IntelligentSource API

User Guide

Trademark Information

Copyright Information

Document Revision

Date	Document edition	Software version	Revision details	Author/Editor
September 2020	1	Delta 2.7	New release	Tom Harvey
September 2020	2	Delta 2.7	Updated Frame Buffer Format	Tom Harvey
October 2020	3	Delta 2.7	Addition of RGBA8 Encode/Decode Method	Tom Harvey

M450-3

www.7thsense.one
info@7thsense.one

Contents

Overview

IntelligentSource is an API that allows the user to render pixel data to their output displays. If the source then fails or freezes, the API will detect this change and return a message to Delta to trigger an action such as switching to a back-up source or playing content on a reserve timeline.

This API allows the developer to make calls to create the pixel data which is then rendered on screen. They can then make calls to encode the pixel data as their source is being displayed. When the source fails, calls to encode the pixel data can be halted, which results in sending Delta a trigger message.

How to use

In your application, first make a call to `_IntelligentSourceInit()`. This will initialise an instance of the API and return the instance ID required for following calls to the API. After initialising the API, make a call to `_IntelligentSourceStart(const unsigned int inst)`. You will need to pass in the instance ID returned from the original call.

With the API initialised and the IntelligentSource process started, you can now find the data size of the pixel array. There are options dependant on your display. For an 8 bpp display, call `_IntelligentSourceGetEncodedDataSize8 bpp(const unsigned int inst)`. There are also options for dual-channel 16 bpp formats and three-channel 24 bpp formats. The returned value from this call will be an unsigned int which is representative of the size of the pixel array. For this example, we shall call the returned value 'sizeOfPixelData'. The next step is to get the overall display pixel value. For example, on an HD display this would be 1920 × 1080 = 2073600. Using this value, create a char pointer representative of this size. For this example we will call the pointer 'pPixelData'.

As the frame count of the source now increments, you will call `_IntelligentSourceNewFrame(const unsigned int inst)` on every frame with the IntelligentSource instance ID as the only parameter. This will create a new IntelligentSource frame and return that frame number. You also need to call `_IntelligentSourceEncodeData8 bpp(const unsigned int inst, unsigned char *pPixelData, unsigned int sizeOfPixelData)` on every frame. This will encode the pixel data and change the pPixelData to update the rendered pixels on screen. You need to pass in your IntelligentSource instance ID as the first parameter, the pPixelData pointer you created which will hold the data, and then the sizeOfPixelData.

In your application you can render this pixel data into your display texture. If for some reason your source fails, you should no longer call `_IntelligentSourceEncodeData8 bpp(const unsigned int inst, unsigned char *pPixelData, unsigned int sizeOfPixelData)` or `_IntelligentSourceNewFrame(const unsigned int inst)` on every frame. Now that you are no longer calling these functions, as the source has failed, you no longer encode and update the pPixelData or increment the IntelligentSource frame count. Internally, this will send a message to Delta to trigger some action. A common use case of IntelligentSource is to monitor capture inputs. If one capture source fails, Delta will receive a message and display a back-up capture source instead.

There are other useful functions such as `_IntelligentSourceGetFrameCount(const unsigned int inst)`. This function will return the overall IntelligentSource frame count. If the pPixelData stops being updated and encoded, this will return the last incremented IntelligentSource overall frame count. You can also call the function `_IntelligentSourceGetFrameTimeMS(const unsigned int inst)`, which will return the frame time in milliseconds for the IntelligentSource frame count.

When closing your application, ensure that within the destructor you call `_IntelligentSourceStop(const unsigned int inst)`. This will stop the IntelligentSource process. The secondary call to make in the destructor is `_IntelligentSourceDestroy(const unsigned int inst)`. This will destroy the instance of IntelligentSource.

API Calls

```
void _IntelligentSourceGetVersion(int &major, int &minor, char &type);
```

Call to retrieve the version of IntelligentSource.

```
unsigned int _IntelligentSourceInit();
```

Call to initiate an instance of the IntelligentSource API. If this fails to initialize, '0' is returned. If not, the instance ID which you will use for following calls will be returned.

```
_IntelligentSourceDestroy(const unsigned int inst);
```

Call to destroy an instance of the IntelligentSource API. By default all running instance of the dll will be destroyed. However, if an instance ID is passed in, only this instance of the dll will be destroyed.

```
bool _IntelligentSourceStart(const unsigned int inst);
```

Call to start the IntelligentSource process. This should be called after initializing an instance of the API. The instance ID of your initialized dll should be passed in.

```
bool _IntelligentSourceStop(const unsigned int inst);
```

Call to stop the IntelligentSource process. The instance ID of your initialized API should be passed in.

```
unsigned __int64 _IntelligentSourceNewFrame(const unsigned int inst);
```

Call to retrieve a new IntelligentSource frame. The instance ID of your initialized dll should be passed in. This will return the incremented IntelligentSource frame count. This should be called on every frame that the source increments and is still alive.

Single Channel 8 bpp Format

```
unsigned int _IntelligentSourceGetEncodedDataSize8 bpp(const unsigned int inst);
```

Call to get the encoded size of the pixel array, assuming 1 byte per pixel. The returned value is used as the third parameter in the call to encode the 8 bpp data.

```
unsigned int _IntelligentSourceEncodeData8 bpp(const unsigned int inst, unsigned char
*pPixelData, unsigned int sizeOfPixelData);
```

Call to encode the data as 8 bpp and place the result in *pPixelData. This function should be called every frame that the source count increments (if in 8 bpp mode).

Dual-Channel 16 bpp Format

```
unsigned int _IntelligentSourceGetEncodedDataSizeYUV422(const unsigned int inst);
```

Call to get the encoded size of the pixel array, assuming 3 bytes per pixel. The returned value is used as the third parameter in the call to encode the 16 bpp data.

```
unsigned int _IntelligentSourceEncodeDataYUV422(const unsigned int inst, unsigned
char *pPixelData, unsigned int sizeOfPixelData);
```

Call to encode the data as 16 bpp and place the result in *pPixelData. This function should be called every frame that the source count increments (if in 16 bpp mode).

Three-Channel 24 bpp Format

```
unsigned int _IntelligentSourceGetEncodedDataSizeRGB8(const unsigned int inst);
```

Call to get the encoded size of the pixel array, assuming 3 bytes per pixel. The returned value is used as the third parameter in the call to encode the 24 bpp data.

```
unsigned int _IntelligentSourceEncodeDataRGB8(const unsigned int inst, unsigned char
*pPixelData, unsigned int sizeOfPixelData);
```

Call to encode the data as 24 bpp and place the result in *pPixelData. This function should be called every frame that the source count increments (if in 24 bpp mode).

Three-Channel 30 bpp Format

```
unsigned int _IntelligentSourceGetEncodedDataSizeRGB10(const unsigned int inst);
```

Call to get the encoded size of the pixel array, assuming 3 bytes per pixel. The returned value is used as the third parameter in the call to encode the 30 bpp data.

```
unsigned int _IntelligentSourceEncodeDataRGB10(const unsigned int inst, unsigned char
*pPixelData, unsigned int sizeOfPixelData);
```

Call to encode the data as 30 bpp and place the result in *pPixelData. This function should be called every frame that the source count increments (if in 30 bpp mode).

Three-Channel 32 bpp Format

```
unsigned int _IntelligentSourceGetEncodedDataSizeRGBA8(const unsigned int inst);
```

Call to get the encoded size of the pixel array, assuming 4 bytes per pixel. The returned value is used as the third parameter in the call to encode the 32 bpp data.

```
unsigned int _IntelligentSourceEncodeDataRGBA8(const unsigned int inst, unsigned char
*pPixelData, unsigned int sizeOfPixelData);
```

Call to encode the data as 32 bpp and place the result in *pPixelData. This function should be called every frame that the source count increments (if in 32 bpp mode).

```
unsigned __int64 _IntelligentSourceGetFrameCount(const unsigned int inst);
```

Call to retrieve the overall IntelligentSource frame count.

```
float _IntelligentSourceGetFrameTimeMS(const unsigned int inst);
```

Call to retrieve the frame time in milliseconds for an IntelligentSource frame.

E: info@7thsense.one
W: 7thsense.one

7thSense Design Ltd

2 The Courtyard Shoreham Road
Upper Beeding
Steyning
West Sussex
BN44 3TN
UK

T: +44 (0) 1903 812299

7thSense LLC

4207 Vineland Rd
Suite M1
Orlando, FL 32811
USA

T: +1 407 505 5200