# WinRTP

Software Unit Design Specification

DirectShow RTP Library for Windows

# Definitions

| | |
|---|---|
| **Introduction** | This section defines words, acronyms, and actions which may not be readily understood. |
| **RTP** | Real-time Transport Protocol. Internet standard for transport of real-time media streams. |
| **MTC** | Media Termination Component. |

# Problem Definition

**Media Termination Component**

The media termination component (MTC) provides an interface to the call control component that allows it to carry out a variety of tasks related to media termination. It can do the following:

- Capture audio from the user's microphone
- Convert sampled audio into other formats like G711, GSM etc.
- Send encoded audio to a predefined destination IP address as an RTP stream
- Receive RTP stream audio from the network and play it through the speaker of the local machine
- Play an audio file from disk to the local speaker and/or send it as an audio stream over the network
- Play sounds like dial tone, DTMF tones and other sounds locally in response to events like the user picking up the phone, pressing a number on the keypad etc.

# Design Considerations

**Introduction**

WinRTP is implemented as an ActiveX component using Microsoft Visual C++. Java applications access it through the J/Direct extensions in the MS Java Virtual Machine.

## Support for Encoders

**Codecs Supported**

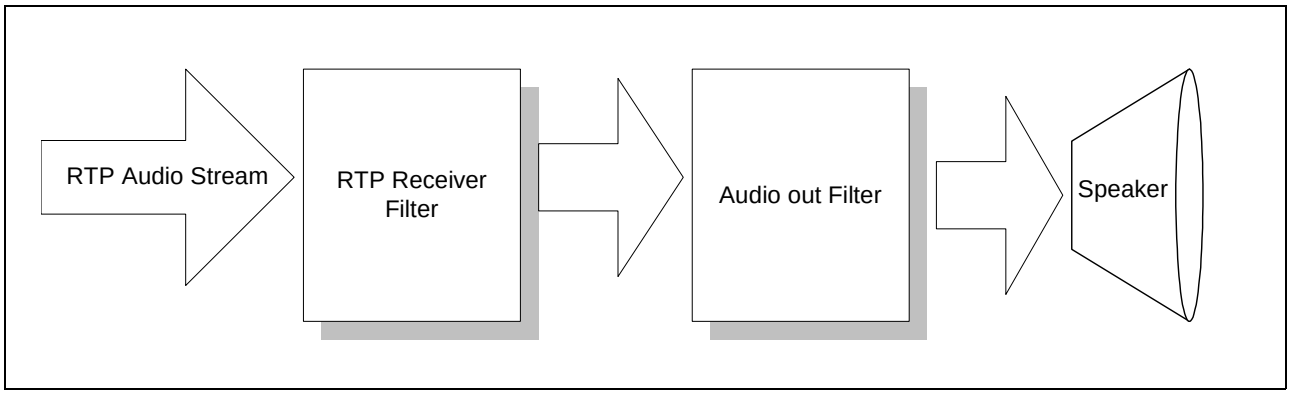
WinRTP can support different codecs, including some high bit-rate audio codecs (in future) which will provide it with an edge over standard telephones. As of now, it supports G.711 (64kbps, both Alaw and Ulaw) G.729 (8kbps), and G.723 (both 5.3kbps and 6.4kbps varieties).

## Audio Processing API

**DirectShow**

For audio processing, Microsoft's DirectShow architecture will be used. The DirectShow API is part of Microsoft's DirectX Media SDK (version 6.0). It is supported in Win 95/98/2000 and also Win NT sp 3+. It allows the user to process audio/video by constructing 'filter graphs'. A filter graph is basically a collection of audio processing blocks (or filters) which are connected together. DirectShow provides a filter graph manager which manages and executes a filter graph. Thus, the application developer has to design and implement certain filters according to the DirectShow specifications and design the proper filter graphs. DirectShow takes care of the rest.

**Filter Graph Illustration**




**Quick Introduction to DirectShow**

For a quick introduction to DirectShow, consider the filter graph, illustrated above, which receives audio from the network and plays it to the speaker

- The RTP receiver filter is a ‘source filter’, i.e. in this case a filter that generates audio. It does it by receiving packets form the network which contain the audio. Its job is to receive the packets, strip off the header part, extract the audio, and send it out
- The Audio out filter plays audio to the speaker. It receives audio in its input, and makes the proper calls to the system for playing it through the speaker
- To execute the filter graph, we need to
  - Instantiate the filters
  - Connect them (by specifying the audio format of the data that will be passed from the rtp server to the audio out filter.)
  - Run the filter graph. The DirectShow “Filter Graph Manager” (FGM) takes care of this. The mechanism is as follows:
    - The FGM creates blank arrays called media samples and passes it to the source filter (in this case the rtp receiver filter) for filling
    - The rtp filter fills the blank sample with audio
    - The FGM passes the sample downstream to the filter connected to the output of the source filter
    - The above 3 steps are repeated as long as the graph is running

## Filter Graph Setup

**Set Up Once and Alter**

Setting up filter graphs is a complex process (in terms of execution) and requires time. Hence it would be better to construct a relatively complete filter graph once and alter it minimally whenever new functionality needs to be turned on or off. This would be faster than making major changes to it or setting up a new filter graph every time. The current design of WinRTP does just that.

# Jitter Suppression Algorithm for Receive Stream

**Preventing A Crackled Sound**

The RTP packets received by WinRTP may not arrive uniformly separated in time. Due to jitter in arrival time, the received audio may sound crackled, reducing its quality. To take care of this, a simple jitter removal algorithm has been implemented by WinRTP, described as follows. When an audio frame is late in arriving and its playing time has arrived, the previous audio frame is replayed. Since the audio frames used for processing within the filter graph are of the order of 30ms in length, the user cannot detect the difference. If the same frame needs to be replayed many times, it is done so with exponentially reducing volume (the volume is halved for every replay after the first replay).

# QoS of Transmitted Stream

## DiffServ Code Point / IP Precedence

**Per Hop Behavior**

WinRTP has the ability to set the DiffServ code point values for outgoing RTP packets. WinRTP sets the Per Hop Behavior (PHB) in the deprecated IP precedence field to “101110b”.

## IEEE 802.1p User Priority

**Not Supported**

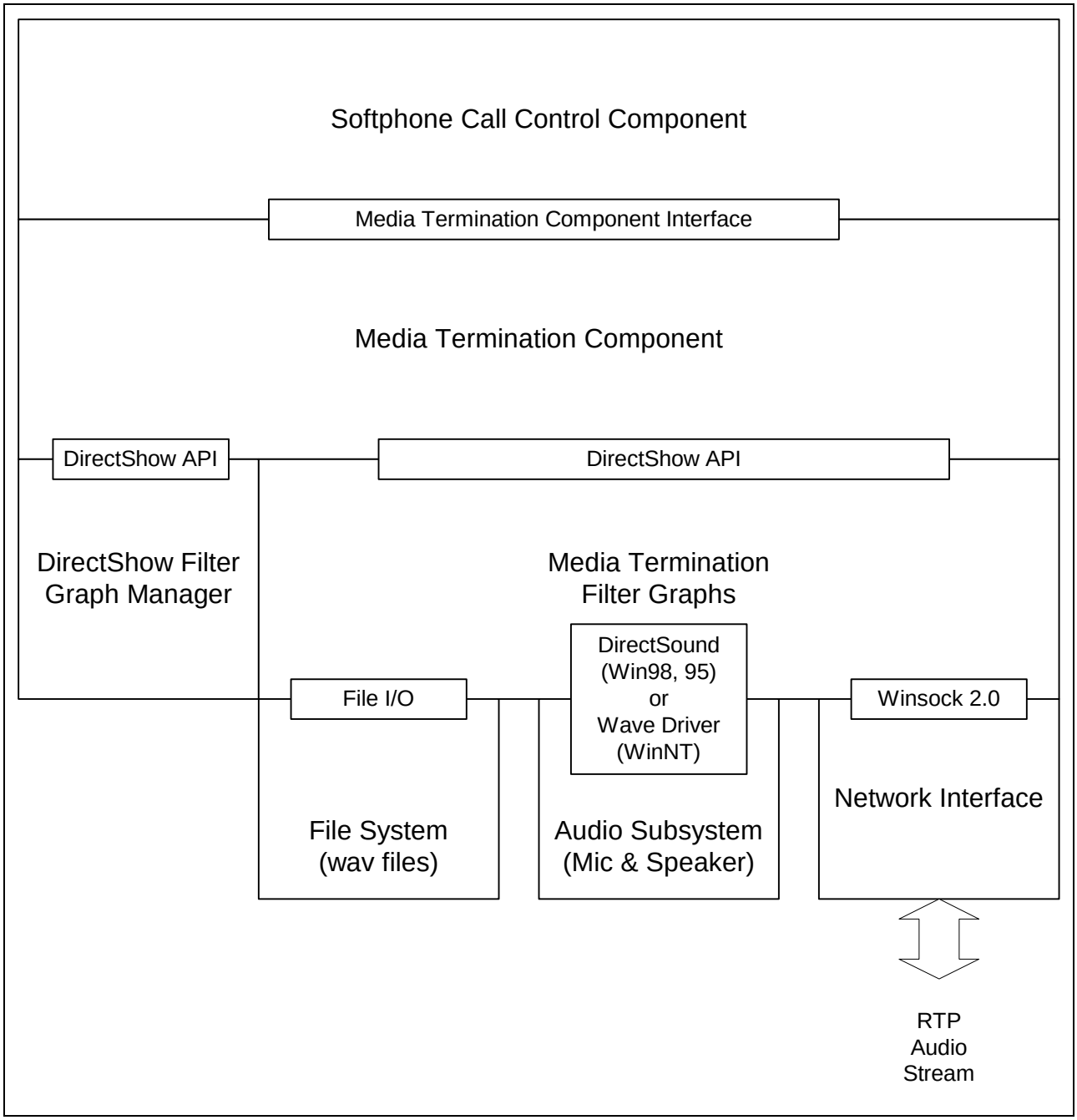
The designers of WinRTP wanted to set the IEEE 802.1p user priority field of outgoing frames to 5. But Windows 95, 98 or NT do not allow the user to set this field. Windows 2000 allows it, but the documentation is very unclear. As such, WinRTP does not support this setting yet.

# Functional Structure

**Illustration**

The following illustration shows the functional structure of a WinRTP system.




**Details from the Illustration**

- WinRTP will initially set up 2 filter graphs, once for the send side, and one for the receive side and start running them with all audio streams disabled. (i.e. switches turned off),
- The call control component will make calls to WinRTP using WinRTP's COM interface.

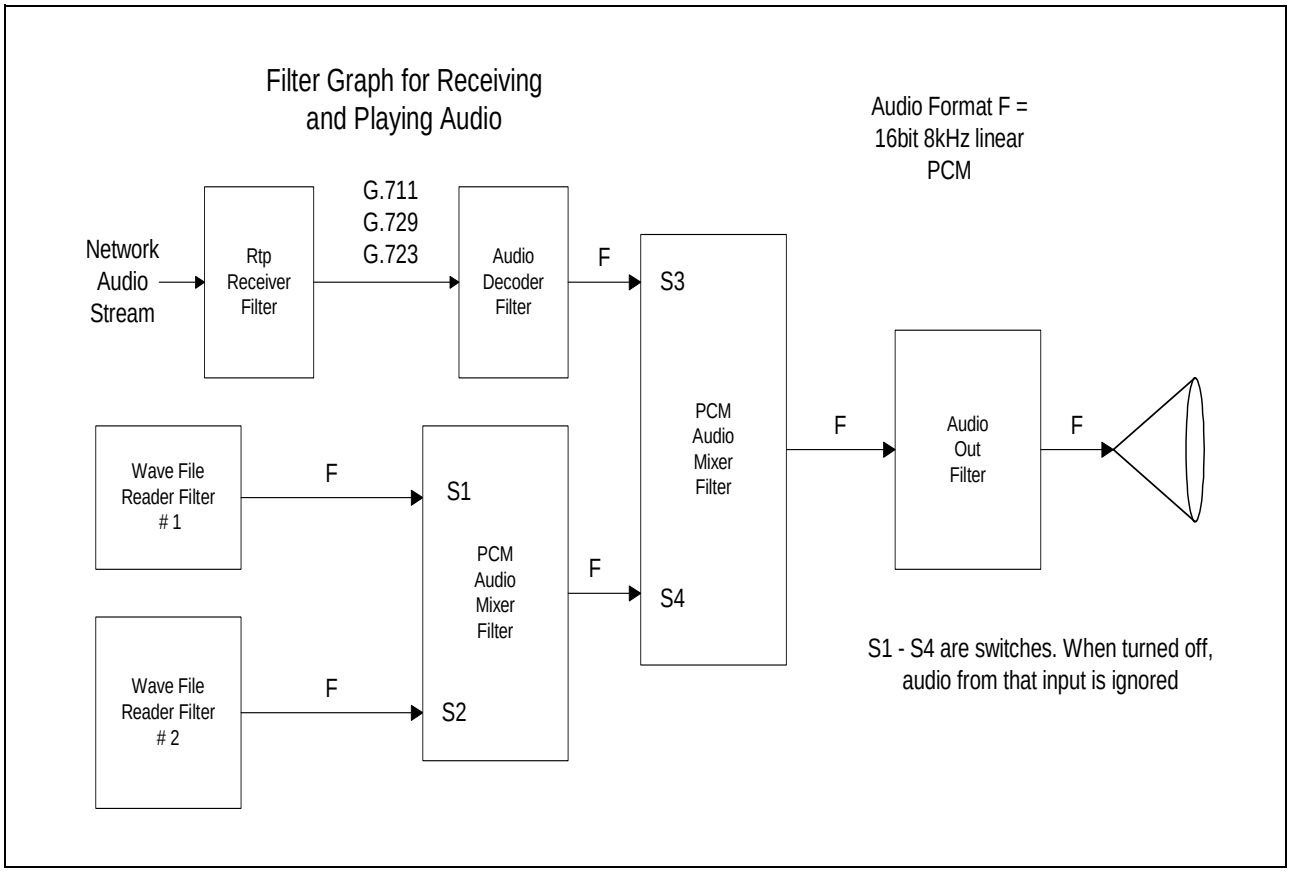
- In response to these calls, the Media Termination component will change the status of the filter graph to turn on or off certain options, like File Playing, DTMF Tone Generation, etc. For e.g. turning on switch S3 in the send side filtergraph turns on Microphone (i.e. starts sending microphone input to the network).

**Filter Graphs**

The following illustrations show the filter graphs used in the component.

**Receiving and Playing Audio**

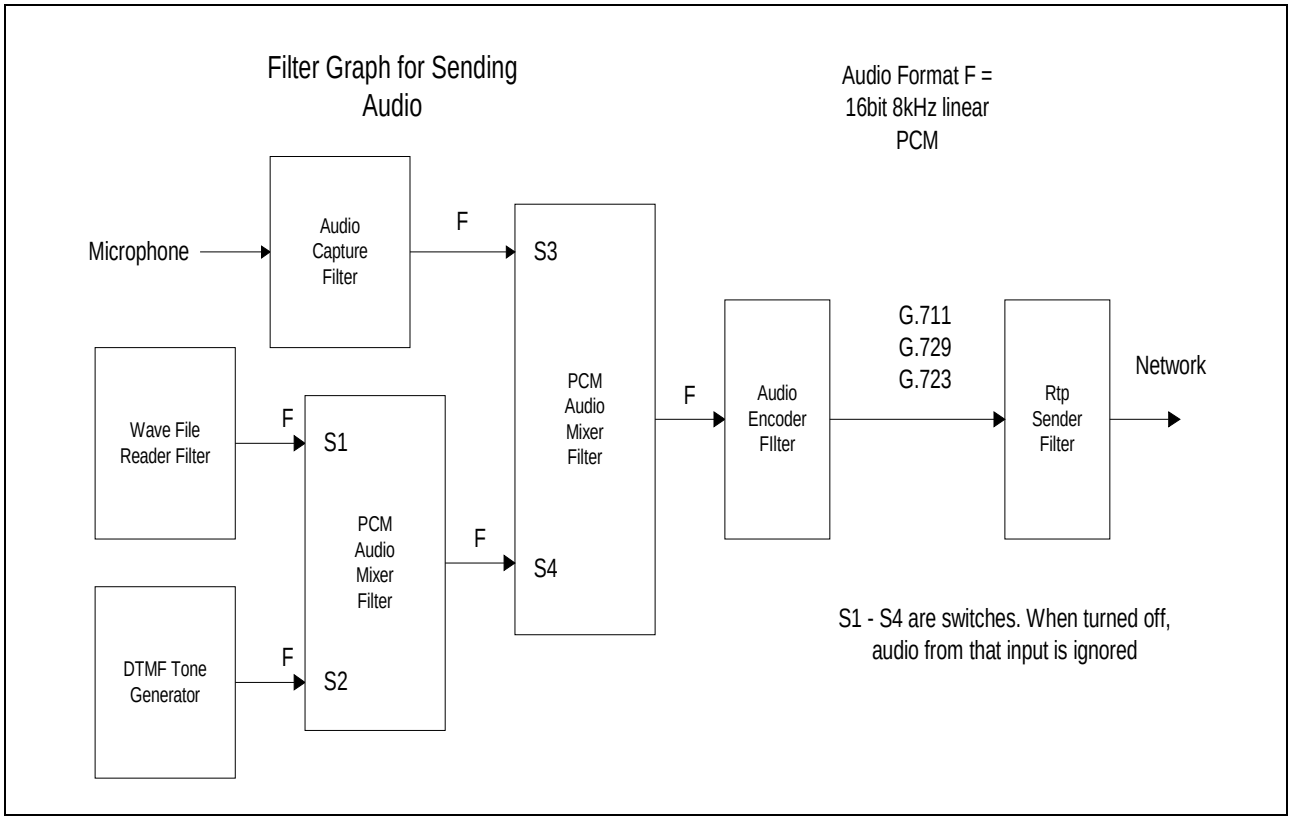
The following illustration shows the filter graph used for receiving and playing audio.



**Sending Audio**

The following illustration shows the filter graph used for sending audio.




# Description of Filters

## RTP Receiver Filter

**Listens for Audio**

Listens to the network for an RTP data stream carrying encoded audio. It extracts the audio data from the packets and forwards it to the next filter. The UDP port number where it should listen has to be specified earlier. This filter has to been implemented.

## Audio Encoder and Decoder Filters

**Transforms PCM Audio**

These filters transform linear PCM audio to and from different formats. Currently G.711, G.723 (both 5.3 and 6.4 kbps) and G.729 codec filters have been implemented. Implementations of G.723 and G.729 codecs must be licensed.

## PCM Audio Mixer Filter

| | |
|---|---|
| **Mixes Streams** | This versatile filter can mix two PCM audio streams. Its inputs are ‘gated’, so that we can enable or disable any one or both of them. This allows us to keep the filter graph running but still ignore certain audio streams and allow some to pass through. This filter has been implemented. |

## Audio Out Filter

| | |
|---|---|
| **Uses DirectSound or Wave Driver** | For Audio Out, MTC uses either DirectSound (for Win98 and 95) or the Wave driver (for Win NT). Both are provided by DirectShow and they drive the computer’s speaker. |

## Audio Capture Filter

| | |
|---|---|
| **Renders Audio as PCM Audio** | Listens to the audio input devices (including microphone) and renders it as PCM audio. Provided by DirectShow. |

## Audio File Source

| | |
|---|---|
| **Renders Audio Files** | Reads an audio file (e.g. wave file) and renders it. All audio files have to be WAV files with the following format:- 16bit 8kHz PCM (linear). This filter has been implemented. |

## RTP Sender Filter

| | |
|---|---|
| **Produces RTP Data Stream** | Receives audio samples at its input, puts them in RTP data packets, and sends them over the network to a predefined destination. The destination IP address and port number can be programmed. This filter does not implement the RTCP control protocol. It only produces an RTP data stream. It has been implemented. |

## DTMF Tone Generator Filter

| | |
|---|---|
| **DTMF Tones in PCM Format** | Generates DTMF tones in PCM format. This filter has been implemented. |

# Filter Graph Operation

## Startup Phase

**Specifying Parameters**

In the startup phase, both send and receive side filter graphs are started. Before the graphs may be started, some other parameters need to be specified. They are the following:

- Destination IP address (for the send side). This has to be set before the Rtp Sender filter starts
- Port # for incoming audio (for the receive side). This is used by the Rtp Receiver filter
- Audio format to be used. This determines which codec filter will be instantiated and put in the filter graph

Once these parameters have been specified, both the send and receive side filter graphs are started.

## Execution Phase

**Send, Receive, Play Files**

In this phase, WinRTP can send and receive audio, play files locally and/or to the remote station, and so on. This phase is pretty simple. In response to requests from the CCC, WinRTP turns on or off some of the switches (S1 – S4) in the send or receive filter graph. For example, if we want to play a file to the remote caller, WinRTP loads the file into the send side filter graph's wave file reader filter, and turns on switches S1 and S4. All this takes place while the filter graph is running, so the voice path from the microphone to the network is undisturbed.

## Stopping Phase

**Filter Graphs are Stopped**

In this phase, the filter graphs are stopped. This takes a few hundred milliseconds worth of time, because quite a few threads have to be stopped, some buffers have to be cleared and so on.

## Stopping and Restarting the Filter Graphs

**Stop, Change Parameters, Restart**

When some of the parameters have to be changed, the calling application needs to stop the filter graph(s), change the parameters, and restart them. Such parameters include the destination IP address, the local Port # for received audio, the audio codec to be used, etc. Stopping is necessary because these changes alter the filter graph in some ways which cannot be done to a running filter graph.

## File Play and DTMF Capabilities

**Play Once or Loop**

As seen from the graphs, WinRTP has the ability to play at most one file to the remote caller (at a time), and two files at a time locally. These files can either be played once, or looped until stopped. Besides, WinRTP can also generate DTMF tones to the audio stream. When DTMF tones are played, all other audio streams (in the send side) are muted. They are resumed when the tone ends. WinRTP has an event mechanism using which it fires events when a file play is finished, so that the application can take appropriate action. When the user wants to play a file to the remote caller, but also hear it so that he/she knows how much the other person has heard, we play the same file on both filter graphs starting at the same time.

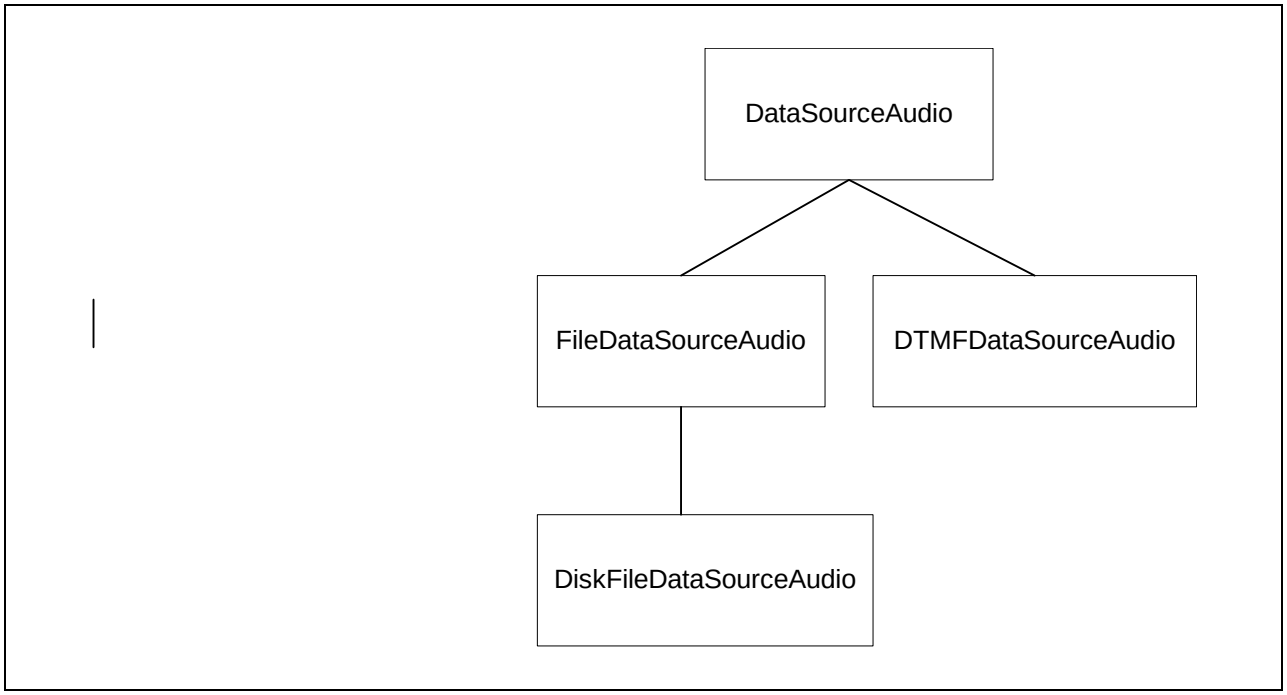
# Data Structures and Code Modules

**Visual C++**

WinRTP is coded using Microsoft Visual C++ 6.0. All the code is organized in one work space called CCNMediaTerm. This work space contains a number of projects. The main project is CCNSMT (CCN Software Media Termination) which is the program that exposes WinRTP interface, creates/runs the filter graphs, etc. There is one project for each filter which was developed, and they are Rtp Receiver Filter, Rtp Sender Filter, Data Source filter, PCM Audio Mixer filter, G.711 codec filter, G.723 codec filter (for 5.3 kbps rate), G.723 codec filter (for 6.4kbps rate), and G.729 codec filter. Here are some notes on the implementation of the above filters:

- PCM Audio Mixer: This filter decides on a sampling time, say N milliseconds, based on the length of audio samples in its input. It then samples its 2 input pins every N milliseconds, mixes them if both input pins had input, and sends it though the output pin. If any input pin is disabled (the corresponding switch is turned off) then this filter consumes the input from that pin but does nothing else with it.
- G.711 codec filter: It does transcoding between G.711 (Alaw or Ulaw) and PCM 16bit 8kHz. Uses table lookups for all conversions.
- G.729 codec filter: Uses libraries (for compiling) and dll's (for runtime) licensed from MiBridge (www.mibridge.com) Source code for this codec must be licensed from MiBridge.
- G.723 codec filters: Again, source and binaries must be licensed from MiBridge.
- Data Source Filter: This is a generic filter which can be coupled with a user specifiable source of data. Sources of data should be derived from a class which has been implemented called DataSourceAudio. Code for WinRTP implements two types of data sources, Wav File data source, and DTMF data source.

**Illustration** The following illustration shows the class hierarchy.



# System Flow

**Introduction**

WinRTP is an ActiveX control and the application makes calls to it through the interface. If any of the calls to WinRTP generates an error condition, WinRTP returns a negative value which raises an exception in the java code calling it.

## Event Passing Mechanism

**Asynchronous Tones**

The calls to play files and DTMF tones are asynchronous. To notify the container when file play has ended.WinRTP passes events implemented using the Connection Point mechanism. Basically the container of the ActiveX control (MTC) implements an event sink interface defined by WinRTP. WinRTP makes calls to this interface when it needs to fire events. Currently events are fired when a file play ends.

# Program Interface

**Functions**

WinRTP's interface contains the following functions. For all these functions, if any parameter is invalid or an error condition occurs during execution, the function returns an error condition (E_FAIL) which raises an exception in the java container of the component.

## StartTX()

**Transmit Side**

Starts the transmit side of the filter graph.

## StopTX()

**Transmit Side**

Stops the transmit side of the filter graph.

## StartRX()

**Receive Side**

Starts the receive side of the filter graph.

## StopRX()

**Receive Side**

Stops the receive side of the filter graph.

## SetAudioCodecRX

**Receive Side Audio Codec**

(long CompressionType,
long MillisecPacketSize,
long EchoCancellationValue,
long G723BitRate)

Sets the Audio codec to be used for the receive side. The received RTP stream should be encoded in this format.

- CompressionType

  G.711 Alaw (64kbps)= 2
  G.711 Ulaw (64kbps)= 4
  G.723.1 (5.3 or 6.4kbps)= 9
  G.729 (8kbps)= 11

- MillisecPacketSize

Length of audio in each received packet in milliseconds Typical values are 20ms or 30ms (for G.711) and 90ms for G.723. This is ignored by MTC because it buffers the received audio.

- EchoCancellationValue

  Echo Cancellation off = 0

  Echo Cancellation On = 1

  Ignored, because Echo Cancellation is currently Not implemented by MTC.

- G723BitRate

  1 = G.723 at 5.3 kbps if CompressionType = 9

  2 = G.723 at 6.4 kbps if CompressionType = 9

  for all other formats.

---

## SetAudioCodecTX

---

**Transmit Side Audio Codec**

long CompressionType,
long MillisecPacketSize,
long PrecedenceValue,
long SilenceSuppression,
unsigned short MaxFramesPerPacket,
long G723BitRate

Sets the audio format of outgoing packets.

- CompressionType
- MillisecPacketSize
- G723BitRate (see SetAudioCodecRX)
- PrecedenceValue

  IP Precedence value of outgoing packets. Ignored because MTC uses the value as described earlier

- SilenceSuppression

  = Silence Suppression OFF

  = Silence Suppression ON

  WinRTP currently does not implement this feature. This parameter is ignored

- MaxFramesPerPacket

  The number of audio frames to put in a packet

---

## SetAudioDestination (String Hostname, long UDPPortNumber)

**Packet Destination Address**

- Hostname: string containing IP address of destination. E.g. “127.0.0.1”
- UDPPortNumber: destination port number.

Set the destination address where the packets will be transmitted. This has an effect instantly.

## SetAudioReceivePort (long UDPPortNumber)

**Port for Listening to Incoming Audio**

- UDPPortNumber = port to receive audio stream.

Set the port to listen to for incoming audio. Has no effect until the next time StartAudioReceive() is called after this call. So if the receive side filter graph is already running, need to stop it and start it again.

## StartMicrophone()

**Capture and Transmission**

Starts capturing Microphone input and transmitting it to destination. No change if already transmitting.

## StopMicrophone()

**Stops Audio**

Stops transmitting audio from the user’s microphone. Does nothing if already not transmitting from the microphone.

## StartAudioReceive()

**Plays Stream to Local Speaker**

Starts playing the received RTP stream to the local speaker. If it is already playing, it does nothing and returns.

## StopAudioReceive()

**Stops Playing Stream**

Stops playing the received RTP audio stream to the speaker. If already stopped, does nothing.

## StartDtmfTone (long ToneAsChar, long OnTime, long OffTime)

**Plays Tone in Output Stream**

- ToneAsChar: The tone to play (char) cast as long. Valid values are '0' – '9', 'a' – 'd', '*', '#'
  It is case insensitive.
- OnTime: The duration of the tone in milliseconds. Typically 50 or 100ms.
- OffTime: Duration of the silence that follows the tone in milliseconds. Typically 50 or 100ms.

Starts playing a DTMF tone in the output stream. All other output streams are temporarily stopped (these include the microphone and file play). Automatically restores previous state when the tone ends or is stopped by calling StopDtmfTone(). If a DTMF tone is already playing, return an error.

## StopDtmfTone()

**Stops Tone**

Stops the DTMF tone being played. If no tone is playing, does nothing.

## StartPlayingFileRX

**Starts Playing WAV File**

Char * Filename
unsigned long Mode
unsigned long StartPosition,
unsigned long StopPosition
long * Cookie

Starts playing the WAV file specified in Filename to the speaker. The file has to be in 16bit 8kHz PCM WAV file format.

- Filename: name of the file to play
- Mode =0 for looping mode (plays the file over and over again until stopped) or
  1 for play once
- StartPosition: Milliseconds of audio to skip at the beginning of the file. 0 plays from beginning
- StopPosition: Position in milliseconds to stop. 0 plays to the end
- Cookie: a number is returned through this parameter. This number may be used later to refer to this stream (for e.g. in StopPlayingFileRX()).

e.g. StartPlayingFileRX("c:\\hello.wav", 0, 100, 2000) will play the audio contained in hello.wav between the positions 100ms and 2 secs.

If start or stop positions are invalid, returns error. Two files can be playing in the receive side at the same time. If two files are already playing, returns an error.

When the file ends (in non looping mode), MTC automatically stops it and fires EndOfFileEventRX(cookie). The container can trap this event and know which stream stopped by looking at cookie.

## StopPlayingFileRX(unsigned long cookie)

**Stops Identified File**

Stops the file play identified by 'cookie' (returned during StartPlayingFileRX()). If stream is already stopped does nothing and returns.

## StartPlayingFileTX

**Plays File to RTP Output Stream**

Char * Filename
unsigned long Mode
unsigned long StartPosition,
unsigned long StopPosition
long * Cookie

Exactly the same as StartPlayingFileRX(), except that it plays the file to the RTP output stream, and that only one file may be played at a time to the output stream. If a file is already playing returns an error.

If the file ends, then stops it and fires the EndOfFileEventTX(Cookie) event to the container of this ActiveX control.

## StopPlayingFileTX (unsigned long cookie)

**Stops Transmission**

See StopEndOfFileRX().

## A Typical Series of Calls to WinRTP

**Examples**

For the send side initialization:

```
SetAudioCodecTX(4, 30, 0, 0, 1, 0);
SetAudioDestination("127.0.0.1", 21243);
StartTX();
```

For receive side initialization:

```
SetAudioCodecRX(4, 30, 0, 0);
SetAudioReceivePort(8283);
StartRX();
```

This will start the send and receive side filter graphs, but still not audio will be sent or received, because no audio stream has been turned on.

To start sending voice to the other side, and then play a file, stop it after 1 second and then play a DTMF tone (*):

```
StartMicrophone();
Unsigned long cookie;
StartPlayingFileTX(“c:\hello.wav”, 1, 0, 0,
&cookie);
Sleep(1000);
StopPlayingFileTX(cookie);
StartDtmfTone(‘*’, 50, 50);
```

To play the received audio from the network along with 2 files, one looping, and the other once from an offset of 1 sec into the file, followed by a change in the receive port when the second file ends playing the receive network audio again:

```
StartAudioReceive();
Unsigned long cookie1, cookie2;
StartPlayingFileRX(“c:\hello.wav”, 0, 0, 0,
&cookie1);
StartPlayingFileRX(“c:\hi.wav”, 1, 1000, 0,
&cookie2);

/* wait for EndOfFileEventRX(cookie2) to be fired */

StopRX();  //needed, as explained earlier in
           //SetAudioReceivePort(). Stops all file
           //plays and playing of network audio
           //permanently until they are started
again

SetAudioReceivePort(23232);

StartRX(); //at this point, again no streams are
           //playing.

StartAudioReceive();
```

For Stopping everything:

```
StopRX();
StopTX();
```

---

# SW Restrictions and Configuration

| | |
|---|---|
| **Software Requirements** | Needs Windows 95, 98, NT service pack 3 or later, or Windows 2000.<br>Needs DirectX Media Run Time version 6.0 or later. |

# HW Restrictions and Configuration

| | |
|---|---|
| **Hardware Requirements** | Sound card compatible with DirectX Media 6.0 or later, microphone, headset/ speakers.<br>At least a 266MHz MMX Pentium microprocessor or equivalent.<br>At least 32 MB RAM. |

# References

**Web Links**

Microsoft DirectX Media SDK 6.0 documentation
http://www.microsoft.com/directx

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, _RTP: a transport protocol for real-time applications,_ Request for Comments (Proposed Standard) 1889, Internet Engineering Task Force, Jan. 1996. http://www.ietf.org/rfc/rfc1889.txt?number=1889