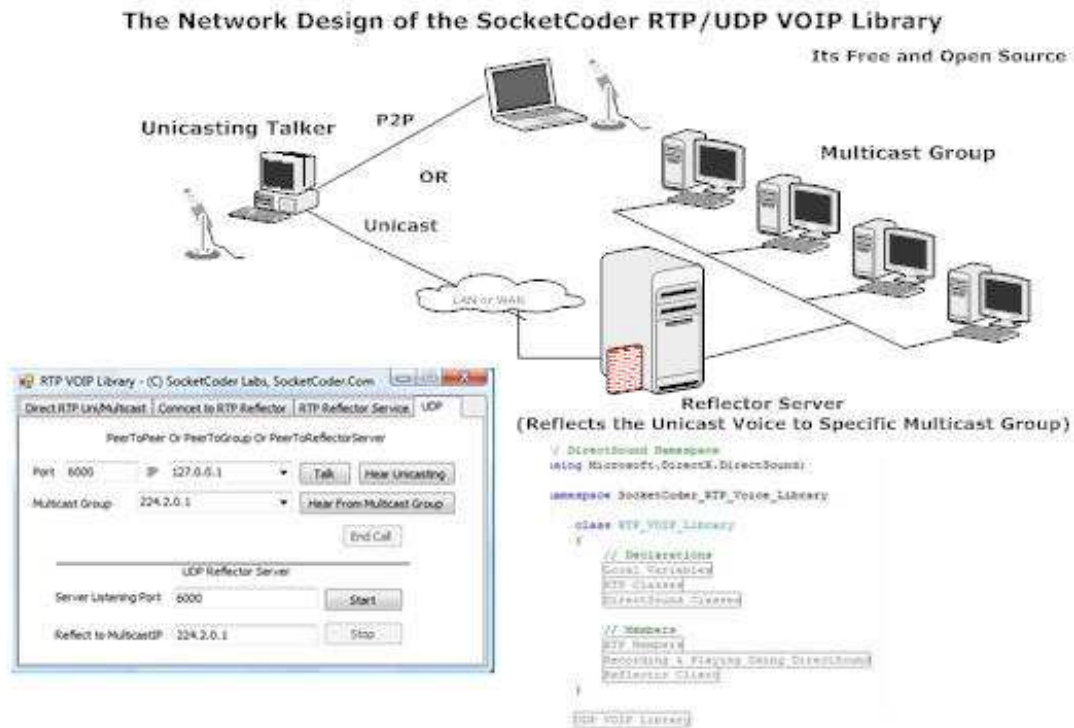


## مشروع تطوير مكتبة مفتوحة المصدر لنقل الصوت عبر شبكات الـMulticasting والـInternet2

هدف المشروع هو بناء مكتبة مجانية ومفتوحة المصدر لنقل الصوت باستخدام بروتوكول النقل في الزمن الحقيقي RTP عبر شبكات الـMulticasting والـInternet2 .

### The Project Features:

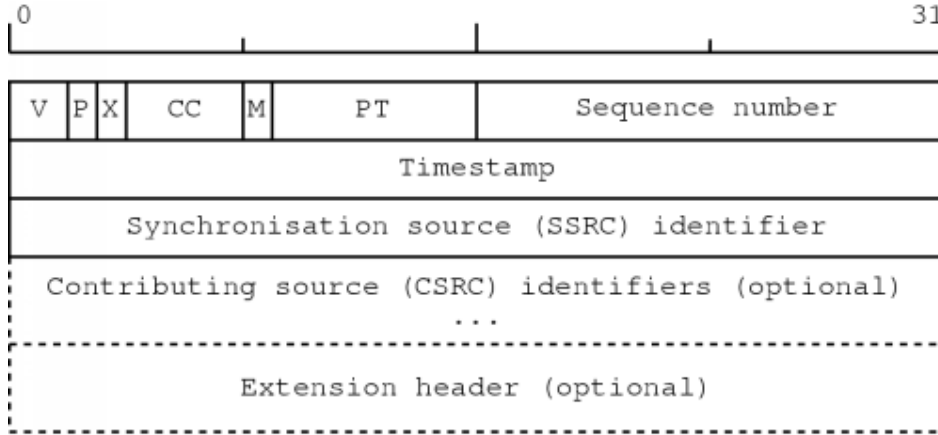
- VOIP Unicast-To- Unicast -Or-Multicast Using RTP
- VOIP Unicast -To- Unicast -Or- Multicast Using UDP
- VOIP Unicast -To-Server & Server-To- Multicast Using RTP



(C) Fadi Abdelqader, www.SocketCoder.Com

## الفصل الأول: مكونات الـ RTP Real-Time Transport Protocol Header:

يتكون الـ RTP Header من 12 Bytes أساسية Fixed مقسمة كما هو واضح في الشكل التالي:



V = Version      CC = CSRC count  
P = Padding      M = Marker  
X = Extension      PT = Payload type

1- **Version**: ويتكون من 2 bits ويوضع فيه الإصدار الـ RTP المستخدم وحاليا نحن نعمل على الإصدار الثاني.

2- **Padding**: ويتكون من 1 bit وهو Flag يبين إذا كنا سنضيف معلومات إضافية على الـ RTP Header وكمثال يمكن أن يستخدم هذا الـ Padding في حالة كنا نريد إرسال الـ RTP Payload بشكل مشفر وبهذه الحالة يضاف الـ Padding والذي سيحتوي على معلومات حول موقع الـ Payload المشفر في أسفل الـ RTP Header.

3- **Extension**: ويتكون من 1 bit وهو Flag يبين إذا كان الـ RTP Header يحتوي على Extensions في نهاية الـ RTP Header أم لا والتي قد تحتوي على معلومات خاصة بالـ Application الذي يستخدم الـ RTP لنقل البيانات.

4- **CSRC Count**: ويتكون من 4 bits وفيه يبين عدد الـ Content Source Identifiers الملحقة مع الـ RTP Header ويستخدم هذا الـ Field

في حالة كنا نريد دمج أكثر من RTP Stream في RTP Stream واحد فمن المعروف أنه لا يمكن تحميل أكثر من Payload Type واحد على الـ RTP Stream وبالتالي في حالة كنا نريد إرسال الصوت والصورة فلا بد من استخدام اثنين من الـ RTP Stream وبالتالي قد تظهر مشاكل لاحقة في عملية المزامنة بين الصوت والصورة ولحل هذه المشكلة يمكن دمج الـ RTP Stream الخاص بالصوت والـ RTP Stream الخاص بالصورة بـ RTP Stream واحد وتميزها بإعطاء Identifier خاص لكل منها ويتكون الـ Identifier للـ CSRC من 32 bits وقد يصل عدد الـ CSRC التي يمكن تحميلها على الـ RTP Header إلى 16 بحجم أعظمي  $32 \times 16 = 480$  bits.

**5- Marker:** ويتكون من 1 bit وهو Flag يبين بداية ونهاية الإرسال لكل مجموعة من البيانات وكمثال في حالة نقل صورة Image Frame على أكثر من RTP Packet فإن ذلك الـ Marker سيحتوي على قيمة 1 في أول Frame يتم إرساله لمعرفة بداية ونهاية الإرسال لتلك الصورة.

**6- Payload Type:** ويتكون من 7 bits توضح فيها نوع البيانات التي سيتم تحميلها على الـ RTP Packet وكما أوضحنا سابقا لا يمكن أن يتم تحميل أكثر من نوع من البيانات على نفس الـ RTP Stream ويبين الجدول التالي أنواع الـ RTP Payload التي يمكن تحميلها على الـ RTP Stream:

## SocketCoder RTP VOIP Library Documentation For R0.2 - Arabic Version

PT	encoding name	audio/video (A/V)	clock rate (Hz)	channels (audio)	
0	PCMU	A	8000	1	[RFC3551]
1	Reserved				
2	Reserved				
3	GSM	A	8000	1	[RFC3551]
4	G723	A	8000	1	[Kumar]
5	DVI4	A	8000	1	[RFC3551]
6	DVI4	A	16000	1	[RFC3551]
7	LPC	A	8000	1	[RFC3551]
8	PCMA	A	8000	1	[RFC3551]
9	G722	A	8000	1	[RFC3551]
10	L16	A	44100	2	[RFC3551]
11	L16	A	44100	1	[RFC3551]
12	QCELP	A	8000	1	
13	CN	A	8000	1	[RFC3389]
14	MPA	A	90000		[RFC3551]
15	G728	A	8000	1	[RFC3551]
16	DVI4	A	11025	1	[DiPol]
17	DVI4	A	22050	1	[DiPol]
18	G729	A	8000	1	
19	reserved	A			
20	unassigned	A			
21	unassigned	A			
22	unassigned	A			
23	unassigned	A			
24	unassigned	V			
25	CelB	V	90000		[RFC2029]
26	JPEG	V	90000		[RFC2435]
27	unassigned	V			
28	nv	V	90000		[RFC3551]
29	unassigned	V			
30	unassigned	V			
31	H261	V	90000		[RFC2032]
32	MPV	V	90000		[RFC2250]
33	MP2T	AV	90000		[RFC2250]
34	H263	V	90000		[Zhu]
35--71	unassigned	?			
72--76	reserved for RTCP conflict avoidance				[RFC3551]
77--95	unassigned	?			
96--127	dynamic	?			[RFC3551]

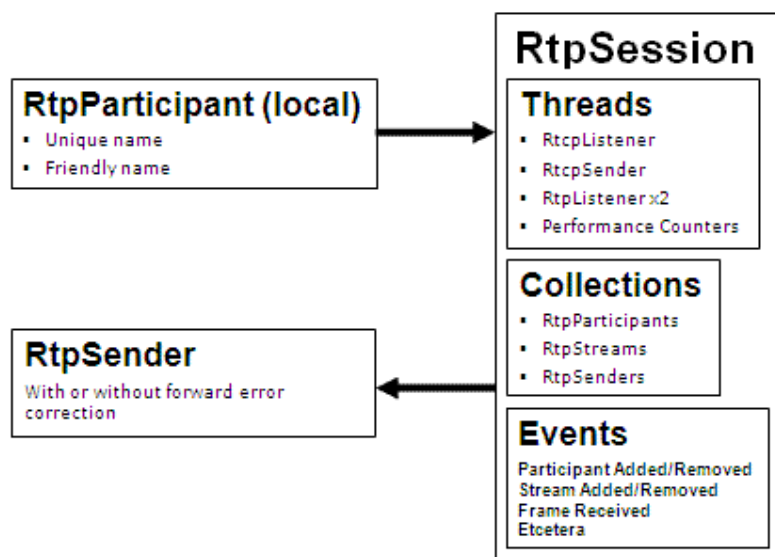
**7- Sequence Number:** ويتكون من 16 bits ويحتوي على الرقم المتسلسل والذي يولد في البداية بشكل عشوائي ويزيد بمقدار واحد لكل RTP Packet يتم إرساله ويستفاد من هذا الرقم لمعرفة موقع الـ RTP Packet عند استلامه لإجراء عملية الترتيب كذلك يستفاد منه في اكتشاف فقدان أي من الـ RTP Packets المرسلة.

**8- Timestamp:** ويتكون من 32 bits ويحتوي على الوقت الزمني الذي تم فيه تحميل البيانات على الـ RTP Packet ويستفاد من هذه البيانات بشكل كبير لتغلب على مشكلات الـ Jitter والمزامنة بين عملية الإرسال والاستقبال والزمن المستغرق بينهما.

**9- Synchronization Source Identifier SSRC:** ويتكون من 32 bits ويحتوي على رقم متسلسل يولد عشوائياً لتمييز كل RTP Stream عن غيره بحيث لا يتشابه RTP Stream مع الآخر على نفس الـ Session مما قد يؤدي إلى ما يسمى بالـ RTP Stream Conflict.

## الفصل الثاني: استخدام الـ RTP In .NET

لا يدعم الـ .NET Framework 3.5 أية Classes للـ RTP Protocol لكن قدمت Microsoft مجموعة من الـ Third Party Kits ضمن مشروعها المفتوح المصدر Microsoft Conference XP والذي يحتوي على نواة الـ RTP Protocol بشكل كامل وقد تم تقسيمه إلى مجموعة من الـ Classes بناء على كيفية عمل الـ RTP وكما في الشكل التالي:



أولاً الـ RTP Session والـ RtpParticipant : يستخدم الـ RTP Session بشكل أساسي في عملية إدارة جلسة الاتصال والتي يمكن أن يتم فيها إرسال مجموعة من الـ RTP Stream كما ويمكن أن يتصل بالـ Session الواحدة مجموعة من المشاركين Participants كذلك يمكن للمشارك الواحد أن يتصل بأكثر من RTP Session وتميز كل RTP Session بالعنوان ورقم الـ Port الذي يتم الإرسال له كذلك يتم إضافة CNAME لتمييز محتويات كل RTP Session عن الأخرى وكمثال في الدوت نيت:

```

RtpSession rtpSession = new RtpSession(endpoint, new
RtpParticipant("My Audio Session", ParticipantName),
true, true);
  
```

إذ يمرر الـ Endpoint Object والذي يحتوي على عنوان Multicast IP بالإضافة إلى Port الاستقبال وينضم المرسل أو المستقبل إلى الـ RTP Session بتمرير الـ RtpParticipant Object إلى الـ RTP Session ولتعريفه يجب أن يتم تمرير الـ CNAME واسم المتصل إليه بعد ذلك نكمل تعريف الـ RTP Session بتمرير قيم True أو False ويحدد في الأول أنك تريد الانضمام إلى الـ RTP Session فقط ويحدد الثاني إذا كنت تنوي الإرسال باستخدام تلك الـ Session.

ثانياً الـ RtpEvents: وهو مجموعة الأحداث التي تحدث داخل الـ RTP Session والتي يمكن أن يوضع Action معين عند حدوث أي منها وتقسم هذه الأحداث إلى ثلاثة أقسام وهي:

أولاً أحداث ترتبط باكتشاف أخطاء أو مشاكل أثناء الإرسال أو الاستقبال ويمكن أن نسميها Exceptions Events وهي:

- 1- DuplicateCNameDetected
- 2- FrameOutOfSequence
- 3- HiddenSocketException
- 4- InvalidPacket
- 5- InvalidPacketInFrame
- 6- NetworkTimeout
- 7- PacketOutOfSequence
- 8- RtpParticipantTimeout
- 9- RtpStreamTimeout

ثانياً حدث وحيد لجلب معلومات Report عن حالة الـ RTP Received Data وهو الـ ReceiverReport.

ثالثاً أحداث تتعلق بالانضمام إلى الـ Session وإضافة وحذف الـ RTP Session Participant & Stream Add/Remove Events وهي:

- 1- RtpParticipantAdded
- 2- RtpParticipantDataChanged
- 3- RtpParticipantRemoved
- 4- RtpStreamRemoved
- 5- RtpStreamAdded

وأكثر ما يهمنا في هذه الأحداث هو ضبط متى يتم الانضمام إلى RTP Session لبدأ عملية الاستقبال وكذلك متى يتم الخروج من RTP Session لإيقاف الاستقبال ولتعامل مع هذه الأحداث يجب أولاً أن نقوم بعمل Hook لها وربط الحدث بالدالة التي سيتم تنفيذها عند حدوث ذلك الحدث وكما يلي كمثال:

```
private void HookRtpParticipantEvents ()
{
    // Add Remove Participant Events
    RtpEvents.RtpParticipantAdded += new
    RtpEvents.RtpParticipantAddedEventHandler (RtpParticipantA
    dded);
    RtpEvents.RtpParticipantRemoved += new
    RtpEvents.RtpParticipantRemovedEventHandler (RtpParticipan
    tRemoved);
}

private void HookRtpStreamEvents ()
{
    // Add Remove Stream Events
    RtpEvents.RtpStreamAdded += new
    RtpEvents.RtpStreamAddedEventHandler (RtpStreamAdded);
    RtpEvents.RtpStreamRemoved += new
    RtpEvents.RtpStreamRemovedEventHandler (RtpStreamRemoved);
}

private void RtpParticipantAdded(object sender,
RtpEvents.RtpParticipantEventArgs ea)
{
    ShowMessage (ea.RtpParticipant.Name + " has joined");
}

private void RtpParticipantRemoved(object sender,
RtpEvents.RtpParticipantEventArgs ea)
{
    ShowMessage (ea.RtpParticipant.Name + " has left");
}
```



```
private void RtpStreamAdded(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived += new
RtpStream.FrameReceivedEventHandler(FrameReceived);
}

private void RtpStreamRemoved(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived -= new
RtpStream.FrameReceivedEventHandler(FrameReceived);
}
```

وينطبق ذلك الشكل في التعامل مع كافة أحداث الـ RTP Events السابقة الذكر.

ثالثا الـ RtpSender والـ RtpListener : يمكن الإرسال مباشرة بعد تعريف الـ RTP Session ولتعريف الـ RtpSender يجب أن يمرر له الـ RTP Session الذي قمنا بتعريفه سابقا ويضاف إليها أيضا الـ Payload Type الذي سيتم إرساله على الـ RTP Session ولتمييز الـ RTP Sender في الـ Session عن غيره يمكن أن يتم تعريف Hash Table لذلك اختياريًا أو قيمة null وتمثيره لدالة الـ CreateRtpSender وكما يلي كمثال:

```
RtpSender rtpsender = rtpSession.CreateRtpSender("My VOIP
Sender", PayloadType.dynamicAudio, null);
```

أو يمكن تعريفه بالشكل التالي بحيث تدعم عملية FEC – Frame Error Correction للـ RTP Packet وذلك لدعم إمكانية تصحيح الـ Packet عند وصوله مع وجود أخطاء بدلا من طلب إعادة الإرسال له ويصبح الشكل العام لدالة كما يلي كمثال:

```
RtpSender rtpsender = rtpSession.CreateRtpSenderFec("My
VOIP Sender With FEC", PayloadType.dynamicAudio, null,
CDataPX, CFecPX);
rtpSender.Send(buffer);
```

وأما بخصوص الاستقبال بالـ RtpListener فتعمل هذه الدالة ضمن الـ RTP Session وذلك بربط تلك الدالة بحدث الانضمام إلى الجلسة وفي هذه الحالة بمجرد انضمامك إلى الجلسة يتم تنفيذ الحدث وبدأ الاستقبال ولتنفيذ ذلك يجب أولا عمل Hook للـ RTP Events التي قمنا بتعريفها سابقا وعند بدأ إرسال بيانات ما من قبل أي طرف من الأطراف يتم تنفيذ الحدث وبدأ الاستقبال ويمكنك الوصول إلى

محتويات البيانات المستقبلية من خلال الخاصية Frame.Buffer والمستلمة من الـ  
: FrameReceivedEventArgs

```
private void FrameReceived(object sender,  
RtpStream.FrameReceivedEventArgs ea)  
{  
    Byte[] Buufer = ea.Frame.Buffer;  
}
```

رابعا الـ RTCP وهو مختصر لـ Real-Time Transport Control Protocol  
ومن أهم استخداماته أنه يعمل مع الـ RTP لإدارة التحكم في العمليات التي تتم في  
أنظمة المؤتمرات وكذلك تقديم تقارير عن حالة تلك العمليات وتقسيم إلى:

- 1- تقارير الإرسال SR وتقارير الاستقبال RR
- 2- تفاصيل مرسل البيانات Source description SDES
- 3- إدارة الانضمام والخروج لـ Add Remove مجموعة Membership
- 4- تعريف application-defined APP جديد على الـ RTP Session.

ويمكن أن نستفيد من كل ذلك من خلال الـ RTCP Namespace وكمثال يمكن  
استخدام الـ RtcpListener Class لتعامل مع البيانات التي يتم استقبالها من  
خلال الـ RTP Session وإدارة عمليات الـ Membership والتي ذكرناها  
سابقا وأيضا تقديم معلومات عن حالة شبكة الاتصال والتي تربط المرسل  
بالمستقبل، وسأقدم بدروس لاحقة معلومات أكبر عن استخدامات الـ RTCP  
Protocol في إدارة أنظمة الـ Conferencing.

## الفصل الثالث: التعامل مع الـ Direct Sound والـ RTP لعلية نقل الصوت في أنظمة الـ Multicast VOIP Conferencing:

تحدث في الفصل السابق عن مكونات الـ RTP وكيفية استخدامه في بيئة الدوت نيت وفي هذا الجزء سنقوم بتطبيق برمجة نظام بسيط لبث الصوت من نقطة إلى مجموعة One-To-Many وسنقوم أيضا بالتحكم بخواص الـ Buffer بهدف المزامنة بين عملية التسجيل والإرسال والعرض والتحكم به لتقليل الـ Delay والـ Loss في عملية الإرسال بناءا على سرعة الشبكة.

### أولا: استخدام الـ DirectSound في التعامل مع الصوت :

لتعامل مع الـ DirectSound لابد أولا من إضافة الـ Microsoft.DirectX والـ Microsoft.DirectX.DirectSound Namespaces إلى الـ References في المشروع ويحتوي الـ DirectSound على عدد من الـ Classes والتي تستخدم في برمجة كل ما يتعلق بالصوت تقريبا مثل التسجيل والـ Encoding والعرض والكثير من الأمور وما يهمنا في هذا المشروع هو استخدام الـ Classes التالية:

**WaveFormat**: ويستخدم لتحديد تفاصيل الـ Wave Format مثل عدد الـ Channels مثلا 1 أو 2 و نوع الـ Modulation المستخدم مثل الـ PCM-Pulse Code Modulation وعدد الـ SamplesPerSecond وعدد الـ BitsPerSample لأستخدم هذه المعلومات في عملية تحويل الذبذبات الصوتية إلى Bits لتمكين نقلها عبر الشبكة.

**CaptureBufferDescription**: ويستخدم لتحديد حجم الـ Bytes Buffer والذي سيتم حجزه في الذاكرة لاستقبال الـ WAVE Bits الملتقطة.

**CaptureDevicesCollection**: وهو الـ Array Of Devices ويستخدم لإرجاع كافة الـ Hardware Devices Info المتعلقة بوحدة الإدخال\الإخراج الصوتية المتاحة لديك لتحديد واحد منها في عملية التقاط الصوت من المايكروفون وعرض الصوت على السماعات.

**DeviceInformation**: ويستخدم لتحديد واحد من الـ Devices التي سيتم إرجاعها من CaptureDevicesCollection حيث سيرجع الـ GUID Globally Unique Identifier الخاص بالـ Sound Driver لديك.

**Capture**: ويعرف به الـ Driver GUID الذي يتم جلبه من DeviceInformation لاستخدامه في الـ CaptureBuffer لالتقاط الصوت وتخزينه في الـ Buffer.

**CaptureBuffer**: وهو الـ Class المسئول عن عملية التقاط الصوت ووضعها في الـ Buffer إذ يمرر له الـ CaptureBufferDescription Object والـ Capture Object لبدأ عملية التقاط الصوت ضمن المعلومات الممررة له وتخزينه في الـ Buffer المحدد.

**BufferPositionNotify**: وهو Handler Notification Class يستخدم لتنفيذ Event محدد عند وصول الـ Buffer إلى منطقة معينة.

**Notify**: ويحدد فيه عدد الـ Bytes التي إن وصل لها معبئ الـ Buffer فسيتم عمل Trigger للـ Event الذي سيقوم بعملية معينة كتفريغ الـ Buffer من جديد.

**Device**: وسيستخدم في طرف المستمع وفيه يتم تحديد الـ Sound Card الذي سيتم عرض الصوت عليه ويربط فيه جزأين أولاً الـ Application الذي سيتم التعامل معه في عملية معالجة وعرض الصوت والثاني الـ CooperativeLevel Priority.

**BufferDescription**: وسيستخدم في طرف المستمع إذ يحتوي على مجموعة من الـ Properties والـ Methods لتحديد وإرجاع معلومات عن الـ Buffer الذي سيستخدم في تجميع الـ Bytes الواردة من مصدر ما.

**SecondaryBuffer**: وسيستخدم في طرف المستمع ويمرر له الـ **BufferDescription** والـ **Device Object** لبدأ عملية تجميع الـ **Buffer** ومن ثم إمكانية عرضه على الـ **Sound Device** المحدد.

## ثانياً: مراحل بناء نظام لنقل الصوت إلى مجموعة من المتصلين باستخدام الـ **DirectSound** والـ **RTP**:

بعدما قمنا بشرح موجز عن استخدام كل Class من الـ **Classes** التي سيتم التعامل معها في الـ **Direct Sound** سنتحدث في هذه الجزء عن كيفية استخدامها برمجياً بشكل مجموعة من المراحل المترابطة (سأقوم بإضافة تعليق باللغة الإنجليزية على كل سطر من اسطر المثال لتسهيل تتبع الكود):

1- التعامل مع الـ **Sound Card** لعملية تسجيل الصوت من المايكروفون والتعامل مع الـ **WAVE Format** :

المرحلة الأولى تعريف الـ **Classes** الخاصة بالـ **Direct Sound** وكما تم شرحها في الأعلى:

```
private CaptureBufferDescription
captureBufferDescription;
private AutoResetEvent autoResetEvent;
private Notify notify;
private WaveFormat waveFormat;
private Capture capture;
private CaptureBuffer captureBuffer;
private Device device;
private SecondaryBuffer playbackBuffer;
private BufferDescription playbackBufferDescription;
```

المرحلة الثانية سنقوم بإنشاء دالة لتعريف الـ **Sound Card** الذي سيستخدم في عملية تسجيل الصوت وكذلك هي نفسها عند المستقبل:

```
public void SetVoiceDevices(System.Windows.Forms.Control
AppForm_TypeThis)
{
// Use The Recommended settings For Sound Devices
```

```
SetVoiceDevices (
0, // Device Number (First Device)
1, // Channels      (2 if Stereo)
AppForm_TypeThis, // Application Form Pointer
16, // BitsPerSample
22050); // SamplesPerSecond
}

public void SetVoiceDevices(int deviceID, short channels,
System.Windows.Forms.Control AppForm_TypeThis, short
bitsPerSample, int samplesPerSecond)
{
// Installization Voice Devices
device = new Device(); // Sound Input Device
device.SetCooperativeLevel(AppForm_TypeThis,
CooperativeLevel.Normal); // Set The Application Form and
Priority
CaptureDevicesCollection captureDeviceCollection = new
CaptureDevicesCollection(); // To Get Available Devices
(Input Sound Card)
DeviceInformation deviceInfo =
captureDeviceCollection[deviceID]; // Set Device Number
capture = new Capture(deviceInfo.DriverGuid); // Get The
Selected Device Driver Information

//Set up the wave format to be captured.
waveFormat = new WaveFormat(); // Wave Format declaration
waveFormat.Channels = channels; // Channels (2 if
Stereo)
waveFormat.FormatTag = WaveFormatTag.Pcm; // PCM - Pulse
Code Modulation
waveFormat.SamplesPerSecond = samplesPerSecond; // The
Number of Samples Peer One Second
waveFormat.BitsPerSample = bitsPerSample; // The Number
of bits for each sample
waveFormat.BlockAlign = (short)(channels * (bitsPerSample
/ (short)8)); // Minimum atomic unit of data in one byte,
Ex: 1 * (16/8) = 2 bits
waveFormat.AverageBytesPerSecond = waveFormat.BlockAlign
* samplesPerSecond; // required Bytes-Peer-Second Ex.
22050*2= 44100
captureBufferDescription = new
CaptureBufferDescription();
captureBufferDescription.BufferBytes =
waveFormat.AverageBytesPerSecond / 5; //Ex. 200
milliseconds of PCM data = 8820 Bytes (In Record)
captureBufferDescription.Format = waveFormat; // Using
Wave Format
```

```
// Playback
playbackBufferDescription = new BufferDescription();
playbackBufferDescription.BufferBytes =
waveFormat.AverageBytesPerSecond / 5; //Ex. 200
milliseconds of PCM data = 8820 Bytes (In Playback)
playbackBufferDescription.Format = waveFormat;
playbackBuffer = new
SecondaryBuffer(playbackBufferDescription, device);
bufferSize = captureBufferDescription.BufferBytes;
}
```

المرحلة الثالثة إنشاء الدالة الخاصة بالتقاط الصوت لتجهيز الضغط والإرسال:

```
private void StartRecordAndSend() // Send Recorded Voice
{
captureBuffer = new
CaptureBuffer(captureBufferDescription, capture); // Set
Buffer Size, Voice Recording Format & Input Voice Device
SetBufferEvents(); // Set the events Positions to Send
While Recording
int halfBuffer = bufferSize / 2; // Take the half buffer
size
captureBuffer.Start(true); // start capturing
bool readFirstBufferPart = true; // to know which part
has been filled (the buufer has been divided into tow
parts)
int offset = 0; // at point 0
MemoryStream memStream = new MemoryStream(halfBuffer); //
set the half buffer size to the memory stream

while (True)
{
//WaitOne() Blocks the current thread until the current
WaitHandle receives a signal
//WaitHandle("Encapsulates operating system-specific
objects that wait for exclusive access to shared
resources")
autoResetEvent.WaitOne();
memStream.Seek(0, SeekOrigin.Begin); //Sets the position
within the current stream to 0
captureBuffer.Read(offset, memStream, halfBuffer,
LockFlag.None); // capturing and set to MemoryStream
readFirstBufferPart = !readFirstBufferPart; // reflecting
the boolean value to set the new comming buffer to the
other part
}
```

```
offset = readFirstBufferPart ? 0 : halfBuffer; // if
readFirstBufferPart set to true then set the offset to 0
else set the offset to the half buffer
byte[] dataToWrite = memStream.GetBuffer;
// Here you can Compress the voice buffer
.
.
// Sending the compressed voice across Network
.
.
}
```

-2 إدارة الـ Buffering وذلك بتجزئ الـ Buffer إلى جزأين الأول يستخدم في عملية تخزين الصوت المسجل والثاني لتجهيزه لعملية المعالجة كإرساله أو ضغطه:

```
protected void SetBufferEvents()
{
// Goal: To Send While Recording
// To Set The Buffer Size to get 200 milliseconds and
divide it in half,
// so that when the first half is filled the data can be
used to send,
// while the second half of the buffer is being filled
with PCM Data

try
{
autoResetEvent = new AutoResetEvent(false); // To wait
for notifications
notify = new Notify(captureBuffer); // The number of
bytes that can trigger the notification event

// the first half
BufferPositionNotify bufferPositionNotify1 = new
BufferPositionNotify(); // to describe the notification
position
bufferPositionNotify1.Offset = bufferSize / 2 - 1; // (=
At the Half of The Buffer) to know where the notify event
will trigger
bufferPositionNotify1.EventNotifyHandle =
autoResetEvent.SafeWaitHandle.DangerousGetHandle(); //
Set The Event that will trigger after the offset reached

// the last half
```



```
BufferPositionNotify bufferPositionNotify2 = new
BufferPositionNotify();
bufferPositionNotify2.Offset = bufferSize - 1; // (= At
The Last Buffer)
bufferPositionNotify2.EventNotifyHandle =
autoResetEvent.SafeWaitHandle.DangerousGetHandle();

notify.SetNotificationPositions(new
BufferPositionNotify[] { bufferPositionNotify1,
bufferPositionNotify2 }); // The Two Positions (First &
Last)
}
catch (Exception) { }
}
```

3- ضغط ال-Buffer المستلم باستخدام معايير الضغط المتاحة: في هذه المرحلة فالخيار متاح لك في اختيار طريقة الضغط المناسبة ويمكنك أيضا في هذه المرحلة القيام بعملية Coding معينة على ال-Buffer مثلا تشفيره ثم ضغطه لكن وجب في البداية دراسة أنواع الضغط والتشفير المناسبة لصوت بهدف المحافظة على ال-QoS Quality Of Server والتي سيتم التطرق لها لاحقا.

4- تغليف ال-Buffering في RTP Packet وإرساله عبر الشبكة إلى  
:Multicast RTP Session  
قمت سابقا بشرح كيفية استخدام ال-RTP Protocol لعملية الانضمام إلى مجموعة وكذلك كيفية تغليف ال-Byte Data بال-RTP Packet لمزيد من المعلومات أنظر الجزء الثاني.

5- في الطرف المقابل (المستقبل) سيقوم بالانضمام إلى RTP Session ثم بدأ عملية ال-Listening على ال-Session لاستقبال كما تم شرح هذه العملية في الجزء الثاني إذ أنه بعد الانضمام إلى RTP Session سيتم تنفيذ ال-RTP Session Event والذي بدوره سيمكنك من استقبال البيانات الواردة من المرسل وبعد ذلك يمكننا تجميع ال-Buffer ثم عرضه.

```
private void RtpStreamAdded(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived += new
RtpStream.FrameReceivedEventHandler(FrameReceived);
}
```

```
}  
  
private void FrameReceived(object sender,  
RtpStream.FrameReceivedEventArgs ea)  
{  
PlayReceivedVoice(ea.Frame.Buffer);  
}  
}
```

6- بعد عملية الاستقبال يتم تجميع الـ Bytes المستقبلية في Buffer لإجراء عملية فك الضغط ومن ثم عرضه على Sound Device لديك.

```
private void PlayReceivedVoice(byte[] VoiceBuffer)  
{  
  
//Decompress the received data  
// byte[] byteDecodedData = Decompress(VoiceBuffer);  
  
//Play it on the speaker device.  
playbackBuffer = new  
SecondaryBuffer(playbackBufferDescription, device);  
playbackBuffer.Write(0, byteDecodedData, LockFlag.None);  
// 0= is the Starting Point (the offset)  
playbackBuffer.Play(0, BufferPlayFlags.Default); // 0 =  
is The Priority of Sound for hardware that mixing the  
voice resources  
  
}
```

## ثالثا: دراسة حجم ال-Buffer وتأثره بسرعة الشبكة وعلاقة ذلك بال-Jitter Loss And Delay:

سنقدم في هذا الجزء كيفية حساب حجم ال-Buffer و التحكم به بناء على عوامل قدرة الشبكة وكذلك الجودة.

بيننا سابقا أن الزيادة في حجم ال-Buffer يساعد المرسل على تقسيم البيانات المراد إرسالها وبالتالي إمكانية إرسال إجمام أكبر من البيانات على سرعات شبكة أقل لكن كما وذكرنا إذ أن ذلك يساعد على أمرين:

الأول: إمكانية ضياع Frames أثناء الإرسال بسبب زيادة عدد تلك ال-Frames.

الثاني: حاجة المرسل إلى وقت أكثر لتقسيم تلك ال-Frames وحاجة المستقبل لوقت و Buffer أكبر لإعادة ترتيب تلك ال-Frames وبالمحصلة زيادة ال-Delay.

في البداية سنقوم بتعريف أهم المصطلحات والتي سنستخدمها في عملية إدارة ال-Buffer والتي قمنا بتخصيصها بدرس السابق في الدالة SetVoiceDevices:

**Channels**: وهو عدد القنوات التي يتم من خلالها تسجيل الصوت وعرضه وكمثال فإن نظام ال-Stereo يستخدم قناتين ويضاعف عدد القنوات المستخدمة حجم ال-Buffer بحيث أنه يضرب بعدد القنوات التي يتم استخدامها.

**Sample Peer Second**: ويسمى أيضا Sampling rate وهي تعبير عن عدد ال-Samples بالثانية الواحدة ومن المعروف أن ال-Sample هو قيمة لموقع معين من الموجة الصوتية في زمن محدد وأما ال-Sampling فهو عملية تحويل ال-continuous signal إلى discrete signal بهدف تحويل تلك ال-Signal إلى النظام الثنائي.

**Bits Peer Sample**: ويعبر عنه بال-bit depth وهو عدد ال-Bits التي ستحتاجها لتحويل ال-Wave Sample واحدة من النظام التناظري إلى الرقمي وكلما زاد عدد ال-bits كلما زادت جودة الصوت وزاد الاحتياج إلى حجم Buffer أكبر.

وبالتالي لحساب حجم الـ Buffer الذي سنحتاجه في عمليات الإرسال والاستقبال ومن خلال المعادلة التالية يمكننا معرفة الحد الأدنى لحجم الـ Buffer الذي سنحتاجه لإجراء عملية الإرسال:

*Bit rate = (bit depth) x (sampling rate) x (number of channels).*

*The Minimum Size Of The Buffer in Bytes = Channels X (BitsPerSample/8).*

*Example: 2 X (16 bits /8) = 4 Bytes*

*The Minimum Size Of The Buffer in One Second = (The Minimum Size Of The Buffer in Bytes) X (Number of Samples Peer One Second)*

*Example: 4Bytes \* 22050 = 88200 Bytes For each Second.*

وهذا يعني أننا سنحتاج إلى Buffer لا يقل عن 88200 Bytes لتخزين ثانية واحدة من الصوت قبل عملية الضغط أو الإرسال وبالتالي ولحساب السرعة المطلوبة من قناة الاتصال في الثانية الواحدة والذي إذا تحقق فسيكون هنالك مزامنة في عملية التحدث وبالتالي وصول الصوت بشكل مستمر وغير متقطع سنطبق الخطوات التالية:

أولا حساب عدد الـ Frames المحتمل والذي سنحتاجها لإرسال 88200 Bytes في الثانية الواحدة:

*(88200 the size of the buffer for each second) / (1500 Bytes the minimum size of the frame In Ethernet as example - 40 Bytes The Minimum Size of The Empty RTP Frame) = 61 Frames*

ثانيا حساب حجم الـ Frames الفارغة:

*61 Frames X 40 Bytes = 2440 Bytes*

ثالثا جمع كافة القيم وضربها بـ 8 لتحويلها إلى Bits وتقسيمها على 1024 لمعرفة السرعة المطلوبة بالـ KB :

*(88200 + 2440) (8) / 1024 = 706 K bit/Second*

وهذا يعني أننا سنحتاج إلى شبكة بسرعة لا تقل عن 706Kb/S لنقل ثنائية واحدة من Wave Voice بمواصفات: قناتين اتصال و 22050 Sampling Rate و 16 Bits لجودة الصوت.

ولمزامنة عملية التسجيل و الإرسال قمنا بتنفيذ الدالة SetBufferEvents وذلك لتقسيم ال-Buffer إلى جزأين الأول سيستخدم في تسجيل الصوت القادم من المايكروفون والثاني سيستخدم في عملية الإرسال، ولتنفيذ ذلك قمنا بإنشاء اثنين من ال-Notifications الأول سينفذ عندما يصل تعبئة ال-Buffer إلى النصف بحيث يعلن فيه عن امتلاء الجزء المخصص لتخزين الصوت وبالتالي إضافته إلى الجزء التالي من ال-Buffer المخصص للإرسال وفي حالة امتلاء الجزء الثاني سيتم إرسال ال-Buffer ومن ثم تنظيفه من جديد لتكرار العملية.

يمكن التحكم أيضا بزمن الإرسال ويعتمد ذلك على سرعة الشبكة من جهة وحجم ال-Buffer من جهة أخرى وقد قمنا في المثال السابق بإرسال كامل ال-Buffer عند كل 200 milliseconds وذلك بعد ضغطه باستخدام المعيار G.711 وهو ITU Standard ويعتبر من إحدى معايير الضغط المخصصة لصوت ويمكن باستخدام ذلك المعيار تقليل حجم ال-Wave Format إلى النصف تقريبا. لمزيد من المعلومات حول كيفية عمل هذا المعيار يمكن الرجوع إلى الرابط التالي:

<http://en.wikipedia.org/wiki/G.711>

<http://www.codeproject.com/KB/security/g711audio.aspx>

**TO Do (Documentations):**

- 1- Reflector Service Documentation
- 2- RTP Video Streaming Documentation
- 3- G.711 Documentation
- 4- Quality of Service Documentation
- 5- VoIP Encryption Documentation

**Languages:**

- 1- Arabic
- 2- English (Soon)

**Project Link:**

<http://www.socketcoder.com/ArticleFile.aspx?index=2&ArticleID=64>

© Fadi Abdelqader 2008 - 2009 [www.SocketCoder.Com](http://www.SocketCoder.Com)