

*\* &#!perl*

Using Perl with Asterisk 1.0

# What is Asterisk?

- An open source PBX / IVR in software
- Supports major VoIP protocols
- Interoperates with standards-based telephony eq.
- Connects disparate technologies together
- Runs on Linux using commodity x86 hardware
- Bridging the gap between Telco and IT

# Major Components

## Media Channels

- SIP, H.323, Skinny, IAX, etc.

## Media Channel Configuration Files

- Contains all channel specific options
- Defines source and destination for media
- Defines real names for use in dialplan
- Declares default dialplan context for channel

# SIP Media Channel

```
[simon]           ; Dialplan channel name
context=local-phones ; Dialplan Context (for outbound)
secret=abc123      ; Password
type=friend        ; Will send and recv. calls
host=dynamic       ; Has dynamic IP
dtmfmode=info      ; Touchtone digits sent oob
disallow=all       ;
allow=ulaw          ; ulaw audio codec is okay
canreinvite=no     ; audio must pass thru server
```

# Major Components

## Dialplan

- Stored in `extensions.conf`
- Comprised of labels, referred to as a "context"
- Contexts contain a list of extension definitions
- Each extension specifies a priority, and a command to run; similar to BASIC

# Simple Dialplan

[context\_name]

exten => <extension>,<priority>,<application>(options)

exten => <extension>,<priority>,<application>(options)

exten => <extension>,<priority>,<application>(options)

[local-phones]

exten => s,1,Wait(1)

exten => s,2,Answer

exten => s,3,Background(hello\_please\_dial)

exten => s,4,WaitExten()

exten => s,5,Goto(local-phones,s,3)

exten => 123,1,Dial(SIP/simon)

exten => 123,2,Voicemail(u123)

# Similar to Basic

```
<priority> <command> <arguments>  
<priority> <command> <arguments>  
<priority> <command> <arguments>
```

```
10 PRINT "Enter a number, zero to stop:";  
20 INPUT A  
30 IF A = 0 THEN GOTO 70  
40 LET A = A + 10  
50 PRINT "The number plus 10 is "; A  
60 GOTO 1  
70 STOP
```

# Adding some Magick

[example]

```
exten => <extension>,1,<application>(options)
exten => <extension>,n,<application>(options)
exten => <extension>,n,<application>(options)
```

[local-phones]

```
exten => s,1,Wait(1)
exten => s,2,Answer
exten => s,3,AGI(say-time.agi)
exten => s,4,Background(hello_please_dial)
exten => s,5,WaitExten()
exten => s,6,Goto(local-phones,s,3)

exten => 123,1,Dial(SIP/simon)
exten => 123,2,Voicemail(u123)
```

# A simple AGI

```
#!/usr/bin/perl -w
```

```
use Asterisk::AGI;  
$AGI = new Asterisk::AGI;
```

```
my %input = $AGI->ReadParse();
```

```
my ($min,$hour) = (localtime(time))[1,2];  
$hour < 12 ? my $mfactor = "a-m" : my $mfactor = "p-m";  
$hour %= 12;
```

```
$AGI->say_number( $hour );  
$AGI->say_number( $min );  
$AGI->say_number( $mfactor );
```

# AGI Methods

## Channel Control

```
$AGI->answer()  
$AGI->hangup($channel)
```

## Communicating with the User

```
$AGI->say_number($number);  
$AGI->say_number($number, $escape_digits)  
$AGI->say_digits($number, $escape_digits)  
$AGI->stream_file($filename, $received_digit)  
$AGI->get_data($filename, $timeout, $maxdigits)
```

# AGI Methods

## Dialplan Interaction

```
$AGI->exec($app, $options)  
$AGI->set_context($context)  
$AGI->set_extension($extension)  
$AGI->set_priority($priority)  
$AGI->set_callerid($number)
```

## Getting/Setting data in the Dialplan

```
$AGI->set_variable($variable, $value)  
$AGI->get_variable($variable)
```

# AGI Gotcha!

- NEVER include newlines in variables passed to AGI calls
- The AGI pipe uses newline to terminate commands
- An empty line apparently means wait indefinitely

# An Interactive AGI

```
#!/usr/bin/perl -w  
[...]
```

```
# <filename> <timeout> <maxdigit>  
$digits = $AGI->get_data( "please-enter", 3000, 20 );
```

```
# <audio filename> <digits pressed>  
$AGI->stream_file( "you-entered", undef );
```

```
# <digits strung together by asterisk>  
$AGI->say_digits( $digits );
```

# Passing Data to Perl

In extensions.conf ...

```
[default]
exten => s,1,Answer
exten => s,2,AGI(readinput.agi|arg1|arg2)
exten => s,3,Hangup
```

In readinput.agi...

```
[...]
my( $arg1, $arg2 ) = @ARGV;
[...]
```

# Returning Data from Perl

auth.agi

```
#!/usr/bin/perl -w  
[...]
```

```
my( $callerid, $password ) = @ARGV;  
if( &validate( $callerid, $password ) ) {  
    $AGI->set_variable('authed', '1')  
} else {  
    $AGI->set_variable('authed', '0')  
}
```

# Returning Data from Perl

```
extension.conf
```

```
[default]
```

```
exten => s,1,Answer()
```

```
exten => s,2,Read(password|please-enter-password)
```

```
exten => s,3,AGI(auth.agi|${CALLERID}|${password})
```

```
exten => s,4,GotoIf($[${authed} = 1]?pass,1:fail:1)
```

```
exten => pass,1,Playback(private-message)
```

```
exten => pass,2,Goto(another-context,s,1)
```

```
exten => fail,1,Playback(invalid-password)
```

```
exten => fail,2,Goto(s,2)
```

# Best Practices

Handle call logic using dialplan whenever possible

- Simple, call logic is not hidden
- It's fast
- Uses fewer resources
- More stable

# Best Practices

Return from AGI's as quickly as possible

- People are impatient
- If someone hangs up in middle of AGI script, `$SIG{HUP}` will be triggered, and should be handled
- Otherwise risk crashing Asterisk
- Crash will result in all calls being dropped, incomplete CDR's, and other bad stuff

# Handling \$SIG{HUP}

```
#!/usr/bin/perl -w

use Asterisk::AGI;
my $AGI = new Asterisk::AGI;
$SIG{HUP} = \&handle_hup

[...]

sub handle_hup {
    $AGI->Hangup();
    exit(0);
}
```

# Fun AGI Hack

```
[incoming]  
exten => s,1,AGI(forecast.agi)  
exten => s,2,Hangup()
```

```
#!/usr/bin/perl -w
```

```
use Asterisk::AGI;  
use Fake::Weather;
```

```
my $AGI = new Asterisk::AGI;  
my %input = $AGI->ReadParse();  
my $weather = new Fake::Weather( 'yyz' );  
my $forecast = $weather->forecast();
```

```
# Asterisk Festival command needs text in quotes  
$AGI->exec('Festival', "\">$forecast\"");
```

# IVR - More on Dialplans

Asterisk has a set of default extensions that should be handled in a dialplan, particular in IVR's

- Start of context
- Invalid extension entered
- Timed out waiting for input
- Caller hung up
- Absolute call length reached

# IVR - More on Dialplans

```
[main]
```

```
; start of context
```

```
exten => s,1,SetVar(callerid=${CALLERID})
```

```
exten => s,2,Playback(welcome)
```

```
exten => s,3,WaitExten(5)
```

```
; invalid extension entered
```

```
exten => i,1,Playback(invalid-option)
```

```
exten => i,2,Goto(s,2)
```

```
; caller option timeout
```

```
exten => t,1,Playback(you-are-slow)
```

```
exten => t,2,Goto(s,2)
```

# IVR - More on Dialplans

```
:[main] continued...
```

```
; caller hung up
```

```
; ensure that we've hung up too!
```

```
exten => h,1,DeadAGI(tidyup_user.agi|${accountid})
```

```
exten => h,2,Hangup()
```

```
; absolute timeout
```

```
exten => T,1,Playback(sorry-time-expired)
```

```
exten => T,2,Hangup()
```

# DeadAGI

Identical to AGI, except that:

- It continues running after caller has hung up
- Only variables set using SetVar (dialplan), and `$AGI->set_variable()` are available
- Variables set by asterisk not available; i.e. `$_CALLERID`, `$_CHANNEL`, `$_CONTEXT`

# Scaleability

Problems with the CGI model similar to CGI

- Every request requires own perl interpreter
- Script startup time is high for realtime applications
- Data persistence across invocations

# Scaleability Solutions

- Embed Perl in Asterisk with res\_perl
- Maps Asterisk C API to Perl

```
exten => 1,1,Perl(some_func:arg1:arg2)
```

- Connect to an AGI server over the network using FastAGI

```
exten => 1,1,AGI(agi://192.168.1.1)
```

# res\_perl Example

- Can call perl functions from special package named Asterisk::Embed
- Early stages of development, docs recommend reading the C files

Calling res\_perl from the dialplan:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Read(password|please-enter-password)
exten => s,3,Perl(auth_func|${CALLERID}|${password})
exten => s,4,GotoIf($[${authed} = 1]?pass,1:fail:1)
```

# res\_perl Example

```
# insert into asterisk_init.pm
```

```
package Asterisk::Embed;
```

```
sub auth_func {
```

```
    my $chan_name = shift;
```

```
    my ( $callerid, $password ) = @_;
```

```
    my $chan = asterisk_get_channel_by_name($chan_name);
```

```
    if( &validate( $callerid, $password ) {
```

```
        asterisk_setvariable( $chan, 'authed', 1);
```

```
    } else {
```

```
        asterisk_setvariable( $chan, 'authed', 0);
```

```
    }
```

```
}
```

# FastAGI Example

- Expects target to be listening on port 4573
- Connects, and waits for AGI commands
- Returns response code '200' on success
- Vaguely follows HTTP response codes

```
[incoming]
```

```
exten => s,1,Answer()
```

```
exten => s,2,Read(password|please-enter-password)
```

```
exten => s,3,AGI(agi://192.168.1.1)
```

```
exten => s,4,GotoIf($[${authed} = 1]?pass,1:fail:1)
```

# FastAGI Example

```
package Demo::FastAGI;

use Net::Server;
use Asterisk::AGI;
@ISA = qw(Net::Server);

sub process_request {
    use vars qw(%input $AGI);
    STDIN->autoflush(1);
    $AGI = new Asterisk::AGI;
    $AGI->setcallback(\&mycallback);
    %input = $AGI->ReadParse();
    my $callerid = $AGI->get_variable('CALLERID');
    my $password = $AGI->get_variable('password');

    if( &validate( $callerid, $password ) ) {
        $AGI->set_variable('authed', '1')
    } else {
        $AGI->set_variable('authed', '0')
    }
}
```

Credit: <http://www.miselconsulting.com/?page=fastagi>

# FastAGI Example

```
sub mycallback {  
  my ($returncode) = @_;  
  &log("CALLBACK");  
  &cleanup;  
  print STDERR "User Disconnected ($returncode)\n";  
  exit($returncode);  
}
```

```
Demo::FastAGI->run(port=>4577, user=>'nobody', group=>'nobody',  
  background=>1);
```

# Local User Groups & Resources

Ontario Asterisk and VoIP Enthusiasts

<http://uc.org/asterisk>

asterisk-subscribe@uc.org

Toronto Asterisk Users Group

<http://taug.ca>

Monthly meeting is tomorrow, see website

VoIP Resources Wiki

<http://www.voip-info.org>

O'Reilly Books

<http://www.oreilly.com/catalog/asterisk/>

<http://www.oreilly.com/catalog/switchingvoip/>

# IVR Applications, a scenario

Interactive Stories Inc.

- Caller is authenticated
- Account details are retrieved, and read back
- Caller is put into interactive story loop
- Disconnects and timeouts should be handled gracefully