# HIGH LEVEL VoIP API

# FOR UNICON LANGUAGE

BY

ZIAD AL-SHARIF

A thesis submitted to the Graduate School

in partial fulfillment of the requirements

for the degree

Master of Science

Major Subject: Computer Science

New Mexico State University

Las Cruces New Mexico

May 2005

"High Level VoIP API for Unicon," a thesis prepared by Ziad Al-Sharif in partial fulfillment of the requirements for the degree, Master of Science, has been approved and accepted by the following:

---

Linda Lacey
Dean of the Graduate School

---

Clinton Jeffery
Chair of the Examining Committee

---

Date

Committee in charge:

Dr. Clinton Jeffery, Chair

Dr. Roger T. Hartley

Dr.

## Introduction

We are building a Collaborative Virtual Environment (CVE) for the computer science department at New Mexico State University [5]. It is a collaborative virtual environment project that integrates a 3D view of a virtual computer science department with 2D collaborative tools for working on various software engineering and development tasks. This project is funded by the NSF. It supports the idea of distance learning and distance education and mainly aims to improve the communication performance inside the department as well as in between the department and the CS community around the New Mexico State.

The audio part of this project requires C or C++ open source cross platform libraries that can handle the audio functions, especially the VoIP. So, we started by describing our experience with libraries such as Port Audio [4], OpenAL [3], and Libsndfile [1], which are written in C and can digitize, write, and read sounds. However, the most interesting work that we found is in Voice over IP (VoIP) libraries. JVOIPLIB stands for "Jori's Voice over IP library". It is Jori Liesenborg's library. JVOIPLIB is an object oriented VoIP library written in C++ and works in both Microsoft Windows and UNIX/Linux systems [2]. The current stable version is jvoiplib-1.3.0. Jvoiplib is some what stable but still under going refinement; the update of 3 September 2004 has major improvements especially in the RTP components.

This CVE will be built and programmed using Unicon [11]. Unicon is a high level goal directed language now being extended to directly support CVEs. We are focusing in adding audio facilities to the Unicon VM; especially VoIP functions on top of the C++ jvoiplib. Since the Unicon VM is written in C, a mechanism for calling a C++ library from C code is needed. So we will discuss how to mix C and C++ and then go forward with the details of the data structures and the API functions that are needed to make the VoIP possible for the Unicon programmers. The Unicon language needs those data structures and API functions in order to establish and maintain VoIP sessions. Those API functions will be added into the Unicon VM through its RTL language.

## 2 Sound Libraries

Those who are looking for sound libraries they may look for a variety of sound and Music files formats which may break down into some categories like

- Purely digital audio file formats such as WAV, AIFF, AU, SND, and CD audio
- Compressed audio formats such as MP3
- Tracker formats such as MOD, IT, and XM

- MIDI files in MID and other formats
- Streaming audio (RealPlayer, Shockwave/Flash, QuickTime, or MP3)

and those who are looking for the efficiency in the space, they may prefer an MP3 or any other compressed audio file format over the gigantic WAV files [8].

However, we are different; we are looking for a useful audio library that can be used in Unicon, This Section examines C /C++ open source cross platform libraries that can digitize, write, and read sounds. So, we have looked in three audio libraries, Libsndfile, OpenAL, and Port Audio. These libraries have a lot in common; they focus on the main sound functions like digitizing and playing back. The main difference between them is in their portability such as the platforms that they can work on, and the sound format that they can read, and write; since some of them cover sound formats more than the others and maybe different from each other. The libraries are evaluated in order to identify the easiest, the simplest, and the most comprehensive one in order to be used in Unicon.

### 2.1 Libsndfile

Libsndfile is an open source C audio library. It is been released under the GNU Lesser General Public License (LGPL). Libsndfile is mainly written to be used and compiled in Linux systems but it can be compiled on any UNIX like system including Macintosh as well as in MS Windows using the Microsoft compiler (MSVC). It is easy to compile in each platform from the clear instructions that come with the library. Libsndfile main goal is to read and write sound files containing sampled sound; which are called formatted sound files. The strong point in this library is the different kinds of sound formats that it can read and write. It can currently read and write 8, 16, 24 and 32-bit PCM files as well as 32 and 64-bit floating point WAV files and a number of compressed formats [1]. However, Libsndfile does not read, write, or play or have any support for MP3 files, since MP3 files are not formatted audio files; MP3 files are a compressed format for audio files that was originally in a common format such as WAV or AIFF. Actually, you do not record an MP3 directly, but you can convert your existing formatted audio files into MP3 using an MP3 encoder [8]. Libsndfile is also designed to be compiled and to run correctly on little-endian processor systems, such as Intel and DEC/Compaq Alpha, as well as big-endian processor systems such as Motorola 68k, Power PC, MIPS and Sparc.

Libsndfile library is able to convert the sound data from one format into another. For example, in a sound playback program, the library caller typically wants the sound data in 16 bit short integers to dump into a sound card even though the data in the file may be 32 bit floating point numbers such as the Microsoft's WAVE_FORMAT_IEEE_FLOAT format. Another example would be someone

doing speech recognition research who has recorded some speech as a 16 bit WAV file but wants to process it as double precision floating point numbers.

Libsndfile is supported and updated; the last stable version is libsndfile-1.0.11. You can find the library and much other important information in its website [1].

| ***Item*** | ***Size*** |
|---|---|
| Libsndfile-1.0.11.tar.gz | 797.0 KB |
| Unzipped folder "libsndfile-1.0.11" | 4.1 MB |
| After running the configure script the folder size is | 4.6 MB |
| Libsndfile.a | 2.07 MB |
| Libsndfile.so.1.0.11 | 1.09 MB |

Table1: Libsndfile

Libsndfile library can provide Unicon with three functions

- Playing formatted sound files.
- Converting sound files from one format into another format.
- Giving a summary of a given sound file formats.

**2.2 OpenAL**

The Open Audio Library (OpenAL) is a software interface to audio hardware. It is an open source C audio library. OpenAL designed to provide a cross-platform open source solution for programming 2D and 3D audio. It is been released under the GNU Lesser General Public License (LGPL). It supports Microsoft Windows, Mac OS, Solaris, and in Linux it supports both Open Sound System (OSS) and Advanced Linux Sound Architecture (ALSA)[3].

OpenAL is concerned only with rendering audio into an output buffer. Many of the design considerations for OpenAL have been driven from the similar considerations for the visual effects in OpenGL, which make its API functions very similar to OpenGL. This makes programmers who are familiar with OpenGL or using OpenGL graphics can more easily use OpenAL in their games and other OpenGL graphics-intensive applications [6], and its complement to OpenGL makes it appropriate for real-time rendering in gaming applications such as the sound effects that associate with OpenGL graphics.[6]

At this time the OpenAL API and library focus only on PCM audio, which makes no support for MIDI (Musical Instrument Digital Interface) and other components usually associated with audio hardware. Programmers must relay on

other mechanisms to obtain audio (e.g. voice) input or generate music. However, it is possible that some future revisions will address CD audio and hardware MIDI synthesis. [6]

OpenAL is supported, documented, and there are many projects using it. It is larger than Port Audio or libsndfile. You can find it and much other vital information in its website [3].

| Item | Size |
|---|---|
| libopenal.a | 1.5 MB |
| libopenal.so.0.0.7 | 870.9 KB |
| openal/linux/src "subfolder" | 2.6 MB |
| openal/win "subfolder" | 2.6 MB |

Table2: OpenAL

### 2.3 Port Audio

It is a free C audio library which is simple, portable, and open source. Using Port Audio is very easy and with small effort lets you write a simple audio program in C that can be compiled and run in different platforms like Windows, Mac, and UNIX. Their project for Linux Open Sound System (OSS) has been released, whereas they already started their project for Advanced Linux Sound Architecture (ALSA), which is going to be released in the next version V19. PortAudio has been designed to promote the exchange of audio synthesis software in between developers on different platforms. It is been recently selected as the audio component of a larger PortMusic project that includes MIDI and sound file support[4].

PortAudio now has no support for reading or writing formatted audio files and they recommend using libsndfile for formatted audio files I/O and conversions between different audio formats.

The most interesting thing in PortAudio is its easiness since it provides very simple API functions for audio tasks like recording and playing back. It is well documented, and lets you understand the low level details for every simple function. It is very useful for building any simple audio program from scratch using a small number of C lines comparing to other libraries like OpenAL or even Libsndfile. Port Audio V19 is under development. The last stable version is V18.1; you can find its status and much more information in its website [4].

| *Item* | *Size* |
|---|---|
| portaudio_v18_1.zip | 537.2 KB |
| portaudio_v18_1 "folder" | 1.6 MB |
| libportaudio.a | 105.9 KB |
| libportaudio.so.0.0.18 | 80.9 KB |

Table 3 : PortAudio

## 3 VoIP Libraries

Digitizing, saving, and playing voice back, in the same machine or in between machines, is very interesting. However, conducting a real-time conversation over an IP network, called Voice over IP (VoIP), is more interesting. We considered JVOIPLIB because it works across the network and it focuses on VoIP. It consists of three sub-libraries JVOIPLIB, JThread, and JRTPLIB, each one is necessary for a specific mission.

### 3.1 JThread

JThread is not a Java Thread. It is the Jori Liesenborg's Thread library which is an object oriented library written in C++ that represent a thread and a mutex. JThread makes the use of threads easy on different platforms like Window and UNIX/Linux. On a UNIX-like platform, the pthread library is the underlying thread mechanism. However, On Microsoft Windows, the native Win32 threads are used. Actually, its classes source code are simple wrappers around existing thread implementations. By using these wrapper classes you can easily create applications which use threads, without having to worry about which platform the program will be running on. JThread is a stand alone library that has been used by JRTPLIB and JVOIPLIB and can be used with any other application. The last stable version is jthread-1.1.1.[2]

| *Item* | *Size* |
|---|---|
| jthread-1.1.0.tar.gz | 35.7 KB |
| jthread-1.1.0 "folder" | 190.9 KB |
| jthread.lib | 34 KB |
| libjthread.a | 6.1 KB |
| libjthread.so.1.1.1 | 7.4 KB |

Table 4: JThread 1.1.1

### 3.2 JRTPLIB

JRTPLIB is an object oriented library written in C++, it uses the previous JThread library. It offers support for a Real-time Transport Protocol (RTP). The current stable version is 3.1.0, which is based on RFC 3550, whereas the previous 2.x versions ware based on RFC 1889 which is now obsolete. JRTPLIB makes it easy for different underlying protocols to be supported. Currently, JRTPLIB support both of UDP over IPv4 and UDP over IPv6. RTP makes it easy to send and receive real time packets over the IP network when the RTCP (RTP Control Protocol) functions are handled internally. JRTPLIB library makes it possible for the user to send and receive data using RTP, without worrying about SSRC collisions, scheduling, and transmitting RTCP data, etc. The user only needs to provide the library with the data to be sent, and the library will give the user access to incoming RTP and RTCP data. [2]

JRTPLIB is an essential part in JVOIPLIB. However, it can work with any other application. It is still under refinement by Jori and others who are interested. The last stable version is jrtplib-3.1.0 which has been released in October 2004.

| ***Item*** | ***Size*** |
|---|---|
| jrtplib-3.0.1.tar.gz | 394.4 KB |
| jrtplib-3.0.1 "folder" | 1.9 MB |
| jrtplib-3.0.1.lib | 416 KB |
| libjrtp.a | 254.9 KB |
| libjrtp.so-3.1.0 | 192.5 KB |

Table 5: JRTPLIB

### 3.3 JVOIPLIB

JVOIPLIB stands for Jori's VoIP Library. It is an object oriented library written in C++, it uses the previous two libraries, JRTPLIB and JThread, and cannot work without them. It works in both MS Windows and UNIX/Linux. The library intended to make the creation and managing of the VoIP session very easy by providing a simple interface such as create a session, destroy a session and set session attributes; all this can be done using simple C++ API methods. Since it is a VoIP library, it cares about the size of the transmitted sound, So, It provides many compression techniques such as Raw PCM, Mu-law, DPCM, GSM, and LPC; some of these compression methods require a special range of sampling rates. The main job of this library is digitizing the input sound according to some attributes like sampling-rate, Sampling-encoding, and sampling-interval, puts them in packets, compresses these packets using one of the compression techniques , and then deliver them to the RTP protocol which will send them to the destinations. At the destination side, it

combines the received packets, decompresses them, and plays them back. The RTP puts the received packets in the correct order as they have been sent, which helps the play back to be fairly good.

The library features are

- Easy VoIP session creation and destruction.
- Highly configurable sessions: sampling rate, sample interval, compression type, etc. These features can also be changed during a session.
- Openness and extensibility.
- Supports for 3D effects.
- Soundcard input and output
- Different compression techniques such as DPCM compression, Mu-law encoding, GSM (06.10) 13 kbps compression, LPC 5.4 kbps compression.
- Using an RTP (Real-time Transport Protocol) for transmitting the sound data.
- It has its own elementary voice mixer

The last stable version is jvoiplib-1.3.0.

| ***Item*** | ***Size*** |
|---|---|
| jvoiplib-1.3.0.tar.gz | 468.2 KB |
| jvoiplib-1.3.0 "folder" | 2.08 MB |
| jvoiplib.lib | 2.82 MB |
| libjvoip.a | 6.4 MB |
| libjvoip.so.1.3.0 | 3.5 MB |

Table 6: JVOIPLIB

## 4 Mixing C and C++

C++ language supports techniques for mixing code that can be compiled using compatible C and C++ compilers with the considerations for different degrees of success during porting such kind of mixed code according to the platforms and the compilers themselves. The first requirement for such a mixing is that both of the C compiler and the C++ compiler must be compatible; which means that they must have the same way of defining the basic types such as int, float, char or even pointers, etc. The second requirement is that the runtime libraries and the header files for the C compiler must be compatible with the C++ compiler. If the C++ compiler provides its own versions of the C headers, the versions of those headers must be compatible with the C compiler [12].

However, if your program is primarily a C code but makes use of C++ libraries, you need to link the C++ runtime support libraries provided with the C++ compiler into your program. The common dynamic C++ runtime libraries such as libstdc++.so.x.x or the static library libstdc++.a is linked automatically if you are compiling using g++. However, if you are compiling using gcc, it will not link automatically; which means in this case you need to link them explicitly. The libraries that you may need to link are vary and very dependent on the options that are used during the compilation of the C++ code and also dependent on the features that you are using in the C++ program itself. [12]

### 4.1 Accessing C from a C++ code

The C programs in general do not require that much modification in order to compile as C++ programs. In fact, both of C and C++ are link compatible languages; which means that it is not required to modify a compiled C code (object code) in order to link it into a C++ code and vice versa [15]. This can be done only if the C runtime libraries used by the C compiler are compatible with the C++ compiler. Also, C++ includes the standard C runtime library as a subset, with a few differences. If the C++ compiler provides its own versions of the C headers, the versions of those headers used by the C compiler must be compatible with those in the C++ compiler [12].

### 4.2 Accessing C++ functions from a C code

It is possible to have a C++ function with all its C++ features and properties and you like to use it within a C code. In this case the C++ function must be declared with a C linkage; C linkage means adding {extern “C”} at the beginning of the function declaration such as

```
extern "C" int print_out(int i, double d)
{
        std::cout<< “i=    ”<<  i  << “and d= ”<< d;
}
```

This C linkage makes this function callable from any other function compiled by the C compiler, even though that it has C++ features inside. This call can be done only if the function's parameters and the function return type are compatible with the C language. For example, if the function declaration has a reference to an IOstream class as a parameter; in this case there is no portable way to explain the parameter type into the C compiler, since the C language does not know any thing about the C++ features such as templates and classes [12].

### 4.3 Accessing C++ Classes from a C code

It is possible to have a C++ class that you need to access from a C code. However, you can not declare a class inside your C code. But C and C++ are compatible in there pointer types; which make the best way to make it possible is to play around using pointers to classes. This way could be very similar to the way that dealing with FILE in C standard I/O. Using the C linkage for C++ functions that can access any C++ property including classes, we can call these functions from any C code.[12][14]

Example

Let us say that we have the following class

```
class Bar{
        public:
                int foo(int);
                // ...
        private:
                int i, j;
};
```

In order to access this class from a C code we need the wrapper function with the C linkage, this function could look like

```
extern "C" int foo_bar(Bar * m, int i) {
        return m->foo(i);
}
```

## 5 API Functions Structure in the C Level

The Unicon Virtual machine is written in C whereas the JVOIPLIB is a C++ Library supporting the VoIP. Using a library written in C++ and compiled using g++ or even MSVC accessible from a C code is very impotent in order to augment the C code in the Unicon VM with VoIP facilities from jvoiplib. Instead of converting the C++ library from C++ into C, which is time consuming and maybe very difficult, it is possible to take advantage of Accessing a C++ code from C code using the C linkage obtained by extern “C”. However, in order to make this possible, First it is important to determine the exact C++ functions and data structures needed to be exposed and to be accessed by the C program that makes the C++ library usable. Second, it is important to write C++ API functions; those API functions are wrapper functions for the C++ functions and structures which are needed to be exposed to the C code in order to make the C++ library usable in a C code. Finally, it is possible to build an

interface library from those API functions on top of the original C++ library. Since the C and C++ are link compatible, we can link this interface library and the original C++ library into the C code in the Unicon VM (iconx) after adding the necessary built-in functions using the Unicon RTL language.

This playing around with the C linkage and the wrapper API functions allow us to use a C++ library inside a C program. And that is what we actually did in order to make the C++ in jvoiplib usable inside the Unicon VM.

### 5.1 The Main jvoiplib Classes (Data Structure)

According to the last update, jvoiplib 1.3.0 has four main classes affecting the VoIP session. Those classes need to be accessed directly in order to make a VoIP session possible.

- JVOIPSession: it encapsulates the essential data and methods which are responsible to create and destroy a VoIP session.
- JVOIPSessionParams: it encapsulates the essential session parameters data and methods needed to setup the VoIP session parameters
- JVOIPRTPTransmissionParams: it encapsulates the essential Real Time Protocol (RTP) parameters settings.
- JVOIPSoundcardParams: it has some of the Soundcard input and output setting parameters that are useful only on Linux OS.

### 5.2 The Data structure needed inside the Unicon VM

According to the previous jvoiplib components, and in order to make the VoIP session possible to be created and maintained in Unicon, the following structure has been defined for being in the Unicon VM

```
#ifndef WIN32
/* for linux OS */
struct VOIPSession{
        JVOIPSession                Session;
        JVOIPSessionParams          sessparams;
        JVOIPRTPTransmissionParams  rtpparams;
        JVOIPSoundcardParams        sndinparams;
        JVOIPSoundcardParams        sndoutparams;
        int                         MaxList;
        int                         InList;
        char                        **ListenerList;
};
```

```
#else
/* for MS Windows OS */
struct VOIPSession{
        JVOIPSession                    Session;
        JVOIPSessionParams              sessparams;
        JVOIPRTPTransmissionParams      rtpparams;
        int                             MaxList;
        int                             InList;
        char                            **ListenerList;
};
#endif

typedef struct VOIPSession    VSESSION;
typedef struct VOIPSession*  PVSESSION;
```

- **VSESSION:** This struct represents the main components required for establishing and managing the VoIP Session in the Unicon VM.
- **PVSESSION**: This data type is a pointer to a struct of type VSESSION. In Unicon, the programmer needs to deal with variables of this type in order to make a VoIP session possible.
- **JVOIPSession** structure has the main exposed components from jvoiplib and some additional fields needed by the Unicon VM; MaxList, InList, and ListenerList. Those fields are added in order to maintain a list of the listener's host-names and their related ports. Whenever there is a new listener will be added to the list and whenever it is been dropped out will be deleted from the listener's list.

### 5.3 The C++ API Function Prototypes

The API functions are C++ wrapper functions with the C linkage working as an interface between C and jvoiplib C++ code. They are focusing on creating, unicasting, multicasting, stopping unicasting, stopping multicasting, setting up, and destroying a VoIP session. In order to give the Unicon programmers the ability to create, maintain, and destroy a VoIP session, the Unicon VM must be extended with the following C API functions, which are designed to make it easy to deal with VoIP sessions. The function prototypes are defined as follows.

**extern "C" PVSESSION CreateVoiceSession(char Port[5], char Destinations[]);**

This function is responsible for allocating and creating an instance object of the VSESSION struct in the heap, which contains the main session components. It takes the base port that the VoIP session should be established on as a parameter and returns a pointer to a VSESSION object. The returned pointer is the main handler for

the created session. This function is also responsible for initializing the main session components with the default values. Then it tries to establish a session on the specified port. It returns a not NULL value if the components and the session are established successfully. Otherwise, it will return a NULL value. The default session components are sessparams, rtpparams, those are in general and there are two more components required in Linux OS; those are sndinparams, and sndoutparams.

sessparams object has the following variables; some of them affect the quality of service. Those values will be set directly using its constructor and can be changed later during the session. Changing any of these values before or during the session will affect the Quality of the Service (QoS). The default values that initialized with the session constructor are working well so far, and we disabled the other values as an optimization in the resulting size. So we can stick with them most of the time. The QoS depends also on the bandwidth and the traffic in the network. The following table shows those session parameters and their default values the possible values that the session can accept.

| Variables | Default Value | Possible Values |
|---|---|---|
| inputsamprate | 8000 | {4000, 8000, 11025, 22050, 44100} |
| outputsamprate | 8000 | {4000, 8000, 11025, 22050, 44100} |
| sampinterval | 20 | 20 |
| inputtype | SoundcardInput | {NoInput, UserDefinedInput, SoundcardInput } |
| outputtype | SoundcardOutput | { NoOutput, UserDefinedOutput , SoundcardOutput } |
| loctype | NoLocalisation | { NoLocalisation, UserDefinedLocalisation, SimpleLocalisation, HRTFLocalisation} |
| comptype | NoCompression | {NoCompression, UserDefinedCompression, ULawEncoding, SilenceSuppression, DPCM,GSM, LPC } |
| mixertype | NormalMixer | { UserDefinedMixer, NormalMixer } |
| transtype | RTP | { UserDefinedTransmission, RTP } |
| receivetype | AcceptAll | { AcceptAll, AcceptSome, IgnoreSome } |
| inputsampenc | EightBit | { EightBit, SixteenBit } |
| outputsampenc | EightBit | { EightBit, SixteenBit } |

Table 7: JVOIPSessionParams Variables

**extern "C" int Unicast(PVSESSION VSession, char Destination[]);**

This function can be used to unicast to any destination which already has a created voice session. The destination can be recognized by its IP address or its host-

name and the base port that should be passed to the function as a string argument separated by a colon. Before using this function, there must be an allocated voice session components and they must be initialized successfully in both the sender and the destination. This function returns 1 if the unicasting has been done successfully, otherwise it returns a 0. Success does not mean that the destination is receiving what the sender is sending, it just means that the sender can send but the destination can benefit, from the received data, only if he already has an active session. Returning 0 means it can not add the destination to the session.

**extern "C" int Multicast(PVSESSION VSession, char Destinations[]);**

While the unicast function is used to cast to a single destination, this function is used to cast into many destinations at once. It takes a punch of destinations separated by a comma.

**extern "C" int StopUnicasting(PVSESSION VSession, char Destination[]);**

This function is used to stop the casting to a specific destination. It takes a pointer to a struct of type VSESSION, an IP address or host-name and its base port as a string. It will return 1 if the destination has been deleted successfully. Otherwise, it will return a 0.

**extern "C" int StopMulticasting(PVSESSION VSession, char Destination[]);**

This function is used to stop the casting to a punch of destinations at once. It takes a pointer to a struct of type VSESSION, and a punch of IP addresses or host-names each one attached to its base port with a colon, the IP's or the host-names are separated by comas in the string parameter. It returns 1 if the destinations has been deleted successfully. Otherwise, it will return a 0.

**extern "C" void CloseVoiceSession(PVSESSION VSession);**

This function is responsible for closing the voice session and freeing the allocated memory for the VSESSION struct back to the heap. It takes the session handler, which is of type PVSESSION, as an argument and returns nothing.

**extern "C" void SetVoiceAttrib(PVSESSION VSession, char StrAttrib[]);**

This function is responsible for adding new destinations into the session, deleting existing destinations and making query about those who are on the VoIP session. It takes the VoIP session handler as the first parameter and the attribute

assigned to some destination's host-names or IP addresses with their port number separated by colons. The attributes are

"cast+=Hostname:BasePort, ...."
This attribute is used to add new destinations into the VoIP session.
"cast-= Hostname:BasePort, ...."
This attribute is used to drop out some existing destinations from the VoIP session.
"cast"
This one is to ask about the people who are in the VoIP session.

This function will analyze the attribute and according to it will decide which function to call such as Unicast, Multicast, Stopunicasting, ... etc.

### 5.4 Using the API VoIP functions in the C Level

The goal of this section is to explain how Those VoIP API functions can be used in order to establish and maintain a voice session. So, first and before anything else, the programmer needs to create and allocate the essential voice components needed for a session. This can be done using the C function.

```
PVSESSION MySession;
MySession = CreateVoiceSession("5000",NULL);
```

Actually, MySession variable is of type PVSESSION, which is a pointer variable to a VSESSION struct. This function allocates the necessary space for the voice components and creates the real voice session object at the base port 5000. the second parameter is NULL, which means that there is no destinations we need to cast to them at this time. This function returns a non NULL value when it is successfully create and initialize the session components; otherwise it returns a NULL value. This function will initialize the session components with the predefined default settings.

At this point, the session is ready for unicasting or multicasting using one of the specified functions. In order to unicast to a specific destination and make a peer-to-peer voice connection, the Unicast function formed to do that.

```
int Status  = Unicast(MySession, "128.123.64.48:5000");
```

The same happen with multicasting, except that it takes a list of IP addresses instead of only one destination IP address and a list of base hosts associated with the IP's. After unicasting or multicasting, it is possible to stop casting to a specific destination using the following function

**Status = StopUnicasting(MySession, "128.123.64.48:5000" );**

It is very important to close the voice session explicitly before the end of the program by calling the function

**CloseVoiceSession(MySession);**

## 6 Compiling Jori's Libraries

Those libraries are easy to compile and to use in both Linux and MS Windows. However, we have an experience with them which may be useful in one way or another.

### 6.1 In Linux

This library is working in Windows and Linux. In Linux it uses the Open Sound System (OSS). Exactly, it opens the audio device called "/dev/dps". According to the OSS specifications, this device can not be opened twice, so, before using this library, it must be sure that is no any other application already using this device, otherwise, the library can not be used and it will give an error says "Can not open sound device". In Redhat Linux 9, where KDE is version 3.1, KDE itself is using this device. So if the "aRTs" in KDE is starting, the library can not be used and it will give a message "Could not open sound device". And in order to fix this and make the library working, the "aRTs" must be disabled. Or you need to switch to GNOME which does not use this device. In linux system that has ALSA on it, ALSA provides an OSS simulation, which allows the library to work and makes no problem with other programs that are using the sound device.

The VoIP library is written in C++ and it needs the g++ compiler not gcc. In the other hand, the Unicon VM is depending hardly on C and the gcc compiler. It is very essential to find a way to use this C++ library inside a set of C functions that can be compiled using gcc not g++. This set of C function will work as an API to the C++ library. Those API functions will be the main functions in the Unicon VM. In order to make this happen, during the compilation of those API function, they must be linked to the VoIP libraries: libjvoip.a, libjthread.a, and libjrtp.a beside the Linux thread library "-lpthread". However, this will not be enough since the VoIP libraries are written in C++, then they will need to be linked dynamically or statically into the C++ standard library "libstdc++.so.x" or "libstdc++.a" respectively.

### 6.2 In Windows

The JRTPLIB library is using sockets. In order to make the API functions, which work with no problem in Linux, also work in Windows, two functions must be added to the API functions, one to start up the socket before using it and the other is to close it before ending it. However, those functions will affect the other sockets used by the program other than the voice ones; if the program is already using windows sockets and the CleanupWinSocket() is called, it will close all of the sockets. So, we need to be careful about this. The two functions are as follows

```
//-------------------------------------------------------
// This is just for starting up windows sockets
        extern "C" int StartupWinSocket(void)
        {       int t;
                WSADATA wsa_data;
                t=WSAStartup(MAKEWORD(1,1),&wsa_data);
                if(t!=0){
                        printf("\nWSAStartup Fails\n");
                        return 0;
                }
        return 1;
        }
//-------------------------------------------------------
// This is just to clean up the windows sockets
        extern "C" int CleanupWinSocket(void)
        {
                int t;
                t=WSACleanup();
                if(t==SOCKET_ERROR){
                        printf("\n Can not sign off from the Windows Socket API \n");
                        return 0;
                }
                return 1;
        }
//-------------------------------------------------------
```

We are using the Windows XP Home Edition, there is a work already done by Jori for building the libraries. He already compiled the VoIP library using the Microsoft Visual Studio 6.0 and built the three libraries; jvoiplib.lib 1.3.0, jthreadlib.lib 1.1.1, and jrtplib.lib 3.1.0. In Windows, using MSVC, it is necessary to have the Microsoft Visual C version 6 updated with the Service Pack 6 in order to compile and build the new update of those three libraries. With out this service pack you will have a compilation error. However, this service pack was not important in the previous 2.x version of the library. Also in windows you need to link into two more additional windows libraries in order to make the VoIP working, those libraries

are ws2_32.lib and winmm.lib, those two libraries used in windows to accomplish the VoIP mission.

## 7 Adding new Built-in Functions into Unicon

From the sense that there is no programming language is totally complete; sometimes there is a need to do things in a language that are not in the language itself or maybe the language itself does not support them. However, those things can be done in another language. This generates two ideas; is this language supporting using programs written in other languages, or is this language has the flexibility for upgrading and improving. Unicon supports the two ideas. In Unicon you can call a C function from a Unicon program. Unicon also has the flexibility to upgrade its VM with new functionalities through its supported RTL language. This may raise a question about the difference between those two ideas which is not that much in the functionality since the two ideas are supporting using things has been done in other languages, but may be the main difference is in the usability. Calling functions written in C from a Unicon program is not that easy for the beginner programmer as using a built-in function supported by the Unicon VM.

### 7.1 Unicon RTL

RTL stands for Run-Time Implementation Language for Icon; it is a language designed to aid and supports the run-time system for the Icon language, the ancestor of Unicon. Originally, the design of RTL was motivated by the needs of the optimizing compiler for Icon, Iconc. This made the RTL supporting a kind of database of information about the Icon operations that is used by the optimizing compiler; since Iconc needs information about run-time routines in order to generate an efficient intermediate C code for Icon programs. However, the RTL is used to define operators, built-in functions, and keywords for the icon language. Actually, RTL consists of two sub-languages, first an interface and a type specification language, and second, a slightly extended version of C. The RTL interface defines what an operation will look like in the Icon language, specifies the type checking and conversion needed for arguments to the operation, and presents the over all structure of the operations implementation. The extended C language includes operations for manipulating, constructing, and returning Icon values. It is impeded within certain constructs of the interface language and provides the low-level of the implementation. [18]

### 7.2 How to add new built-in functions into Unicon VM

Adding new VoIP functions into the Unicon VM needs a programming using the RTL language, and an efficient C API functions and structures on top of the VoIP

libraries. The RTL programmer will determine the format of the built-in functions such as their names, parameters and return types. In order to make those new functions consistent with the Unicon high level and goal directed specifications, an optimization that hides all the low level details in the language itself is needed, which makes those functions very abstract and very easy to use by the Union programmer.

We had to extend and overload two Unicon functions that already in Unicon, open and close, to work with VoIP. So, now open with the right arguments will open a VoIP session and close will check if the passed parameter is a VoIP session to wrap that session up and clean every thing. This way of extension function and overload them with new jobs instead of adding new dedicated functions makes the language easer to work with; since there are less functions to remember.

However, we had to add two more new functions that are dedicated for the VoIP and with ability to be extended in the future for new additions in the field of audio. Those functions are VAttrib and VWho; VAttrib is setting up the VoIP attributes such as adding new host-names into the session, dropping some host-names out of the session, and querying about those who are in the session. The other function is VWho, it creates a list of the host-names and their base ports for those who are listening to the VoIP session.

### 7.3 The Increase in the VM Size

However, there is always a tradeoff between the added functions into the Unicon VM and the cost of this addition. Our cost here is in the size of the VM itself. So, always we are looking for a reasonable increase in the VM size for any new addition, which made Dr. Jeffery from the beginning very conservative about the Jori's VoIP libraries since they are three big libraries written in C++ and there is an extra overhead for another API library on top of them. Which made us worried about whether this new addition into the Unicon VM will be part of the standard Unicon distribution or not. This made us watching the size of those libraries and the increase in the VM caused by them. So, the changes in the Unicon VM "iconx" after adding the VoIP functions in the Linux and windows systems is in the following table:

| Item | Size before adding VoIP | Size after adding VoIP |
|---|---|---|
| /src/runtime/iconx | 628.4 KB | 4.2 MB |
| /bin/iconx | 539.1 KB ( after strip) | 1.5 MB |

Table: The changes in the Linux Unicon VM after adding the VoIP functions

And the changes in the VM in windows machines are in the following table

| Item | Size before adding VoIP | Size after adding VoIP |
|---|---|---|
| iconx.exe | 597 KB | 1.23 MB |
| wiconx.exe | 605 KB | 1.42 MB |

Table: The changes in the Windows Unicon VM after adding the VoIP functions

So, the increase in the iconx in Linux system is about **0.9609 MB** which is about **178%** of the original size. Also, the increase in the windows VM in iconx.exe is about **0.633 MB** which is about **106%** of the original size of the iconx.exe. And in wiconx.exe is about **0.815 MB** which is about **135%** of the original wiconx.exe.

This is not acceptable at all for the Unicon VM in both Linux and MS Windows. This situation encouraged us to find a way to reduce this size. The structure of the Jori's VoIP libraries and their components encouraged us to disable some of those components which are not necessary for us such as the compression techniques and the localizations which are the 3D sound effects on the VoIP.
However, those additions where built on top of the Jori's libraries which their sizes are in the following table.

| ***Library*** | ***Windows Static library size*** | ***Linux Static library Size*** | ***Linux Dynamic library Size*** |
|---|---|---|---|
| jvoip | ***2.82 MB*** | ***6.4 MB*** | ***3.5 MB*** |
| JThread | 34 KB | 6.1 KB | 7.4 KB |
| jrtp | 416 KB | 254.9 KB | 192.5 KB |

Table: Jori's Libraries before disabling any component.

And after disabling some components, the libraries are in the following table

| ***Library*** | ***Windows Static library size*** | ***Linux Static library Size*** | ***Linux Dynamic library Size*** |
|---|---|---|---|
| jvoip | ***1.7 MB*** | ***3.9 MB*** | ***2.0 MB*** |
| JThread | 34 KB | 6.1 KB | 7.4 KB |
| jrtp | 416 KB | 254.9 KB | 192.5 KB |

Table: Jori's Libraries after disabling some component

The new changes after disabling those components to be in the following table

| Item | Size before adding VoIP | Size after adding VoIP |
|---|---|---|
| /src/runtime/iconx | 628.4 KB | 2.8 MB |
| /bin/iconx | 539.1 KB ( after strip) | 978 KB |

Table: The changes in the Linux Unicon VM after disabling some components from the VoIP library

| Item | Size before adding VoIP | Size after adding VoIP |
|---|---|---|
| iconx.exe | 597 KB | 784 KB |
| wiconx.exe | 605 KB | 988 KB |

Table: The changes in the Windows Unicon VM after disabling some components from the VoIP functions

So, now after disabling some components from the jvoiplib, the increase in the iconx in Linux system is about **438.9 KB** which is about **81%** of the iconx original size. Also, the increase in the windows VM in iconx.exe is about **187 KB** which is about **31%** of the iconx.exe original size. And in wiconx.exe is about **383 KB** which is about **63%** of the wiconx.exe original size. These new changes seems to be reasonable changes and in some who acceptable for the Unicon VM.

## 8 Using the new Unicon VoIP functions

Unicon is always under development, and recently has been developed in the area of sounds and VoIP facilities in order to accomplish its new era in the field of Collaborative Virtual Environments (CVE). These new development focused on playing sounds and making a voice chatting which is now a necessity for a language like Unicon since Unicon is a goal directed high level language meant to save the programmer time and to make his development and maintenance so cheap.

Like in text chatting, In order to make a bidirectional voice chatting, both of the end users must be in the session at the same time. So, in order to make a voice chatting between two users and to talk to each other, both of them must have an open VoIP session and both of them must add the other IP address or the host-name into the listeners list of broadcasting using the VAttrib function with the "cast+=" attribute. However, if they want to make a one directional voice chatting the listener

must have an open voice session on some port and the sender mast also have an open session on some port and he must talk to the listener on the listener port.

Unicon now is supporting the on-line conferencing. It supports connecting n-users together, which means that they can talk and communicate using a VoIP session through the very simple high level Unicon built-in functions. Unicon also supports the idea of multicasting, one user can cast to n-users which maybe some of them servers who can extend the casting into another m-users and so on. Also away from the VoIP, Unicon can play wave files locally in both windows and Linux. The playing sound can be accomplished using the PlaySound() function. This function has been built on top of the open source library

Unicon provides the VoIP facility in a very simple way through three main functions

### 8.1 open()

open function opens a VoIP session and returns a handler into that session which should be saved into a variable. It takes the port number that you want to establish the voice the session on and the mode which is the letter “v” or “V” as an acronym for voice; the previous two are required parameters. However, there is an additional optional parameter that allows the programmer to specify the listeners at the opening time. This optional parameter is a string of one or more destinations attached to their ports.

The syntax of open is

Variable: = open (“base port”, mod [, string of one or more listeners])

Example

```
# very simple Unicon program lets you connect to your self
procedure main()
      local VSession
      VSession := open("4500", "v", "localhost:4500")
      read()
      close(VSession)
end
```

### 8.2 close()

In order to make the VoIP chatting, you have to, at first, open a VoIP session using the previous open function , and to closes the VoIP session that is already opened there is a close function. close function takes the session handler as a

parameter and returns nothing. It should close and clean every thing related to the session.

The syntax of close is

close( variable holding the session handler)

Example

```
# A Unicon program lets you make a VoIP with two listeners
# zorro.cs.nmsu.edu at port 5000 and
# 128.123.64.49 at port 4500
procedure main()
      local VoIP
      VoIP := open("5000","V","zorro.cs.nmsu.edu:5000, 128.123.64.49:4500")
      read()
      close(VoIP)
end
```

### 8.3 VAttrib()

VAttrib is an essential function manages many attributes such as adding new host into the session, dropping off some hosts out of the session and making a query about those who are listening to the session at that time.

The VAttrib function takes two parameters, the first is the session handler which has been obtained by the previous open function and the second is a string which has an attribute that may be attached into a list of destinations especially in the case of adding and dropping. Those attributes are

- "cast+=" : used to add one or more listener into the open VoIP session
- "cast-=" : used to drop out one or more listeners from the open VoIP session
- "cast" : used to ask about those who are listening to the session. In this case the VAttrib will return a string contains all the listeners and their session ports, the listeners will be separated with a comma.

The syntax of VAttrib is

VAttrib(Session handler variable, "attribute[one or more destinations]")

Example

```
# A Unicon program lets you make a VoIP with two listeners
# zorro.cs.nmsu.edu at port 5000 and
#128.123.64.49 at port 4500
procedure main()
	local VoIP, listener
	VoIP := open("5000", "V")
	Vattrib(VoIP,"cast+=zorro.cs.nmsu.edu:5000,128.123.64.49:4500")
	write("To Stop Casting , JUST Press Enter”)
	read()
	Vattrib(VoIP,"cast-=zorro.cs.nmsu.edu:5000")
	write("To See The Listeners: JUST Press Enter")
	read()
	listener := Vattrib(VoIP,"cast")
	write("The Listener Is" ,listener)
	close(VoIP)
end
```

### 8.4 VWho()

VWho is a function creating a list with the listener's host-names or their IPs at that time. It takes the session handler that has been saved from the open function and returns a Unicon list with those listeners.

The syntax of VWho is

```
	VWho(Session handler variable)
```

Example

```
procedure main()
	local voice , list
	voice := open( "4500","V","128.123.64.49:4500")
	VAttrib(voice, "cast+=128.123.64.48:4500,localhost:4500")
	# This will create a list with the destination IP's
	list := VWho(voice)
	# This will write out the contents of the list.
	every write(! list)
	close(voice)
end
```

This function is very important since it allows the programmer to see and remember those who are listening to the current VoIP session especially in the case of multicasting situation or even n-user conferencing.

### 8.5 PlaySound()

This function is not for VoIP, it is only to play sounds like wav files. It takes the name of the file ant it returns nothing.

The syntax of PlaySound is

```
PlaySound(filename)
```

where the file name is a string and may contain the path where the file is located.\\

Example

```
procedure main()
    PlaySound(“sample.wav”)
end
```

**References**


1. Libsndfile
   http://www.mega-nerd.com/libsndfile

2. Liesenborg, J., Voice over IP in networked virtual environments, 2000.
   http://lumumba.luc.ac.be/jori/thesis/thesis.html

3. OpenAL
   http://www.openal.org

4. Port Audio
   http://www.portaudio.com

5. Virtual Computer Science Community (VCSC)
   http://www.cs.nmsu.edu/~jeffery/vcsc

6. OpenAL Explained by Dave Phillips , 10/13/2000
   http://www.linuxdevcenter.com/pub/a/linux/2000/10/13/oa_openal.html

7. PortMusic APIs, Platform Independent Libraries for Sound and MIDI
   http://www-2.cs.cmu.edu/~music/portmusic

8. Sound and Music Software, by Dave Phillips, in Category Reviews - Sat, Dec 15th 2001 00:00 PDT
   http://freshmeat.net/articles/view/354

9. NCH Swift Sound
   http://www.nch.com.au/acm/formats.html

10. Audio Compression
    http://www.vocal.com/data_sheets/audio_compression.html

11. The official website for Unicon programming language
    http://unicon.org

12. Technical Articles & Tips, Mixing C and C++ Code in the Same Program
    http://developers.sun.com/tools/cc/articles/mixing.html

13. Jeffery, Mohamed, Pereda & Parlett, Programming with Unicon, September first, 2004
    http://gcc.gnu.org/ml/gcc-help/2004-07/msg00107.html

14. C++ 4.1 User's Guide: using C and C++
http://docs.sun.com/app/docs/doc/802\-3043/6i6vej1nd?a=view

15. C++ portability guide
http://www.mozilla.org/hacking/portable-cpp.html

16. Ralph E. Griswold, Supplementary Information for the Implementation of Version 9 of Icon, April 7, 1996.
http://www.cs.arizona.edu/icon/docs/ipd239.htm

17. The Icon Programming Language
http://www.cs.arizona.edu/icon/index.htm

18. The Run-Time Implementation Language for Icon, Kenneth Walker Department of Computer Science, The University of Arizona
http://www.cs.arizona.edu/icon/ftp/doc/ipd261.pdf

19. Calling C Functions from Version 9 of Icon, Gregg M. Townsend and Ralph E. Griswold, march 1996,
http://www.cs.arizona.edu/icon/docs/ipd240.htm