadioButton
RadioButton1(TRUE,callback,0,0,8,2,3,0,1,"text1","text1");
```

Slider Controller

The slider can be used to specify numerical – integer or float – values interactively. Sliders are horizontal on the GUI.

Parameters for instancing:

- Initial value
- Minimal value
- Max value
- Step value (the smallest increment)
- Format string to display the current value of the slider (for example "%.2f" to display float numbers)
- Callback function
- Control Page identifier (it is recommended to be always 0, since currently it is the only control page)
- Grid number: the available horizontal space is split into sections with the number of points defined
- Start point (where the label starts, minimum is 0)
- End point (where the label ends, maximum is the grid number defined above)
- The line the label resides on (0 – 7)
- The height of the label (lines)
- Type of resolution dependency. Indicates if a slider controller depends on image size and/or position. These values are needed for handling the image coordinates correctly when using the "Pan and Scan","Anamorphic" and "Full-Half Resolution".

Resolution dependency	Possible values
NON_DEP	The slider value does not depend on the image size and position.

Resolution dependency	Possible values
RES_DEP	The slider value depends on the image dimensions. For example: If this value is 100 in the half resolution and then you switch to full resolution, this value will be compensated for by the full resolution, and the value will be 200.
X_COORD	The slider with this attribute is treated as a horizontal image x coordinate. So, for example, if you use "Pan and Scan", then the x coordinate will be compensated to the point to the same pixel before the pan and scan. So, this slider depends on the width of the image and the x position of the image.
Y_COORD	Same as the X_COORD, except that it refers to a vertical image coordinate. So, this slider depends on the height of the image and the y position of the image.
X_SIZE	Use this if your slider represents a horizontal size value. This means that this value depends on the width of the image. For example: blur x kernel size.
Y_SIZE	Same as the X_SIZE, except that it refers to a vertical size. This means that this value depends on the height of the image.

NOTE If you want to use controllers for denoting image coordinates, set the controllers to X_COORD or Y_COORD. If any parameters depend on image dimensions, use X_SIZE or Y_SIZE. For example: x or y kernel size of a blur effect. If you do not want to compensate the parameter for "Pan and Scan" or "Unsqueeze", use the RES_DEP attribute.

- Normal text
- Selected text
- Type of interaction, it may be:

Interaction type	Processing method
PROCESS_WHILE_DRAGGING	The image processing will be computed continuously while dragging a slider. Use this only if your image processing code is fast.
PROCESS_AFTER_RELEASE	The image processing will be computed ONLY at the end of the slider dragging. Use this if the image processing code is slow in order to prevent loss of interactivity.

Sample:

```
PLUGIN_API Cslider Slider1(0,-0.2f,0.2f,0.1f,"%f",Callbk, 0,9,2,5,1,1,
RES_DEP,"txt","txt",PROCESS_AFTER_RELEASE);
```

Data Controller

This is not really a controller, as it has no representation on the GUI. Use this controller if you want to define values which must be stored even when a shot change happens. Presently ONLY the controller values (i.e. instances of the base controller class and its subclasses) will be stored when a shot change occurs. The other member variables of the plug-in will lose their values and cannot be restored automatically (i.e. it is the developers exercise to maintain the state of these variables). Its value is also saved in the grade file, so for the time being this is the only way to save any data you want. No callback is needed.

Parameters for instancing:

- Type of resolution dependency (for details see the Slider controller)

Sample:

```
PLUGIN_API CData Data1(RES_DEP);
```

Example Plug-in File Structure

Your header file should look like this. Bold indicates names that you define.

```
#ifndef __defined_name_you_like__
#define __defined_name_you_like__

#ifdef PLUGIN_EXPORTS
#define PLUGIN_API extern "C" __declspec(dllexport)
#else
#define PLUGIN_API extern "C" __declspec(dllimport)
#endif

#ifdef PLUGIN_EXPORTS
#define CPLUGIN_API __declspec(dllexport)
#else
#define CPLUGIN_API __declspec(dllimport)
#endif

#include "CFront.h"

// This class is exported from the dll
class CPLUGIN_API YourPlugin : CPlugin
{
private:

// add your private methods here.

public:
  YourPlugin (void);
  ~YourPlugin (void);


  void Init();
  void Interact(int event_type, int data, float sx, float sy);
  void Overlay();
  void Process();
  void PreProcess();
  void PostProcess();

// add your public methods here.

};

PLUGIN_API YourPlugin *CreateCPlugin();

PLUGIN_API void DeleteCPlugin(CPlugin *pPlugin);
// add your callback functions


void sampleCallbackFunction(CPlugin *p);

// for multithreaded processing the image processing code is in a separate method

void MyMultiThreadProcess(unsigned long *start,unsigned long *size, your_type *passed_data);
```

```
#endif /* #ifndef __defined_name_you_like__ */
```

The *.c* file corresponding to the header above looks like this:

```
#include "YourPlugin.h" // header of this file

////////////////////////////////////////////////
// Every plugin MUST CONTAIN these 3 functions!!! //
////////////////////////////////////////////////

PLUGIN_API CTrackerPlugin *CreateCPlugin()
{
  return new CTrackerPlugin;
}
PLUGIN_API void DeleteCPlugin(CPlugin *pPlugin)
{
  delete pPlugin;
}
PLUGIN_API float GetPluginVersion(CPlugin *pPlugin)
{
  return(PLUGIN_API_VERSION_NUMBER);
}


// Constructor of the plugin
YourPlugin::YourPlugin()
{}
// Destructor of the plugin
YourPlugin::~YourPlugin()
{}

//   GUI CONTROLLER VARIABLES, YOU CAN DEFINE YOUR GUI ELEMENTS HERE
// Defining a label:
PLUGIN_API CLabel Label1(0,8,0,1,0,1,"Some text","Some text");

//   Defining a radio button:
PLUGIN_API CRadioButton RadioButton2(FALSE, someCallbackMethod,0,0,8,1,2,0,1,"text1","text2");

//   Defining a push button:
PLUGIN_API CPushButton
PushButton1(0,0,someCallbackMethod,10,6,7,6,1,"text1","text2");

//   Defining a boolean button:
PLUGIN_API CBooleanButton BooleanButton1(FALSE, someCallbackMethod, 0, 10,8,10,5,1,"work","not
 work");

//   Defining a slider:
PLUGIN_API Cslider Slider1(0,-0.2f,0.2f,0.1f,"%f", someCallback, 0,10,2,5,1,1, RES_DEP,
"txt", "txt",
PROCESS_AFTER_RELEASE);
```

```
//    CALLBACK FUNCTIONS
// this sample callback function shows how to call a method of the plugin class

void someCallbackMethod(CPlugin *plugin)
{
  YourPlugin *myplugin=dynamic_cast<YourPlugin *>(plugin); myplugin->someFunctionOfYourPlu
ginClass();
}


//   API SPECIFIC METHODS
void YourPlugin::Init()
{}
void YourPlugin::Interact(int event_type, int data, float sx, float sy)
{
  float vx,vy,XZoom,YZoom,XPicPos,YPicPos;

  CTime Time;
  CImage Image;

  //   Querying the current Zoom Factors and picture position coordinates
  GetZoom(&XZoom,&YZoom);
  GetPicPos(&XPicPos,&YPicPos);

  vx = (-XPicPos+sx)/XZoom; // set the actual coordinates of the mouse pointer
  vy = (-YPicPos+sy)/YZoom;


  switch(event_type)
  {
  case BUTTONPRESS_EVENT:
   if(data & Cplugin::BUTTON_LEFT) // this example shows how to detect the press event of
the left button
   {
   }
   break;
  case BUTTONRELEASE_EVENT:
   break;
  case MOTIONNOTIFY_EVENT:
   break;
  default:
   break;
  }
}

void YourPlugin::Overlay()
{
  float XPicPos,YPicPos;
  float XZoom,YZoom;

  GetZoom(&XZoom,&YZoom);
  GetPicPos(&XPicPos,&YPicPos);

  // add your openGL code here
}
```

```
void YourPlugin::Process()
{
  RGB16BIT *InBuff,*OutBuff;
  OutBuff = m_Output;    // The destination buffer
  InBuff = m_Input;     // The source buffer

  int number_of_columns = m_Output.m_width; // width and height of the image
  int number_of_lines = m_Output.m_height;

  // if you want to use more processors use MpCallThread
  // in this example one parameter is passed to MyMultiThreadProcess,
  // thus all the desired data (for example the value of sliders) is
  // stored in a custom structure - which you must define yourself
  your_type *passed_data;
  passed_data.value1_to_be_passed=1;
  passed_data.value2_to_be_passed=2;
  MpCallThread((void (*)())(MyMultiThreadProcess),number_of_lines,1, &passed_data);


  // if you don't want to alter the image at all
  memcpy(OutBuff,m_Input,m_Output.m_pixels*sizeof(RGB16BIT));
}

void YourPlugin::PreProcess()
{}
void YourPlugin::PostProcess()
{}



void MyMultiThreadProcess(unsigned long *start,unsigned long *size, your_type *passed_data)
{
   // image processing code (usually for lines starting at *start until *start+*size)
}
```