openBOXware

Crash Course

21 Nov 2010

Lisbon

1. Introduction

The goal of the following document is to provide a quick and dirty introduction to the **openBOXware** framework, in order to speedily be able to develop application and extensions for the system, integrating with its multimedia capabilities.

openBOXware is an open-source multimedia framework, currently based on Mono/.NET, GStreamer and Qt. It provides a set of APIs that enable third-party add-in writers to easily leverage the framework's functionalities to stream multimedia data from and to a wide set of devices, including remote Internet TV providers on the Internet, content sharing services, local NAS or UPnP devices and many more.

The programming interface has been streamlined to ease exposure and consumption of multimedia resources, that can be easily shared between add-ins. The framework also enables third-party applications to be run and used by the user of the **openBOXware** enabled device.

2. Development

The `openBOXware` framework provides an API that allows third-parties to develop add-ins. Each add-in may expose one or more components, called "*extension-points*" which may be of the following kinds:

- **Applications:** have a graphical user interface the user can interact with. They can be started and terminated at runtime, either in fullscreen mode or by using a "sidebar" mode. In both modes, an application can display complex GUI elements by leveraging the power of the Qt libraries.

- **Media Sources:** expose a browsable tree of multimedia resources in an abstract fashion. The contents are provided to the framework and can be accessed by other add-ins (for instance an application) or the framework itself. Multimedia resources thus exposed can be played back by the framework.

- **Media Targets:** expose a multimedia capable device to the rest of the framework. Each `openBOXware` installation has a local media target that displays video on the default screen and renders audio on the default output device. Additional media targets can install remote devices, UPnP media centers on the local net and so on. The framework will then be able to stream multimedia resources to any media target.

2.1. Requirements

The framework has been tested on Linux systems (Ubuntu 10.04 and 10.10) with the following components:

- Mono 2.6,
- GStreamer 0.10.30,
- Qt 4.5,
- X11 / XV overlay.

At its current state, **openBOXware** has been tested only on x86 based architectures. However, the required components can be deployed to ARM devices (attempts at porting are currently in progress).

The requirements are on-par with the packages provided by all core MeeGo installations. Thus **openBOXware** should be able to run on most MeeGo devices.

2.2. Installation

A distributable package, containing the **openBOXware** framework, development files and documentation, can be downloaded from: http://www.openboxware.net/developers-area

Source code can be obtained by registering on the project's Trac server, thus enabling read-only access to its SVN repository.

2.3. Application Store

Add-ins developed by third-parties will be available for download on a dedicated application store (in development).

Several applications developed during a preview course on `openBOXware` are available, along with a set of tutorials.

3. Tutorials

The following tutorials will be shown during the presentation course you are attending. All source code that will be shown is listed in the following pages with some comments that were deemed helpful to understand the framework's APIs and functions.

For complete documentation and the full API reference, please check out the official website at http://www.openboxware.net/developers-area.

3.1. Hello World

Objective

This basic application will demonstrate how an add-in project is created, which are the basic files that must be included, how an add-in is loaded by the framework and how GUI elements are created.

Features

- Project creation in MonoDevelop,
- Including API references to openBOXware,
- Writing an application,
- Registering the application via manifest file.

HelloWorldApp.cs

```
using System;
using Qyoto;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.Gui;

namespace HelloWorld {

      public class HelloWorldApp : Application {

            public override bool SetupGui (GuiWidget container,
                                           GuiMode mode){

                  this.GetLogger().InfoFormat("Setup GUI call.");

                  var layout = new QVBoxLayout(container);
                  layout.AddWidget(new QLabel("Hello World!"), 0,
                                   (uint)Qt.AlignmentFlag.AlignHCenter);

                  return true;
            }
      }
}
```

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<addin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <name>Hello World</name>
  <description>Hello World application.</description>
  <author>openBOXware Team</author>
  <version>1.0</version>
  <iconPath>hello-icon.png</iconPath>
  <assemblyName>HelloWorld</assemblyName>
  <application qualifiedName="HelloWorld.HelloWorldApp">
     <gui>
           <fullscreen maintainSystemBar="true" />
     </gui>
  </application>
</addin>
```

3.2. Web Radio

Objective

This second add-in will expose a “media source” to the framework, allowing it to receive an incoming Web Radio stream and play it back using the default facilities provided by the core openBOXware installation.

The source stream is provided by an IceCast server, as an HTTP stream encoded using Ogg/Vorbis.

Features

- Using the MediaSource API,
- Exposing a simple multimedia resource using openBOXware's abstract representation of a content tree,
- Registering a media source via manifest file.

HttpMediaSource.cs

```
using System;
using OpenBoxWare.Interface;
using Qyoto;

namespace HttpMediaSource {

	public class HttpMediaSource : MediaSource {

		public override MediaSourceNode GetRootNode(){
			return new LiveSourceNode(this);
		}

		public override string Name {
			get {
				return "IceCast web radio";
			}
		}

		QPixmap _icon;

		public override QPixmap CreateIcon() {
			if(_icon == null){
				var url = this.GetContext().UrlService;
				_icon = new QPixmap(url.ResolveUrl("icon.png"));
			}
			return _icon;
		}
	}
}
```

LiveSourceNode.cs

```
using System;
using System.Collections.Generic;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaResources;

namespace HttpMediaSource {

      public class LiveSourceNode : MediaSourceNode  {

            MediaSource _source;
            public LiveSourceNode(MediaSource source) {
                  _source = source;
            }

            public override IEnumerable<MediaElement> MediaElements {
                  get {
                        return new MediaElement[] {
                              new MediaElement(new HttpMediaResource  {
                                    Uri = new
Uri("http://192.168.1.1:8000/stream.ogg")
                              }, MediaElementContent.OggContainer)
                        };
                  }
            }

            public override IEnumerable<MediaSourceNode> Children {
                  get { return null; }
            }

            public override bool HasChildren {
                  get { return false; }
            }

            public override MediaSource ParentMediaSource {
                  get { return _source; }
            }

            public override string Name {
                  get { return "Live web radio"; }
            }

            public override MediaSourceNodeType NodeType {
                  get { return MediaSourceNodeType.Channel; }
            }
      }
}
```

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<addin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <name>IceCast web radio media source</name>
  <description>Media source for an IceCast web radio.</description>
  <author>openBOXware Team</author>
  <version>1.0</version>
  <assemblyName>HttpMediaSource</assemblyName>
  <extensions>
     <mediaSource qualifiedName="HttpMediaSource.HttpMediaSource" />
  </extensions>
</addin>
```

3.3. UDP audio streaming

Objective

In this example, openBOXware will be operating both as the streaming server and the streaming client.

The tutorial will show how almost any media resource (exposed by any media source add-in) can be streamed to a media target add-in. The output stream can in turn be received by openBOXware itself, if it is registered as a media source.

We will continue the previous IceCast sample: the HTTP stream generated by the first computer will be received by a second computer. Audio data will be transcoded and streamed using a raw UDP stream to the first computer, where it will be received and played back using openBOXware.

Features

- Overview of the MediaTarget API,
- Registering a media target via the manifest file,
- Interactions between media sources and targets.

LocalUdpTarget.cs

```
using System;
using System.Collections.Generic;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaSinks;
using OpenBoxWare.Interface.MediaTargets;

namespace UdpStreaming {

      public class LocalUdpTarget : MediaTarget {

            public LocalUdpTarget(){
                  var udpsink = new UdpMediaSink {
                        Host = "192.168.1.2",
                        Port = 3030
                  } ;

                  _audioTarget = new AudioMediaTarget(udpsink,
                                                      AudioEncoding.Aac);
            }

            public override string Name {
                  get {
                        return "Remote UDP target";
                  }
            }

            AudioMediaTarget _audioTarget;

            public override AudioMediaTarget AudioTarget {
                  get {
                        return _audioTarget;
                  }
            }

      }

}
```

LocalUdpSource.cs

```
using System;
using System.Collections.Generic;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaResources;

namespace UdpStreaming {

      public class LocalUdpSource : MediaSource {

            public override string Name {
                  get {
                        return "Incoming UDP stream";
                  }
            }

            public override MediaSourceNode GetRootNode(){
                  return new LocalUdpNode();
            }

      }

      public class LocalUdpNode : MediaSourceNode {

            public override string Name {
                  get {
                        return "Stream node";
                  }
            }

            public override IEnumerable<MediaSourceNode> Children {
                  get {
                        return null;
                  }
            }

            public override IEnumerable<MediaElement> MediaElements {
                  get {
                        return new MediaElement[] {
                              new MediaElement(
                                  new UdpMediaResource {
                                         Port = 3030
                                  }, MediaElementContent.AacStream
                              )
                        } ;
                  }
            }
```

```
            public override MediaSource ParentMediaSource {
                  get {
                        return null;
                  }
            }

      }

}
```

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<addin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <name>UDP target/source streaming pair</name>
  <description>Media target and source pair that stream audio through an
UDP connection.</description>
  <author>Lorenz Cuno Klopfenstein</author>
  <version>1.0</version>
  <assemblyName>UdpStreaming</assemblyName>
  <extensions>
     <mediaTarget qualifiedName="UdpStreaming.LocalUdpTarget" />
     <mediaSource qualifiedName="UdpStreaming.LocalUdpSource" />
  </extensions>
</addin>
```

3.4. RSS Feed Reader

Objective

After writing the first “Hello World” application, we will expand on how applications work with a slightly more complex example. Add-ins which decide to implement this API can rely on the whole host of libraries provided by .NET (the subset implemented by Mono, more precisely) and all graphical features of the Qt GUI toolkit.

One of the most common ways to retrieve news and updates across the Internet is reading a “feed”, usually either in RSS or in Atom format. In this sample, we'll show you a very simple RSS Feed reader that can retrieve remote feeds through an HTTP request, parse the results and present the news items to the user.

Features

- Using System.Net.HttpWebRequest to issue an HTTP request,
- Using LINQ to parse the RSS XML format,
- Using some more advanced Qt widgets.

Code intentionally missing

This project is slightly too convoluted and complex to be comfortably printed out here.

Please refer to the online tutorials available at
http://openboxware.net/developers-area

3.5. Custom pipeline

Objective

In tutorial 3.2. we have shown how media targets and media sources interact between each other and how a multimedia stream can be easily bounced off between multiple nodes.

By implementing the media source and media target API, add-ins can expose a rich set of remote devices and/or contents to the rest of the framework, providing a sufficient level of abstraction to enable consumption by every other installed add-in.

However, sometimes it is needed to have more control over how your media is played back: in this case, you might want to use so-called “custom pipelines” inside your application's code.

Features

- Use a custom pipeline to play back a web radio stream,
- Expose remote controls via web server that can control the radio playback.

WebRadioApp.cs

```
using System;
using Qyoto;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.Gui;

namespace PluginWebRadio {

      public class WebRadioApp : Application {

            public override bool SetupGui (GuiWidget container,
                                           GuiMode mode){

                  if(mode != GuiMode.Fullscreen)
                        return false;

                  ServerController.Player =
                        new WebRadioPlayer(container, this);

                  return true;
            }

            public override void OnInitialization (){
                  this.GetContext().WebServer
                        .Register(typeof(ServerController), "radio");
            }

            public override void OnShutdown (){
                  this.GetContext().WebServer
                        .Deregister(typeof(ServerController));
            }

      }
}
```

WebRadioPlayer.cs

```
using System;
using System.Linq;
using Qyoto;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaResources;
using OpenBoxWare.Interface.MediaTargets;
using OpenBoxWare.Interface.Gui;

namespace PluginWebRadio {

      public class WebRadioPlayer : GuiWidget {

            QStackedLayout _layout;
            const string _radioUrl =
"http://192.168.1.1:8000/stream.ogg"; // audio stream URL
            CustomMediaPipeline _pipeline;
            QPushButton _play, _stop;

            public WebRadioPlayer(QWidget parent, Application owner)
                  : base(parent, owner) {

                  // getting an url service
                  var url = owner.GetContext().UrlService;

                  // creating the buttons
                  _play = new QPushButton(this);
                  var _playIcon = new QPixmap(url.ResolveUrl("play.png"));
                  _play.icon = new QIcon(_playIcon);
                  _play.SetIconSize(_playIcon.Size());

                  _stop = new QPushButton(this);
                  var _stopIcon = new QPixmap(url.ResolveUrl("pausa.png"));
                  _stop.icon = new QIcon(_stopIcon);
                  _stop.SetIconSize(_stopIcon.Size());

                  // setting up the stacked layout
                  _layout = new QStackedLayout();
                  _layout.AddWidget(_stop);
                  _layout.AddWidget(_play);
                  this.SetLayout(_layout);
                  _layout.SetCurrentWidget(_play);

                  // connecting button's signals
                  Connect(_play,SIGNAL("clicked()"),SLOT("PlayRadio()"));
                  Connect(_stop,SIGNAL("clicked()"),SLOT("StopRadio()"));
```

```
                // connecting signals which belongs to the web server
                Connect(this,SIGNAL("StartPlay()"),SLOT("PlayRadio()"));
                Connect(this,SIGNAL("StopPlay()"),SLOT("StopRadio()"));

                // pipeline creation
                var context = OwningApplication.GetContext();
                _pipeline = context.MediaPlayer.CreateNewPipeline();
                _pipeline.StatusChange += PipelineStatusHandler;
                var audioTarget =
context.MediaPlayer.CreateLocalAudioTarget();
                var finalTarget = MediaTarget.Create(audioTarget);
                _pipeline.SetMediaTarget(finalTarget);
          }

          // preferred widget size is 200x200 px
          public override QSize SizeHint (){
                return new QSize(200,200);
          }

          // widget will be always centered
          protected override void ResizeEvent (QResizeEvent arg1){
                base.ResizeEvent (arg1);
                Move((ParentWidget().Width()-Width())/2,
                      (ParentWidget().Height()-Height())/2);
          }

          // starts audio playback
          [Q_SLOT("PlayRadio()")]
          private void PlayRadio(){
                if(_pipeline != null){
                      var element = new MediaElement(
                            new HttpMediaResource {
                                  Uri = new Uri(_radioUrl)
                            } , MediaElementContent.OggContainer);

                      _pipeline.Play(new MediaPlaylist(element));
                }
          }


          // stops audio playback
          [Q_SLOT("StopRadio()")]
          private void StopRadio(){
                _pipeline.Stop();
          }
```

```
        // emits the start/stop playback signals
        public void SwitchPlay(){
            if(_pipeline.Status == MediaPlayerStatus.Playing){
                Emit.StopPlay();
            }
            else{
                Emit.StartPlay();
            }
        }

        // check whether the pipeline is playing back
        public bool IsPlaying(){
            if(_pipeline != null){
                if(_pipeline.Status == MediaPlayerStatus.Playing){
                    return true;
                }
                else{
                    return false;
                }
            }
            return false;
        }

        // emits custom signals using Qyoto/Qt
        protected WebRadioPlayerSignal Emit {
            get {
                return (WebRadioPlayerSignal) Q_EMIT;
            }
        }

        // callback that handles pipeline status changes
        public void PipelineStatusHandler(object sender, EventArgs e){
            switch(_pipeline.Status){
                case MediaPlayerStatus.Playing:
                    _layout.SetCurrentWidget(_stop);
                    break;
                default:
                    _layout.SetCurrentWidget(_play);
                    break;
            }
        }
    }

    // useful in order to use custom signals in Qyoto
    public interface WebRadioPlayerSignal  {
        [Q_SIGNAL("StartPlay()")]
        void StartPlay();

        [Q_SIGNAL("StopPlay()")]
        void StopPlay();
    }
}
```

ServerController.cs

```
using System;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.WebServer;

namespace PluginWebRadio {

      public class ServerController : Controller {

            // reference to the singleton media player istance
            public static WebRadioPlayer Player;

            // default front web page
            [Path("/")]
            public object Media(){
                  return View("Media", new {
                        Now = DateTime.Now,
                        isPlaying = Player.IsPlaying()
                  });
            }

            // command that starts/stops the player
            [Path("/switch")]
            [AcceptVerb(HttpVerb.Post)]
            public object Switch(){
                  var response = Player.IsPlaying() ? "play" : "stop";
                  Player.SwitchPlay();

                  return response;
            }

            // info about player status
            [Path("/status")]
            public object Status(string w){
                  switch(w){
                        case "m":
                              return string.Format("The radio is {0} playing",
                              Player.IsPlaying() ? "currently" : "not");
                        case "b":
                              return Player.IsPlaying() ? "stop" : "play";
                        default:
                              return string.Empty;
                  }
            }
```

```
            // get the local copy of the jquery library
            [Path("/jquery")]
            public object Jquery(){
                  return new FileResult{
                        FileName = "jquery-1.4.4.min.js"
                  } ;
            }
      }
}
```

Media.django

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Media Content</title>
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="-1" />
<meta http-equiv="Cache-control" content="no-cache" />
<script src="/radio/jquery"></script>
<script>
      $(document).ready(function() {
            $("#nownext").load("/radio/status");

            refreshId = setInterval(function() {
                  $("#nownext").html($("#nownext").text()+"...");
                  $("#nownext").load("/radio/status?w=m",
                        function(response){
                              if(response == ""){
                                    alert("Server may be down");
                                    clearInterval(refreshId);
                              }
                              $.get("/radio/status?w=b", function(data){
                                    $('#switch').html(data);
                              } );
                        }
                  );
            } , 1000);

            $('#switch').click(function(){
                  $.post("/radio/switch", function(data) {
                        $('#switch').html(data);
                  } );
            } );

      } );
</script>
```

```
<style>
      body{text-align: center;}
      #main_container{
            text-align:left;
            margin:20 auto;
            width: 300px;
      }
      div{margin: 10 0;}
      #nownext{
            color:#FECC0A;
            font-size:16px;
      }
      #switch{color:red;}
      #switch:hover{text-decoration:underline;}
</style>
</head>
<body>

<div id="main_container">
      <div id="datetime">Today is {{ Now }}</div>
      <div id="commands_container">
            Here you can control the local radio playout.<br/>
            Just take a look!

            <div id="nownext"></div>

            <div id="switch">
            { % if isPlaying %}
            stop
            { % else %}
                  play
            { % endif %}
            </div>
      </div>

      <div id="res"></div>
</div>

</body>
</html>
```

3.6. Multicast

Objective

In previous samples, streaming of multimedia data was always done using unicast connections. However, in many scenarios, using multicast streams can allow more users to enjoy the same multimedia resources by using the same limited amount of network resources.

The openBOXware framework supports multicast directly: a video stream via RTP can be generated in a matter of a couple of lines of code. The same multicast stream can then be received by another openBOXware instance or a common video player.

Features

- Streaming to a multicast enabled media target,
- Writing an SDP file that describes the multicast stream and publishing via web server,
- Playback using a common video player (VLC),
- Playback using openBOXware.

MulticastStreamerApp.cs

```
using System;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.Gui;

namespace PluginMulticastStreamer {

      public class MulticastStreamerApp : Application {

            public override void OnActivation(){
                  this.GetContext().WebServer
                        .Register(typeof(MulticastController), "multicast");
            }

            public override void OnShutdown(){
                  this.GetContext().WebServer
                        .Deregister(typeof(MulticastController));
            }

            public override bool SetupGui(GuiWidget container,
                                          GuiMode mode){
                  return false;
            }

      }

}
```

MulticastMediaTarget.cs

```
using System;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaTargets;
using OpenBoxWare.Interface.MediaSinks;

namespace PluginMulticastStreamer {

      public class MulticastMediaTarget : MediaTarget {

            public override MuxingMediaTarget MuxingTarget {
                  get {
                        return new MuxingMediaTarget(new RtpMediaSink {
                              VideoHostAddress = "232.0.0.1",
                              VideoPort = 6000,
                              AudioHostAddress = "232.0.0.1",
                              AudioPort = 6002
                        } , ContainerFormat.None);
                  }
            }

            public override VideoMediaTarget VideoTarget {
                  get {
                        return new VideoMediaTarget(null,
                                                VideoEncoding.H264);
                  }
            }

            public override AudioMediaTarget AudioTarget {
                  get {
                        return new AudioMediaTarget(null,
                                                AudioEncoding.Aac);
                  }
            }

            public override string Name {
                  get {
                        return "RTP to Multicast";
                  }
            }

      }

}
```

MulticastMediaSource.cs

```
using System;
using System.Collections.Generic;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.MediaResources;

namespace PluginMulticastStreamer {

      public class MulticastMediaSource : MediaSource {

            public override MediaSourceNode GetRootNode() {
                  return new MulticastMediaSourceNode(this);
            }

            public override string Name {
                  get {
                        return "Multicast group";
                  }
            }

      }

      class MulticastMediaSourceNode : MediaSourceNode {

            MediaSource _parent;
            public MulticastMediaSourceNode(MediaSource parent){
                  _parent = parent;
            }

            public override string Name {
                  get { return "Multicast node"; }
            }

            public override IEnumerable<MediaSourceNode> Children {
                  get { return null; }
            }

            public override IEnumerable<MediaElement> MediaElements {
                  get {
                        return new MediaElement[] {
                              new MediaElement(new SdpMediaResource {
                                    Location =
                                          "http://localhost:8080/multicast/"
                              }, MediaElementContent.Autodetect)
                        };
                  }
            }
```

```
            public override MediaSource ParentMediaSource {
                  get {
                        return _parent;
                  }
            }

      }

}
```

MulticastController.cs

```
using System;
using OpenBoxWare.Interface;
using OpenBoxWare.Interface.WebServer;

namespace PluginMulticastStreamer {

      public class MulticastController : Controller {

            const string BaseSdp = @"v=0
o=multicast 2890844526 2890842807 IN IP4 232.0.0.1
s=MulticastProxySession
c=IN IP4 232.0.0.1
m=video 6000 RTP/AVP 96
a=rtpmap:96 H264/90000";

            const string AudioSdp = @"m=audio 6002 RTP/AVP 96
a=rtpmap:96 MP4A-LATM/44100";

            [Path("/")]
            public object Sdp(){
                  return string.Format("{0}\n{1}", BaseSdp, AudioSdp);
            }

            [Path("/no-audio")]
            public object SdpNoAudio(){
                  return BaseSdp;
            }

      }
}
```