WASP3D

Shotbox Implementation

Introduction

The Wasp3D SDK helps developers create custom solutions for Broadcast & Interactive graphics. Live Event Graphics Developers can use the API to data-bind wasp scenes to various sources of data, scoring applications, wire feeds and databases and push it to the Wasp3D rendering engine - Sting Server for on-air playout. The Shotbox implementation document helps customers load and playout the wasp scene on the real-time rending server – Sting Server.

The Wasp3D Shotbox is a simple API that gives users a high level control over the Sting Server. The API provides classes for developers to load and meld data with wasp design templates on Sting Server. The API wraps the underlying protocol for communication through helper functions, enabling developers to create custom solutions for Broadcast & Interactive graphics. You can connect to multiple Sting-Server(s) using different communication channels. You can create from simple to complex applications managing multiple scenes on multiple Sting-Server(s) from a single application using WASP3D Shotbox SDK. The Shotbox implementation document helps customers load and playout the wasp scene on the Sting Server.

Shotbox Features

- Connect or disconnect from a Sting server on the wasp network.
- Load and Unload wasp scenes on the Sting Server.
- Play and Pause wasp scenes on the Sting Server.
- Update data of wasp scenes at run-time on the Sting server, ensuring on-air data changes.
- Receive update/acknowledgement information from the Sting server.

Shotbox Components

Link Manager

The *LinkManager* class is the top most layer in the WASP3D Shotbox SDK. You should create only one instance of this class in your application. This class is responsible for creating and managing instances of Link class.

Link

Link class is responsible for communication with the Sting-Server. It is also responsible for creating an instance of Shotbox class. An instance of Link is created through *LinkManager* class.

Shotbox

Shotbox class instance represents a scene on a Sting Server. You can create multiple instances of Shotbox class in a project. *Shotbox* class exposes various methods to manage the wasp scene on Sting Server. It also exposes a variety of events to notify the status of scene on Sting Server.

Creating an instance of Link Manager.

```
LinkManager lnkManager = null;

public void CreateLinkManager()
{
lnkManager = new LinkManager();
}
```

Creating an instance of Link

```
Link  link = null;
public void CreateLink()
{
string linkid = null;
link = lnkManager.GetLink(LINKTYPE.TCP, out linkid);
}
```

Connecting to Sting Servers.

The following sample code illustrates the process to connect with a Sting-Server (IP 192.168.1.151) and listening on port: 7001 using an instance of Link class. Developers may handle OnEngineConnected event of link class to verify a successful connection with the Sting-Server.

```
public void Connect()
{
link.Connect("net.tcp://192.168.1.151:7001/TcpBinding/WcfTcpLink");   }
}
```

Creating an instance of Shotbox

To load a scene on Sting Server, you have to create an instance of ShotBox class. In the following sample developer is loading a scene (GettingStarted.wsl) available on X: (drive letter).

```
ShotBox shotbox = null;
public void CreateShotbox()
{
string    sgxml = null; string   shotBoxID = null;
bool      isTicker = false;

Util.getSGFromWSL(@"X:\GettingStarted.wsl", out sgxml);
if (!string.IsNullOrEmpty(sgxml))
{
shotbox = link.GetShotBox(sgxml, out shotBoxID, out isTicker) as ShotBox;
}
}
```

Loading a Scene

Developer can handle the events of ShotBox class instance to receive the current state/ update of scene from Sting-Server.

***Note**: The drive/directory where the scene is located should be accessible by the Sting-Server.*

```
public void LoadScene()
{
if (shotbox != null)
{
shotbox.SetEngineUrl("net.tcp://192.168.1.151:7001/TcpBinding/WcfTcpLink"
);
(shotbox as IaddinInfo).Init(new InstanceInfo()
Type = "wsl",
InstanceId = string.Empty,
TemplateId = @"X:\GettingStarted.wsl", ThemeId = "default"
}
);


deleted etc.

//Handle the ShotBox events for maintaining the status of ShotBox
//User will receive the Status event of scene in the ShotbOx Status
event, i.e., prepared,

shotbox.OnShotBoxStatus += new

EventHandler<SHOTBOXARGS>(shotbox_OnShotBoxStatus);

//User will receive the controller and object events in
ShotControllerStatus event, i.e, playing, pause, pause infinite
shotbox.OnShotBoxControllerStatus += new
EventHandler<SHOTBOXARGS>(shotbox_OnShotBoxControllerStatus);

//User shall then call the prepare method to load the scene on Sting-
Server.
shotbox.Prepare("net.tcp://192.168.1.151:7001/TcpBinding/WcfTcpLink", 0,
RENDERMODE.PROGRAM);
}
}

void shotbox_OnShotBoxControllerStatus(object sender, SHOTBOXARGS e)
{
switch (e.SHOTBOXRESPONSE)
{
case SHOTBOXMSG.PLAYING: MessageBox.Show("Playing");
break;
case SHOTBOXMSG.PAUSEINFINITE: MessageBox.Show("Pause");
break;
}
}

void shotbox_OnShotBoxStatus(object sender, SHOTBOXARGS e)
{

if (e.SHOTBOXRESPONSE == SHOTBOXMSG.PREPARED)
isprepared = true;
}
```

Playing a scene

To play a scene on Sting Server developer should use Play method of instance of ShotBox class. It is must that developer should call all the actions like play, pause, etc. on the scene once the scene is prepared successfully on Sting-Server and its event is received.

```
public void PlayScene()
{

//Play method takes 2 boolean parameters.
//If first parameter is true, shotbox will initialize the scene on sting-
server otherwise not
//If second parameter is true, shotbox will play the scene from first
frame otherwise not

if (shotbox != null && isprepared )
shotbox.
```

Pause a scene

Developer may require pausing a scene playing on the Sting Server. Developer shall call the pause method to achieve the same.

```
public void PauseScene()
{
if (shotbox != null)
```

Resume a paused scene

To resume a paused scene from its current frame developer shall call the play method again with different parameters.

```
public void ResumeScene()
{
//Play method takes 2 boolean parameters.
//If first parameter is true, shotbox will initialize the scene on sting-
server otherwise not
//If second parameter is true, shotbox will play the scene from first
frame otherwise not if (shotbox != null && isprepared )
shotbox.Play(false, false);
}
```

Unloading a Scene

Once a scene has been played (and is no longer required to be played out), it should be unloaded from the sting server toensure that Sting Server memory usage is maintained. It is a method call to unload the scene from Sting-Server.

```
public void UnloadScene()
{
if (shotbox != null)
{
shotbox.DeleteSg();
shotbox.Dispose();
}
shotbox = null;
}
```

Disconnecting with the Sting-Server

Developer should disconnect with a Sting-Server, once the playout of the graphics is complete. It is recommended for developer to call the Disconnect method of Link class after the work of Link is complete. Developer may call the Disconnect method of LinkManagerclass when closing his/her application to remove all the reference of scenes and connections from Sting-Server.

```
public void Disconnect()
{
if (link != null)
{
link.DisconnectAll();
link.Dispose();
} if (lnkManager ! null)
{
lnkManager.DisConnect();
lnkManager.Dispose();
```

Taking Scene On-Air

Developer can call following method when he wish to go on-air.

```
shotbox.SetRender(true);
```

On-Air Update

Developer can update the data at the time of on-air calling the UpdateSceneGraph method of shotbox class.

```
TagData tagData;

 if (!Equals(shotbox, null))
 {
         tagData = new TagData();
         tagData.IsOnAirUpdate = true;
         tagData.SgXml = Util.getSGFromWSL(filepath);
         tagData.TagType = new DataTargetType[] {  DataTargetType.UserTag };
         tagData.UserTags = new string[] { "user tag" };
         tagData.Values = new string[] { "updated value" };
         tagData.Indexes = new string[] { "-1" };
                shotbox.UpdateSceneGraph(tagData);
 }
```

Page Handling

Developer need to handle onShotBoxControllerStatus of shotbox in order to handle paging.

```
shotbox.OnShotBoxControllerStatus += new
EventHandler<SHOTBOXARGS>(shotbox_OnShotBoxControllerStatus);

   void shotbox_OnShotBoxControllerStatus(object sender, SHOTBOXARGS e)
        {
            switch (e.SHOTBOXRESPONSE)
            {
                case SHOTBOXMSG.PAGEOUT:

                    break;

            }
}
```

Developer can handle controller available in paging scene with following shotbox methods

```
 shotbox.Controllers[0].Play();

shotbox.Controllers[0].Pause();

shotbox.Controllers[0].Stop();
```

Loading a Ticker Scene

Developer can handle the events of CTickerShotBox class instance to receive the current state/ update of scene from Sting-Server.

***Note**: The drive/directory where the scene is located, should be accessible by the Sting-Server.*

```
CTickerShotBox shotbox = null;
public void CreateShotbox()
{
string    sgxml = null; string   shotBoxID = null;
bool      isTicker = false;

Util.getSGFromWSL(@"X:\GettingStarted.wsl", out sgxml);
if (!string.IsNullOrEmpty(sgxml))
{
shotbox = link.GetShotBox(sgxml, out shotBoxID, out isTicker) as
CTickerShotBox;
}
}

public void LoadScene()
{
if (shotbox != null)
{
shotbox.SetEngineUrl("net.tcp://192.168.1.151:7001/TcpBinding/WcfTcpLink"
);
(shotbox as IaddinInfo).Init(new InstanceInfo()
Type = "wsl",
InstanceId = string.Empty,
TemplateId = @"X:\TickerScene.wsl", ThemeId = "default"
}
);

//Handle the ShotBox events for maintaining the status of ShotBox
//User will receive the Status event of scene in the ShotbOx Status
event, i.e., prepared,

shotbox.OnShotBoxStatus += new

EventHandler<SHOTBOXARGS>(shotbox_OnShotBoxStatus);



//User shall then call the prepare method to load the scene on Sting-
Server.
shotbox.Prepare("net.tcp://192.168.1.151:7001/TcpBinding/WcfTcpLink", 0,
RENDERMODE.PROGRAM);
}
}
```

Ticker Events

```
void m_objShotBox_OnShotBoxStatus(object sender, SHOTBOXARGS e)
        {
            switch (e.SHOTBOXRESPONSE)
            {
                case SHOTBOXMSG.TICKERPREPARED:
                    //ensures ticker is prepared
                    break;
                case SHOTBOXMSG.TICKERREADYTOPLAY:
                    if (tickerstopped)
                    {
                            m_objShotBox.Play();
                       tickerstopped = false;
                    }
                    break;


                case SHOTBOXMSG.TICKERSTARVING:
                    //feed the ticker

                    break;


            }
        }


//create instance of TagData class with data developer wants to be
displayed and call the feedstarvingticker method of CTickerShotBox class.

 shotbox.FeedStarvingTicker(SgXml, tagData, out feedId);
```

Set Ticker Direction

Developer can set the direction of ticker calling SetDirection method of shotbox class.

```
shotbox.SetDirection(TickerDirection.RIGHT_TO_LEFT);
```

Set Ticker Speed

Developer can set the speed of ticker calling SetSpeed method of shotbox class.

```
shotbox.SetSpeed("8");
```