ircII Scripting

ircII (pronounced irk-two) is the second of the two scripting languages that Snak supports.   The language is used in several irc clients with minor variations which means that there is a large body of existing scripts that can be adapted or used directly.

Scripts made in this language can be used side by side with scripts written in AppleScript. A small ircII script is frequently the fastest way to create simple command aliases.

Some very large and complex ircII scripts are available, but they can be hard to understand. It helps to have a programming background and be familiar with a language like C if you want to make complicated scripts in ircII or modify existing ircII script packages.

One of the largest and most powerful such packages is called PurePak and is included with Snak. PurePak can be a rich source for tips and inspiration once you have grasped the basics.

The ircII language is rich, expressive and complex. It has many intricacies and while simple aliases and commands are easily within the grasp of a beginner, large programs and script packages should be written in AppleScript unless you really want to learn the ircII language.

There are a number of websites that contain scripts for the ircII client, and Snak is able to use most of those with only minor modifications.

The site http://www.irchelp.org contains scripts and also an extensive command reference and script guide that is highly recommended. Its location is:

http://www.irchelp.org/irchelp/ircii/irciiman.wri

It has chapters on the full syntax of the commands and the correct way to use them. Among the wealth of information in the guide is the full description of the various conditional operators like "if", branching commands like "while", and the full range of expressions that can be written in ircII. If you want to learn ircII scripting, the guide is highly recommended.

It is not possible to access data outside of Snak using the ircII language because it is processed entirely inside the application. AppleScript can be used to tie together multiple applications and access outside data because it is processed by the operating system in cooperation with the application.

Some examples

Snak comes with a number of script files from the regular distribution of ircII, and you are encouraged to open them in a text editor like SimpleText to see what you can do.

You will notice the useful /oops alias in the file called basical which uses an user defined variable.

In the file action you will find an example of the "if" statement. This example sets a string containing the possessive (his/her/the) depending on a gender variable.

That gender variable is defined in the file action, so if you are a woman, you may want to change the gender to "F" in order to get the correct possesive.

For more elaborate examples, look in the script files that came with the program.

The /J <channel> alias

A simple example is the /j alias. It is simply used as an abbreviation for the normal /join command. If you look in the script file basical you'll see that alias j is defined as /join.   /join is a built in command and can be used on the input line with a parameter, and so can /j.

When you type /j, the program will replace the alias with its definition, which in this case is /join

The /op <nick> alias

A slightly more elaborate example is /op. In order to convey operator status on someone you normally have to type

```
/mode <channelname> +o <nick>
```

However the /op alias is defined as "/mode $C +o" and can be used to simplify this.

You can now convey op status by typing "/op <nick>". The alias makes use of a built-in scripting variable $C, and when the programs finds a variable in an alias the variable will be replaced by its value before executing the command.

Later in this chapter you will find a full list of the available variables and see what they represent, and $C   contains the name of the current channel.

The <nick> part of the /mode command shown above is supplied by the <nick> parameter you typed after /op .

The /oops <nick> alias

The alias /oops is is used to correct a message that went to, let's say John when it should have gone to Mary. The syntax is simply "/oops <intended nick>"

/oops is defined as

```
alias oops

^assign alias.oops $B

msg $. Sorry, that wasn't meant for you.

msg $0 $alias.oops
```

Two new variables appear here : $B which always contain the text of your last message, and $. which is the last nick you messaged. The ^ character makes the assignment "silent", meaning no message is output when the line is processed.

The first line copies the text of the last message to an intermediate variable called "alias.oops".
The second line sends a message to the last nick we messaged (John) to say "Sorry, that wasn't meant for you"."
Third line resends the original text (stored in the intermediate variable) to the nick that was the first parameter to the alias (Mary)

In addition to variables like $C and $B there is also $0 ($zero) to $9 ($nine). Numerical cariables like that will be replaced by the corresponding argument from the input line. $0 will be replaced by the first argument, $1 will be replaced by the second argument and so on.

The consoleClick script

The aliases can be run from the input field, and you can also run a particular script when you double click on a user in either the user list or the notify list.

The default command for a double click in the notify list is "ConsoleClick". ConsoleClick is an alias, which is defined as "/msg $E $0-".

This looks cryptic, but remember that a /msg always takes two pieces of input : the nick and the message. When Snak interprets the alias it replaces $E with the currently selected nick, and $0- with the entire contents of the input line.

The Possessive script variable

In the file action you can see an example of the how to affect the execution of the alias depending on outside values. As mentioned before, scripting is really a form of programming, and a programming languages have conditional statements.

This example demonstrates the "if" statement which takes one of two branches depending on a test. The syntax for this statement is

```
IF (<variable-expression>) <true-command/s>

[<false-command/s>]
```

and the example in the file is

```
if (GENDER)




if ([$GENDER] == [F])








assign POSSESSIVE her
assign POSSESSIVE his
```

```
assign POSSESSIVE the
```

This is actually two "if" statements inside each other. The outermost one test if the variable GENDER is defined at all. If not then it assigns "the" to the variable POSSESSIVE.

If GENDER is defined then the program runs the innermost "if" which tests its value, and assigns "her" to POSSESSIVE if it is "F". Otherwise POSSESSIVE will be set to "his"

Scripting Variables

Snak supports the numerical variables $0 thru $9
The numerical variables can be used individually or in ranges:
$n- gives argument n through the last argument
$n-m gives arguments n through m
$-m gives the first argument through m
$* gives the entire contents of the input line. Same as $0-

$.
 Nick of the last person to whom you sent a message
$,
Nick of the last person who sent you a message
$:
   Nick of the last person who joined the channel
$;
   Nick of the last person who sent a message to the channel

$?   Brings up a dialog where you can enter text.

Syntax is $?="explanation"

$B
 Text of the last message you sent
$C   The name of the channel
$E
 Nick of the first selected user in the userlist
$T   Name of the current channel or query window.
$F
User and host information about the first selected user in the userlist

$I
Name of the channel you were last invited to join
$J The text of the last private message you received.
$N
Your nick
$K returns the character (/) that is used to make the program process some input as a command

$J
The text of the last private message you received. Used in the Respond contextual menu item to pass the text into the query window.

The unsupported language features are:
Most event handlers are supported, except for handlers relating to DCC which are not yet implemented.

Keywords:
/Bind

Built-in functions:
$winnum, $winnam, $connect, $listen, $curpos,$pid, $ppid, $$,, $Q, $R, $S, $U, $V, $W

I prioritize the implementation of these features according to the needs of the scripts that people want to run. Please let me know which language features you need most for your favorite scripts to work.