**ChatScript App Client Manual**
**© Bruce Wilcox, gowilcox@gmail.com**
**Revision 2-11-2014 cs4.03**

Building a web-based interface to ChatScript is different from building an app involving ChatScript. In the web-based version, the ChatScript engine will always be a server on a reasonably powerful machine and have available the full resources of the web.
In building an app-based NLP application with ChatScript, there are two general architectures, client-based and server-based.

## Client-based

In a client-based architecture the client is not required to communicate with a server during use. ChatScript is installed locally on the client with all needed data. If voice-to-text exists, it must be performed locally. The actual client app controls the screen and inputs and when it gets a response from the user (text or voice-translated-to-text) it calls a ChatScript subroutine which returns a text output which may embed commands to be obeyed by the app (gestures, device functions, etc).

The extra considerations in a client-based app are three-fold: memory size, logging, and upgrading.

Things like iOS devices may restrict your app to somewhere in the 20MB size, and if you go beyond that you risk having your app killed by the OS at any moment. In that context, you cannot afford a full ChatScript dictionary and need a “miniaturized” one that occupies maybe 1/3 the normal space. A large ChatScript memory footprint including the script for the bot is around 15-18MB. The miniaturized dictionary is not something you can easily build yourself and must be created by me (normally for a fee).

Log files are generated locally in the device and must somehow migrate to a server if you want analytics. Typically this is done when the user launches the app and if an internet connection exists and it has been long enough since the last upload, the log files are uploaded to a server and cleared from the client.

Upgrade issues involve the desire to fix or augment ChatScript content without requiring the user download a new edition of the app. That is, on startup the app would check with the server and if specific files (eg the entire TOPIC folder) have changed, it downloads those changed files. It either then initializes ChatScript (the changes take place immediately) or if it has already initialized ChatScript the changes will take place starting with the next user startup of the app.

## Server-based

The other app configuration always requires an internet connection and communicates with a ChatScript server over it. In addition, this means it can easily to voice-to-text over

it and text-to-voice back.  All log files are already on the server so complete access to all conversations is conveniently available and all ChatScript data files are also easy to update as desired.  Of course the requirement of having an internet connection available is a severe one on the client and may also mean his app is generating data charges he may not like.

You still have a client app which sends and receives the communications (voices and/or texts) and must still decode return text from ChatScript to retrieve app commands for avatar behavior and other functions.

If you are using voice-to-text on the server, then you would create a server app that receives inputs from the client, communicates with the voice-to-text server, then hands appropriate text over to the ChatScript server. The server app, on receiving the ChatScript output, may need to send it over to a text-to-voice conversion, and then ship all results back to the client. If you are not using voice in any capacity, the client app could communicate directly with the ChatScript server.