AVENTURA™
AWARENESS COMPUTING™

# Aventura VoIP Module

Integrating Asterisk Voice over Internet Protocol Phone System with Aventura Roaming Awareness Software

Final Report Document

Stephen Kennicutt
Tyler Thorn
June 15, 2015
Aventura VoIP Team

# Table of Contents

## Introduction

Aventura is a Denver-based software company that focuses on developing customized solutions for medical facilities. Aventura's flagship product is the called the "Roaming Aware Desktop" (RAD) which allows medical personnel access to secure, consistent and customized workflows to help reduce time spent on mundane and repetitive technical functions. The RAD is comprised of a central server that hosts the system's core functions and user endpoints that communicate with the core. This server-endpoint interaction provides a way for doctors and nurses to have a type of "roaming profile" service that allows these medical staff's unique desktops to travel around the hospital as they login with personalized Radio-Frequency Identification (RFID) cards to card readers attached to the endpoints placed in various rooms. In addition to presenting necessary patient information and transferring user settings to the current endpoint, the RAD system can also set default printers to the one that is the closest to the current location. For this project, Aventura has asked us to add a Voice over Internet Protocol (VoIP) module to the core system that gives the RAD the ability to have a dedicated internal phone line follow users as they travel around the medical facility.

## Requirements

There were two functional abstract services that we needed to implement for this project, which we broke into smaller sub-problems:

### *Aventura RAD Core:*

- Collecting information about user when signing in
- Set appropriate phone number to user
- Merge maps of phones with maps in Aventura core database
- Query information about current location
- Maintain proper security

### *VoIP Server:*

- Phone number will follow provider
- Calling a number results in the phone ringing in the intended recipient's current location
- If no available phone, call will be forwarded to the recipient's cell phone.
- Mapping phones to locations
- Matching users to phones

### *Non-Functional*:

- Module written in C++
- Module to be used with existing, stable, release of software
- Will package into module for release with product
- Development on separate branch of software
- Use Visual Studio for development
- Asterisk service used for VOIP server

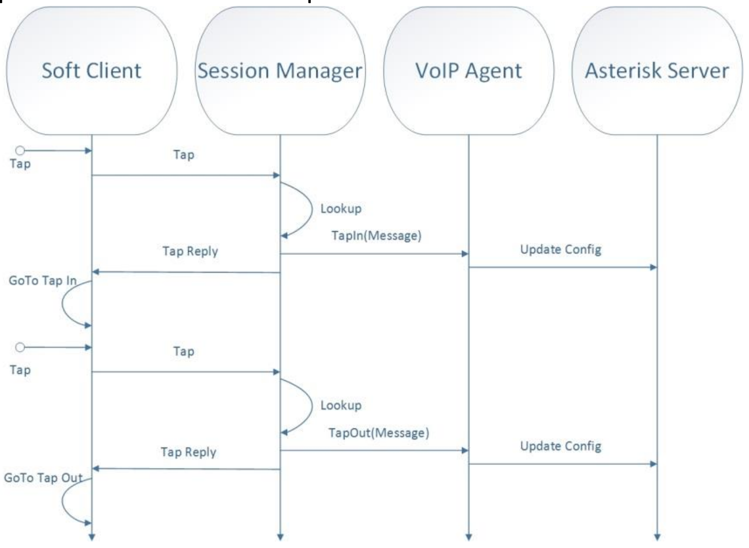
### *Potential Risks*:

- Asterisk and Aventura are fairly complicated technologies and there was no guarantee that integration was possible
- Testing the overall framework was very difficult, as it couldn't be simulated virtually
- Neither one of us had experience with implementing a VOIP server

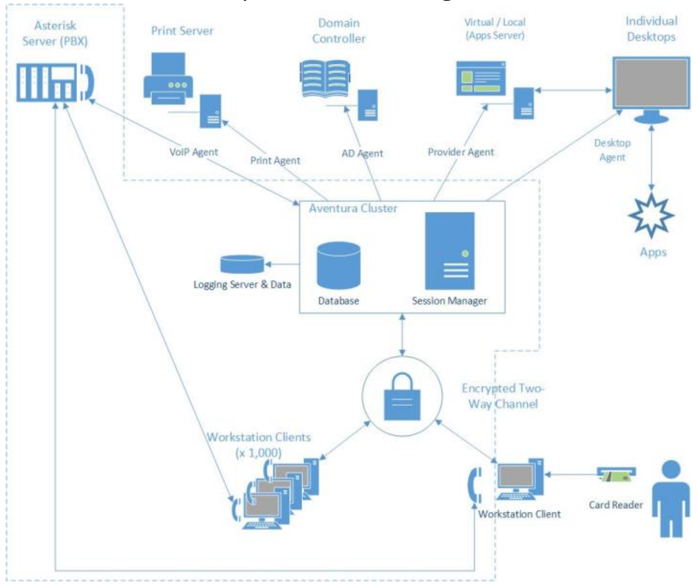
The design of the Aventura Voice over Internet Protocol (VoIP) module was largely driven by a "do what works quickly" philosophy. The functionality of our module has not been implemented before (or at least no internet footprint was left by any similar pre-existing implementation) so we had to invent the system ourselves.

## System Architecture

From an abstract view, our module will need to recognize a 'tap' from a user, take that tap and the location and be able to match the user's extension to the phone in his location. On the Aventura side, a soft client runs on an end point, the user taps his card and the soft client sends a tap notification to the Session Manager (Aventura server). From there, the Session Manager does a look up and figures out if it was a "tap-in" or a "tap-out". From there, it will message the VoIP Agent and then the VoIP Agent will then tell the Asterisk Server what changes to make so that the extension will be correctly routed. The following diagram depicts the situation of a "tap."



The Module will sit on the Aventura Server and will be like all the other agents that they have. The module will not need to be there for the system to work but will work with an existing system. The following diagram shows the Aventura setup and the VoIP Agent and its interactions.



## Design and Implementation

Our first task was to understand how Asterisk, the open-source VoIP server that the client requested that we implement, functioned and how to correctly create roaming phones in the system. After understanding how phones communicated with each other, we were able to create and evaluate two viable methods for phones to properly roam from location to location. We provided an explanation of options document that is included in Appendix A:

1. Have the physical phones register as a new user (that matched the person entering the room) system wide whenever a user logged in to a terminal.

2. Have two sets of users in the system, one user representing the location of the phone, and the other user representing the personnel in the hospital. With this configuration, we would forward the human users to the location users, edit the appropriate configuration files each time a user changed locations and reload the dial plan through the Asterisk system commands on the server.

The first scenario was our initial plan which followed the model of how a Voice over IP system is typically used. The issue with this system is that the phones need to be rebooted in order for them to receive their new instructions and dial plan from the server. This restart was discovered to be very time consuming (on the best case we achieved about a 1:15 minute reboot). VoIP systems and phones were not designed to be moved or change users regularly, therefore there was no feasible way to speed this up, nor could the phones be told to pull a new configuration remotely without a reboot. Conversely, when trying out our second option, we achieved reload times of the server configuration of no more than one second. Consequently, the second path is the one we chose.

Knowing that our routing changes were going to be implemented through changes in a user's configuration file, we needed to create a way to dynamically edit these files in our client agent. This became pretty tricky as the Aventura client operated on Windows Server-based machines while the Asterisk server is typically deployed on a Linux distribution, and there was no clean cut way to achieve file transfers between these operating systems in an autonomous fashion in code. We went and researched some open source libraries to prevent us from having to write a transfer protocol from scratch. We found two libraries that suited our needs: libssh and libcurl. Both of these transfer libraries worked for our purposes for similar time costs, but we decided to go with libssh as the code was more readable than libcurl and provided more options for the file transfer from trivial file transfer (tftp) to secure copy (scp) and secure file transfer (sftp), including hypertext transfer (secure) (http(s)).

Ultimately, we decided to use the scp functionality of libssh. One of our requirements was that the transfer be somewhat secure, (not unbreakable encryption, but also not a plaintext transfer), which ruled out the http, tftp and ftp protocols. In addition, we wanted the system to be able to work in a wide variety of implementations. Using https would require setup and the creation of certificates as well as trying to get the certificate to authenticate in our program, which was beyond our desired scope. Lastly, with sftp we would have to have the Asterisk server also be set up as an ftp server and again certificates could be a problem. This left us with the scp protocol through an ssh tunnel. After the connection is verified the first time and the server is added to known hosts, there is no other security issues (other than passing credentials which all implementations would require) and it only requires that the Asterisk server allows ssh connections from one specified user and only access to the directory that the configuration files are located in.

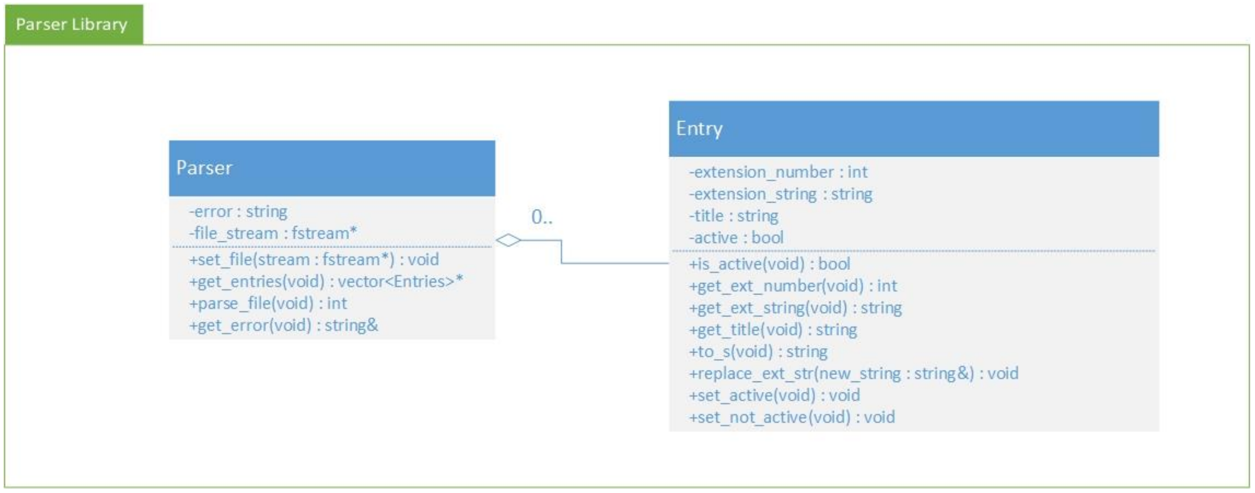
We wrote throwaway code to test out the efficiency and usage of libssh using scp and ssh tunnel. We came up with a way to further abstract it to make the use easier and better fit our system. We ended up writing a class that creates the connection and has methods for pulling and pushing files. We then had a decision to just put the class into our main program or to pull it out and use it as a library. Around the same time we were also working on a file parser. We broke that out into two classes. One that performed the parsing, and the other was an object to hold one entry of the file. There was then a container to hold multiple of these objects per file. This too we had to decide if we will use simply in the project or create a library out of it as well. Because we made the file transfer easy to use and generic enough to be used almost anywhere file transfer needed to happen, we decided to make it a library. We also decided to make the parser and entry class as a library. With some guidance from the team at Aventura, we decided to go with a dynamic library. The main reason for this is that we can make changes to the existing functions to make them better or to fix a bug and they can just be a drop in replacement rather than having to recompile the entire project as would be the case with a static library.

Next, we needed a way for the server to reload the dial plan after the configuration files had been edited. Though we could have come up with a way for the client to issue a reload to the server, we decided it would be best if the server had a listener script that watched the directory with the configuration files to see if any files had been edited and then call the reload function. This was due to the fact that we wanted the server to be in control of the configuration files whenever possible to prevent a possible “dining philosophers” problem where the contents of the configuration files would be altered incorrectly and avoid multiple, simultaneous connections to the server, one for the scp and one for the reload.

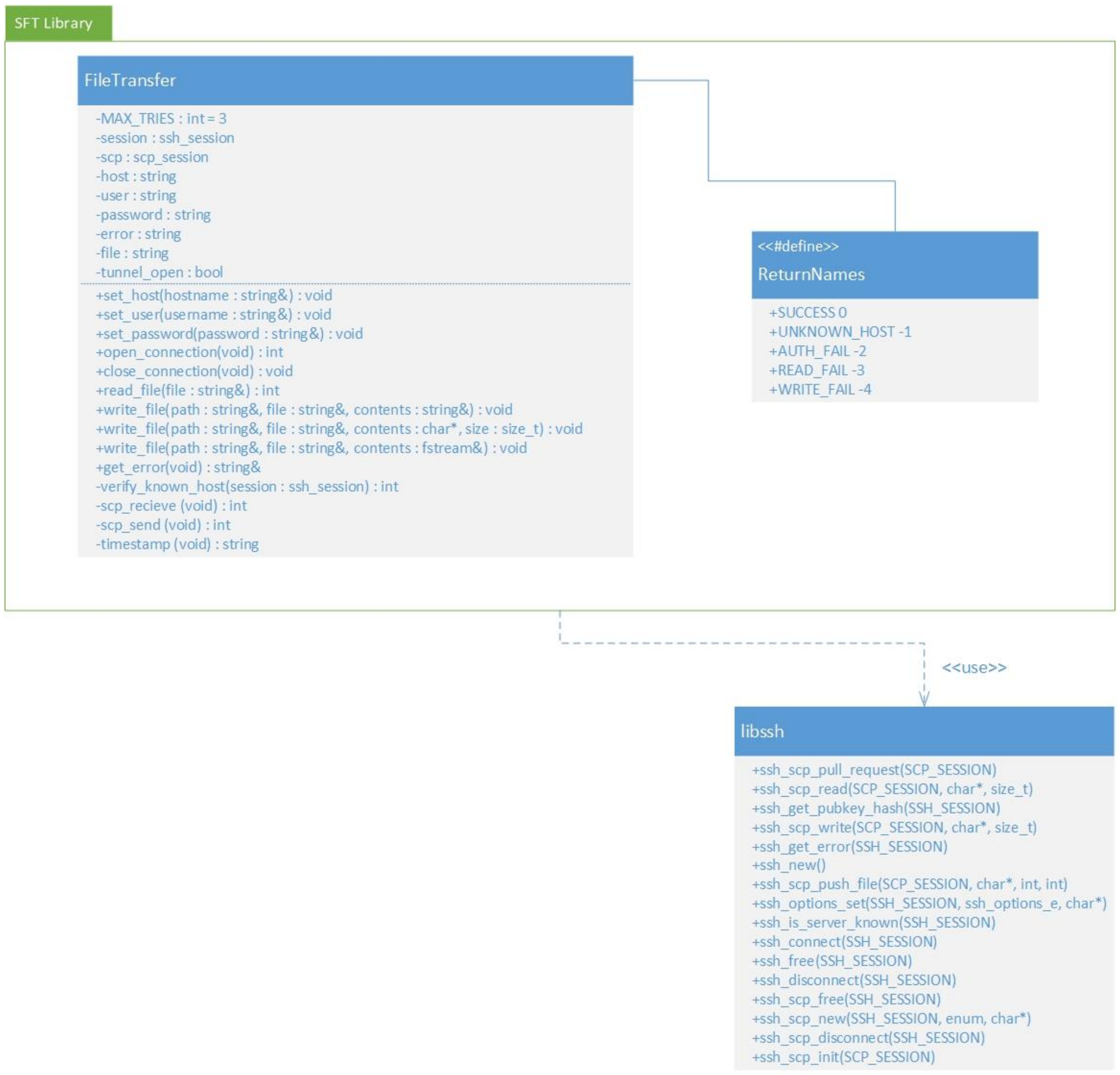
Finally, we needed a way to wrap our client functions into an Aventura agent. We deferred to the design guidance of the Aventura engineers, as they were most familiar with the design decisions implemented in existing agents. We are unable to elaborate on the design decisions made in regards to agent creation due to a non-disclosure agreement with the client.

## Technical Design

From all of the decisions we made, we were able to produce the following UML diagrams of how our agent will work.

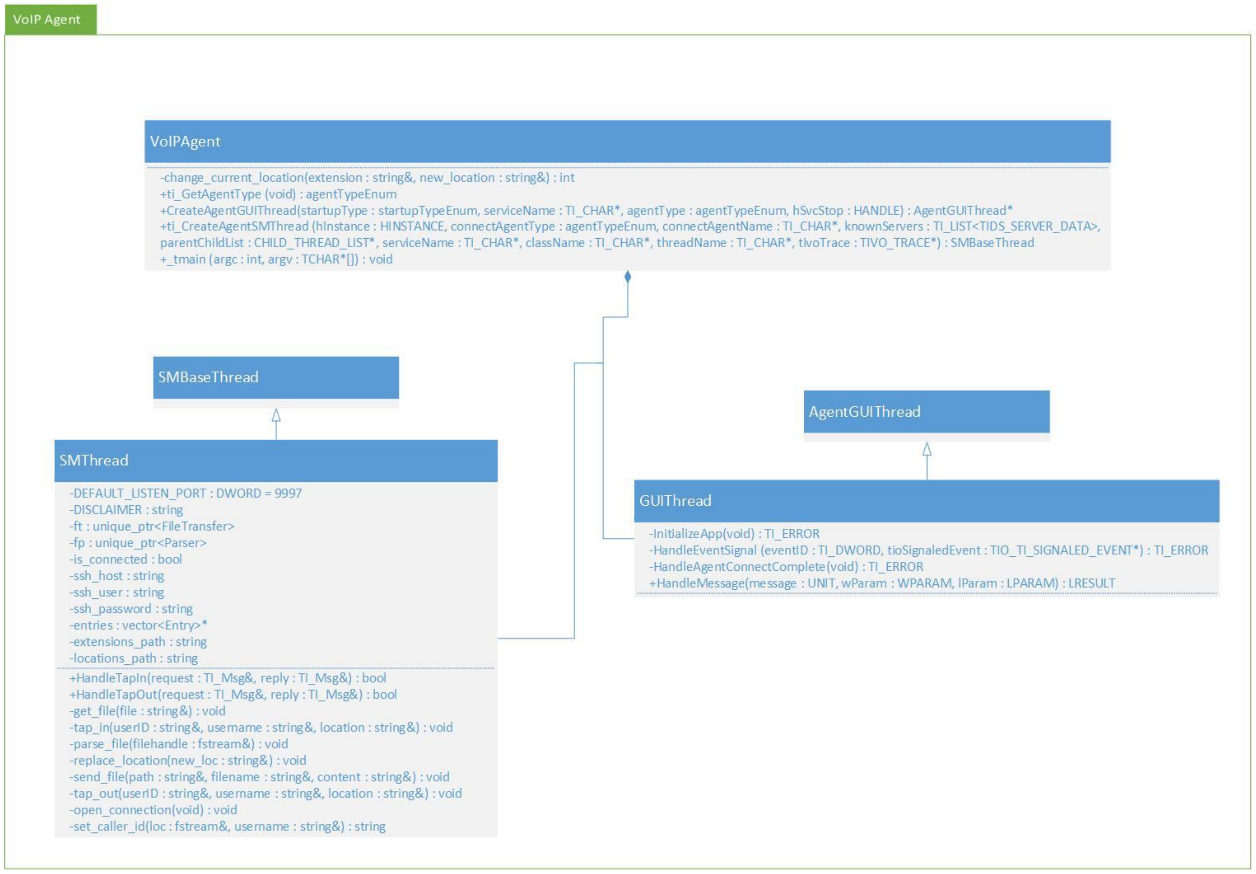

Parser Library
Parser
-error : string
-file_stream : fstream*
+set_file(stream : fstream*) : void
+get_entries(void) : vector<Entries>*
+parse_file(void) : int
+get_error(void) : string&
0..
Entry
-extension_number : int
-extension_string : string
-title : string
-active : bool
+is_active(void) : bool
+get_ext_number(void) : int
+get_ext_string(void) : string
+get_title(void) : string
+to_s(void) : string
+replace_ext_str(new_string : string&) : void
+set_active(void) : void
+set_not_active(void) : void

UML 1 Parser Dynamic Library


SFT Library
FileTransfer
-MAX_TRIES : int = 3
-session : ssh_session
-scp : scp_session
-host : string
-user : string
-password : string
-error : string
-file : string
-tunnel_open : bool
+set_host(hostname : string&) : void
+set_user(username : string&) : void
+set_password(password : string&) : void
+open_connection(void) : int
+close_connection(void) : void
+read_file(file : string&) : int
+write_file(path : string&, file : string&, contents : string&) : void
+write_file(path : string&, file : string&, contents : char*, size : size_t) : void
+write_file(path : string&, file : string&, contents : fstream&) : void
+get_error(void) : string&
-verify_known_host(session : ssh_session) : int
-scp_recieve (void) : int
-scp_send (void) : int
-timestamp (void) : string
<<#define>>
ReturnNames
+SUCCESS 0
+UNKNOWN_HOST -1
+AUTH_FAIL -2
+READ_FAIL -3
+WRITE_FAIL -4
<<use>>
libssh
+ssh_scp_pull_request(SCP_SESSION)
+ssh_scp_read(SCP_SESSION, char*, size_t)
+ssh_get_pubkey_hash(SSH_SESSION)
+ssh_scp_write(SCP_SESSION, char*, size_t)
+ssh_get_error(SSH_SESSION)
+ssh_new()
+ssh_scp_push_file(SCP_SESSION, char*, int, int)
+ssh_options_set(SSH_SESSION, ssh_options_e, char*)
+ssh_is_server_known(SSH_SESSION)
+ssh_connect(SSH_SESSION)
+ssh_free(SSH_SESSION)
+ssh_disconnect(SSH_SESSION)
+ssh_scp_free(SSH_SESSION)
+ssh_scp_new(SSH_SESSION, enum, char*)
+ssh_scp_disconnect(SSH_SESSION)
+ssh_scp_init(SCP_SESSION)

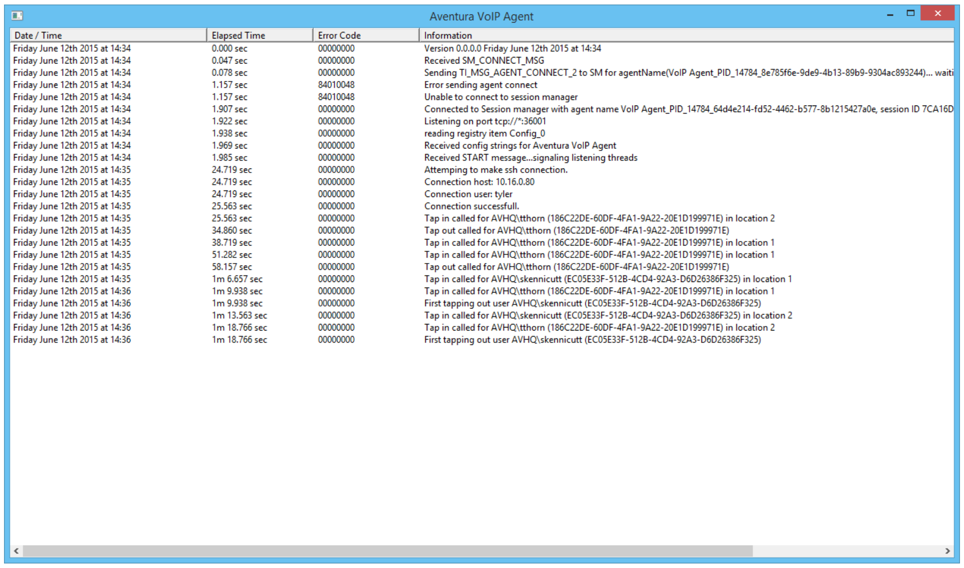
**UML 2 Secure File Transfer Dynamic Library**

The two UMLs above are of our two libraries, the first is our SFT library. It is a wrapper for libssh. We decided to make it into a class rather than using static functions (how libssh is) so that one connection can be made and we can use that connection the entire time that the agent is running. There are set functions for setting the information of the ssh tunnel including the host address, username and password. Then there are also functions for reading and writing files through the scp protocol. The write function is overloaded for a total of three different ways of passing in a file to write. The read file method will read the given file and write it locally to the current directory. The second library is the Parser library. There are two classes in this library. Rather than just having strings passed from the parser, there is an Entry class which is one entry in the dial plan for the current user. The parser, when parsing, will make a collection of Entries that can then be retrieved by the library user. These entries also have modifiers and one of which, the change location function, we use to set the route for the new location. There are also set active and set inactive modifiers so that a tap in can revert to the default location for the user.


VoIP Agent
VoIPAgent
-change_current_location(extension : string&, new_location : string&) : int
+ti_GetAgentType (void) : agentTypeEnum
+CreateAgentGUIThread(startupType : startupTypeEnum, serviceName : TI_CHAR*, agentType : agentTypeEnum, hSvcStop : HANDLE) : AgentGUIThread*
+ti_CreateAgentSMThread (hInstance : HINSTANCE, connectAgentType : agentTypeEnum, connectAgentName : TI_CHAR*, knownServers : TI_LIST<TIDS_SERVER_DATA>, parentChildList : CHILD_THREAD_LIST*, serviceName : TI_CHAR*, className : TI_CHAR*, threadName : TI_CHAR*, tivoTrace : TIVO_TRACE*) : SMBaseThread
+_tmain (argc : int, argv : TCHAR*[]) : void
SMBaseThread
SMThread
-DEFAULT_LISTEN_PORT : DWORD = 9997
-DISCLAIMER : string
-ft : unique_ptr<FileTransfer>
-fp : unique_ptr<Parser>
-is_connected : bool
-ssh_host : string
-ssh_user : string
-ssh_password : string
-entries : vector<Entry>*
-extensions_path : string
-locations_path : string
+HandleTapIn(request : TI_Msg&, reply : TI_Msg&) : bool
+HandleTapOut(request : TI_Msg&, reply : TI_Msg&) : bool
-get_file(file : string&) : void
-tap_in(userID : string&, username : string&, location : string&) : void
-parse_file(filehandle : fstream&) : void
-replace_location(new_loc : string&) : void
-send_file(path : string&, filename : string&, content : string&) : void
-tap_out(userID : string&, username : string&, location : string&) : void
-open_connection(void) : void
-set_caller_id(loc : fstream&, username : string&) : string
AgentGUIThread
GUIThread
-InitializeApp(void) : TI_ERROR
-HandleEventSignal (eventID : TI_DWORD, tioSignaledEvent : TIO_TI_SIGNALED_EVENT*) : TI_ERROR
-HandleAgentConnectComplete(void) : TI_ERROR
+HandleMessage(message : UNIT, wParam : WPARAM, lParam : LPARAM) : LRESULT

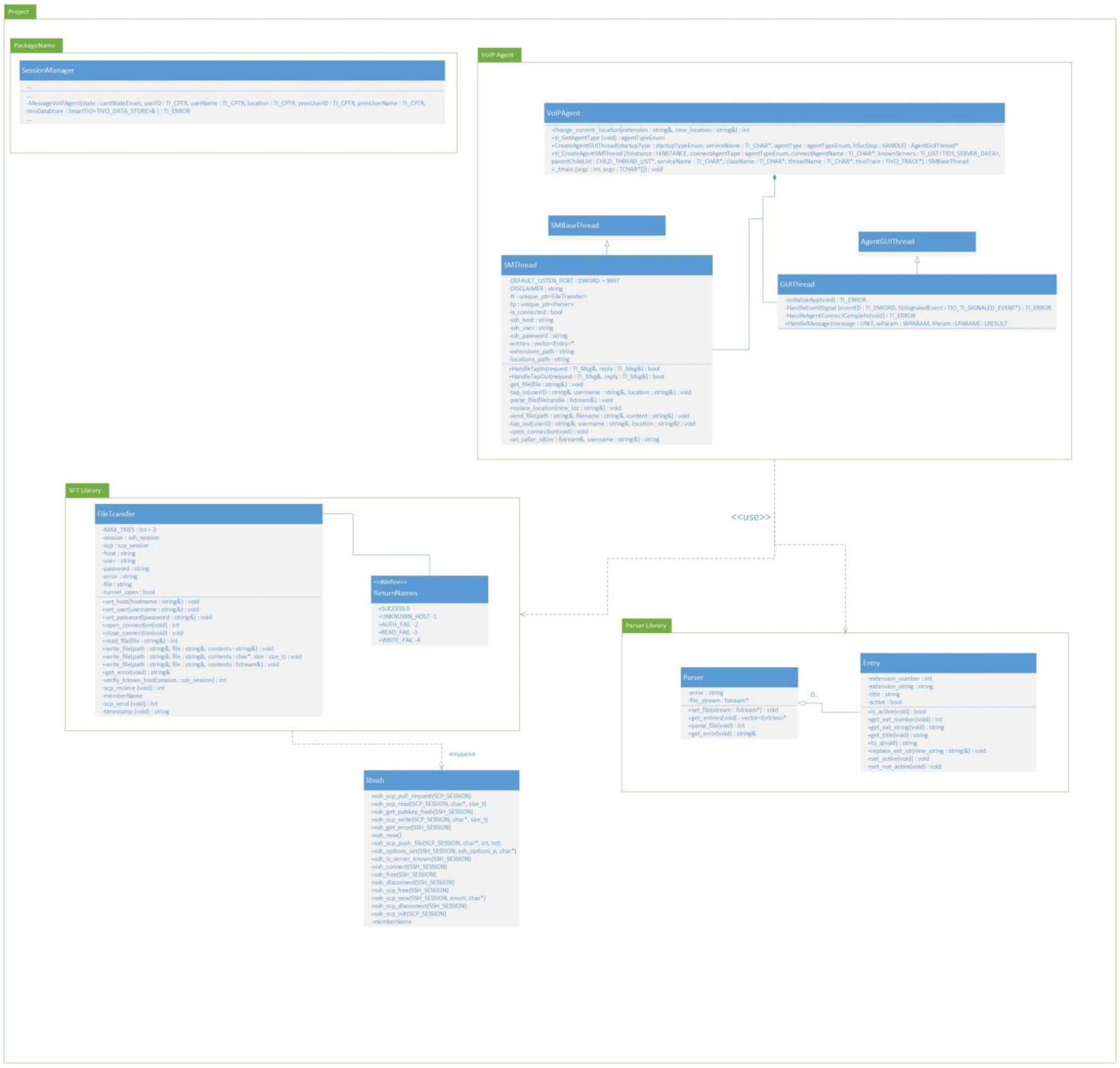
**UML 3 VoIP Agent**

Above is the UML for the VoIP Agent. The agent was made to fit the model of the existing Aventura Agents. The agent is comprised of a SMThread (SessionManager Thread) which is then wraped into a GUIThread. Both SMThread and GUIThread are child classes of the SMBaseThread and AgentGUIThread respectively. Most of the methods are inherited and some are virtual and need an instantiation by the child class. The following picture shows a sample of what the GUIThread looks like.



```
Date / Time                     Elapsed Time   Error Code   Information
Friday June 12th 2015 at 14:34  0.000 sec      00000000     Version 0.0.0.0 Friday June 12th 2015 at 14:34
Friday June 12th 2015 at 14:34  0.047 sec      00000000     Received SM_CONNECT_MSG
Friday June 12th 2015 at 14:34  0.078 sec      00000000     Sending TI_MSG_AGENT_CONNECT_2 to SM for agentName(VoIP Agent_PID_14784_8e785f6e-9de9-4b13-89b9-9304ac893244)... waiti
Friday June 12th 2015 at 14:34  1.157 sec      84010048     Error sending agent connect
Friday June 12th 2015 at 14:34  1.157 sec      84010048     Unable to connect to session manager
Friday June 12th 2015 at 14:34  1.907 sec      00000000     Connected to Session manager with agent name VoIP Agent_PID_14784_64d4e214-fd52-4462-b577-8b1215427a0e, session ID 7CA16D
Friday June 12th 2015 at 14:34  1.922 sec      00000000     Listening on port tcp://*:36001
Friday June 12th 2015 at 14:34  1.938 sec      00000000     reading registry item Config_0
Friday June 12th 2015 at 14:34  1.969 sec      00000000     Received config strings for Aventura VoIP Agent
Friday June 12th 2015 at 14:34  1.985 sec      00000000     Received START message...signaling listening threads
Friday June 12th 2015 at 14:35  24.719 sec     00000000     Attemping to make ssh connection.
Friday June 12th 2015 at 14:35  24.719 sec     00000000     Connection host: 10.16.0.80
Friday June 12th 2015 at 14:35  24.719 sec     00000000     Connection user: tyler
Friday June 12th 2015 at 14:35  25.563 sec     00000000     Connection successfull.
Friday June 12th 2015 at 14:35  25.563 sec     00000000     Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 2
Friday June 12th 2015 at 14:35  34.860 sec     00000000     Tap out called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E)
Friday June 12th 2015 at 14:35  38.719 sec     00000000     Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 1
Friday June 12th 2015 at 14:35  51.282 sec     00000000     Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 1
Friday June 12th 2015 at 14:35  58.157 sec     00000000     Tap out called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E)
Friday June 12th 2015 at 14:35  1m 6.657 sec   00000000     Tap in called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325) in location 1
Friday June 12th 2015 at 14:36  1m 9.938 sec   00000000     Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 1
Friday June 12th 2015 at 14:36  1m 9.938 sec   00000000     First tapping out user AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325)
Friday June 12th 2015 at 14:36  1m 13.563 sec  00000000     Tap in called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325) in location 2
Friday June 12th 2015 at 14:36  1m 18.766 sec  00000000     Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 2
Friday June 12th 2015 at 14:36  1m 18.766 sec  00000000     First tapping out user AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325)
```

**UML 4 GUI Screen**

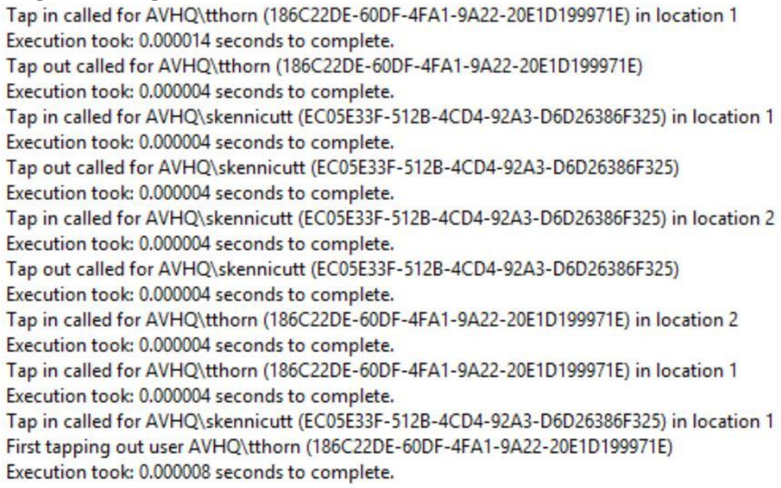

Project
PackageName
SessionManager
...
...
-MessageVoIPAgent(state : cardStateEnum, userID : TI_CPTR, userName : TI_CPTR, location : TI_CPTR, prevUserID : TI_CPTR, prevUserName : TI_CPTR, tivoDataStore : SmartTIO<TIVO_DATA_STORE>& ) : TI_ERROR
...
VoIP Agent
VoIPAgent
-change_current_location(extension : string&, new_location : string&) : int
+ti_GetAgentType (void) : agentTypeEnum
+CreateAgentGUIThread(startupType : startupTypeEnum, serviceName : TI_CHAR*, agentType : agentTypeEnum, hSvcStop : HANDLE) : AgentGUIThread*
+ti_CreateAgentSMThread (hInstance : HINSTANCE, connectAgentType : agentTypeEnum, connectAgentName : TI_CHAR*, knownServers : TI_LIST<TIDS_SERVER_DATA>, parentChildList : CHILD_THREAD_LIST*, serviceName : TI_CHAR*, className : TI_CHAR*, threadName : TI_CHAR*, tivoTrace : TIVO_TRACE*) : SMBaseThread
+_tmain (argc : int, argv : TCHAR*[]) : void
SMBaseThread
AgentGUIThread
SMThread
-DEFAULT_LISTEN_PORT : DWORD = 9997
-DISCLAIMER : string
-ft : unique_ptr<FileTransfer>
-fp : unique_ptr<Parser>
-is_connected : bool
-ssh_host : string
-ssh_user : string
-ssh_password : string
-entries : vector<Entry>*
-extensions_path : string
-locations_path : string
+HandleTapIn(request : TI_Msg&, reply : TI_Msg&) : bool
+HandleTapOut(request : TI_Msg&, reply : TI_Msg&) : bool
-get_file(file : string&) : void
-tap_in(userID : string&, username : string&, location : string&) : void
-parse_file(filehandle : fstream&) : void
-replace_location(new_loc : string&) : void
-send_file(path : string&, filename : string&, content : string&) : void
-tap_out(userID : string&, username : string&, location : string&) : void
-open_connection(void) : void
-set_caller_id(loc : fstream&, username : string&) : string
GUIThread
-InitializeApp(void) : TI_ERROR
-HandleEventSignal (eventID : TI_DWORD, tioSignaledEvent : TIO_TI_SIGNALED_EVENT*) : TI_ERROR
-HandleAgentConnectComplete(void) : TI_ERROR
+HandleMessage(message : UNIT, wParam : WPARAM, lParam : LPARAM) : LRESULT
<<use>>
SFT Library
FileTransfer
-MAX_TRIES : int = 3
-session : ssh_session
-scp : scp_session
-host : string
-user : string
-password : string
-error : string
-file : string
-tunnel_open : bool
+set_host(hostname : string&) : void
+set_user(username : string&) : void
+set_password(password : string&) : void
+open_connection(void) : int
+close_connection(void) : void
+read_file(file : string&) : int
+write_file(path : string&, file : string&, contents : string&) : void
+write_file(path : string&, file : string&, contents : char*, size : size_t) : void
+write_file(path : string&, file : string&, contents : fstream&) : void
+get_error(void) : string&
-verify_known_host(session : ssh_session) : int
-scp_recieve (void) : int
-memberName
-scp_send (void) : int
-timestamp (void) : string
<<#define>>
ReturnNames
+SUCCESS 0
+UNKNOWN_HOST -1
+AUTH_FAIL -2
+READ_FAIL -3
+WRITE_FAIL -4
Parser Library
Parser
-error : string
-file_stream : fstream*
+set_file(stream : fstream*) : void
+get_entries(void) : vector<Entries>*
+parse_file(void) : int
+get_error(void) : string&
0..
Entry
-extension_number : int
-extension_string : string
-title : string
-active : bool
+is_active(void) : bool
+get_ext_number(void) : int
+get_ext_string(void) : string
+get_title(void) : string
+to_s(void) : string
+replace_ext_str(new_string : string&) : void
+set_active(void) : void
+set_not_active(void) : void
<<use>>
libssh
+ssh_scp_pull_request(SCP_SESSION)
+ssh_scp_read(SCP_SESSION, char*, size_t)
+ssh_get_pubkey_hash(SSH_SESSION)
+ssh_scp_write(SCP_SESSION, char*, size_t)
+ssh_get_error(SSH_SESSION)
+ssh_new()
+ssh_scp_push_file(SCP_SESSION, char*, int, int)
+ssh_options_set(SSH_SESSION, ssh_options_e, char*)
+ssh_is_server_known(SSH_SESSION)
+ssh_connect(SSH_SESSION)
+ssh_free(SSH_SESSION)
+ssh_disconnect(SSH_SESSION)
+ssh_scp_free(SSH_SESSION)
+ssh_scp_new(SSH_SESSION, enum, char*)
+ssh_scp_disconnect(SSH_SESSION)
+ssh_scp_init(SCP_SESSION)
-memberName

**UML 5 Full Project UML**

Putting all of the UML diagrams together, we arrive at this final UML with a MessageVoIPAgent function to send a tap in or tap out message to the VoIP Agent.

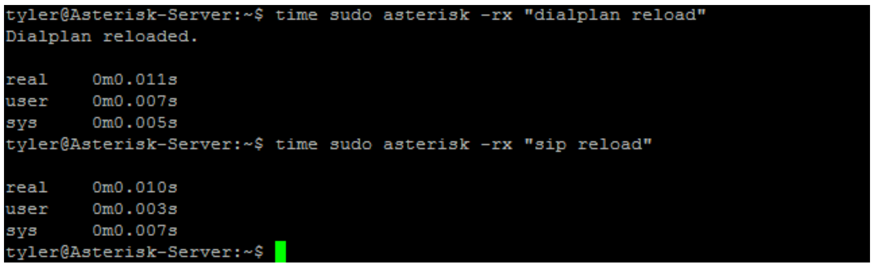
## Results

We were able to successfully create a VoIP server that works as desired with Aventura's managing service, with log ins, log outs and log overs being appropriately handled and reflected in phone behavior. When a user logs in to a terminal, our VoIP server correctly routes that user's extension to the nearest phone based off of information stored in the RAD's database. When a user logs out, the VoIP server cancels the established route. In addition, we were able to add caller identification functionality, which was not one of our given requirements. This interaction is executed within 0.02 seconds which is ideal for already impatient healthcare personnel. This number can be broken down into parts. The first is the time it takes from when a user taps to when the corresponding files have been modified and written to the Asterisk server. The first time a user taps, this takes 14 microseconds to complete because the ssh tunnel is not created until the first tap. Then, the following taps will only take 4 microseconds. A tap over will take 8 microseconds (because it is a tap out and a tap in). Sample output is provided for the timing of the Agent.



```
Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 1
Execution took: 0.000014 seconds to complete.
Tap out called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E)
Execution took: 0.000004 seconds to complete.
Tap in called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325) in location 1
Execution took: 0.000004 seconds to complete.
Tap out called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325)
Execution took: 0.000004 seconds to complete.
Tap in called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325) in location 2
Execution took: 0.000004 seconds to complete.
Tap out called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325)
Execution took: 0.000004 seconds to complete.
Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 2
Execution took: 0.000004 seconds to complete.
Tap in called for AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E) in location 1
Execution took: 0.000004 seconds to complete.
Tap in called for AVHQ\skennicutt (EC05E33F-512B-4CD4-92A3-D6D26386F325) in location 1
First tapping out user AVHQ\tthorn (186C22DE-60DF-4FA1-9A22-20E1D199971E)
Execution took: 0.000008 seconds to complete.
```

The slower of the timing is the reload that the Asterisk server takes, each tap requires a dial plan and a SIP reload, these each take approximately 10 milliseconds to complete making the total 20 milliseconds (we are truncating the time the Agent takes as it is minimal compared to the Asterisk Reload). The asterisk reload is shown below.



```
tyler@Asterisk-Server:~$ time sudo asterisk -rx "dialplan reload"
Dialplan reloaded.

real    0m0.011s
user    0m0.007s
sys     0m0.005s
tyler@Asterisk-Server:~$ time sudo asterisk -rx "sip reload"

real    0m0.010s
user    0m0.003s
sys     0m0.007s
tyler@Asterisk-Server:~$
```

If Aventura decides that they want to package our system with the existing RAD system, they will have to create a way properly update the database with the list of phones at each location (which they have done with printers) and implement a way to call an outside line with the Asterisk server. Overall the project was a success and all of the requirements were met and fully accepted by Aventura.

## Appendix A – Explanation of Options Document

### *Exploration of Options*

Preliminary –

IP phone systems are not meant to be moved around often. They are designed to be linked to a user and stick to that user for life. VoIP and digital phone systems are also designed that a user and an extension are synonymous. Asterisk especially, following best practice guides, the user is actually the same as an extension, the username is the extension (display names can be of the actual user).

High Level VoIP Overview –

VoIP phones are initially set up with an ftp server and credentials for the server as well as the PBX server. At boot time, the phone goes to the ftp server and retrieves the appropriate files for its configuration. These files have information on the extension they are supposed to handle, as well as a user and secret to use to register against the PBX server (Asterisk). Then the PBX server will know the IP address of the registered phone and route calls appropriately.

The following three options are what we have come up with as plausible solutions. None have been fully tested to ensure capability.

1. Changing of ftp files –

   In this use case, the config files for the phones on the ftp site will be changed accordingly. The extensions will be tied to users and we will change the extension in the file of the MAC address of the phone.

   Pros –

   a. Most like existing systems
   b. Able to use web configuration GUI and can (probably) be extended to other VoIP systems
   c. Able to use user control panel web gui
   d. Can (probably) integrate existing system
   e. Normal configuration more suitable if stationary positions exist (Receptionist)
   f. Intuitive system management
   g. Probably can have all features of VoIP system
   h. Soft Phone capability

   Cons –

   a. Phones need to reboot each time extension changes (slow)
   b. Might only work with Polycom phones
   c. Need FTP library to interact with module

2. Changing PBX config files –

   In this case, the users will be the rooms that the phones are in. The people using the phones will have their extensions and when entering a room, the config files in the PBX system will be modified, by changing the Dial Plan of the extension. The config is then reloaded and the extension ported to the right room.

   Pros –

   a. Faster reload time

b. Closest to description of action
c. Very proprietary

Cons –
a. Outside of design and functionality
b. Will not work with existing systems
c. Will not work with a web GUI
d. Very proprietary
e. Less intuitive system management
f. Need to create ssh tunnel in C++ code
g. Need to be able to shell out in said tunnel
h. Cannot have many features (Probably not even voicemail)

3. Soft Phones –

There is some software that allows a computer to act as a phone, the phone registers as a user to the server and can make and receive calls. We may be able to register the soft phone as the user logs in and have the calls come through the soft phone and not have hardware phones in the room.

Pros –
a. Only changing config files at the workstation, not on a server
b. Registers correctly with PBX
c. Can use any system with SIP configuration
d. Easily implements in existing system, just need credentials
e. Can use with any VoIP system (depending on system, soft phone may need different manufacture)

Cons –
a. No real phone
b. Need to bring microphone and speakers forward from Terminal Session
c. Not as private as hard phone
d. Less feature rich

## Appendix B – Administrator Guide

## Section 1: Installing Asterisk Server

There are two main options that can be used to install the Asterisk Phone Server. The first option is to install Asterisk on its own without any easy configurator front end. This is the preferred method as the integration with the Aventura system becomes much easier. Although the other way is also possible, but less preferred.

### Installing Asterisk Solo (without use of configuration front end)

Asterisk is open source and runs best on Linux Operating systems. The common environment is to install Ubuntu server 12 or 14 LTS. Asterisk will work on both versions. There are other distributions that will work, including CentOS and there is a package in the Arch Repositories for Arch Linux (although the module has not been tested in an Arch environment). Once the host OS is installed, Asterisk can be installed. Installation archives can be found at this link. The download is of a .tar.gz file. Extract the contents with the following command (it is recommended to extract the tar ball to the /usr/local/src directory, and this guide will assume that you do):

```
$ tar -zxvf {asterisk}.tar.gz
```

Using the correct filename for the chosen version. There are a few dependencies that are

```
$ cd dahdi-linux-complete-{VersionNumber}
```

included in the Asterisk tar ball. The first is DAHDI. Change the directory to the DAHDI directory.

Then DAHDI can be installed with the following commands (as root or elevated user):

```
# make
# make install
# make config
```

The next dependency is LibPRI.

NOTE: DAHDI Linux MUST BE installed prior to LibPRI, LibPRI may install without it but the configuration will not be correct.

Move into the LibPRI directory just as before.

```
$ cd libpri-{VersionNumber}
```

Then LibPRI can be installed using the following commands (again as an elevated user)

```
# make
# make install
# make config
```

One more set of non-repo supported dependencies is pjproject. Asterisk versions 11 and later will handle pjproject accordingly, if using an earlier version, installation instructions can be found here.

Then change to the directory containing the Asterisk source files.

```
$ cd /usr/local/src/asterisk-{Version}
```

In the Asterisk directory there is a script called configure that will check the system for required libraries and other dependencies. All of the remaining dependencies should be in your choice OS repositories. Running configure with:

```
# ./configure
```

Will check the dependencies and will alert of any missing packages. Install the missing packages and run configure again.

NOTE: Running configure multiple times causes a lot of data to be cached. Sometimes, this will output wrong results, the cache can be cleared by running:

```
# make distclean
```

And configure should be re-run after to verify everything is installed.

Once all of the dependencies are installed and found, the configure script will output the following screen.

```
                .$$$$$$$$$$$$$$$=..
              .$7$7..        .7$$7:.
            .$7$7..           .7$$7:.
          .$$:.                 ,$7.7
        .$7.     7$$$$           .$$77
     ..$$.       $$$$$            .$$$7
    ..7$   .?.   $$$$$   .?.       7$$$.
   $.$.   .$$$7. $$$$7 .7$$$.      .$$$.
 .777.   .$$$$$$77$$$77$$$$$7.      $$$,
 $$$~      .7$$$$$$$$$$$$$7.       .$$$.
.$$7          .7$$$$$$$7:          ?$$$.
$$$          ?7$$$$$$$$$$I        .$$$7
$$$       .7$$$$$$$$$$$$$$$$      :$$$.
$$$       $$$$$$7$$$$$$$$$$$$    .$$$.
$$$        $$$   7$$$7  .$$$    .$$$.
$$$$             $$$$7         .$$$.
7$$$7            7$$$$        7$$$
 $$$$$                        $$$
  $$$$7.                       $$  (TM)
   $$$$$$$.           .7$$$$$$  $$
     $$$$$$$$$$$$7$$$$$$$$$.$$$$$$
       $$$$$$$$$$$$$$$$.
configure: Package configured for:&nbsp;
configure: OS type &nbsp;: linux-gnu
configure: Host CPU : x86_64
configure: build-cpu:vendor:os: x86_64 : unknown : linux
configure: host-cpu:vendor:os: x86_64 : unknown : linux
```

NOTE: Output may be different depending on the host computer.

Then you must select the packages for Asterisk to install. A graphical window will display and you can select what packages to install. A detailed list of the options can be found here.

```
# make menuselect
```

You are then ready to compile Asterisk, to compile, simply run the following command

```
# make
```

This can take a few to several minutes to complete. When the compilation is complete, there will be a message that looks like:

```
+--------- Asterisk Build Complete ---------+
+ Asterisk has successfully been built, and +
+ can be installed by running:              +
+                                           +
+               make install                +
+-------------------------------------------+
+--------- Asterisk Build Complete ---------+
```

Then, as the message states, you can install the binaries using:

```
# make install
```

Once that has finished, you have successfully installed Asterisk. There are sample configuration files that can be installed by typing:

```
# make samples
```

Once finished, there are a few things to check to be sure the install was successful and the service can run. First, we need to make sure that DAHDI is loaded and running. To do this, check the lsmod function.

```
# lsmod | grep dahdi
```

lsmod should output something like the following:

```
dahdi_transcode  7928 1 wctc4xxp
dahdi_voicebus  40464 2 wctdm24xxp,wcte12xp
dahdi          196544 12 wctdm24xxp,wcte11xp,wct1xxp,wcte12xp
crc_ccitt        2096 1 dahdi
```

If the output is blank (e.g. nothing is written to the screen) look to your distros instructions on starting services, and start the dahdi service.

To check if Asterisk is running, there is a supplied init file that can be checked with:

```
# /etc/init.d/asterisk status
```

If Asterisk is stopped, you can start it with the same script.

```
# /etc/init.d/asterisk start
```

Congratulations! You now have a running Asterisk server!

### Installing Asterisk with a Graphical Front-end

NOTE: This is NOT the desired method and you can expect to see performance changes and incompatibility issues with Aventura software if using a GUI front end.

There are many different options for you to install Asterisk with a more user friendly configuration interface. The most common and easiest is probably AsteriskNOW. It includes a distribution, Asterisk and FreePBX (a web interface for Asterisk). The AsteriskNOW installation is very straight forward and easy, but for instructions, follow this link.

Most other systems can be used. There are a few caveats that must be met including the ability to #include other files for the SIP and extensions configurations as well as making users as the location of the phone and using typical naming conventions and not making a user as an extension (as most systems will do by default).

## Section 2: Configuration for the Asterisk Server

### SSH Connection

It is required for the Asterisk Server to allow SSH connections, although security will not be put at risk. The SSH connection is for the transfer of files (using Secure CoPy) to and from the

NOTE: Root privileges are not required and are urged against. A user can be created with only read and write permissions to specific directories for the Aventura Module.

Aventura Server.

Software such as fail2ban can be used with the Asterisk server (although under normal circumstances, there would not be issues, it is recommended that the ip address of the Aventura Server be an exclusion because of the retry nature of the module).

OpenSSH is the most common SSH Server Daemon for Linux systems and is available for most distros. Check your distribution's documentation for installation and set-up instructions.

### Configuration Files

There are two sets of files that are needed both by Asterisk and the Aventura Module. The first of these is the location users. These are the configuration files for the users (as the room) that the phone will register to Asterisk as. The recommended set up is to have a domain config Sample Domain Configuration File with the static information and then each of the locations as

NOTE: The domain file can be named anything although it is recommend to be 0.conf or start with an underscore as it needs to be loaded first into the Asterisk Configuration. Another option is to #include it separately in the sip.conf first before including the entire directory.

NOTE: To reduce the number of files. All the location users can be declared in one file.

users config Location User Sample Configuration File with the user information. The sip.conf file will then need to have an inclusion to the directory <Sample>.

The second set of files is the person's extension dial plan files. These files outline the dial plan of the user's extension. These files must be one per user and a sample can be found Sample

NOTE: The person-user config files must be matched to the users Active Directory ID.

Extension Configuration for One User.

All of these files (after the initial set up) should never be changed. They are automatically changed by the VoIP Module and any non-automated changes can break the system.

## Directories

The Aventura VoIP Module requires two directories to be used only by Aventura. The first directory is for the location user config files, this is known as the 'locations directory'. The user definitions for the locations of the phones' users go here. The domain file (if included) and all of the user definitions will go in this directory.

The second directory is for the user's extension files. This directory is known as the extensions directory. The files that have the dial plan routing for the user's extensions go in this directory.

Both directories, and all of the files within, must be owned by the set up SSH user for the Aventura Module. Permissions of the files will be preserved if they are more premissive than rw_rw_r__ and need to be set to at least ___rw____ for the module to work. Again, the files in these directories (after the initial set up) should never be changed. They are automatically changed by the VoIP Module and any changes can break the system.

### FTP Server (Optional)

For hard phones that are configured using provisioning, they need to have a boot server. The recommended setup is to have the boot server on the Asterisk server. You can make a new user and use a daemon like VSFTPD to provide the ftp service.

NOTE: Set the root to be a folder outside of the root of the Linux System. Users with only read and write permissions to that directory are sufficient.

## Section 3: Integrating Asterisk with Aventura

The Aventura VoIP Module is designed to provide roaming phones to go along with the roaming desktops. There are a few required items for the VoIP Module to work correctly. There are two sides, the Asterisk Server and the Aventura Server.

### Asterisk Requirements

The Asterisk server must have SSH enabled and a user for the VoIP Module to use. The configuration files must be created with the correct syntax and format and must be in the appropriate directories. The directories and the files within must be accessible to the SSH user. When a change is made by the VoIP Module, both the dial plan and the SIP configuration must be reload to the Asterisk Server. A script to do this automatically is provided <here>. One script must be used for each of the directories and each of them needs to be run as root.

### Aventura Requirements

The information for the SSH is required. In the database, ConfigStringEntries must be made with titles of VoIP_Host, VoIP_User, VoIP_Password, VoIP_Locations_Directory, and VoIP_Extensions_Directory with each of their respective values as well as the phone to location mapping. The mapping takes the form of VoIP_Phone_ {LocationName} as the title and the user of that phone as the value. Sample entries of the database can be found <here>.

If the above has been installed and configured correctly, the Aventura VoIP Agent, when connected to Session Manager will provide phone roaming with the Aventura System.

## Section 4: Polycom Phones

Our setup uses Polycom phones and they can be non-trivial to set up with Asterisk. There are multiple configuration files that the phones require. These all need to be in the ftp root of the boot server. There is an XXXXXXXXXXXX.cfg file that corresponds to the hardware MAC address of the phone (the letters need to be lowercase to work correctly). In that file it specifies the username that the phone is to use to register to the Asterisk Server. This is the same username as the Location username talked about in the docs. There is then a {username}.cfg file that the phone also pulls. There is then a directory that contains language information and a default sound, .wav, file. Then you need three sip files, sip.cfg, sip.ver, and a binary sip.ld. If using different model phones or different versions of firmware, these need to have unique names and need to be noted in the MAC address config file. The phone will then write two sets of log files, one for the boot information and then one for apps. A sample of the FTP root directory can be found in Section 5: Sample Configuration Files

## Section 5: Sample Configuration Files

The following are sample configuration files pulled from our working setup of the Aventura VoIP module. Unless otherwise noted, the configuration files are from the Asterisk Server.

### Sample Domain Configuration File

```
[aventura](!)
type=friend
context=from-internal
host=dynamic
qualify=yes
nat=no
```

**Sample 1 - domain.conf**

### Location User Sample Configuration File

```
;This file is automatically created and edited by Aventura VoIP Module.
;Do not make any changes to this file, as they will be overwritten
;and can cause system failure.
;This is the config file for stephen_surface

[stephen_surface](aventura)
secret=password1234
canreinvite=no
callerid=AVHQ\skennicutt
```

**Sample 2 - stephen_surface.conf**

## SIP Configuration Sample

```
[basic-options](!)
        dtmfmode=rfc2833
        context=from-internal
        type=friend

[public-phone](!,basic-options)
        directmedia=yes

[my-codecs](!)
        disallow=all
        allow=ilbc
        allow=g729
        allow=gsm
        allow=g723
        allow=ulaw

#include "/locations/domain.conf"
#include "/locations/*.conf"
```

Sample 3 - sip.conf

## Sample Extension Configuration for One User

```
;This file is automatically created and edited by Aventura VoIP Module.
;Do not make any changes to this file, as they will be overwritten
;and can cause system failure.
;This is the config file for AVHQ\tthorn

;[Default]
exten => 2000,1,Dial(SIP/tyler_desk)

;[CurrentLocation]
;exten => 2000,1,Dial(SIP/tyler_surface,15)

;[CellPhone]
;exten => 2000,2,Dial(SIP/TRUNK ID7202433010,15)

;[Voicemail]
exten => 2000,2,Voicemail(1000)
```

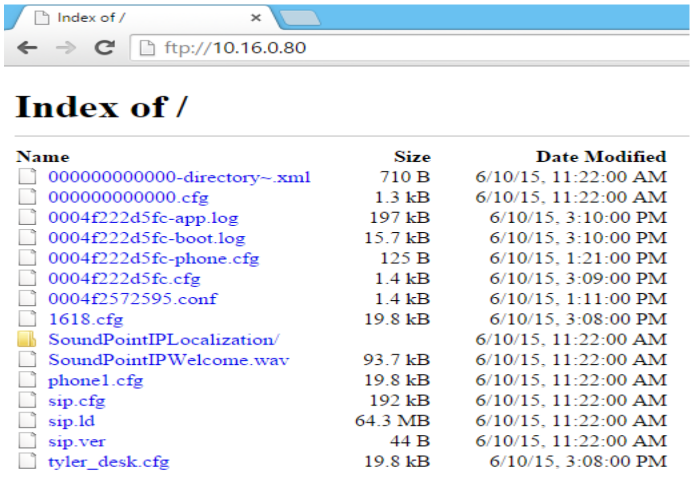
Sample 4 - 186C22DE-60DF-4FA1-9A22-20E1D199971E.conf

## Sample Database Table

| ID | Name | Parsing Expression | Config String | Owner ID |
|---|---|---|---|---|
| 0CF9A064-84B4-4857-B28D-9B4375A418C3 | PINFilter | NULL | .{4,} | NULL |
| 22E5BEF7-589D-44FE-9A88-F32B1E6CE6CD | DeviceInactivityTimeout | NULL | 60 | NULL |
| 25553A3D-DACE-42A2-BC79-651D83829E49 | VoIP_Locations_Directory | NULL | /locations/ | NULL |
| 7ED0548-7118-4092-845F-4BC8B5EEB654 | VoIP_Extensions_Directory | NULL | /extensions/ | NULL |
| 61ED4BE6-710D-4EA8-A6BF-A0AAADA8A0AB | DefaultDomain | NULL | avhq.local | NULL |
| 78CF572D-4911-430F-819D-568B221B402F | VoIP_Host | NULL | 10.16.0.80 | NULL |
| 80812BE0-652B-441A-AA38-6E574B1D9D71 | VoIP_User | NULL | tyler | NULL |
| 80861F01-F8FB-4F0A-9D3C-CED72D1CEF2E | VoIP_Password | NULL | aventura | NULL |
| 9470DFA3-7151-48E5-8D27-E0BCC2099108 | isStepDownAuthEnabled | NULL | false | NULL |
| BB83B5EE-3B36-4489-AD53-F04E1B303A4D | isAlternateAccessAlwaysAvailable | NULL | false | NULL |
| CF345002-C96E-40EF-9D2B-1DABF04C66FC | VoIP_Phone_1 | NULL | tyler_surface | NULL |
| E34AE8A7-660C-44AA-A969-31DD3F941CE9 | keyboardAccessDurationHours | NULL | 24 | NULL |
| EB5EA8FD-D0B9-4794-B1FF-B114B90E33B5 | isPinEnabled | NULL | false | NULL |
| FFE34C09-8FE9-4372-8B23-8B2CB6D4CDC6 | VoIP_Phone_2 | NULL | stephen_surface | NULL |

Sample 5 - configString.data

## Sample FTP Root Directory



Index of /

| Name | Size | Date Modified |
|---|---|---|
| 000000000000-directory~.xml | 710 B | 6/10/15, 11:22:00 AM |
| 000000000000.cfg | 1.3 kB | 6/10/15, 11:22:00 AM |
| 0004f222d5fc-app.log | 197 kB | 6/10/15, 3:10:00 PM |
| 0004f222d5fc-boot.log | 15.7 kB | 6/10/15, 3:10:00 PM |
| 0004f222d5fc-phone.cfg | 125 B | 6/10/15, 1:21:00 PM |
| 0004f222d5fc.cfg | 1.4 kB | 6/10/15, 3:09:00 PM |
| 0004f2572595.conf | 1.4 kB | 6/10/15, 1:11:00 PM |
| 1618.cfg | 19.8 kB | 6/10/15, 3:08:00 PM |
| SoundPointIPLocalization/ | | 6/10/15, 11:22:00 AM |
| SoundPointIPWelcome.wav | 93.7 kB | 6/10/15, 11:22:00 AM |
| phone1.cfg | 19.8 kB | 6/10/15, 11:22:00 AM |
| sip.cfg | 192 kB | 6/10/15, 11:22:00 AM |
| sip.ld | 64.3 MB | 6/10/15, 11:22:00 AM |
| sip.ver | 44 B | 6/10/15, 11:22:00 AM |
| tyler_desk.cfg | 19.8 kB | 6/10/15, 3:08:00 PM |

Sample 6 - FPT Root

## Section 6: Scripts

### File Change Watcher

```
#!/bin/sh

## Set initial time of file
LTIME=`stat -c %Z ./*`
while true
do
       ATIME=`stat -c %Z ./*`

       if [[ "$ATIME" != "$LTIME" ]]
       then
              echo "Dialplan reloading"
              SYSTIME=$(date)
              echo "Dialplan reloaded at "$SYSTIME" ,check
                     /var/log/asterisk/messages for details." >>
                     ~/logs/listenerlog.txt
              sudo asterisk -vvvvvvrx "dialplan reload" >>
                     ~/logs/loadinglog.txt
              LTIME=$ATIME
       fi
       sleep 1
done
```

Script 1 - filemod.sh

## Appendix C – Parser Library API

*Parse Library API*

### Parser object (uses parse namespace)

Public class methods

Passes in a predefined filestream (must open file outside of the class)
```
void set_file(std::fstream*)
```

Returns a pointer to the list of entries parsed from the config file. Can use methods
```
std::vector<Entry>* get_entries()
```

Primary function call to parse the config file.
```
int parse()
```

Returns the appropriate error string
```
std::string get_error()
```

### Entry object

Public class methods

Constructor (class doesn’t have a default constructor)
```
Entry(int ext_num, std::string ext_str, std::string title, bool is_active)
```

Class getter methods
```
bool isActive()
int get_ext_number()
std::string get_ext_str()
std::string get_title()
```

Converts attributes to one string.
```
std::string to_s()
```

## Appendix D – Secure File Transfer Library

### *Secure File Transfer API*

```
namespace
namespace sft

Return Values
SUCCESS 0
UNKNOWN_HOST -1
AUTH_FAIL -2
READ_FAIL -3
WRITE_FAIL -4

Constructor
FileTransfer();

Methods for creating and connecting the tunnel
int returns are error codes.
void  set_host(std::string);
void  set_user(std::string);
void  set_password(std::string);
int   open_connection();
void  close_connection();

Methods for reading file, the full path to the file and to the full
name of file
int returns are error codes.
int   read_file(std::string);

Methods for writing file, first two are always path to file and name
of file
int returns are error codes.
int   write_file(std::string, std::string, std::string);
int   write_file(std::string, std::string, const char*, size_t);
int   write_file(std::string, std::string, std::fstream&);

Method for error reporting, sets to most recent error if one of
previous calls returns something other than SUCCESS
std::string get_error();

Retrieve the file, mostly for testing purposes, the file is written to
the location of the executing program. This will return a string of
the file.
std::string get_file();
```