# Introduction

The purpose of this document is to describe the design of the user input device plugin template that will be used to create dynamic link libraries for the VSA S/W application. A user input device plugin will be created for each user input device supported by the VSA application which includes the following devices:

* Mouse
* Keyboard
* Joystick
* VCB
* Touch screen
  + Avalex
  + Atlas
  + GDC SDU-16

Each user input device plugin will be composed of an aggregation of 4 required interfaces implemented by plugins (IDataStream, IDataLink, IGraphicsEvent, and IUserInputDevice) and 1 optional interface (ICommandPlugin) that is also implemented by a plugin. In addition, each user input device plugin is expected to launch at least one thread that handles receiving data from the user input device. In cases where data flow between the VSA application and the user input device is bidirectional, the user input device plugin may launch an additional thread to handle transmitting messages to the device.

# Component Hardware

User input device plugin modules shall be made to support the following user input devices:

* Mouse
* Keyboard
* Touch screen:
  + Avalex
  + Atlas
  + GDC SDU-16
* VBC
* Joystick

Since user input devices utilize many different communication buses, the user input device plugins shall use plugin interface modules to communicate with the input device. Communication plugin modules shall be made to support the following communication buses:

* RS-232
* RS-422
* RS-485
* USB
* Ethernet

# Component Software

## Device communication plugins.

Communication to a user input device shall be implemented using 2 interfaces corresponding to the OSI communication model layers 1 (Physical) and OSI layer 2 (Data Link) or 4 (Transport):

### IDataStream – OSI Layer 1 (Physical):

namespace: NAMESPACE\_COMMUNICATION

bool Load( VPluginConfigurationPtr pConfig, VPluginLibraryPtr pLib )

IPluginLoadConfiguration Load() method override. The implementation of this method will load all member data from the XML file that defines the plugin. For serial data streams, this includes:

* Serial port name – e.g. “COM1”, “/dev/tts/1”
* Baud rate
* Parity
* Stop bits
* Data bits
* Flow control

For TCP/IP networks, the data includes:

* IP address – e.g. “192.168.0.100”
* IP port number – e.g. 2000

Returns:

* true = Success
* false = Failure

NResult Initialize( void )

Perform any initialization to prepare to connect to the data stream. The Initialize() function shall not attempt to connect to the input device during initialization.

Returns:

0 : Success

-1 : Failure

NResult Connect( void )

This function performs the initial connection to the communication bus. This function will execute in the context of the user input device RX thread and may contain an infinite wait to establish the physical connection to the communication bus.

Returns:

0 : Success

-1 : Failure

NResult Read( void \*pBuf, int maxBytes )

Receive up to nBytes of data from the device communication bus. If the interface is not currently connected to the communication bus, the Read() function shall automatically attempt to re-establish the physical layer communication to the device.

Returns:

>0 : Success, number of bytes placed in the read buffer

<= 0 : Failure

NResult Write( void \*pBuf, int nBytes )

Transmit nBytes of data from pBuf via the communication bus to the input device.

Returns:

>0 : Success, number of bytes written

<= 0 : Failure

### IDataMessage – Device Data Message Interface:

The Device Data Message interface performs the message construction and transmission protocol checks of the data received from the IDataStream interface. This interface shall contain an IDataStreamPtr member variable that holds the physical stream interface to the communication bus object.

namespace: NAMESPACE\_COMMUNICATION

bool Load( VPluginConfigurationPtr pConfig, VPluginLibraryPtr pLib )

IPluginLoadConfiguration Load() method override. The implementation of this method will load all member data from the XML file that defines the plugin.

Returns:

* true = Success
* false = Failure

NResult Initialize( void )

Perform any initialization that needs to be done prior to prepare to connect to the data stream. This includes querying for the IDataStream interface to use for communicating with the user input device and initializing the data stream:

m\_pDataStream = static\_case<IDataStream \*>(QueryInterface(IXID\_IDataStream, &nr));

if (m\_pDataStream)

{

m\_pDataStream->Initialize();

m\_pDataStream->Release();

}

Returns:

0 : Success

-1 : Failure

NResult Connect( void )

This function performs the initial connection to the communication bus. This function will execute in the context of the RX thread and may contain a infinite wait to establish the physical connection to the communication bus. The IDataMessage::Connect() must execute the IDataStream::Connect() of the associated physical data stream communication bus.

Returns:

0 : Success

-1 : Failure

NResult ReadMessage( void \*pMsg, int maxMsgSize )

This function reads a message from the input device. The function uses the IDataStream interface to access the input device data stream. This function is responsible for constructing a message and performing protocol validation – e.g. checksum validation.

Returns:

>0 : nBytes put in pBuf (size of message read)

-1 : Failure

NResult WriteMessage( void \*pMsg, int msgSize )

This function writes a message to the input device. The function uses the IDataStream interface to access the input device data stream.

Returns:

N : Number of byte written

<0 : Failure

## Graphics Event Plugin

namespace : NAMESPACE\_GRAPHICS\_EVENT

### IGraphicsEvent

namespace : NAMESPACE\_GRAPHICS\_EVENT

typedef enum

{

BUTTON\_UP,

BUTTON\_DOWN,

KEY\_UP,

KEY\_DOWN,

MOUSE\_UP,

MOUSE\_DOWN,

MOUSE\_MOVE,

DATA\_AVAILABLE,

VIDEO\_CNTL,

CAMERA\_CNTL,

EXT\_CMD,

USER\_INPUT\_DEVICE\_STATUS,

LAST\_GRAPHICS\_EVENT

} tGraphicsEventType;

typedef struct graphics\_event

{

tGraphicsEventType eventType;

unsigned long eventData;

unsigned char eventArray[64];

} tGraphicsEvent;

typedef enum

{

DEVICE\_STATUS\_UNKNOWN,

DEVICE\_CONNECTED,

DEVICE\_DISCONNECTED,

DEVICE\_COMM\_ERROR,

DEVICE\_STATUS\_LAST

} tUserInputDeviceStatus;

bool Load( VPluginConfigurationPtr pConfig, VPluginLibraryPtr pLib )

IPluginLoadConfiguration Load() method override. The implementation of this method will load all member data from the XML file that defines the plugin.

Returns:

* true = Success
* false = Failure

NResult Initialize( int graphicsThreadID )

Perform any initialization that needs to be done prior to prepare to establishing an event connection to the graphics thread.

graphicsThreadID : Graphics thread ID. For the OmniScape, the ID is also the DSP index (0 – 3).

Returns:

0 : Success

-1 : Failure

NResult Connect( void )

Establishes the initial connection graphics thread. This function will execute in the context of the user input device RX thread and may have an infinite wait to connect to the graphics thread.

Returns:

0 : Success

-1 : Failure

NResult SendMouseEvent( tGraphicsEventType mouseEvent, int x, int y )

NResult SendButtonEvent( tGraphicsEventType buttonEvent, int buttonID )

NResult SendKeyEvent( tGraphicsEventType keyEvent, void \*pKey, int sizeKeyData )

NResult SendDataAvailableEvent()

NResult SendVideoControlEvent( int length, void \*pVideoCntlData )

NResult SendCameraControlEvent( int length, void \*pCameraCntlData )

NResult SendUserInputDeviceStatusEvent(tUserInputDeviceStatus status, int deviceID)

## ICommandPlugin: Optional special commands

namespace : NAMESPACE\_GADGET

bool Initialize( CApplicationGaugePtr pApplication, VCommandSetupPtr pCommandSetup )

bool ProcessRequest( VDictionary::Selector selCommand, ConstCVariantDataRef vData )

## IUserInputDevice: User input device interface

namespace : NAMESPACE\_USER\_INPUT\_DEVICE

bool Load( VPluginConfigurationPtr pConfig, VPluginLibraryPtr pLib )

IPluginLoadConfiguration Load() method override. The implementation of this method will load all member data from the XML file that defines the plugin.

Returns:

* true = Success
* false = Failure

NResult Initialize( int graphicsThreadID, int userInputDeviceID )

The Initialize() method shall use the QueryInterface() method to set the following member interface pointers:

1. IDataMessage : m\_pDevice – pointer to the input device data message interface.
2. IGraphicsEvent : m\_pGraphicsEvent – pointer to the VSA graphics event interface.

graphicsThreadID : ID of the associated graphics thread which will process the user input device events.

userInputDeviceID : ID of the user input device. This value is typically the index for the input device in the user input device array maintained by the device manager.

Returns:

0 : Success

-1 : Failure

General Algorithm:

NResult Initialize( int graphicsThreadID,int userInputDeviceID )

{

NResult rtnResult = 0;

NResult nr;

...

m\_graphicsThreadID = graphicsThreadID;

m\_inputDeviceID = userInputDeviceID;

// Get IDataLink interface

m\_pDevice = static\_cast<IDataLink \*>(QueryInterface(IXID\_IDataMessage, &nr));

if (m\_pDevice)

{

m\_pDevice->Initialize();

}

else

{

rtnResult = -1;

}

// Get the Graphics Event interface

m\_pGraphicsEvent = static\_cast<IGraphicsEvent\*>(QueryInterface(IXID\_IGraphicsEvent,&nr);

if (m\_pGraphicsEvent)

{

m\_pGraphicsEvent->Initialize(graphicsThreadID);

}

else

{

rtnResult = -1;

}

...

if (rtnResult == 0)

{

// Launch RX thread

// Launch TX thread (if needed)

}

return nResult;

}

### RX Thread Algorithm:

void RxThread(void)

{

NResult nr = -1;

tDeviceMsg msg;

tUserInputDeviceStatus deviceStatus = DEVICE\_DISCONNECTED;

...

do

{

nr = m\_pGraphicsEvent->Connect();

if (nr < 0)

{

sleep(1);

}

} while (nr < 0);

m\_pGraphicsEvent-> SendUserInputDeviceStatusEvent(deviceStatus, m\_inputDeviceID);

nr = -1;

do

{

nr = m\_pDevice->Connect();

if (nr < 0)

{

sleep(1);

}

} while (nr < 0);

deviceStatus = DEVICE\_CONNECTED;

m\_pGraphicsEvent-> SendUserInputDeviceStatusEvent(deviceStatus, m\_inputDeviceID);

...

while (1)

{

nr = m\_pDevice->ReadMessage(&msg, sizeof(msg));

if (nr > 0)

{

// Process message: coordinate transforms, filter, ...

//

switch (msg.type)

{

case UP\_ARROW\_PRESS:

m\_pGraphicsEvent->SendKeyEvent(KEY\_DOWN, (void \*)msg.data, 2);

break;

...

}

if (deviceStatus != DEVICE\_CONNECTED)

{

deviceStatus = DEVICE\_CONNECTED;

m\_pGraphicsEvent-> SendUserInputDeviceStatusEvent(deviceStatus,

m\_inputDeviceID);

}

}

else

{

deviceStatus = DEVICE\_COMM\_ERROR;

m\_pGraphicsEvent-> SendUserInputDeviceStatusEvent(deviceStatus,

m\_inputDeviceID);

}

}

...

}

# Input Device Manager (OmniScape)

Class that manages the graphics thread input devices. This class is project specific – i.e. separate implementations for OmniScape and Windows. On the VSA OmniScape application, the class shall be added to the graphics thread class – CVdds. The class shall contain a TArray of IUserInputDevicePtr. The class shall implement the following public method:

void Initialize( int graphicsThreadID,

CApplicationGaugeVDDSPtr pApplicationGauge,

CCommandProcessorPtr pCmdProc )

graphicsThreadID : ID for this graphics thread (OmniScape DSP index/ID)

pApplicationGauge : pointer to the VDDS application gauge object. This pointer is needed to load the XML data for the user input device(s).

Algorithm:

void Initialize( int graphicsThreadID,

CApplicationGaugeVDDSPtr pApplicationGauge

CCommandProcessorPtr pCmdProc )

{

...

ISupportsArray \*tmpList;

tmpList = factoryPlugin.Make(sPathName, &m\_tUserInputDevicePluginLibrary);

if (tmpList)

{

for (int i = 0; i < tmpList->Length(); i++)

{

ISupportsPtr pSupports = \*tmpList->PointItem(i);

m\_deviceList[i] = static\_cast<IUserInputDevice\*>

(pSupports-> QueryInterface(IXID\_IUserInputDevice, &nr));

if (m\_deviceList[i])

{

m\_deviceList[i].Initialize( graphicsThreadID, i );

}

ICommandPluginPtr pCmdPlugin;

pCmdPlugin = static\_cast<ICommandPlugin\*>

(pSupports-> QueryInterface(IXID\_ICommandPlugin, &nr));

if (pCmdPlugin)

{

pCmdProc->AddCommandPlugin( pSupports );

pCmddPlugin->Release();

}

pSupports->Release();

}

delete tmpList;

}

...

}

# XML

/dictionary/configuration0.xml :

<Item ID=”Input Devices” Type=”STRING”>input\_devices\_0</Item>

/plugins/inputdevices/input\_devices\_0.xml :

<Plugins Group=””>

...

<!-- Avalex touch screen input device -->

<Plugin Class=”AvalexTouchscreen” ID=”Touch Screen 0”>

<!--IDataStream plugin -->

<Plugin Class=”RS232VDDS”>

<Param Name=”Port” Type=”STRING”>/dev/tts/2</Param>

<Param Name=”Baud” Type=”INTEGER”>115200</Param>

<Param Name=”Parity” Type=”STRING”>None</Param>

<Param Name=”Stop Bits” Type=”INTEGER”>1</Param>

<Param Name=”Data Bits” Type=”INTEGER”>8</Param>

<Param Name=”Flow Control” Type=”STRING”>None</Param>

</Plugin>

<!--IDataLink plugin -->

<Plugin Class=”AvalexTouchscreenMsg”>

</Plugin>

<!--IGraphicsEvent plugin -->

<Plugin Class=”GraphicsEventVDDS”>

</Plugin>

</Plugin>

<!-- VCB input device -->

<Plugin Class=”VCB” ID=”VCB 0”>

<!-- Data Stream plugin -->

<Plugin Class=”RS232VDDS”>

<Param Name=”Port” Type=”STRING”>/dev/tts/3</Param>

<Param Name=”Baud” Type=”INTEGER”>115200</Param>

<Param Name=”Parity” Type=”STRING”>None</Param>

<Param Name=”Stop Bits” Type=”INTEGER”>1</Param>

<Param Name=”Data Bits” Type=”INTEGER”>8</Param>

<Param Name=”Flow Control” Type=”STRING”>None</Param>

</Plugin>

<!-- Device Data Message plugin -->

<Plugin Class=”VcbMsg”>

</Plugin>

<!-- VDDS Graphics Event plugin -->

<Plugin Class=”GraphicsEventVDDS”>

</Plugin>

<!-- Command Plugin -->

<Plugin Class=”CmdVcbBacklight” Group=”” ID=”VCB Backlight 0”>

</Plugin>

</Plugin>

...

</Plugins>

# Directory Structure:

## UserInputDevices – VSA subdirectory.

Contains the following source files:

1. IUserInputDevice.h – User input devices interface definition
2. NamespaceUserInputDevices.h

Subdirectories under UserInputDevices for each input device implementation (all subdirectories, projects, and binaries are prefixed with ‘Input’). For example:

* InputAvalexEloVDDS
* InputAtlasVDDS
* InputVcbVDDS

## GraphicsEvent – VSA subdirectory

Contains the following source files:

1. IGraphicsEvent.h
2. NamespaceGraphicsEvent.h ?

Subdirectories under GraphicsEvent for each plugin implementation (all subdirectories, projects, and binaries are prefixed with ‘Event’):

* GraphicsEventVDDS
* GraphicsEventWin

## Communication – VSA subdirectory

Contains the following source files:

* IDataMessage.h
* IDataStream.h
* NamespaceCommunication.h?

Subdirectories under Communication are:

* DataMessage
  + Subdirectories under Message are for each Data Message plugin implementation (all subdirectories, projects, and binaries are prefixed with ‘Msg’):
    - MsgAvalexElo
    - MsgVcb
    - MsgAtlas
* DataStream
  + Subdirectories under DataStream are for each Data Stream plugin implementation (all subdirectories, projects, and binaries are prefixed with ‘Comm’):
    - CommSerialVDDS