**VoIP Recorder**

**Project Overview**

**Authorized for release by:**

………………………………………………………………………….

**Philip D. Hill**

Author : Ian Spinks

E-mail of author : ian.spinks@irl-recording.com

Issue of specification : 1

Date of specification : 17th June 2010

This work is copyright © 2010 Intelligent Recording. No part of it may be reproduced or used in any form without the express permission in writing of an authorized officer of Intelligent Recording.

These restrictions extend to all media in which this information may be contained.

Date & Issue: Issue 2 – August 2010

[www.usbcallrecord.com](http://www.usbcallrecord.com)

US Office +1 (949) 587 1226

UK Office +44 (0) 1909 512 148

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Release | Date | Author | Comment |
| Issue 1  Issue 2 | 17-06-2010  02-08-2010 | Ian Spinks  Ian Spinks | First release  Add modification to the Audio codecs |

**Table of Contents**

[INTRODUCTION 4](#_Toc266185777)

[PROJECT AIMS 5](#_Toc266185778)

[PROJECT DELIVERABLES 6](#_Toc266185779)

[Dynamic Link Library’s 6](#_Toc266185780)

[BackOffice Recorder 12](#_Toc266185781)

[Licensing 12](#_Toc266185782)

[Port Configuration 12](#_Toc266185783)

[Phone registering 12](#_Toc266185784)

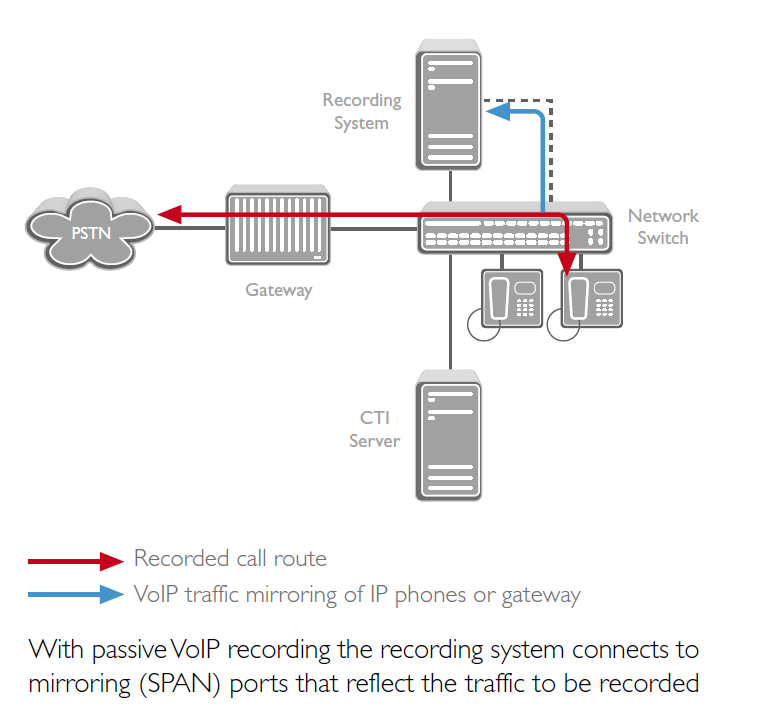
[Playback 12](#_Toc266185785)

[GENERAL ISSUES WITH VOIP 13](#_Toc266185786)

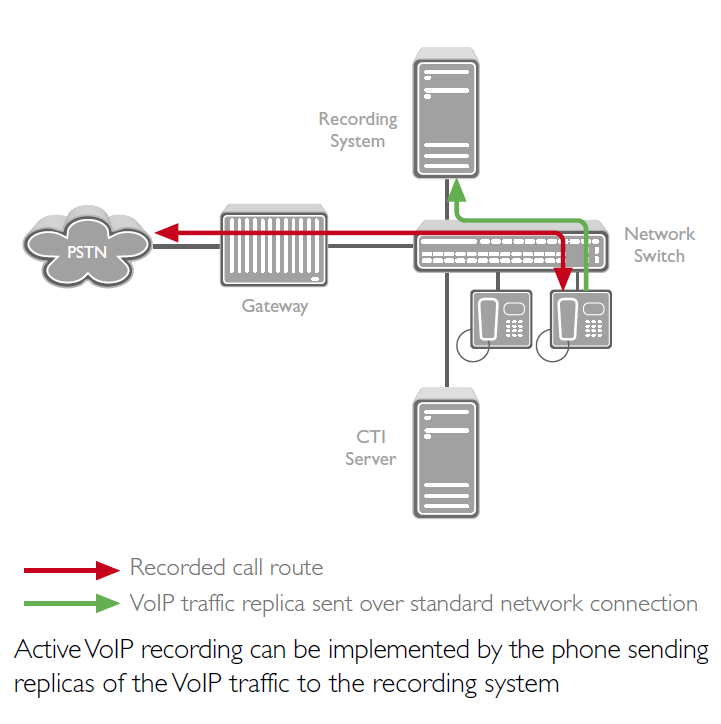
# ****INTRODUCTION****

Intelligent Recording Limited business has traditionally supplied USB based recorders that interface to propriety digital and analogue systems these being either single port or multi port devices. Theses recorders have been highly successfully, however as telephony technology changes and with several PBX manufactures starting to release IP-PBX (VoIP) systems they has been a need to venture into the world of Voice over Internet Protocol(VoIP). VoIP supports two main methods of recording – passive and active.

**Passive VoIP recording** the recorders are connected to mirroring Switched Port Analyzer (SPAN) ports of the network switches that reflect the traffic to be recorded. Passive VoIP recording is available with every telephony environment that uses standard Real Time Protocol (RTP) for the voice packets.



**Active VoIP recording** are transparent to the network as a whole. Active recording involves sending a replica of captured audio interactions from a gateway, conference bridge or IP-phone to the recording system using standard network routing.



The VoIP Recorder is expected to support both methods of VoIP recording.

# PROJECT AIMS

The aim the project is support the following VoIP protocols within the Intelligent Recording Limited BackOffice recorder:

**NEC-SIP** - KTS enhancing SIP is SIP by which NEC-Infrontia added an original definition necessary for the KTS function based on NEC enhancing SIP for which NEC provides. Because KTS-SIP uses NEC-SIP and a common parameter about the basic function, it can be compatible with NEC-SIP.

**SIP** - The Session Initiation Protocol (SIP) is an [signalling](http://en.wikipedia.org/wiki/Signalling_(telecommunications)) [protocol](http://en.wikipedia.org/wiki/Communications_protocol), widely used for controlling [multimedia](http://en.wikipedia.org/wiki/Multimedia) [communication sessions](http://en.wikipedia.org/wiki/Communication_session) such as [voice](http://en.wikipedia.org/wiki/Internet_telephony) and [video](http://en.wikipedia.org/wiki/Video) calls over [Internet Protocol](http://en.wikipedia.org/wiki/Internet_Protocol) (IP). The protocol can be used for creating, modifying and terminating two-party ([unicast](http://en.wikipedia.org/wiki/Unicast" \o "Unicast)) or multiparty ([multicast](http://en.wikipedia.org/wiki/Multicast)) [sessions](http://en.wikipedia.org/wiki/Session_(computer_science)) consisting of one or several [media](http://en.wikipedia.org/wiki/Media_(communication)) [streams](http://en.wikipedia.org/wiki/Flow_(computer_networking)).

**Skinny** -The Skinny Call Control Protocol (SCCP, or short Skinny) is a proprietary network terminal control protocol. The SCCP technology is now owned and defined by [Cisco Systems, Inc.](http://en.wikipedia.org/wiki/Cisco_Systems,_Inc.) as a messaging system between a Skinny client and the Cisco [CallManager](http://en.wikipedia.org/wiki/CallManager).

**RTP** - The Real-time Transport (RTP) Protocol provides end-to-end network transport functions suitable for applications transmitting real-time data such as audio, video or simulation data, over multicast or unicast network services.

**UNISTIM** - UNIStim (or Unified Networks IP Stimulus) is a [Telecommunications](http://en.wikipedia.org/wiki/Telecomunication) [protocol](http://en.wikipedia.org/wiki/Communications_protocol) developed by [Nortel](http://en.wikipedia.org/wiki/Nortel) (now acquired by [Avaya](http://en.wikipedia.org/wiki/Avaya)) for IP Phone (terminals and soft phones) and IP PBX communications.

With the above VoIP protocols the BackOffice recorder can support the follow IP-PBX manufactures:-

* NEC
* Cisco
* Siemens
* Nortel Networks
* Avaya

**The first stage of the project is to support NEC-SIP.**

# ****PROJECT DELIVERABLES****

**The project consists of:**

## ****Dynamic Link Library’s****

The low level interface to the telephone will be provided as a Windows Dynamic Link Library. Intelligent Recording will provide the header file. This file is guaranteed to be compatible with the latest version of Microsoft C++ Developer Studio. No guarantees can made for other systems. The Header file will be ‘MultiXtR.h’, and will contain all the function definitions. The library file will be MultiXtR.lib. This contains definitions used for importing the DLL at compile time.

The Dynamic Link Library will be ‘MultiXtR.DLL’. This will contain the code necessary for driving the appropriate VoIP protocol interface. There will be different versions of the DLL file for different VoIP protocol interfaces. The Lib and Header files for all VoIP protocol interfaces will be identical.

**Functions which affect the entire system**

* MP\_Open
* MP\_Close
* MP\_GetSwType
* MP\_GetSwVersion
* MP\_SetCallback
* MP\_GetDataFormat
* MP\_ConvertData

**Functions to determine Port numbers of connected devices**

* MP\_GetMaxNumPorts
* MP\_GetIfSerialNo

**Functions which affect a single connected port**

* MP\_GetStatus
* MP\_GetCallState
* MP\_GetIfType
* MP\_SetAudioGain
* MP\_GetAudioData
* MP\_AudioMonitor
* MP\_AutoRecord
* MP\_ControlMonitor
* MP\_GetControlDataPBX
* MP\_GetControlDataPhone
* MP\_GetCallerID

MP\_Open and MP\_Close should be called from the same thread as each other. All other functions should be thread safe and may be called from any thread at any time. Function calls should be protect internally to prevent any data corruption between threads.

**void MP\_Open( void )**

This function will attempt to open the link to the Telephone system. Since it cannot immediately determine success or failure, this function has no return value. The MP\_GetStatus function should be called after MP\_Open to determine whether required devices are correctly connected.

**void MP\_Close( void )**

This function will close the link to the Telephone system. It will release the ports and all allocated software devices, returning them to the Windows system.

**void MP\_SetCallback( (void \*MyProc)( int SerialNumber, int Event, int Data ) )**

This function sets a callback which will be used by the library to identify significant events taking place on the telephone lines. The events provided to the callback are described below. MyProc may be NULL, which disables the further sending of callbacks. Note that callbacks should come from an independent thread.

**LPCSTR MP\_GetSwType( void )**

This function will return the version type of the current software library. This is a Intelligent Recording product code, which is a string containing manufacturer, hardware interface and audio encoding type. This string consists of 5 characters e.g. AVfUS, TOfUK. Currently supported types are as follows. There may be more in the future.

|  |  |  |
| --- | --- | --- |
| Position | Chars | Meaning |
| 1st and 2nd | AV | Avaya Definity |
| 1st and 2nd | AM | Avaya Magix (Merlin Magix) |
| 1st and 2nd | NE | NEC |
| 1st and 2nd | NO | Nortel Norstar |
| 1st and 2nd | NM | Nortel Meridian |
| 1st and 2nd | PA | Panasonic DBS |
| 1st and 2nd | PK | Panasonic KXTD |
| 1st and 2nd | SI | Siemens (HiPath & HiCom) |
| 1st and 2nd | TO | Toshiba (DK + CTX) |
| 3rd | f | Always lower case f |
| 4th and 5th | US | United States |
| 4th and 5th | UK | United Kingdom |

**LPCSTR MP\_GetSwVersion( void )**

This function will return the version number of the current software library. This is an ASCII string of the format “nn.nn.nn”

**int MP\_GetDataFormat( void )**

This function will return the data format being used by the attached telephone system. Current supported values are WAVE\_FORMAT\_MULAW for American µ-law or WAVE\_FORMAT\_ALAW for European A-law.

**short MP\_ConvertData( const char Data )**

This function converts received telephone data into 16 bit data suitable for use in Windows WAV files, or by a soundcard. This function works correctly regardless of whether the data received from the telephone is in European A-law or American µ-law.

**int MP\_GetStatus( int SerialNumber )**

This function will return the current status of the requested device, and the status of the connected exchange and telephone. Possible return values from this function are as follows

| Value | Meaning |
| --- | --- |
| STATUS\_GOOD | The device, Exchange and Telephone are all functioning correctly |
| +1 to +100 | The device initialising, percentage complete |
| STATUS\_NOTPRESENT | Unable to open the link to the device |
| STATUS\_UNLICENSED | Incorrect device type. i.e. MP\_GetIfType not compatible with MP\_GetSwType |
| STATUS\_NOEXCHANGE | Device ok, Exchange not connected. |
| STATUS\_NOPHONE | Device ok, Exchange ok, Keyset not connected |
| STATUS\_UNKNOWNERROR | Device ok. Keyset OR Exchange not connected or not communicating. Unable to determine which is at fault. |
| STATUS\_NOTINITIALISED | Interface not initialised by Open, or has been Closed. |
| STATUS\_INTERNALERROR | Internal error. Some error has occurred which has stopped the device functioning. May have just been disconnected |
| STATUS\_TOOMANY | Too many devices are connected to this PC. So unable to initialise this one. |

**NOTE**: Unless MP\_GetStatus shows the device is connected and initialised, most of the following functions are to return defaults. The status can change to faulty after being connected ok.

**int MP\_GetCallState( int SerialNumber )**

This function will return the state of the keyset connected to the device. Some systems can return more information on the connected telephone than others. So you should ensure you cope with all the possible return values from this function.

| Value | Meaning |
| --- | --- |
| CALL\_IDLE | The keyset is at rest, and not in use. |
| CALL\_ACTIVE | The keyset is in use. The DLL is unable to determine the type of activity in progress |
| CALL\_INCOMING | The keyset is in use on an incoming call |
| CALL\_OUTGOING | The keyset is in use on an outgoing call |
| CALL\_INCOMING\_INTERNAL | The keyset is in use on an incoming call from an internal extension |
| CALL\_INCOMING\_EXTERNAL | The keyset is in use on an incoming call from an external line |
| CALL\_OUTGOING\_INTERNAL | The keyset is in use on an outgoing call to an internal extension |
| CALL\_OUTGOING\_EXTERNAL | The keyset is in use on an outgoing call to an external line |

**const char \* MP\_GetIfType( int SerialNumber )**

This function will return a pointer to a string describing the currently connected device type. i.e. What manufacturer or phone types it is compatible with. Return values exactly the format as MP\_GetSwType.

**void MP\_AudioMonitor( int SerialNumber, int WhatToMonitor )**

Use this function to manually start and stop monitoring of Bearer Channel (audio) data. No B Channel Data or events can be received until recording is started. Once started, audio data will come in as standard telephone data rate of 8kHz.

The WhatToMonitor parameter is used to tell the DLL which information is required.

**void MP\_AutoRecord(int SerialNumber, int WhatToMonitor )**

Auto record will automatically start recording when data is available. As soon as the telephone is in use, the system will automatically start streaming data. When the line again becomes idle the streaming of audio data will stop. The WhatToMonitor parameter is exactly as described for MP\_AudioMonitor above.

**int MP\_GetAudioData( int Port, int MaxLength, unsigned char \*pPBXData, unsigned char \*pPhoneData, unsigned char \*pMergedData )**

Use this function to recover monitored Bearer (audio) channel data. This method delivers audio data sent in either/both directions to and from the telephone and/or a merged data channel. The amount of available data for all three channels is always identical. This function returns the amount of data available (the same for each channel). This may be 0, and will be if Bearer Channel monitoring with MP\_AudioMonitor is not enabled. This function erases all returned data from the low level interface buffer.

Any or all of the three pointers may be NULL, in which case MaxLength bytes are deleted from all received channels, in order that the amount of data available from each channel remains identical.

You can check the amount of data available by calling this function with MaxLength=0. This returns the amount of data available from each separate buffer. This does not erase any data from any channel.

**void MP\_SetAudioGain( int SerialNumber, BOOL Remote, int Gain )**

This function boosts or reduces the audio signal. There are two separate gains for each port. Remote=TRUE sets the gain on the speaker i.e. the remote caller. Remote=FALSE sets the gain from the local microphone. Gain is a fixed point modifier, and will be divided by 256. A value of 256 gives no boost, i.e. exactly what is seen on the line. A value >256 will amplify the signal. A value <256 will attenuate the signal. Both values default to unmodified. i.e. 256.

**void MP\_ControlMonitor( int SerialNumber, BOOL OnOff )**

Use this function to manually start and stop monitoring of Data Channel (control) data.

**int MP\_GetControlDataPBX(int SerialNumber, int MaxLength, unsigned char \*pPBXData)**

Use this function to recover monitored D channel data sent to the telephone from the exchange. The low level interface can buffer approx 8kbytes of data. A buffer is provided with the maximum amount of data required. This function returns the amount of data available, which may be 0, and will be if you have not enabled monitoring with MP\_ControlMonitor. This method erases any returned data from the low level interface buffer.

If pPBXData is NULL, calling this function will erase MaxLength bytes of data without returning them.

You can check the amount of data available by calling this function with MaxLength=0. This does not erase any data from the buffer.

**int MP\_GetControlDataPhone(int SerialNumber, int MaxLength, unsigned char \*pPhoneData)**

Use this function to recover monitored D channel data sent from the telephone to the exchange. All other details are exactly as for GetControlDataPBX above.

**LPCSTR MP\_GetCallerID( int SerialNumber )**

This function returns the caller ID, of the current active telephone call. This may be an empty string if no call is currently in progress. Note that caller id changes do not necessarily occur at the beginning and end of calls. They may be delayed, or on some systems may even precede active call data.

Callbacks

**void MyProc(int SerialNumber, int Event, int Data )**

This is the method which is provided to allow the low level interface to tell about received data events. Providing a pointer to the function to MP\_SetCallback is set, it will be called whenever any significant events occur.

Note that the Callback function will be called from an independent child thread, not from the Application’s

All the functions provided by this library should be thread safe (except MP\_Start and MP\_Stop) and may be called from the callback’s thread, or from the main Windows thread (or indeed from any other thread). So it is possible to recover data, manipulate and even record it from the callback, if desired.

The constants for the Event parameter are defined in MultiXtR.h. Possible values and their meanings are discussed in the following sections.

**MP\_STATUS**

The callback function will be called with this event whenever a interface device changes status. The Data parameter is the new status, and the possible values and meanings of these values are described in MP\_GetStatus.

**MP\_BCHANNELDATA**

The Callback function will be called with this event when B channel data recording is enabled and data is available from this device. The Data parameter gives the number of available bytes.

**MP\_DCHANNELDATAPBX**

The Callback function will be called with this event when a D channel data packet is sent from the exchange to the telephone. The Data parameter will give the number of bytes available.

**MP\_DCHANNELDATAPHONE**

Same as above, except it reports data from the phone to the exchange.

**MP\_CALLSTATE**

This event is sent whenever the call state changes on an interface. Note that you may receive multiple call state messages for a single call, as the interface determines what the user is doing with his telephone. Possible Data values are any of the return values from MP\_GetCallState.

**MP\_CALLERIDUPDATE**

This event is sent whenever the caller id string returned by MP\_GetCallerID changes. Note that caller id changes do not necessarily occur at the beginning and end of calls. They may be delayed, or on some systems may even precede active call data.

**int MP\_GetMaxNumPorts ( void )**

This function will return the maximum number of connected devices connected to the PC. Not necessarily the number which currently are connected.

**int MP\_GetIfSerialNo( int Virtual )**

This function will return the serial number of the connected virtual device. If no device is present in this slot it returns 0. Even if the device has a serial number, it might still be faulty. You will need to check the status to ensure that the device is working.

Please note that this information will NOT be available at program start up, due to the time delay necessary to initialise the devices.

## BackOffice Recorder

Following changes will have to be added to the BackOffice recorder application

### Licensing

Each recordable VoIP extensions needs to be licensed by protocol and channel by software protected USB Dongle.

### Port Configuration

Each port will need to be enabled or disabled for recording and a new section is to be added to incorporate any VoIP channel configuration.

### Phone registering

IP Phones must be "registered" with an IP-PBX before they can place and receive calls. "Registered" means that the phone can communicate with the IP-PBX over the Local Area Network (LAN) or over the Internet (if the phone is remote).

The IP Phones must be “registered” with the BackOffice recorder before they can be recorded. The process of registering phones with the BackOffice recorder will have to be an automatic process either by picking each phone up (Off-Hook) or via the BackOffice recorder via single button action.

### Playback

Each of the playback application should not be affected as the audio conversion and format will be performed in the VoIP DLL’s or the BackOffice recorder

# GENERAL ISSUES WITH VOIP

VoIP may be the best technology in place of PSTN with its amazing features and cheap phone offers worldwide, yet there are always two sides of a story. VoIP users maybe on the rise, yet there is no denying that there are some general problems being based by ‘internet phone’ or VoIP phone users.

In some cases, VoIP phone users are facing problems in Voice quality, Bandwidth dependency, Power dependency, Emergency calls and in terms of security. Normally, Quality of Service (QoS) in VoIP phones is equivalent and sometimes better than the level of ‘voice quality’ being offered by the service to make phone calls, without delays, interruptions, sounds, noise or even echo. Voice quality most of the times depends on your broadband connection, your requisite hardware, your VoIP phone service provider, and lastly, your call destination.

The [internet phone](http://www.axvoice.com/) works best with a broadband connection. If the internet connection is sluggish, that means the VoIP phone will slow down alongside. It’s simple: No internet means, no VoIP phone. In case of poor broadband quality, using VoIP service can get pretty annoying, ultimately discouraging, instead. The moral is that a good broadband service must be selected.

The VoIP service needs a modem, router, ATA or other VoIP hardware whatever it is, to be connected to the electric power supply to make it function. This means, the VoIP service works on electricity. In case of power outage, the VoIP phone is of no use.

The biggest issue in an Internet Phone service is of security, which is the same for other Internet technologies. However, here the difference is that large companies are more exposed to viruses and hackers as compared to the small companies. Identity and service theft, viruses and malware, denial of service, spamming, call-tampering issues and phishing attacks are among the many problems.