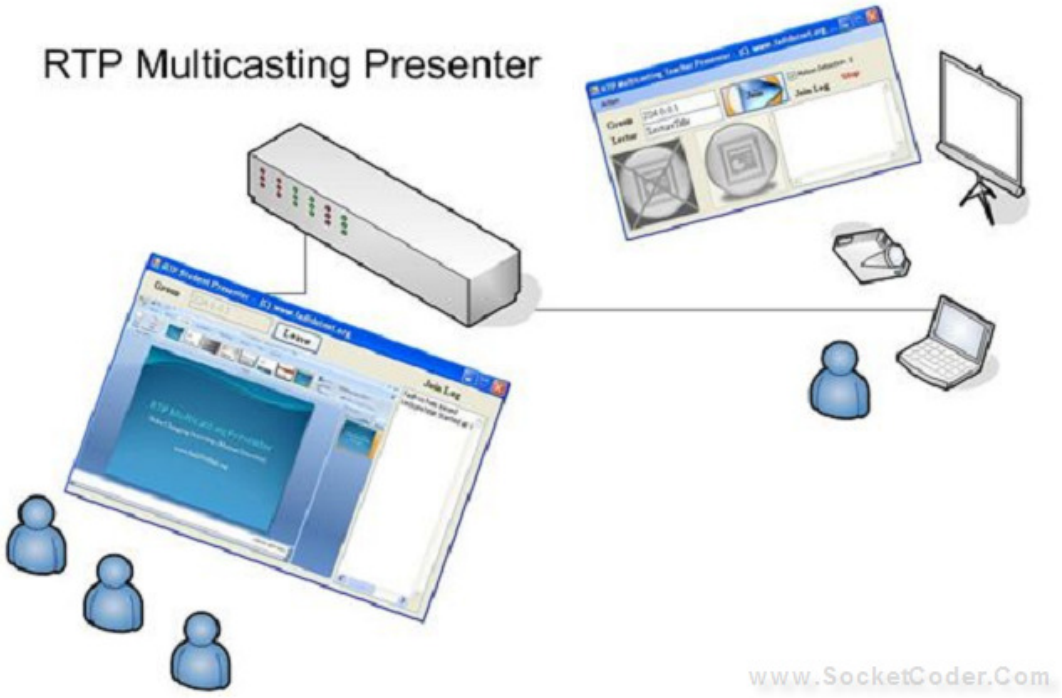
# How to use the managed RTP API classes in .NET to create your multicasting systems

**Create an RTP multicasting presenter (with motion detection)**



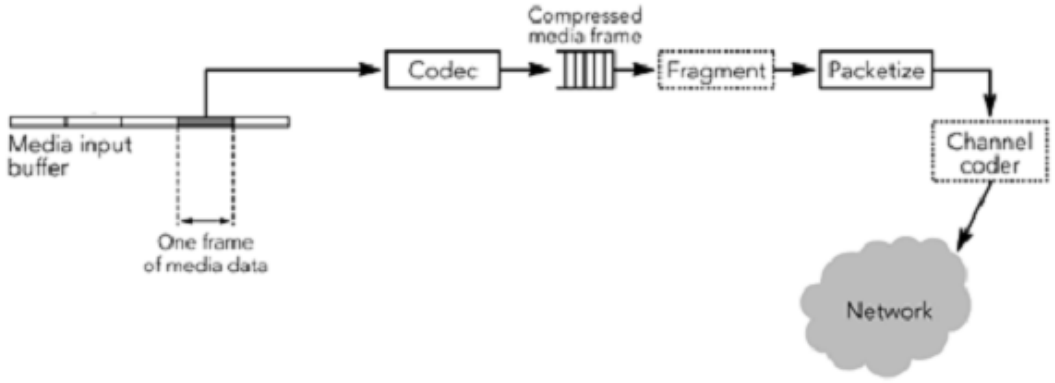

## Introduction

In this article, I will describe the architecture of RTP – Real Time Transport Protocol, and discuss the RTP managed classes for the Microsoft Conference XP Project to multicast JPEG images. For more examples of RTP programming, see www.SocketCoder.com.

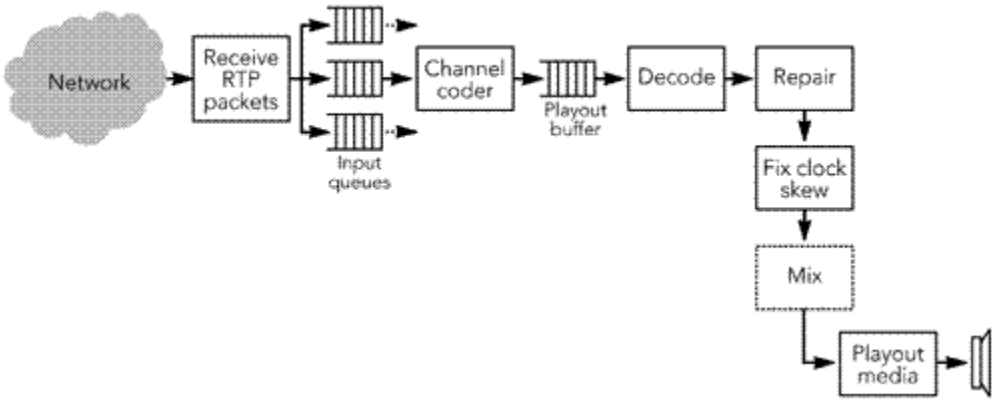
The key standard for data audio/video transport in IP networks is the Real-time Transport Protocol (RTP), along with its associated profiles and payload formats. RTP aims to provide services useful for the transport of real-time media, such as audio and video, over IP networks. These services include timing recovery, loss detection and correction, payload and source identification, reception quality feedback, media synchronization, and membership management. RTP was originally designed for use in multicast conferences, using the lightweight sessions model. Since that time, it has proven useful for a range of other applications: in H.323 video conferencing, webcasting, and TV distribution; and in both wired and cellular telephony. The protocol has been demonstrated to scale from point-to-point use to multicast sessions with thousands of users.

# Background

### How does RTP work?




A sender is responsible for capturing and transforming audiovisual data for transmission, as well as for generating RTP packets. It may also participate in error correction and congestion control by adapting the transmitted media stream in response to receiver feedback. The frames will be loaded into RTP packets, ready for sending. If frames are large, they may be fragmented into several RTP packets; if they are small, several frames may be bundled into a single RTP packet. Depending on the error correction scheme in use, a channel coder may be used to generate error correction packets or to reorder packets before transmission. After the RTP packets have been sent, the buffered media data corresponding to those packets is eventually freed. The sender must not discard data that might be needed for error correction or for the encoding process. This requirement may mean that the sender must buffer data for some time after the corresponding packets have been sent, depending on the codec and error correction scheme used. The sender is responsible for generating periodic status reports for the media streams it is generating, including those required for lip synchronization. It also receives reception quality feedback from other participants, and may use that information to adapt its transmission. A receiver is responsible for collecting RTP packets from the network, correcting any losses, recovering the timing, decompressing the media, and presenting the result to the user. It also sends reception quality feedback, allowing the sender to adapt the transmission to the receiver, and it maintains a database of participants in the session. A possible block diagram for the receiving process is shown in the figure below; implementations sometimes perform the operations in a different order depending on their needs.

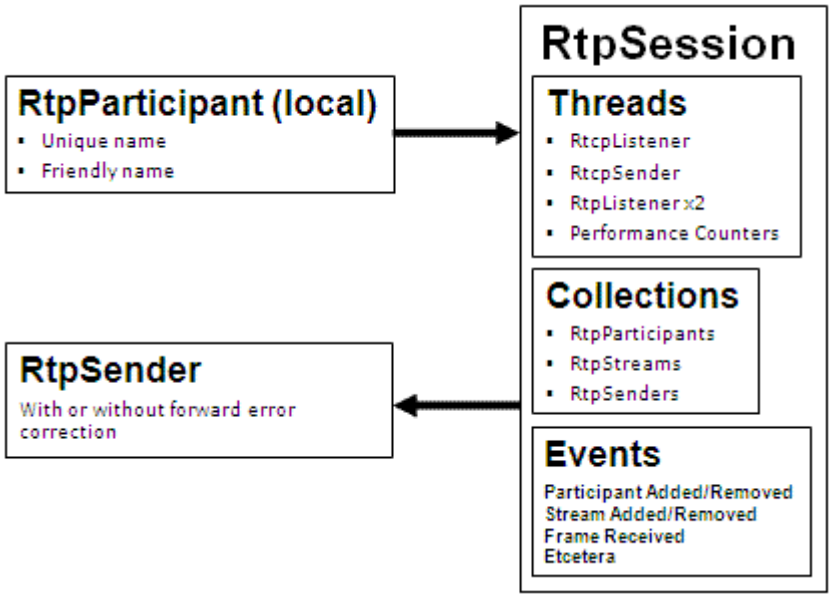



In my next article, I will provide more info about the RTP architecture to transmit audio and video. For more information about RTP, see the following links:

- SocketCoder.com
- RFC 3550 (RFC3550)

## Using the code

Microsoft has implemented the RTP in its Conference XP project. The following diagram illustrates the architecture design of the RTP in Conference XP 3.0:




These steps will give you a brief of how to use the RTP on your multicasting projects:

1. Using `RtpSession` and `RtpParticipant`: A session consists of a group of participants who are communicating using RTP. A participant may be active in multiple RTP sessions—for instance, one session for exchanging audio data, and another session for exchanging video data. For each participant, the session is identified by a network address, and a port pair

to which data should be sent and a port pair on which data is received. The send and receive ports may be the same. Each port pair comprises of two adjacent ports: an even-numbered port for RTP data packets, and the next higher (odd-numbered) port for RTCP control packets. The default port pair is 5004 and 5005 for UDP/IP, but many applications dynamically allocate ports during session setup and ignore the default. RTP sessions are designed to transport a single type of media; in a multimedia communication, each media type should be carried in a separate RTP session. We will use the `RtpSession` and `RtpParticipant` classes to:

- Manages all of the RTP objects and data.
- Hold information about a user.
- To send or receive RTP data.

We have first hook some RTP events to communicate for what is happening in the RTP API process. These are:

```
// Manage the join to the session Ex.Add/Remove a User To/From the RTP
Session
RtpEvents.RtpParticipantAdded += new
   RtpEvents.RtpParticipantAddedEventHandler(RtpParticipantAdded);
RtpEvents.RtpParticipantRemoved += new
   RtpEvents.RtpParticipantRemovedEventHandler(RtpParticipantRemoved);
```

**Examples of using the above event declarations:**

```
private void RtpParticipantAdded(object sender,
RtpEvents.RtpParticipantEventArgs ea)
{
  MessageBox.Show (ea.RtpParticipant.Name + " has joined to the
session");
}

private void RtpParticipantRemoved(object sender,
             RtpEvents.RtpParticipantEventArgs ea)
{
  MessageBox.Show(ea.RtpParticipant.Name + " has left the
session");
}

// Manage (Add/Remove)Sessions Ex.Activeate multiple RTP
sessions
RtpEvents.RtpStreamAdded +=
  new RtpEvents.RtpStreamAddedEventHandler(RtpStreamAdded);
RtpEvents.RtpStreamRemoved +=
  new RtpEvents.RtpStreamRemovedEventHandler(RtpStreamRemoved);
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);
}
```

Then to join the RTP session, we have to specify the type of the RTP packet payload. Each session can contain only payload type:

```
RtpSession rtpSession;

private void JoinRtpSession(string SessionName,string name)
{
  rtpSession = new RtpSession(ep, new
RtpParticipant(SessionName, name),
                              true, true);
  rtpSender = rtpSession.CreateRtpSenderFec(name,
PayloadType.JPEG, null, 0, 200);
}
private void LeaveRtpSession()
{
  // Clean up all outstanding objects owned by the RtpSession
  rtpSession.Dispose();
}
```

Finally, after joining the session, we can get the RTP stream buffer, as below:

```
private void FrameReceived(object sender,
RtpStream.FrameReceivedEventArgs ea)
{
  System.IO.MemoryStream ms = new
MemoryStream(ea.Frame.Buffer);
  pictureBox_Receive.Image = Image.FromStream(ms);
}
```

- Using `RtpSender` and `RtpListener`: The sender is responsible for sending the captured data and generating RTP packets - the data can be from live capturing or from a file - compressing it for transmission. For example, converting a bitmap image to a JPEG compressed image, as I used in my example. The sender starts by reading media data, such as video frames, into a buffer from which encoded frames are produced. In the Managed RTP Library, the `RtpSender` is used for:
  - Sending data across the network.
  - Send with or without data being forward error corrected.

  And, in the `RTPListener`:

  - One thread receives data off the network.
  - One thread distributes packets to the appropriate stream for processing.

  Use the `RTPSender` as shown below:

```
RtpSender rtpSender;

MemoryStream ms = new MemoryStream();
// Compressed  the captured image as JPEG image format
pictureBox_sender.Image.Save(ms, ImageFormat.Jpeg);
// Send The The Comressed Image as Bytes stream
rtpSender.Send(ms.GetBuffer());
```

**RTCP Management:** RTCP packets are defined in the RTP specification: receiver report (RR), sender report (SR), source description (SDES), membership management (BYE), and application-defined (APP). The RTCP Sender is used for outgoing RTCP data where:

- Local participants and streams are joining or leaving.
- Sender and receiver reports for network status.

`RtcpListener` is used to:

- Processes incoming RTCP (RTP Control Protocol) data.
- Participants and streams join or leave.
- Sender and receiver reports for network status.

- Performance Counters

Used to keep track of interesting network statistics, such as bytes, packets, frames per second, lost bytes, and recovered bytes. All of these are contained in the RTP API class properties.

**Increase network usage performance:**

To increase the performance in my example, I used a mechanism to send just only the new screen-captured image that is different than the previous captured image. This will reduce the usage of the network resources so I compare the pixels of the image before sending it, and to speed up the comparison, I resize the image to 100X100 and then I compare it pixel by pixel, as shown below:

```
public float difference(Image OrginalImage, Image SecoundImage)
{
    float percent = 0;
    try
    {
        float counter = 0;
        Bitmap bt1 = new Bitmap(OrginalImage);
        Bitmap bt2 = new Bitmap(SecoundImage);
        int size_H = bt1.Size.Height;
        int size_W = bt1.Size.Width;
        float total = size_H * size_W; Color pixel_image1;
        Color pixel_image2;
        for (int x = 0; x != size_W; x++)
        {
            for (int y = 0; y != size_H; y++)
            {
                pixel_image1 = bt1.GetPixel(x, y);
                pixel_image2 = bt2.GetPixel(x, y);
                if (pixel_image1 != pixel_image2)
                {counter++; }
            }
        }
        percent = (counter / total) * 100;
    }
    catch (Exception) {percent=100;}
    return percent;
}
```

The above method will calculate the difference pixels in the new captured image so we can decide to send it or not depending on the returned difference percentage.

## References


- Microsoft Conference XP Project
- RTP: Audio and Video for the Internet, Addison Wesley 2003.
- The Complete Reference to Network Programming, FADI Abdelqader (under writing). See: www.SocketCoder.Com.
- RFC3550.


## License



## About the Author

Fadi M. Abdelqader
Personal Site: www.SocketCoder.com

Education:
- MA. Degree In CINS DePaul University

- Bsc. In Computer Networking From Phildelphia University Graduate With Honor.

Researches:
He Published a Set of Researches to Develop a Set of Advanced Systems Such as Remote Monitoring System, Remote Controlling System, Advanced Video Conferencing System, Voice Communication & Voice Conferencing System & Remote Classrooms & Distance Learning Systems

Books:
He Authored a Book in .Net Network, Distributed Systems & TCP/IP Programming ; Read all information about this book in his site
http://www.socketcoder.com/ReleasedBooks.aspx?index=1