mIRC

Scripting

Documentos recopilados por Ferran Sarrió

INDICE

INTRODUCCIÓN

El mIRC es sin duda el programa cliente para IRC más extendido en la actualidad, y no solo por su distribución libre y gratuita, puesto que hay otros muchos que también cuentan con esta ventaja, ni por estar creado para Windows, el sistema operativo más extendido actualmente, puesto que no es ni el mejor sistema ni el más económico; los factores más decisivos son, bajo muestro punto de vista, la gran capacidad del programa para poderlo adaptar a los gustos y necesidades de cada usuario mediante rutinas de script, lo que hace de él una herramienta casi a la medida, así como la gran cantidad de estos scripts, para todos los gustos y necesidades, que se encuentran hoy a disposición de todos los usuarios.

Derivado de los anteriores, otro factor fundamental para que mIRC sea el cliente de IRC más popular es la enorme cantidad de documentación sobre este programa que podemos encontrar en la red. En efecto, aunque mIRC cuenta con completo fichero de ayuda en formato .hlp este no es absolutamente completo ni agota todas sus posibilidades, además, tanto esta ayuda como la inmensa mayoría de la información disponible en Internet se encuentra en idioma inglés. La principal razón que podemos alegar para justificar la difusión del programa entre los usuarios de habla española no es otra más que la gran cantidad de scripts en esta lengua que han ido surgiendo en los últimos dos o tres años, gracias al trabajo meticuloso y desinteresado de docenas o centenares de creadores más o menos anónimos.

El objetivo de estas páginas es documentar en español las herramientas y técnicas necesarias para la creación de scripts para mIRC. De este modo, además de incrementar la escasa información de que se dispone en nuestro idioma, pretendemos poner estos conocimientos al alcance de los nuevos y viejos creadores, y de todos los usuarios que los necesiten, tanto para desarrollar y distribuir su propio script como para aplicarlos a la personalización de las propias sesiones en el IRC.

En general podemos decir que un script es una secuencia de instrucciones que un programa es capaz de seguir, interpretar, y ejecutar. Un script puede estar formado por una o más rutinas o grupos independientes de instrucciones. El mIRC dispone de un amplio entorno en el que podemos llevar a cabo la programación de rutinas de script, logrando así que el programa realice las funciones más diversas, permitiéndonos avanzar mucho más allá de las capacidades que este cliente de IRC implementa ya de 'serie'.

Prácticamente la totalidad de la configuración básica, tanto la que el programa realiza por sí mismo como la que nosotros especificamos en las distintas ventanas de diálogo, es depositada en ficheros de texto con extensión .ini en el mismo directorio que el fichero ejecutable (mirc16.exe ó mirc32.exe). Es posible por tanto acceder a todos esos parámetros de configuración y editarlos según nuestros deseos. Así mismo, todos los scripts que creemos para el mIRC, con todas sus rutinas e instrucciones, se depositan también en ficheros. La creación, modificación, y carga de estos ficheros se puede hacer directamente desde el propio mIRC en las distintas opciones del menú TOOLS.

La creación de scripts para mIRC potentes y útiles exige un buen conocimiento de los comandos generales de IRC, de los comandos propios del mIRC, y de las distintas áreas configurables de este programa. En este manual veremos el uso y utilidad de cada una de esas áreas configurables y como debe de trabajar con ellas; también tocaremos en capítulos aparte el uso de las herramientas imprescindibles para explotar todas las capacidades de este entorno.

Si usted parte de cero, o desea iniciar un proceso de aprendizaje ordenado es siempre recomendable comenzar estudiando detenidamente los comandos, estas ordenes le darán una buena idea inicial de lo que el programa puede hacer y serán los ladrillos con los que posteriormente deberá de construir su script. Es indispensable un buen nivel de conocimientos de órdenes y comandos antes de avanzar a las siguientes etapas.

El siguiente paso lógico es comenzar a crear Alias y Popups, aquí ya descubrirá como, con unos pocos comandos, usted puede lograr efectos espectaculares en su programa. Tanto los alias como los popups se crearán en ficheros de texto independientes y cargados en el mIRC (ver comando Load y Unload). La capacidad de estos ficheros es grande pero limitada, ahora bien, usted podrá crear una buena cantidad de ellos si lo necesita.

LLegados a este punto ya se habrá encontrado con lo Identificadores, pero es el momento de estudiarlos más a fondo y observar lo que le permiten hacer. sus Alias y Popups multiplicarán su capacidad y eficacia con un buen uso de estos Identificadores.

Si hasta ahora no se ha sentido tentado de estudiar el uso de Variables y de la estructura IF-ELSE es el momento de hacerlo. Si ya está familiarizado con algún tipo de programación los encontrará extremadamente simples, si no quizás le parezcan algo más complicados pero tómelo con calma, verá que una vez comprendidos son conceptos muy elementales y le permiten dar a sus rutinas una flexibilidad que antes no imaginaba.

Si su practica y estudio ha llegado hasta aquí solo le queda adentrarse en el área que mIRC denomina "Remotes", es lo que le queda para poder hacer un script a la altura de los mejores. Estudie con detenimiento los Eventos, los Números Raw y los Sucesos CTCP. Esta lectura y el análisis cuidadoso de otros scripts le introducirán ya como miembro de pleno derecho en el grupo de "chalados" que alguien ha denominado "scripters" :-).

Los capítulos de Ventanas Personalizadas y el Servidor DDE completan de momento los textos de este manual, recurra a estas técnicas solamente cuando su necesidad lo justifique plenamente. El uso de Ventanas Personalizadas es un recurso muy vistoso y que da al script un aspecto original, pero el abuso de ellas, empleandolas para fines superfluos o cuando el programa ya dispone de una forma cómoda de suministrar la misma información delata a los creadores principiantes y hace más engorroso el manejo de su script.

Debo de pedir disculpas por no tener aún disponibles alguno de los apartados del manual. Están en elaboración y espero poder mostrarlos en breve, como contrapartida y justificación solo puedo alegar que he dado prioridad a la actualización del material para adaptarlo a la versión 5.31 del mIRC, que ya aporta grandes cambios y novedades con respecto a los textos originales que escribí en principio.

No puedo terminar sin agradecer a todas las personas que se interesan por este trabajo, a los que con sus preguntas me animan a continuar con él no enseñando sino aprendiendo, a los que me han ayudado con sus conocimientos y experiencia y son los culpables de gran parte de lo que sé, y de modo muy especial a hPm, que me demuestra cada día lo que el interés y la afición pueden hacer en el mundo del scripting y que los maestros han de ser los primeros alumnos.

Documento escrito por SoMaTiC · www.ayuda-irc.net · sucubus@ctv.es

POPUPS

El mIRC permite también la creación de popups o menús personalizados; cada opción que diseñamos para estos menús ejecutará las órdenes, alias, comandos,...etc. que les introduzcamos, y podrá así mismo hacer uso de identificadores, variables, etc. Disponemos inicialmente de 5 menús que podemos diseñar a muestro gusto con las opciones que deseemos:

MENUBAR:
Es el tercer menú de la barra principal de menús que se encuentraen la parte superior de la pantalla del mIRC.

CHANNEL:
Es el menú emergente que aparece al pulsar con el botón derecho del ratón sobre la pantalla del canal.

NICKNAME LIST: Es el menú emergente de la lista de nicks del canal.

STATUS: Es el menú emergente de la ventana de Status del programa.

QUERY/CHAT: Es el menú emergente de la ventana de querys y de DCC chat.

Accederemos al código que genera los popups desde el menú TOOLS opción "popups". En la ventana que nos aparece tenemos a su vez una barra superior de opciones, si pulsamos "view" podremos elegir, de entre los 5 anteriores aquel popup del que queremos ver las "interioridades" ;-). Ahí podremos crearlos, modificarlos o eliminarlos. Terminado el trabajo pulsaremos OK, y desde ese momento ya podremos hacer uso de las modificaciones introducidas. La mejor forma de comprender lo popups es observando el código de los ya existentes, cada línea de este código suele corresponder con una opción del menú, veamos un ejemplo:

```
Canales
.Entrar a #ayuda_irc: /join #ayuda_irc
.Entrar a #programación: /join #programacion
.-
.Elegir canal: /join #$$?="Introduce canal"
.Cambiar modos de canal.
.Solo op’s cambian topic: /mode # +t.
.No mensajes externos: /mode # +n.
.Cambiar el Topic: /topic # $$?="Introduce nuevo topic"
Nicks: /nick $$?="Introduce nuevo nick"
```

Como se puede observar cada línea describe en primer lugar el nombre de la opción de menú con la que aparecerá cuando se despliegue este. Como es sabido cada opción de menú puede tener a su vez otras subopciones, y estas a su vez otras tantas, y así sucesivamente; esto se especifica en el diseño del menú mediante los puntos que preceden a cada una de las líneas: las líneas sin puntos delante son opciones principales que aparecerán nada más desplegarse el menú. Si tienen un punto delante significará que son subopciones de las anteriores, y se desplegarán al pulsar sobre aquella inmediatamente anterior que no tenga punto; si tienen dos puntos serán una subopción de la inmediatamente anterior que solo tuviese uno... y así sucesivamente.
En virtud de lo anterior una línea de opción de un popup deberá de tener su nombre (el que queramos) seguido del símbolo ":" y la acción o comando que se ejecutará al hacer click sobre esta opción.

Si solo está el nombre, no hay dos pontos y luego un comando, es porque al seleccionarla se desplegará un submenú, así pues deberá de estar seguida de otras líneas de opciones de rango inferior , es decir, con más puntos que ella por delante, que serán las que se desplieguen al seleccionarla.

Opcionalmente podemos poner tan solo un guión en lugar del nombre de una opción, esto lo único que hará será insertar una línea de separación en la lista de opciones del menú.

Así pues, en el ejemplo anterior, al desplegar el popup observaremos tan solo dos opciones: Canales y Nicks. Si pulsamos la segunda podremos cambiar nuestro nick introduciendo uno nuevo en la ventana que se nos presentará; en cambio, al pulsar la primera se desplegarán a su vez otras cuatro opciones, separadas las dos primeras, por una línea horizontal, de las dos últimas. De estas cuatro las tres primeras ejecutarán ya una acción cada una, sin embargo la cuarta desplegará otras tres nuevas opciones.

Este razonamiento quizás pueda parecer complejo en principio, pero con un mínimo de práctica y observación descubrirá que domina las sutilezas de los popups mucho antes que cualquiera de las otras áreas de creación de script para el mIRC. Todo lo dicho para la definición de alias es también válido para construir la definición de opciones de menús, incluido el uso de identificadores.

Algunos identificadores (palabras especiales con el símbolo $ delante) cobran un valor especial cuando se usan en un determinado popup; por ejemplo, el identificador posicional $1, cuando se usa en una opción del popup de QUERY/CHAT, se cargará automáticamente con el nick del la persona con la que tenemos abierto el query o DCC Chat; y en el menú NICKNAME LIST con el primer nick que tengamos seleccionado en la lista.

Recuerde, para dominar los popups lo mejor es un mínimo de práctica y observar otros que ya están hechos, el mIRC dispone de algunos, y además son muchos los scripts en circulación que nos proporcionarán excelentes ejemplos de la creación y ordenación de esta excelente herramienta.

Documento escrito por SoMaTiC · www.ayuda-irc.net · sucubus@ctv.es

ALIASES

El mIRC te permite crear aliases y scripts para hacer más rápido tu sesión IRC o para hacer más fáciles y rapidas las funciones del script. Para crear los aliases debes conocer los comandos básicos del mIRC.

Un aliases puede ser llamado desde una linia de comandos, desde otro alias, desde un popup y un remote script. Un alias no se puede llamar a si mismo principalmente porque esto parece causar más problemas para los usuarios de los que soluciona. No hay nada mejor para entenderlo que algunos ejemplos:

```
/hola /msg $chan Hola gente, que tal os va?
```

Si ahora tu escribes **/hola** en un canal, automaticamente es lo mismo que decir:

Hola gente, que tal os va?

NOTA: El simbolo de doble «**$$**» significa que el comando no se ejecuta si no se indican los parametros:

```
/j /join $1 o /j /join $$1
```

Ahora si escribes **/j #ayuda_para_novatos** es lo mismo que escribir **/join #ayuda_para_novatos**. El simbolo «**$1**» se refiere al primer parámetro de la linia que tu escribes. Si el simbolo fuera «**$3**» se referiria al 3er parámetro que has escrito y asi con cada número.

```
/canal /join $?="En que canal quieres entrar?"
```

El parámetro «**$?**» sirve para que te pida información que tu debes escribir, para indicar que pregunta quieres hacer debes poner **="La pregunta a hacer"**. Y si quieres que vuelva a aparecer el texto que ha puesto en la ventana esa, debes poner «**$!**», mira este ejemplo, que asi te sera más fácil comprenderlo:

```
/canal /join $?="En que canal quieres entrar?" | /echo Ahora entraras en $!
```

NOTA: Este simbolo «**|**» sirve para separar diferentes comandos que esten en una linia.

```
/mejor /say Yo opino, que soy el ME $+ J $+ OR $+ !!!
```

Los parametros normalmente estan separados por espacios, pero para unir palabras o parametros puedes usar el simbolo «**$+**» dejando un espacio antes y despues de este simbolo, se debe poner entre las palabras o parametros que quieres unir.
En los aliases también puedes usar las llaves **{ }**, que se explica ocn detalle en el apartado Popups. También puedes usar los estamentos del **if-then-else**, el cual se explica en el apartado que lleva su nombre. Pero igualmente aqui te enseño un ejemplo para que no te queden dudas:

```
/ayuda {
if ( $chan == #Ayuda_para_novatos ) { /say Hola, necesito ayuda. }
else { /echo $chan Este canal no es de ayuda, o almenos eso creo :) } }
```

El comando GOTO

El comando **GOTO**, te hace saltar de un punto del alias a otro punto.

```
/numero {
if ($1 == 1) goto one
elseif ($1 == 2) goto two
else goto unknown
:one
echo El numero uno
halt
:two
echo El numero dos
halt
:unknown
echo Numero desconocido!
Halt }
```

Remote Script

También puedes escribir los alias en un remote, lo has de hacer poniendo **alias** delante seguido de la palabra del alias y todos los comandos que ha de ejecutar. Mira este ejemplo:

```
alias Sarrio {
/say este curso de scripting...
/say lo ha creado [SaRRiO] x'D }
```

Teclas de Función

Tu puedes redefinir las teclas de funcion, F1, Shift+F2... Mira estos ejemplos de aqui:

```
/F1 /join #Ayuda_para_novatos
/SF10 /nick [SaRRiO]
/CF1 /quit Me largo, que telefonica nos tima cada dia
```

El **/SF10** es **Shift+F10**, el **/CF1** es **Control+F1**, **/F1**, asi a secas es la tecla F1 y asi con las otras teclas. Esto a parte de lo que hemos visto, puedes meter comentarios en los aliases y remotes, ejemplos:

```
;Esto es un comentario que hago
:Te puedes servir para recordar para que sirve cada alias o script en general!
;Para distingir entre protecciones, juegos...
```

El simbolo «;», sirve para esto, para que el mIRC sepa que no son comandos sino comentarios.

Documento escrito por [SaRRiO] · www.sarrio.org · admin@sarrio.org

VARIABLES

El uso de variables es el recurso fundamental en programación. Podemos ver una variable informática como un recipiente en el que podemos depositar un valor cualquiera, para después hacer uso de él dentro de una instrucción, de esta forma no será necesario tocar la instrucción cada vez que queremos alterar el valor a manejar, sino tan solo asignar antes un nuevo valor a la variable.

Técnicamente una variable no es más que un espacio de nuestra memoria RAM (una dirección de memoria) que queda reservado en el momento en que se crea la variable, e identificado temporalmente con el nombre de esta. Cada vez que asignamos un valor a la variable el programa lo único que hace es depositar ese valor en la dirección reservada. Más adelante, cuando ejecutemos una instrucción que contenga esa variable, el programa irá a ese espacio de memoria y extraerá el valor que contenga, poniéndolo en el lugar del nombre de la variable, dentro de la instrucción en cuestión.

Por ejemplo, si en un script para mIRC hemos creado una variable llamada %num, a la que hemos asignado el número 3.14159 ,podemos tener una instrucción como la siguiente:

```
echo -a El valor asignado es: %num
```

Esta orden presentará en la ventana activa un mensaje que diga:

```
"El valor asignado es: 3.14159"
```

Si queremos que el valor que nos presente el mensaje sea diferente, no tenemos más que asignar previamente un valor distinto a la variable.

Como se ve el uso de las variables es muy sencillo. En mIRC lo único que hace falta es identificarlas con un nombre de nuestra elección y precederlo del símbolo: %.

Pero, ¿Como creamos las variables y les asignamos valores?. El entorno del mIRC provee una manera muy simple de hacerlo mediante el uso de la orden /set, cuya sintaxis es la siguiente:

```
set <%variable> [valor]
```

Por ejemplo:

```
set %num 3.14159
```

Es decir, es posible crear la variable (reservar el área correspondiente de memoria) en el mismo momento que le asignamos un valor. Si no especificamos valor la variable se creará, pero no contendrá nada de momento; y si usamos esta orden sobre una variable que ya existe, esta recibirá el nuevo valor especificado.

En el mIRC no es necesario preocuparse de qué tipo es la información que depositamos dentro de la variable, es decir, si se trata de una cadena de caracteres, un número entero, o un número con parte entera y parte decimal. Esta distinción la hace el programa en el momento que hagamos uso de ella; en principio todas las variables se consideran como una simple cadena de caracteres, pero si empleamos para manejarlas operadores aritméticos, el programa las transforma internamente en el valor numérico que corresponde; la única limitación en este caso es que solo se tendrán en cuenta los cinco primeros valores decimales, es decir, los cinco primeros dígitos tras la coma decimal.

Recursos de asignación

Asignar valores a una variable se puede hacer de forma directa, como hemos visto, o de forma indirecta a través de algún recurso del programa. Por ejemplo, si queremos que la variable %MiNick reciba el nick que estamos utilizando en ese momento, sin tener que especificárselo nosotros, podemos recurrir a la función $me, que nos devuelve precisamente nuestro nick actual. La asignación sería pues:

```
set %MiNick $me
```

Otro ejemplo, queremos asignar a una variable el contenido de otra variable:

```
set %var2 %var1
```

Incluso podemos hacer que el programa nos pregunte qué valor queremos que reciba determinada variable, cuando el script llegue a ese punto; no hay mas que hacer uso del identificador $$?, por ejemplo:

```
set %variable $$?="Asigna un valor a la variable:"
```

Así mismo es posible también hacer que una variable reciba directamente el resultado de operaciones aritméticas sencillas:

```
%x = 5 + 1          Suma
%x = 5 - %y         Resta
%x = %x * 2         Producto
%x = %z / $2        División
%x = $1 % %valor    Módulo (Resto de la división: $1 entre %valor)
%x = 2 ^ %w         Potencia
```

Solo es posible asignar el resultado de operaciones simples, para el uso de operaciones más complicada es preciso recurrir a la función $calc que podemos ver en el capítulo IDENTIFICADORES.

Finalmente, podemos hacer incrementos o decrementos automáticos del valor numérico de una variable con las órdenes:

```
inc <%variable> [incremento] dec <%variable> [decremento]
```

Estas órdenes incrementarán o decrementarán el contenido de una variable en una cantidad especificada. Si no se especifica cantidad el incremento o decremento se hará en la unidad. Por ejemplo:

- ```
  set %valor 5
  set %cifra 2
  dec %valor %cifra
  echo -a El valor actual es: %valor
  ```

En la pantalla aparecerá el mensaje:

"El valor actual es: 3"

Si en algún momento hacemos uso de una variable que aún no ha sido creada, o bien, que no contiene ningún valor, está devolverá el parámetro $null (nulo), esto solo será útil en las comparaciones de tipo if que se estudian en su correspondiente apartado.

NOTA: podemos consultar todas las variables existentes en un momento determinado, así como el valor que contienen (alterándolo manualmente si lo deseamos) desde el menú TOOLS, opción "Remote...", en la solapa "Variables".
  ```

Eliminación de variables

Eventualmente es posible también eliminar la variable, es decir, hacer que desaparezca la reserva de memoria que la creación de esta supuso. La orden a este efecto es la siguiente:

```
unset <%variable_1>
```

Es posible también utilizar el carácter * para eliminar grupos de variables, por ejemplo:

```
unset %nom*
```

Esta orden eliminará todas las variables cuyo nombre comience por "nom".

Finalmente podemos, si lo deseamos, eliminar todas la variables instaladas son la orden:

```
unsetall
```

Documento escrito por SoMaTiC · www.ayuda-irc.net · sucubus@ctv.es

EVENTOS REMOTOS

Se conocen como EVENTOS, en informática, todos aquellos sucesos que se producen a lo largo de una sesión de trabajo de un programa, y que éste es capaz de detectar y controlar. La programación orientada a eventos es aquella en la que podemos definir previamente las acciones que el programa debe de emprender en el momento en que se produzca un evento concreto. Es decir, predisponemos al programa a que preste especial atención a algunos sucesos, y le obligamos a que analice dichos eventos a fin de determinar si se reúnen las circunstancias que hemos establecido, y de ser así, a ejecutar las acciones que para ese caso le hemos programado.

El mIRC es un programa altamente configuráble y personalizáble a través de la definición de Alias, Popups, y Remotes. Dentro de este último grupo provee una de sus herramientas más potentes: la posibilidad de programar acciones orientadas a un grupo de eventos propios de una sesión de IRC; es decir, es posible programar de modo sencillo acciones de cierta complejidad como respuesta a algunas situaciones.

Existen tres tipos de eventos en el mIRC, los que llamaremos estándar, que son los que se describen aquí, y los eventos CTCP que describiré en un capítulo aparte, y los eventos RAW que se explicarán en el apartado Números Raw.

En este manual de referencia se incluyen los eventos descritos para la versión 5.31 del mIRC.

Un ejemplo

Por ejemplo, es muy sencillo hacer que el programa detecte cada vez que, en cualquier canal en el que estemos, alguien escriba nuestro nick (supongamos que es SomaTic), y en consecuencia, si lo deseamos, comunicar privadamente a esa persona que no estamos prestando atención en ese momento. No habría más que recurrir al evento "onTEXT", quizás el más importante de todos.

Para esto debemos de ir al menú "TOOLS" que se encuentra en la barra de menús de la parte superior de la pantalla y pulsar la opción "REMOTE...". Nos aparece un cuadro con una gran ventana de texto y una nueva barra de menús en la parte superior. Vayamos al menú "VIEW" y seleccionemos la opción "events"; ya podemos escribir en el cuadro de texto nuestras órdenes para el control de eventos (hágalo a ser posible al principio de esta caja de texto). Escribiremos la siguiente línea:

```
on 1:TEXT:Somatic:#: notice $nick Perdona $nick no estoy prestando atención!!!.
```

Pulsamos el botón "OK" y trabajo concluido. Desde este momento, si {_PePe_} menciona el nombre de SomaTic recibirá un mensaje que solo podrá ver él (notice), en el que le decimos: "Perdona {_PePe_} no estoy prestando atención!!!".

Al margen de la sintaxis de la línea, que veremos más adelante, la objeción es evidente: ¿Y que pasa si SÍ estamos prestando atención? ¿tenemos que ir otra ver a la ventana de edición de eventos y borrar lo que pusimos?. Bueno, es una posibilidad, pero afortunadamente también tenemos la facilidad de hacer que la acción de los eventos que deseemos se active y desactive a nuestra voluntad. Vayamos de nuevo a la ventana de edición de eventos y escribamos lo siguiente en las líneas anterior y posterior a la que escribimos antes: `#ausente off` y `#ausente end`. El bloque total (grupo) quedará de la siguiente forma:

```
#ausente off
on 1:TEXT:*Somatic*:#:/notice $nick Perdona $nick no estoy prestando atención!!!.
#ausente end
```

De esta forma podemos crear bloques o grupos de eventos con tantas líneas como deseemos en su interior. Para activar o desactivar su efecto debemos de incluir las

instrucciones: .enable #ausente y .disable #ausente dentro de algún alias o de un menú (si esto le causa problemas ahora es el momento que estudiar a fondo las secciones de aliases y popups del mIRC ;)). (el punto delante de las cláusulas enable y disable tiene el único fin de inhibir el mensaje que el programa presenta en pantalla anunciando la activación o desactivación del grupo).

Pruebe los efectos de la programación de eventos aplicando los distintos comandos del IRC y del mIRC en su interior (por motivos que desconozco algunos de ellos no funcionan en este contexto, como por ejemplo /me, /ame, /say, ... etc). También puede hacer uso de alias préviamenre definidos, o de la estructura IF-ELSE, pero, en cualquier caso, si desea incluir varias líneas de órdenes vinculadas a un mismo evento use siempre el separador de comandos (el carácter "|", que se obtiene pulsando las teclas ALTGR y 1).

Sintaxis general

Como ya se ha podido observar en el ejemplo anterior, lo que hemos dado en llamar un la línea de evento se forma mediante una cadena de caracteres en la que los distintos campos de información están separados por el símbolo de los dos puntos. Los campos son los siguientes:

```
on <nivel>:<EVENTO>:<texto>:<lugar>:<comando/os>
```

En el primer campo es necesario especificar en nivel de acceso remoto que ese usuario tiene a nuestro sistema. Este es un tema algo complejo en el que no procede entrar ahora, baste saber que este nivel es un número que marca la capacidad de acceso que cada usuario tiene para ejecutar comandos del IRC de forma remota dentro de nuestro ordenador. No se alarme, nadie tendrá un nivel que usted no le haya concedido previamente, y este para los comandos que usted especifique. Para comprobar quien tiene nivel de acceso remoto a su sistema no tiene más que mirar en la solapa de "USERS" de la misma ventana desde donde introducimos los eventos, ahí puede añadir o borrar lo que desee. En principio, y para todos los efectos de este documento, podemos considerar el nivel de acceso 1 que es el que otorga el programa por defecto a todos lo usuarios; por tanto, en este caso el primer campo sería: on 1.

también es posible anteponer el símbolo @ antes del número de nivel, si hacemos esto, la definición del evento no se ejecutará nada más que en caso de que dispongamos del estatus de operador del canal.

El segundo campo es el identificador del evento que estamos configurando, en el ejemplo anterior era TEXT, pero descubrirá que hay otros muchos.

El tercer campo es el "texto", este no tiene sentido en todos los eventos, gran parte de ellos no están vinculados a texto alguno, y simplemente se omite este campo. El cuarto campo es "lugar", es decir, el canal, privado...etc. donde el mIRC vigilará la circunstancia del evento, podemos especificar de este modo que determinado evento produzca las acciones especificadas solo si se da en un determinado lugar. Como en el campo anterior no siempre tiene sentido y por tanto se omite en algunas definiciones de eventos.

El último campo es el más grande, incluye el comando o comandos, alias, etc... que se ejecutarán en caso de que se produzca el evento y las circunstancias especificadas. En este campo se pueden concatenar varios comandos utilizando el símbolo "entubador": "|" (no olvide dejar espacios en blanco a ambos lados del entubador).

Para aumentar aun más la potencia de la configuración de eventos, estos admiten el uso de funciones, son todos aquellos identificadores que comienzan por el símbolo $ (la referencia del mIRC los llama parámetros, el tema es mas bien terminológico, yo prefiero verlas como funciones puesto que junto con las internas del programa el mIRC permite definir otras propias, con una sintaxis muy similar a la del lenguaje C).

No se relaciona en este documento una lista exhaustiva de dichas funciones, puesto que muchas de ellas son propias de un solo evento concreto, mientras que otros no las admiten. No obstante, si presta atención a los ejemplos irá viendo la forma y utilidad de la mayoría de ellas, en los lugares en los que son efectivas; el resto lo debe de aportar la experiencia personal. (las funciones deberán de tener siempre espacios en blanco a ambos lados de su nombre).

Es perfectamente posible, así mismo, el empleo de variables y estructuras IF-ELSE dentro de las definiciones de eventos, encontrará numerosos ejemplos de este uso en los scripts para mIRC que circulan a libre disposición en la WWW y el IRC.

Es hora de entrar a conocer una a una las definiciones de eventos que proporciona la versión 5.31 del mIRC. Esta lista está basada en la ayuda del propio programa y no la considero cerrada, por tanto agradeceré cualquier crítica y sugerencia, por supuesto las dudas también serán atendidas en mi e-mail: sucubus@ctv.es, o en el canal #Ayuda_IRC del IRC Hispano, no tiene más que preguntar por SomaTic ;).

Relación de eventos

ON TEXT

Este evento se produce al recibirse un mensaje especificado, bien en un privado, o bien en un canal en el que nos encontramos.

Formato: `on <level>:TEXT:<texto>:<*><?><#[,#]>:<comandos>`

Ejemplo: `on 1:TEXT:*ayuda*:*:/msg $nick ¿Cual es el problema?`

El texto del mensaje que deseamos que produzca el evento se puede especificar de varias maneras:

*	El evento se producirá ante cualquier texto.
&	El evento se producirá ante cualquier palabra.
Texto	El evento se produce anta una palabra concreta y no otra.
Texto*	Se produce ante una cadena de caracteres que comience por el texto especificado.
*Texto	Se produce ante una cadena de caracteres que finalice por el texto especificado.
Texto	El evento se produce si el texto contiene la cadena especificada, sea cual sea su posición.

Se puede especificar el lugar donde se controlará el evento de la siguiente forma:

?	Controlara la producción del evento dentro de un query o de un DCC chat
#	Controlara la producción del evento dentro de un canal cualquiera en el que nos encontremos
#canal_1,#canal_2,...	Controlará la producción del evento dentro de un canal especificado.
*	Controla el evento en un privado o canal indistintamente.

Ejemplos:

```
on 1:TEXT:Hola*:#:/msg $chan ¡Bienvenido al canal $chan $nick $+ !
```

Esta instrucción detectará cuando alguien diga la palabra "Hola" en cualquier canal que nos encontremos, y automáticamente le dará la bienvenida al canal en cuestión. El identificador $+ elimina los espacios en blanco que tiene a ambos lados, de ese modo podemos hacer que el nick de la persona que ha dicho "hola" aparezca en nuestra respuesta seguido del símbolo "!", si escribimos directamente este símbolo a continuación de $nick este no será reconocido como un identificador del nick del usuario, sino que aparecerá literalmente como está en la orden.

También es posible emplear variables para dar más flexibilidad al formato del evento, por ejemplo:

```
on 1:TEXT:%text:%canal:/msg $nick Tú has escrito %text en el canal: %channel
```

Este ejemplo trabajará con los valores previamente guardados en las variables %text, y %canal.

Nota: Este evento se producirá solo ante textos de otros usuarios, los de usted no desencadenarán el evento

ON ACTION Y ON NOTICE

Tienen el mismo formato que on TEXT, y se producen a partir de una acción o de una noticia respectivamente. Recordemos que una noticia se produce cuando alguien nos envía un mensaje con el comando /notice, y una acción cuando alguien efectúa una acción de control dentro del canal; por ejemplo:

```
on 1:ACTION:*set mode*:#:/msg $chan ¡Ya estamos jugando con los modos!
```

Esta orden detectará cualquier acción que contenga la cadena "Set mode", y emitirá un mensaje al canal en que se produce.

```
on 1:NOTICE:*:?:/msg $nick Estoy ocupado, espera un momento!
```

Esta orden detecta cualquier /notice recibido estando en un privado, y responde con otro /notice al autor.

ON BAN Y ON UNBAN

Estos eventos se producirán cuando un usuario sea baneado o desbaneado en el canal.

Formato: on <level>:BAN:<#[,#]>:<comandos>

Ejemplos:

```
on 1:BAN:#:/msg $nick Has sido baneado de: $chan
on 9:BAN:#:/mode $chan -o $nick | /mode $chan -b $banmask
```

El segundo ejemplo es una acción múltiple que quita el estatus de operador al usuario que puso el ban, y elimina este cuando el usuario baneado tiene nivel 9 o superior en la lista de usuarios de nuestro programa (la función $banmask devuelve la máscara de baneo que fue empleada, y se usa, en este caso, para eliminar dicho baneo).

```
on 1:UNBAN:#:/msg $bnick Has sido desbaneado por $nick
```

Observe en este ejemplo el uso de la función $bnick. Esta devuelve el nick del usuario baneado o desbaneado, pero solo en caso de que dicho nick este incluido dentro de la máscara del baneo. En caso contrario $bnick = $null (nulo).

Es posible tener en cuenta en la sintaxis de la instrucción los niveles del baneador y del baneado. A este fin se puede preceder el número de l nivel de los signos: <,>,<=,=>,<>, ó =. Por ejemplo:

```
on >=2:BAN:#:/msg $chan $nick baneado $banmask (legal)
```

En esta situación, si el nivel del baneador es mayor o igual que el del usuario baneado, se comunica queeste es un ban legal (como se podría comunicar cualquier otra cosa). Recuerde, esta sintaxis compara los niveles de baneado y baneador, nunca que este tenga o no nivel superior o igual a 2.

Nota: Estos eventos solo operan cuando el server de IRC proporciona los nicks de baneador y baneado, y no su cuando da sus direcciones.
Observe el siguiente ejemplo:

```
on 1:BAN:#: {
 if ($banmask iswm $address($me,0) || ($me isin $banmask) {
  echo -s $nick te ha baneado con esta máscara: $banmask
  if ( $me isop $chan ) { mode $chan -ob+b $nick $banmask $address($nick,2) }
 }
}
```

Esta rutina se inicia cada vez que se produce una acción de baneo en un canal en que nos encontramos. En primer lugar comprueba que la máscara del ban coincide con la nuestra o bien tiene incluido nuestro nick, en tal caso emite un mensaje a la ventana de estatus especificando la persona que lo ha realizado y la máscara con la que se ha hecho el ban, a continuación comprueba si somos operadores del canal, y en tal caso retira el estatus de operador a quien nos ha baneado, y le banea a su vez.

ON CHAT Y ON SERV

Los eventos on CHAT y on SERV se producen cada vez que es enviado un mensaje la una ventana de DCC chat, o a un Fserve respectivamente. Es similar al evento on TEXT, el cual no funciona en un chat ni en un servidor de ficheros.

Formato: on <level>:<CHAT/SERV>:<texto>:<comandos>

Ejemplo: on 1:CHAT:*ayuda*:/msg $nick Cual es el problema?

La especificación del texto que producirá el evento se hace de la misma forma que en ON TEXT.

ON CONNECT Y ON DISCONNECT

El evento on CONNECT se produce cuando mIRC conecta al server de IRC, después de visualizar el MOTD (mensaje del día del servidor).

Formato: on <level>:CONNECT:<comandos>

Ejemplo: on 1:CONNECT:/join #ayuda_irc

on DISCONNECT es otro evento con el mismo formato, que se produce al desconectarse del server.

Ejemplo: on 1:CONNECT:/echo Conectado a $server a las $time Tu nick es: $nick

ON CTCPREPLY

Se produce cuando otro usuario responde a una pregunta CTCP por parte de usted.

Formato: on <level>:CTCPREPLY:<texto>:<comandos>

Ejemplo: on 1:CTCPREPLY:VERSION*:/echo $nick está usando IRC cliente: $1-

Ver evento on TEXT para los formatos de texto aceptados.

Ejemplo: on 1:CTCPREPLY:PING*:/echo -s $nick ha respondido al ping!

ON OP Y ON DEOP

Se producen cuando un usuario del canal es opeado o deopeado.

Formato: on <level>:OP:<#[,#]>:<comandos>

Ejemplo: on 1:OP:#ayuda_irc:/msg $nick No abuses del status!

Ejemplo2: on 9:OP:#:/mode $chan -o $opnick | /msg $nick no opees a esa persona!

En este ejemplo el evento se produce cuando es opeado un usuario de nivel de acceso 9 en nuestra sección de remotes. $opnick devuelve el nick de la persona opeada o deopeada, y $nick el de la que lo opeó o deopeó.

```
on 1:DEOP:#ayuda_irc:/mode $chan +o $opnick
```

Con esta orden devolvemos el op a cualquier persona que sea deopeada en el canal #ayuda_irc.

Es posible tener en cuenta en la sintaxis de la instrucción los niveles del opeador y del opeado. A este fin se puede preceder el número de nivel de los signos: <,>,<=,=>,<>, ó =. Por ejemplo:

```
on >=2:OP:#mIRC:/msg $chan $nick opeado (legal)
```

En esta situación, si el nivel del opeador es mayor o igual que el del usuario opeado, se comunica que este es un op legal (como se podría comunicar cualquier otra cosa). Recuerde, esta sintaxis compara los niveles de opeado y opeador, nunca que este tenga o no nivel superior o igual a 2.

Nota: Estos eventos solo operan cuando el server de IRC proporciona los nicks de opeador y opeado, y no su cuando da sus direcciones.

ON VOICE Y ON DEVOICE

Se produce cuando un usuario recibe o es privado de voz en un canal moderado en que nos encontramos.

Formato: on <level>:VOICE:<#[,#]>:<comandos>

Ejemplo: on 1:VOICE:#:/msg $vnick ya tienes voz en: $chan

Ejemplo2: on 9:VOICE:#:/mode $chan -v $vnick | /msg $nick No des voz o ese tío!

Esta orden se ejecuta cuando el que recibe voz tiene nivel 9 o superior.

```
on 1:DEVOICE:#mIRC:/mode $chan +v $vnick
```

Devuelve la voz al que se le quite en el canal. Son aplicables también los prefijos de comparación de niveles descritos en on BAN y on OP.

ON DNS

Se produce cuando hacemos un requerimiento DNS (comando /dns).

Formato: on <level>:DNS:<commandos>

Ejemplo: on 1:DNS:/notice $me resuelto: $raddress

Ejemplo2:

```
on 1:DNS:{
 echo –s DNS de $nick | echo -s ip address: $iaddress
 echo -s named address: $naddress | echo -s resolved address: $raddress
}
```

ON ERROR

Se produce cuando el server envía un mensaje de error, por ejemplo, en una desconexión.

Formato: on <level>:ERROR:<texto>:<commandos>

Ejemplo: on 1:ERROR:*server full*:/echo Has salido del server!

Ver el evento on TEXT para el formato del texto.

Ejemplos: on 1:ERROR:*banned*:/echo Has sido baneado de este server!

ON FILESEND y ON FILERCVD. ON SENDFAIL y ON GETFAIL

Se producen al completarse un dcc send o un dcc get (envio o recepción de ficheros reapectivamente).

Formato: on <level>:FILESENT:<filename[,filename]>:<comandos>

Ejemplo: on 1:FILESENT:*.txt:/msg $nick ahí te mando: $filename !!

Los eventos on SENDFAIL y on GETFAIL se producen cuando un dcc send o un dcc get son fallidos.

Ejemplos: on 1:FILESENT:*.txt,*.ini:/echo Enviado $filename a $nick $address

Se produce cuando enviamos un fichero .TXT o INI. $filename devuelve el nombre del fichero enviado.

```
on 1:FILERCVD:*:echo Recibido $filename de $nick | run notepad.exe $filename
```

Esta orden avisa cuando hemos recibido ficheros .TXT e .INI, y abre el bloc de notas para ver su contenido.

```
on 1:SENDFAIL:*.txt:/echo Ha fallado el envío de $filename a $nick!!
```

ON INPUT

Se produce cuando introducimos texto en una caja de texto y pulsamos INTRO.

Formato: on <level>:INPUT:<commandos>

Ejemplo: on 1:INPUT:/echo Texto introducido: $1-

La función $1- devuelve en este caso toda la cadena de texto introducida

ON INVITE

Se produce cuando un usuario nos invita a un canal.

Formato: on <level>:INVITE:<#[,#]>:<comandos>

Ejemplo: on 1:INVITE:#mIRC:/join $chan

ON JOIN Y ON PART

Se producen al entrar o salir un usuario de un canal en que nos encontramos

Formato: on <level>:JOIN:<#[,#]>:<comandos>

Ejemplo: on 1:JOIN:#:/msg $nick Hola!

ON KICK

Se produce cuando un usuario es expulsado de un canal en el que nos encontramos.

Formato: on <level>:KICK:<#[,#]>:<comandos>

Ejemplo: on 5:KICK:#:invite $knick $chan | msg $nick Hey, $knick ese es mi amigo!

Es aplicable también aquí la regla de comparación de niveles de los nicks descrita el los eventos on BAN y on OP.

ON LOAD Y ON START

Estos eventos se producen cuando un fichero de script es cargado o descargado.

Formato: on <level>:LOAD:<commandos>

Ejemplo: on 1:LOAD:/echo mIRC Script cargado!

On start se produce cuando inicies el mIRC...

ON MIDIEND Y ON WAVEEND

Se produce cuando el mIRC termina la reproducción de un fichero MIDI o WAV.

Formato: on <level>:MIDIEND:<commandos>

Ejemplo: on 1:MIDIEND:/splay jazzy.mid

Este evento no tiene funciones asociadas. No se producirá si la reproducción se detiene a causa de una interrupción, solo lo hace cuando la reproducción es completada.

ON MODE

Se produce cuando un usuario cambia los modos del canal (pero no los de usuario).

Formato: on <level>:MODE:<#[,#]>:<comandos>

Ejemplo: on 1:MODE:#ayuda_IRC:/notice $me $nick Cambia modos de $chan a: $1-

Esta instrucción se produce cuando alguien cambia los modos del canal teniendo nosotros también el estatus de operador.

ON NICK

Se produce cuando un usuario cambia de nick dentro del canal.

Formato: on <level>:NICK:<commandos>

Ejemplo: on 1:NICK:/msg $nick Hi $newnick!

$nick devuelve el nick antiguo; $newnick el nuevo.

ON NOSOUND

Este evento se produce cuando alguien activa un sonido en el canal y el mIRC no encuentra el fichero midi o wav en nuestros directorios de sonidos, los especificados en el dialogo FILE/OPTONS/SOUNDS

Formato: on <level>:NOSOUND:<comandos>

Ejemplo: on 1:NOSOUND:/notice $nick No tengo el fichero $filename

ON NOTIFY Y ON UNOTIFY

Se producen al entrar o salir del IRC un usuario que se encuentra en nuestra lista de notificaciones.

Formato: on <level>:NOTIFY:<commandos>

Ejemplos:

on 1:NOTIFY:/msg $nick Hola nick, ya te veo venir!!! :)

on 1:NOTIFY:/notice $me $nick se va del IRC *sniff* :-(

ON QUIT

Se produce cuando un usuario que está en nuestro canal sale del IRC.

Formato: on <level>:QUIT:<commandos>

Ejemplos:

on 1:QUIT: notice $me $nick sale del IRC con el mensaje: $1-

ON SNOTICE

Se produce cuando recibimos una noticia del server.

Formato: on <level>:SNOTICE:<texto>:<commandos>

Ejemplo: on 1:SNOTICE:*client connecting*:/halt

Para una explicación del formato del texto, ver el evento on TEXT.

Ejemplo: on 1:SNOTICE:*split*:/splay netsplit.wav

Activa un fichero de sonido en caso de una noticia del server anunciando un split.

ON TOPIC

Se produce cuando algún usuario cambia el tópico del canal.

Formato: on <level>:TOPIC:<#[,#]>:<comandos>

Ejemplo: on 1:TOPIC:#mIRC:/msg $chan Hmm, que tópico mas cutre!!!: $1-

La función $1- devuelve el texto completo del nuevo tópico.

ON USERMODE

Se produce cuando cambiamos nuestros modos de usuario.

Formato: on <level>:USERMODE:<commandos>

Ejemplo: on 1:USERMODE: echo -s nuevos modos: $1-

ON WALLOPS

Se produce cuando recibimos un wallops message.

Formato: on <level>:WALLOPS:<texto>:<commandos>

Ejemplo: on 1:WALLOPS:*aviso*: echo –s $timestamp WallOp de aviso: $1-

Ver el evento on TEXT para una explicación de los posibles formatos del texto.

ON HELP Y ON DEHELP

Se produce cuando un usuario recibe o es privado de halfop en un canal en que nos encontramos. Este modo tan especial se puede encontrar en la red Univers.Org...

Formato: on <level>:HELP:<#[,#]>:<comandos>

Ejemplo: on 1:HELP:#: echo # $nick ha puesto status de halfop a $opnick

ON PING

Se produce cuando el servidor te manda un PING con un cierto mensaje para comprobar si sigues conectado o has cerrado la conexión.

Formato: on <level>:PING:<comandos>

Ejemplo: on 1:PING:echo –s El servidor de ha hecho un ping: $1-

ON RAWMODE

Se produce cuando un usuario cambia cualquier modo en un canal donde estas tu.

Formato: on <level>:RAWMODE:<#[,#]>:<comandos>

Ejemplo: on 1:RAWMODE:#: echo # $nick pone los modos $1-

$1 : los modos que cambia

$2 : los parametros de los modos (nicks, key, limite de usuarios...)

ON KEYUP Y ON KEYDOWN

Se produce cuando el usuario del mIRC pulsa o deja de pulsar una tecla especificada en una ventana especificada...

Formato: on <level>:KEYDOWN:<@>:<key,...,keyN>:<comandos>

Ejemplo: on 1:KEYDOWN:@:*:echo –S Has pulsado la tecla $keyval en $window

El ejemplo indicado es un estándar, funciona en cualquier ventana y tecla...

Mira este otro ejemplo:

on 1:KEYDOWN:@keys:32:echo –s has pulsado la barra espaiadora en la ventana @keys

ON HOTLINK

Se produce cuando el usuario del mIRC pasa el cursor por encima de un texto que contiene la cadena de texto especificada. Puede usarse en cualquier tipo de ventana, query, canales, ventanas personalizadas... en todas si quieres...

Formato: on <level>:HOTLINK:<texto>:<*#?=!@>:<comandos>

Ejemplo: on 1:HOTLINK:*univers*:*:echo Algun dia todas las redes seran como Univers

ON SEROVEROP

Se producen cuando un usuario del canal es opeado por un servidor de la red.

Formato: on <level>:SERVEROP:<#[,#]>:<comandos>

Ejemplo: on 1:SERVEROP:#ayuda_irc: if ($me isop #) mode # -o $opnick

Lee el apartado del evento Op/Deop para más información...

ON SERVERMODE

Se produce cuando un servidor cambia los modos del canal...

Formato: on <level>:SERVERMODE:<#[,#]>:<comandos>

Ejemplo:

```
on 1:SERVERMODE:#:{
 if ($me isop #) {
  mode # $replace($replace($replace($1,-,<),+,-),<,+)
 }
}
```

En este ejemplo hacemos que cuando el servidor cambia los modos del canal nosotros activaremos o desactivaremos los modos que ha cambiado, eso si tenemos la suerte de ser op del canal...

Documento escrito por SoMaTiC · www.ayuda-irc.net · sucubus@ctv.es

Modificado y actualizado por [SaRRiO] · www.sarrio.org · admin@sarrio.org

IF-ELSE

Que son los If-Else

Los “If-Else” no son comandos que se puedan ejecutar automaticamente, como por ejemplo, el **/mode #scripting +m**. Los “If-Else” deben de ser ejecutados dentro de un alias, popup, o remote, ose a, que hay que escribirlos en código.

Los “If-Else” son unas condiciones que creamos ente dos objetos. Este comando, sera conocido para el que haya programado alguna vez.

La sintaxis

La sintaxis básica de los If-Else es la siguiente : **IF (%objeto.1 comparador %objeto.2) { comandos }**. El comando IF, abre el comando. La comparación que queramos hacer debe de ir entre paréntesis.

La comparación, consta de tres partes: El primer objeto a comparar, el comparador y el segundo objeto a comparar. Los objetos a comparar pueden ser cualquier tipo de texto, un número, un nick, el nombre de un archivo… El comparador, es lo que crea la condición. Pueden ser muchos los comparadores, *igual que*, *menor que*, *esta dentro de*…

Bueno, no os habreis enterado demasiado y aquí os va un ejemplo sencillo

```
/comprobar {
IF ( $$1 == $$2 ) { echo –a $$1 es igual a $$2 }
ELSE { echo –a $$1 no es igual a $$2 }
}
```

Si os habéis fijado, ya hemos incluido un comando. El ELSE. Creo que, lo que he hecho no necesita más explicaciones. Pero bueno, aquí va:

IF ($$1 == $$2) // Si lo primero puesto, es igual que el segundo
{ echo –a $$1 es igual a $$2 } // Haz lo siguiente
ELSE // o sino
{echo –a $$1 no es igual a $$2 } //Haz lo siguiente

EL comando ELSE, no es necesario. Si no lo hubiéramos puesto, y si $$1 no fuera igual que $$2, no hubiera pasado nada. Pero que pasa, cuando queremos hacer que si los números no son iguales, mire si son mayores o menores. ¿Tenemos que poner ELSE y despues seguido IF? Pues no.

Para eso esta el comando ELSEIF, que es el unido de los comandos ELSE y IF. Con esto, mejoramos nuestro ejemplo:

```
IF ( $$1 == $$2 ) { echo –a $$1 es igual a $$2 }
ELSEIF { $$1 > $$2 ) { echo –a $$1 es mayor que $$2 }
ELSEIF { $$1 < $$2 ) { echo –a $$1 es menor que $$2 }
ELSE { echo –a PERO KE KLASE DE NUMEROS SON ESOS? }
```

Este nuevo ejemplo, mira si el primer número es igual que el segundo, si no es así, mira si es mayor, si no es así, mira si es menor, y si tampoco es así (cosa difícil si son números), dice lo de PERO KE KLASE DE NUMEROS SON ESOS?.

Los comparadores

mIRC admite comparadores de muchos tipos. Para números, para letras…

Aquí teneis los comparadores que acepta el mIRC 5.41 Algunos estan omitidos, puesto que no se para que son o no se usan mucho. :)

== Es igual que
=== Es igual que (mayusculas y minusculas incluidas)
!= No es igual que
< Es menor que
> Es mayor que
>= Mayor o igual que
<= Menor o igual que
// es multiplo de
\\ no es multiplo de

isin Esta dentro de*
isincs Esta dentro de (mayusculas y minusculas incluidas)*
isnum Esta dentro del rango*
isletter Esta dentro de las letras*
ison Esta dentro de canal*
isop Es op en canal
isvo Es voice en canal
ischan Es un canal en el que estas*

isauto Esta en tu lista de auto-ops del canal
isignore Esta en tu lista de ignores
isprotect Esta en tu lista de protects
isnotify Esta en tu lista de notify

El primer grupo, son para los números principalmente, menos los tres primeros. Estos se pueden usar con números y texto. El tercero, es para decir que no es igual, o sea lo contrario del ==. Este es una excepción, puesto que con los demás, si quieres lo contrario del comparador en si, se consigue mediante un ¡ antes del comando, por ejemplo, ¡isop .

El segundo grupo es para comparar distintos grupos de textos., menos el isnum. Este es para mirar si el número $$1 esta dentro del rango $$2. Este rango tiene que estar en este formato: N1-N2. Por ejemplo: 1-10. El comando isletter hace lo mismo, pero con letras, en este, el formato de $2 es el siguiente: L1L2L3L4. Por ejemplo: abdgklt.

El isin y el isinc son para mirar si un texto esta dentro de otro. Por ejemplo, si queremos hacer el juego de “la ruleta de la fortuna” y alguien, dice una palabra, por ejemplo, casa. La condición seria:

IF (%lokehandixo isin %eltextoentero) { say Has acertado una palabra!!! }

El isinc, seria lo mismo, pero si quisieras que también acertar con las mayúsculas.

El ison, aunque se parezca mucho al isin, es para mirar si alguien esta dentro de un canal. Por ejemplo:

IF (Flubbers ison #scripting) { msg $chan Joer Flubbers, siempre estas aquí! }

El isop, y el isvo se usan de la misma forma, para mirar si alguien tiene op o si tiene voz o esta voice. El ischan, no necesita un segundo objeto, su sintaxis es la siguiene: if (canal ischan) { }. Esto mira si el canal es un o en el que estes tu.

El último grupo de comparadores, es para nuestras listas internas de notify, ignore… Estos tiene el mismo sintaxis que el ischan.

Comandos Especiales

Y que pasa si queremos hacer, por ejemplo, que cuando haya dos condicones para hacer algo, o cuando alguno de los dos valga…? Pues para eso existen los comandos || y &&. El primero, junta dos comparaciones, siendo valida cualquiera de las dos para que pase algo. Por ejemplo:

```
IF( $nick == Scytale)||( $nick == Scytale) { echo –a $Nick es un bot }
```

El && es como un "Y". Es usado cuando se necesita que pasen las dos cosas. Por ejemplo, cuando un protegido te pide op mediante query:

```
ON 1:TEXT:opme*:?:{
If  ( $nick isprotect )&&( $me isop $$2 ) {
Mode $$2 +o $nick
}
```

Por último, hay un identificador especifico de los IF-ELSE. Es el $ifmatch. Este, es el primer objeto de la comparacion. Por ejemplo:

```
IF ( %TEXT != HOLA KARAKOLA ) { echo –a $ifmatch no es igual que HOLA KARAKOLA }
```

Documento escrito por genars para el canal #Scripting con fecha 1-12-1998

EVENTOS CTCP

Introducción

Las siglas **CTCP** significan *"Client to Client Protocol"* o *"Protocolo de cliente a cliente"* en castellano, y básicamente se trata de un tipo escpecial de comunicación entre los usuarios de un server de IRC que será usada para provocar que los usuarios de el script que estemos haciendo ejecuten ciertas acciones automáticamente al recibir cierta información por CTCP.

Por otra parte los usuarios remotos se refiere a algo que usted seguro que ya ha visto anteriormente en el tutorial de enventos remotos, se trata de ese numerito que siempre ponemos en el evento por ejemplo "on 1:OPEN:" , ¿qué es ese '1'? Pues sencillamente es el nivel que necesita un usuario para hacer que salte ese evento con los comandos que se hayan especificado, despues entraré en más detalle en esto, ahora volviendo al principio vamos a ver todo lo referente a los eventos CTCP:

Comandos CTCP

Para mandar información a otro usuario mediante **CTCP** lo haremos de la siguiente forma: **/ctcp <nick> <mensaje>**

Donde <nick> es el nick de la otra persona y <mensaje> es cualquier mensaje que queramos enviar por ese protocolo. Lógicamente no nos pondremos a hablar con un usuario mediante **CTCP's** ya que seria absurdo estando los dos conectados al **IRC**. Los **CTCP's** tienen otra utilidad... que es la de que el otro usuario reaccione automáticamente de una cierta manera al usted enviarle ese **CTCP**. Por ejemplo, existe uno que tiene el **mIRC** ya implementado: el famoso ***CTCP PING*** y consiste en enviar un ping al otro usuario:

```
/ctcp <nick> ping
```

El programa del otro usuario responderá automaticamente al ***CTCP PING*** y lo hará devolviendo una información, que al llegarnos de nuevo el **mIRC** nos muestra en pantalla. En este caso en pantalla se muestra el "Lag" o retardo de la línea que hay entre usted y la persona a la que envió el ping. Puede probar con los otros **CTCP** implementados ya en **mIRC**, el funciona-miento de todos es similar; solo varia la respuesta que proporcionan.

A continuación aprenderá a crear sus propias respuestas a ciertos **CTCP's**, ya que el **mIRC** solo trae unas cuantas ya definidas (como son PING, VERSION, TIME ...) pero usted quizás quiera hacer otras con otros nombres, o tal vez cambiar las respues-tas que a los ya existendes dará su programa.

Eventos CTCP

Vamos a ver ya como usamos este tipo de eventos para que la explicación sea más fácil de entender. En la sección "Remotes" del editor del **mIRC** es donde definiremos estos eventos y se hacen de una forma parecida al resto de eventos remotos. La sintaxis es:

ctcp <nivel>:<texto>:<#,?,*>:{ comandos }

Este tipo de eventos haran que nuestro programa se comporte de cierta manera (es decir, que ejecute los comandos que le especifiquemos) cuando recibamos un ***CTCP <texto>*** de otro usuario. El <nivel> de momento lo dejaremos siempre en '1' , y el otro parámetro ha de ser o bien un '#' si nos referimos a un canal, un '?' para un privado(query) o un '*' para "en cualquier lado".

Con un pequeño ejemplo lo veremos más claro, copie lo siguiente en el editor del **mIRC**, pestaña "Remotes":

```
ctcp 1:*hora*:*:{
 msg $nick Son las $time
}
```

Ese evento hará como ya habrá imaginado que cuando un usuario le haga un ***/ctcp <nick> hora*** , usted automáticamente le responda enviándole un query en el que diga por ejemplo "Son las 19:45:23" . Como ve se pueden usar '*' en el parámetro <texto> para indicar que si la palabra "hola" del mensaje **CTCP** viniera precedida de cualquier otra, o despues de esa palabra hubiera alguna palabra más, se ejcutaría de todas formas en comando. En este ejemplo en concreto eso no es de mucha utlilidad, pero en el siguiente si que lo será:

```
ctcp 1:dime*:*:{
 msg $nick Lo siento estoy ocupado
}
```

Este evento hará que cuando un usuario le envíe un /ctcp DIME , usted le responda diciendole que está ocupado. Por ejemplo un usuario le podría hacer un /ctcp <sunick> dime la hora o quizas /ctcp <sunick> dime tu nombre. En cualquier caso la respuesta será la misma.

Lo que hemos visto hasta ahora se refiere a crear eventos **CTCP** propios, que no existían antes en el **mIRC** y a los que el script responderá de la forma que le hemos especificado, pero también si quisiera, podría cambiar su respuesta a algunos de los eventos **CTCP** ya definidos, como es el caso del PING, para ello tendremos que especificar al final de los comandos, el comando /halt , por ejemplo:

```
ctcp 1:PING:*:{
 notice $nick Nada de Pings, gracias! | halt
}
```

Este evento hará que cuando usted reciba un /ctcp ping de algún usuario, le enviará un /notice diciéndole: "Nada de Pings, gracias!", y mediante el comando /halt haremos que el script deje de procesar ese evento, y de esa forma que no procese la parte que ya estaba hecha en el **mIRC** (la que nos devuelve el lag). También podríamos usar este procedimiento para otros **CTCPs** ya definidos como son TIME, USERINFO ... etc.

Otra utilidad de estos eventos puede ser la de controlar nuestro **mIRC** "a distancia", y me explico, si abrimos dos **mIRCs**, podremos controlar a uno de ellos mediante **CTCPs** mientras que el otro lo controlaremos normalmente, se pueden usar por lo tanto para controlar a nuestros clones, por ejemplo si copiamos el siguiente código en la sección Remotes y abrimos dos **mIRCs**:

```
ctcp 1:habla*:#:{ /say $1- }
```

Cuando desde uno de los **mIRCs** escribamos ***/ctcp <nick_clon> HABLA <mensaje>*** el otro **mIRC** que hemos abierto enviará el mensaje que pongamos después del "HABLA" al canal, por ejemplo si ponemos /ctcp <nickclon> habla soy un bot, me manejan con ctcps! hará que nuestro clon diga ese mensaje al canal.

```
ctcp 1:quit:*:{ /quit $1- }
```

Este nuevo ejemplo hará que al recibirlo el **CTCP**, el clon cierre el **mIRC** con el mensaje especificado en ***/ctcp <nickclon> quit <mensaje_de_quit>***

```
ctcp 1:entra:*:{ /join $1 }
```

Este hará que el clon entre en el canal que especifiquemos en /ctcp <nickclon> entra #canal

```
ctcp 1:comosoy:#:{ /say Me llamo $1 , tengo $2 años y soy $3 }
```

Este último hará que el clon diga en el canal ese mensaje usando las tres siguientes palabras que pongamos despues del ***/ctcp <nickclon> comosoy*** , por ejemplo si ponemos /ctcp <nickbot> comosoy Pepe 20 alto , el bot pondrá en el canal "*Me llamo Pepe, tengo 20 años y soy alto*".

Con esto hemos matado dos pájaros de un tiro, no sólo ya sabemos manejar los eventos **CTCP** y como evitar las respuestas predeterminadas de algunos de ellos, sino que hemos aprendido sobre su principal utilidad que es la creación de Clones que obedezcan nuestras órdenes, tambien conocidos como "bots".

Antes de pasar a la siguiente sección hay que comentar también que hay un tipo especial de eventos **CTCP** que sirven exclusivamente para cambiar la apariencia de las respuestas estándar de los **CTCPs** predefinidos en el **mIRC**... es decir que por ejemplo cuando usted hace un ping a alguien, ese alguien le devuelve la información del ping, y usted ve en pantalla algo como:

[TeMpEsT PING reply]: 0 secs

Pero quizás para hacer más bonito el script le gustaría que pusiera:

Lag con TeMpEsT: 0 segundos.

para ello usamos el evento **ON CTCPREPLY** que tiene la siguiente sintaxis:

on 1:CTCPREPLY:<ctcp>:{ comandos }

Donde **<ctcp>** pondremos el **CTCP** predefinido al que nos referimos, y en comandos la secuencia de comandos que queremos ejecutar. Generalmente para este tipo de acciones usaremos /echo para poner lineas de texto en pantalla. Vamos a ver como conseguiriamos hacer que la respuesta del PING nos fuera mostrada como hemos visto antes, debemos escribir en los "remotes":

```
on 1:CTCPREPLY:*PING*:{
 %lag = $ctime - $2
 echo –s Lag con $nick : %lag
 halt
}
```

Lo que hemos hecho es primero calcular el lag basándonos en a información que nos devuelve el nick al que le hemos hecho el PING. En este caso nos devuelve: "PING 919197981" . ¿Y que es ese numero tan largo? . Ese numero corresponde a una referencia de tiempo, indicada como el numero de segundos transcurridos desde el 1 de enero de 1970 . El instante al que se refiere ese número es el momento en que la persona recibio el PING, por lo tanto si restamos a la hora actual en el formato **$ctime** (que nos devolvera la hora actual como numero de segundos desde el 1 de enero de 1970) de la fecha en la que el nick recibio el ctcp, nos quedará un numero más pequeño y corresponderá al LAG en segundos. Guardamos ese dato en la variable **%lag** y a continuacion, mediante un /echo, ponemos la información en ***Status***, y el comando /halt. Se debe estar preguntando ¿ese halt no parará el proceso del PING y nos dejará sin ver la información? La respuesta es no, puesto que cuando este evento "salta" la información del PING ya nos ha sido devuelta por la otra persona, así que en este tipo de eventos el /halt al final lo único que hará será evitar que veamos, además del mensaje que hemos especificado, el que ya había por defecto. Pruebe ese ejemplo, y después pruebelo de nuevo suprimiendo el /halt para que vea usted mismo a que me refiero.

Documento escrito por TeMpEsT · www.relativo.com · tempest@mixmail.com

USUARIOS REMOTOS

Ya le he dado una pista antes de que son los usuarios remotos... nos referimos a un usuario remoto cada vez que especificamos un evento remoto , por ejemplo en el evento "on 1:JOIN" le estamos diciendo al **mIRC** "cuando un usaurio de nivel 1 entre a un canal..." . Ésta es una sección completamente opcional y no necesariamente todos los scripts harán uso de ella, puesto que sólo es realmente útil para ciertas tareas muy específicas. Antes que nada debe saber, que por defecto, nivel 1 quiere decir "todos los usuarios", es por eso que hasta ahora todos los eventos remotos se han declarado con "on 1:..." para que tengan efecto sobre todos los usuarios, pero podría darse la ocasión en que usted quiera que algun o algunos usuarios en concreto tengan acceso a unos eventos y no lo tenga el resto, para ello les tendremos que asignar un nivel.

- Asignación de niveles a usuarios

Para asignar un nivel determinado a un usuario iremos al editor del **mIRC**, pero esta vez a una pestaña que seguramente tendremos en blanco, la pestaña "***USERS***". La sintaxis para declarar que un usuario tiene un cierto nivel es:

<nivel>:<nick>!<user>@<host>/<ip>

por ejemplo:

10:TeMpEsT!*@* 20:Scytale!*@theilax.arrakis.es

Hemos creado dos usuarios remotos, el primero TeMpEsT le hemos dado el nivel '10' y cualquier persona con ese nick tendrá acceso a los privilegios de ese nivel 10, que los especificaremos más adelante. El segundo nivel que hemos es asignado es más especifico porque se lo asignamos a un nick y a una máscara, es decir que el nivel 20 solo lo tendra aquel que su nick sea "Scytale" y su host sea "theilax.arrakis.es", de esa forma nos aseguramos que los provilegios que especifiquemos para el nivel 20 solo los Scytale y no alguna persona que se ponga ese nick. Una cosa importante a recordar es que una persona con nivel 20 tendrá acceso no solo a los privilegios del nivel 20, sino también a los de nivel 19,18,17....etc, es decir que en el ejemplo anterior, Scytale tendria acceso a los eventos de nivel 20, pero tb a los de nivel 10. Si quisieramos que Scytale tuviera acceso únicamente a los eventos de nivel 20, tendríamos que escribirlo de la siguiente forma:

=20:Scytale!*@theilax.arrakis.es

el '=' delante del nivel indica que la persona especificada solo podra acceder a los eventos que marquemos con un nivel 20, es decir que los de nivel 1, o 2, o 10 no los podrá acceder el usuario Scytale.

Pues bien así se asignan los niveles de una forma "estática" es decir, vamos al editor del **mIRC** y los intriducimos a mano, pero tambien podríamos hacerlo de una forma "dinámica" mediante comandos del **mIRC**, digo dinámica porque nos pueden servir estos comandos para más adelante permitir al usuario cambiar el nivel de cierta persona o añadir mas gente con un determinado nivel. Los comandos son:

/auser [-a] <niveles> <nick/host>

Añade un usuario con el nivel o niveles que especifiquemos a la lista de usuarios remotos, si especificamos el parámetro **[-a]** hará que si el usuario ya existía, se le añada el nuevo nivel al que ya tenía. Por ejemplo:

/auser 10 TeMpEsT

Añade a TeMpEsT a la lista de usuarios remotos con nivel 10, si TeMpEsT ya estaba en esa lista, será borrado y sustituido por la nueva entrada

/auser –a 12,13 TeMpEsT

Añade los niveles 12 y 13 a los que ya tenía el usuario TeMpEsT, por lo tanto la sección users quedará asi:

10,12,13:TeMpEsT

En lugar de un nick podriamos haber especificado una máscara con el modelo nick!user@host

/flush [niveles]

Este comando borrará a todos los nicks de niveles especificados que no estén actualmente en ninguno de nuestros canales. Por ejemplo:

/flush 1,2,3

Borrará de a lista de usuarios remotos a todas las personas que tengan nivel 1,2 o 3 y no estén en ninguno de nuestros canales.

/flush

Cuando se especifique este comando sin argumentos borrará todas las entradas (en la pestaña “Users” del **mIRC**) de gente que no esté actualmente en ninguno de nuestros canales

/guser [-a] <niveles> <nick> [tipo]

Este comando trabaja de la misma forma que /auser con la única diferencia de que solo le podemos especificar el nick de la persona y el mIRC mirará su máscara actual y la añadirá al nick, para ello tenemos que especificarle también el [tipo] de máscara que será un número de 0 a 9

/guser 10 TipoX 4

Añade al nick TipoX con nivel 10 y una máscara del tipo 4 (*!*@*.dominio)

/ruser [niveles] <nick / host> [tipo]

Borrará los niveles que especifiquemos del nick o host que especifiquemos, podemos tambien darle solo el nick y especificar el [tipo] de máscara para que el **mIRC** la mire y borre los niveles de los usuarios que tengan esa máscara.

/ruser 10 Ytreme

Borrará el nivel 10 que le hayamos dado al nick Ytreme

/ruser Ytreme

Borrará a Ytreme de la lista de usuarios remotos (o lo que es lo mismo le quitará todos los niveles)

/ruser 25 Ytreme 4

El **mIRC** buscará la información de Ytreme y borrará el nivel 25 de todas las entradas en la lista de usuarios remotos que tengan esa máscara.

/rlevel [-r] <niveles>

Borra a todos los usuarios de la lista de usuarios remotos cuyo primer nivel sea el que especifiquemos en <niveles>. Si usamos el parámetro **[-r]** borrará a todos los usuarios que tengan el nivel <niveles> en cualquier lugar. Partiendo de:

10,12,15:TeMpEsT 12,20:Ytreme

El comando: /rlevel 12 borrará al usuario Ytreme puesto que su primer nivel el 12

/rlevel –r 12

Borrará tanto a Ytreme como a TeMpEsT puesto que tienen el nivel 12 , no importa en que posición

/ulist [< / >] <nivel>

Lista a los usuarios de nivel <nivel> , o bien podemos especificar el parámetro **[< / >]** como “>4” o “<10” . Por ejemplo:

/ulist >10

Lista todos los usuarios cuyo nivel sea menor o igual a 10

/ulist >20

Lista a todos los usuarios remotos cuyo nivel sea mayor o igual a 20

Restricciones en el acceso a eventos

Vista ya la forma en la que asignamos niveles a usuarios, ahora veremos como puede hacer que ciertos eventos solo sean accesibles por usuarios especificos (con un nivel especifico), para ello simplemente cambiaremos ese “1” que soliamos poner en todos los eventos remotos/ctcps por el nivel mínimo que necesitará el usuario para acceder al evento:

on 10:JOIN:#:{ echo –s Ha entrado un usuario de nivel 10 o superior }

Este evento hará que cuando un usuario de nivel 10 o superior entre en un canal en el que usted esté, le salga un mensaje en en ***Status*** avisándole. Pero recuerde que este evento no sólo lo accederan los usuarios de nivel 10, sino los de 20, 30, etc... Pero también podemos hacer que un evento solo sea accesible por los usuarios de un nivel determinado, y no por aquellos usuarios que tengan más nivel, lo haremos mediante el prefijo ‘+’ :

on +5:JOIN:#:{ echo –s Ha entrado un usario de nivel 5 }

De esta forma si entra un usuario con nivel 10 por ejemplo, este evento no se ejecutará puesto que esta restringido a los usuarios de nivel 5 .

Otra forma de restringir el acceso a eventos es mediante el sufijo ‘!’ al nivel, que hará que el evento no se ejecute si fue accionado por usted mismo, por ejemplo:

on 1!:OP:#:{ /echo $nick le dio op a $opnick }

Este evento hará que cuando un usuario le de op a otro nos sea notificado en la ventana activa, a excepción de cuando sea usted el usuario que da op

Y al igual que restringimos el accceso a ciertos eventos también podemos tener eventos de “libre acceso” por todos los usuarios, independientemente del nivel que tengan usando un ‘*’ en lugar del nivel:

```
on *:JOIN:#:{ echo $chan Ha entrado $nick }
```

Ese evento mostrará un mensaje en pantalla cada vez que entre un usuario a un canal, sin tener en cuenta su nivel.

También podemos usar el prefijo '@' para indicarle al script que ese evento solo salte cuando tengamos OP en ese canal, por ejemplo:

```
on @10:JOIN:#:{ op $nick }
```

Este evento hará que cuando un usuario de nivel 10 entre a un canal en el que estemos, y en el que seamos op, le demos op automaticamente. Resumiendo un poco esta última parte en la siguiente tabla de ejmplos:

Evento Accesible por usuarios de nivel:

- **on 100:PART** 100 o superior
- **on +30:DEOP** Únicamente 30
- **on 5!:QUIT** 5 o superior (excepto usted!)
- **on *:JOIN** Todos
- **on @30:NICK** 30 o superior (solo si usted es OP en el canal)

Visto esto, y para finalizar, se va a realizar un ejemplo que mezcle los Usuarios Remotos con lo que vimos al principio de este documento sobre eventos **CTCP**, el objetivo será crear un medidor de LAG, algo que es fundamental en cualquier script.

Ejemplo 1: Creación de un medidor de LAG

Para medir el lag que se tiene con una cierta persona, lo que se hace, como ya sabe, es hacerle a esa persona un CTCP PING, pues bien si usted quiere hallar el lag con usted mismo, es decir su propio lag con el server, lo que tendrá que hacer es como ya habrá adivinado, hacerse un CTCP PING a usted mismo, por lo tano analicemos un poco en que se basará nuestro medidor para que despues nos sea más fácil construirlo:

Primero necesitaremos un TIMER , es decir que cada 'X' segundos el mIRC nos haga automaticamente un PING para hallar nuestro lag.

Despues necesitaremos que la respuesta a ese ping se nos muestre en un formato diferente al habitual y que la respuesta habitual se omita, pero habrá que tener en cuenta que cuando hagamos un ping a nosotros mismos, la respuesta que recibiremos será del tipo "Tu lag es de X segundos", para ello usaremos el evento "**ctcp 1:ping**". El problema es que si en ese evento ponemos una linea que diga "Tu lag es de X segundos" ese ctcp saltará cada vez que alguien nos haga un ping, sea quien sea, y lo que haremos en este ejemplo es que ese evento solo se ejecute cuando la persona que le haga el ping sea usted misma, de esa forma conseguiremos que cada vez que usted se haga un ping (automaticamente mediante el timer) se active el evento ctcp del ping que usted habrá diseñado y que cuando sea otra persona la que le haga un ping, ésta reciba el mensaje estándar. Pasemos ya a ver el código para este ejemplo, y posteriormente lo acabaremos de comentar:

Copie lo siguiente en "Aliases":

```
/medidor {
 /timer 0 30 ctcp $me PING
 /auser 55 $me
}
```

Copie lo siguiente en “Remotes”:

```
ctcp 55:PING:{
 %lag = $ctime - $2
 titlebar Lag: %lag
 halt
}

on 1:CONNECT:{ medidor }
```

Y ya está, lógicamente este es un medidor de lag bastante primitivo en el sentido de que se le pueden añadir muchas más cosas, pero esta es la base y a partir de ella usted podrá ir elaborándo su propio medidor de lag con sus conocimientos.

La explicación es bastante simple, cuando usted conecte a un servidor de **IRC** (salta el evento on 1:connect) se ejcutará el alias “medidor” que iniciará un timer infinito (de ahí el ‘0’ como primer parámetro) que consistirá en hacerse a usted ($me) un CTCP PING cada 30 segundos. Además cuando conecte usted será añadido a la lista de usuarios remotos con nivel 55. Al cabo de 30 segundos se producirá por primera vez ese CTCP PING que pusimos en el timer, y puesto que el ping lo manda usted que es un usuario de nivel 55, saltará el evento “CTCP 55” que calculará el lag y lo mostrará en la barra de título del mIRC mediante el comando “titlebar”, por ultimo se usa un “halt” para que la respuesta predeterminada del mIRC al PING quede escondida. Y obviamente si cualquiero otro usuario le hiciera un ping, como usted será el único con nivel 55, este evento no saltará y por tanto el otro usuario recibirá la respuesta estándar del PING.

Documento escrito por TeMpEsT · www.relativo.com · tempest@mixmail.com

GRUPOS

Los grupos son partes del Script que sólo funcionan cuando se activan, y que no molestan en absoluto cuando están desactivados. Precisamente por eso, los grupos son muy indicados para controlar ciertas aplicaciones o ciertos eventos que queramos que se produzcan sólo cuando nosotros lo deseamos. Por ejemplo el F-Server, el Scan de Clones, los Add-Ons, etc.

Una vez dicho para que los queremos, toca explicar como funcionan y como se programan. Pues bien, los grupos deben progamarse en el apartado de remotes, aunque luego se activen mediante aliases o popups, y deben escribirse según la siguiente estructura:

```
#grupo [on/off]
comando.1
comando.2
...
ultimo comando
#grupo end
```

Donde grupo sea el nombre del grupo que queramos y comandos sean las operaciones que queramos que realice, pudiendo incluir toda clase de eventos, raw y ctcps. Hay que recordar que los grupos forman parte de los remotes, y por lo tanto, deben programarse como tales.

Donde pone [on/off] hay que escoger uno de los dos valores:

on -> Activo

off -> Inactivo

Aunque luego podremos cambiarlos facilmente con los comandos enable y disable.

enable #grupo -> Activa un grupo

disable #grupo -> Descativa un grupo

Para evitar que mIRC nos devuelva un mensage del tipo "Group has been enabled" podemos poner un punto (.) delante del alias. (.enable / .disable)

Si queremos saber si un grupo está activo o no, deberemos usar el IDENTIFICADOR $group:

$group(0) ----------> da el número de grupos

$group(1) ----------> da el nombre del primer grupo

$group(1).status ----------> da el estado on/off del primer grupo

$group(#grupo) ----------> da el on/off del grupo #grupo

$group(#grupo).fname ----> da el nombre del archivo script en el grupo si existe

$group(3).name ----------> representa el nombre del tercer grupo

Pero claro, esto es un IDENTIFICADOR, no un ALIAS, por lo que solo sirve para que mIRC pueda actuar de un modo o de otro en una pregunta condicional (IF-THEN-ELSE). Para saber nosotros si un grupo está o no activo debemos usar el comando /echo o alguno similar.

Ahora que ya hemos visto el funcionamiento de los grupos, podemos mirar un ejemplo real para ver de que manera puede sernos útil cuando scripteamos:

menú canal:

```
Fserver
.Activar: {
 /set %gets $?="Número de Gets" | set %dir $sdir?="Directorio"
 /msg # Mi FileServer está activado. Pulsa !Fserver para acceder a él.
 .enable #Fservice
}
.Desactivar: {
 /msg # Mi FileServer está desactivado.
 .disable #Fservice
 unset %gets %dir
}
```

en remotes:

```
#Fservice off
on 1:text:!Fserver:#:{
 /fserve $nick %gets %dir fserve.txt
}
#Fservice end
```

Este Script haría que, mediante el menú del canal, pudiesemos activar el grupo #Fservice cuando quisieramos activar el Fserver, y desactivarlo cuando quisieramos apagarlo.

Documento escrito por CaiN para #Scripting

VENTANAS PERSONALIZADAS

Introducción

El mIRC nos permite la creación de ventanas personalizadas que no son más que "ventanas" con diferentes atributos (que aprenderemos a asignar y modificar) y que pueden ser usadas para cualquier tipo de acción, generalmente para mostrar informacion al usuario, ya que se hace de una manera más limpia y ordenada que mostrando toda la información en la ventana de ***Status***.

Después veremos que tambien pueden ser usadas para la creación de **GUI's** (Interfaces Graficas de Usuario) mediante las ***picture Windows*** o ventanas de imagen.

Este documento está pensado para que el lector y aprendiz de scriptter tenga un primer contacto con las ventanas personalizadas y sea capaz de crearlas y manipularlas a su gusto. Se han suprimido algunas funciones del comando **window** bien por el escaso uso de estas o porque suponen alguna dificultad añadida que solo la experiencia con el scripting nos ayudaria a superar. De cualquier forma una vez el lector ya domine todos los comandos y eventos expuestos en este tutorial, le remito a la ayuda del **mIRC** (existe una versión en castellano disponible en esta misma página para la versión 5.41, que puede ser bajada desde la sección "*Zona de descarga*") en ella encontrara informacion (de una forma mas resumida, eso sí) de todas las funciones del comando **window**.

En el presente texto se utilizarán las sigientes convenciones para presentar la sintaxis de instrucciones y comandos: los argumentos entre < y > representarán valores que es necesario introducir para la correcta ejecucion del comando, mientras que los argumentos entre [y] son opcionales, y pueden ser omitidos a la hora de ejecutar el comando.

La manera de seguir el documento, y aprender con él, es leerlo de arriba a abajo, siguiendo todos los ejemplos y probandolos todos antes de seguir. Tambien es conveniente que el lector modifique a su gusto los ejemplos que aquí encontrará para que vaya experimentando con los comandos de creacion/modificacion de ventanas personalizadas.

En este tutorial aparecen mucho las palabras "editbox" y "listbox", se refieren, respectivamente, al campo de texto que tienen en la parte inferior ciertas ventanas (como las ventanas de los canales), y a las listas de 'objetos' seleccionables que tienen tambien ciertas ventanas (como la lista de nicks en los canales).

Y, por último, antes de empezar, en este tutorial se presupone que el lector ya es medianamente experto en el uso de Aliases, Popups y Eventos Remotos.

Creación de ventanas

Para crear una ventana personalizada usaremos el comando **window** con la siguente sintaxis:

```
/window [-acdeEhkl[N]noprswx] [+bdelLmnstx] <@nombre> [x y [w h]] [popup]
```

Por ejemplo:

```
/window –ad @miventana 10 10 200 100
```

Introduzca esta orden en **mIRC** y descubrirá que ha creado una nueva ventana de **windows**, verá que su nombre es "miventana" (el símbolo @ debe de especificarse siempre al principio del nombre, esto corresponde al argunento <@nombre> en la sintaxis general). Hemos utilizado dos parámetros en esta orden: -a y -d, esto ha supuesto que se ha mostrado como ventana activa de **windows** y su icono ha aparecido en la barra de tareas de la parte inferior de la pantalla. Observe que al utilizar dos parámetros segidos solo hemos necesitado un símbolo "-" delante de ellos. Los números 10 10 200 100 especifican la ubicación y dimensiones de la ventana (corresponden respectivamente a los parámetros x, y, w, h); los dos prineros deterninan la posición dentro de la pantalla, los dos últimos sus dimensiones (alto y ancho).

La posición de la ventana se basa en un sistema de coordenadas que toma como origen la esquina superior izquierda de nuestra pantalla, y sitúa la esquina superior izquierda de la ventana en el punto indicado, en este caso coordenadas 10,10.

Volvamos a la sintaxis general del comando **window** para analizarla más despacio, resulta un poco imponente pero veremos como examinando cada una de sus partes es bastante más asequible de lo que parece :-).

Existen dos grupos de parámetros que podemos emplear, por un lado [-acdeEhkl[N]noprswx] y por otro [+bdelLmnstx]. De inmediato veremos la utilidad de cada elemento de estos grupos. Una vez escritos los parámetros deseados especificaremos el nombre que queremos dar la la ventana (siempre con @ delante). A continuación no queda sino concretar ubicación y tamaño de la ventana, no es necesario especificar ninguno de los dos, en ambos casos se asumen unos valores por defecto. También tenemos la posibilidad de incluir al final del comando el nombre de un fichero de *popup's* si deseamos que la ventana cuente con sus propios menús emergentes, es decir, aquellos que aparecen pulsando sobre ella con el botón derecho del ratón.

A continuacion puede ver una descripcion de para que sirve la primera tanda de parametros del comando **window** que definiran el estado de la ventana:

-a : Activa la ventana.
-c : Cierra la ventana.
-d : Abre la ventana como ventana de escritorio (se puede acceder a ella desde la barra de ventanas de Windows).
-e : Añade a la ventana un editbox o línea de escritura (como la que tienen en la parte inferior los canales).
-E : Añade a la ventana un editbox de múltiples lineas líneas.
-h : Esconde la ventana.
-k : Esconde el prefijo '@' del nombre de la ventana.
-w : Devuelve el prefijo '@' al nombre de la ventana.
-l[N] : Añade una listbox (lista de elementos) a la ventana. Si se especifica N (opcional) se crea una listbox con N caracteres de largo.
-n : Minimiza la ventana.
-o : Si la ventana es una ventana de escritorio, este parametro la pone y mantiene siempre encima de todas las ventanas (on top).
-u : Desactiva el poner la ventana encima de todas (on top).
-p : Crea una ventana de imagen o "picture window" (esta opción será analizada en un capitulo aparte).
-r : Restaura una ventana despues de minimizarla.
-s : Organiza por orden alfabético la ventana.
-S : Organiza por orden alfabético la listbox de una ventana.
-x : Maximiza la ventana.

Hasta aqui hemos visto la primera tanda de argumentos del comando **window** , no se asuste, la inmensa mayoria de las veces solo usará uno o dos de estos argumentos, pero esta bien que los conozca por si los necesitara para alguna ocasión en especial.

Ahora vamos con la segunda tanda de argumentos del comando, estos definirán la apariencia de la ventana:

+b : Dibuja un pequeño borde alrededor de la ventana.
+d : Ventana sin borde.
+e : Dibuja un borde un poco mas ancho y redondeado.
+l : Ventana de utilidades.
+n : Boton de minimizar.
+s : Se puede cambiar de tamaño.
+t : Con barra de titulo.
+x : Con boton de maximizar.

Después tenemos que indicar el nombre de la ventana a crear/modificar:

<@nombre> : Por ejemplo, @mi_ventana

Por último indicamos (opcionalmente) las coordinadas de la posición inicial de la ventana (x y) así como su longiud y altura (en pixels). w (longitud) h (altura)

[x y [w h]] : por ejemplo, 100 100 300 100

Las coordinadas x y significan la distancia en pixels desde el extremo izquierdo de la pantalla, y desde el extremo superior de la ventana principal del mIRC (¡ojo, que no es lo mismo que el extremo superior del monitor!).

Por ultimo el parámetro [popup] sirve para especificar un archivo que contenga el menú popup de la ventana (que aparecera al hacer click con boton derecho sobre la misma). Se puede especificar un archivo (cuya extension NO sea .ini) o bien el mismo nombre de la ventana (Ej: @mi_ventana) si se quiere implementar el popup dentro de los remotes del script.

Modificando las ventanas

Ahora que el lector ya sabe las posibilidades del comando **window**, veremos, ya creada la ventana, que comandos podemos usar para modificar la informacion que se muestre en ella.

El parámetro [c] esta presente en todos estos comandos, e indica el color principal de la linea que añadirá, borrará, o modificará . En cualquier caso también se peden especificar controles de color (ctrl + k + color), negrita (ctrl + b) y subrayado (ctrl + u) en el parámetro <texto>.

Estos comandos se refieren siempre a actuar sobre la ventana principal, en caso de que queramos actuar sobre una listbox a un lado de la ventana, incluiremos el parametro -l entre el nombre del comando y el parametro [c], por ejemplo /aline -l @mi_ventana lo ke sea... :).

/aline [c] <@nombre> <texto>

Añade la línea <texto> a la ventana <@nombre>
Ejemplo: /aline 4 @mi_ventana Hola mundo!!

/cline [c] <@nombre> <N>

Cambia el color de la línea número <N> al color número [c]
Ejemplo: /cline 4 #ayuda_irc 3

/dline [c] <@nombre> <N>

Borra la línea (o el rango de lineas) numero <N>
Ejemplo: /dline @nombre 1-10

/iline [c] <@nombre> <N> <texto>

Inserta una línea (<texto>) en la posicion <N>
Ejemplo: /iline @mi_ventana 2 Insertando una linea...

/rline [c] <@nombre> <N> <texto>

Sustituye la línea <N> por <texto>
Ejemplo: /rline @mi_ventana 1 Hasta luegoorr!!

/sline [c] <@nombre> <N>

Selecciona la línea numero <N>
Ejemplo: /sline @mi_ventana 3

/renwin <@nombre> <@nuevo_nombre> [topic]

Cambia de nombre la ventana <@nombre> y (opcionalmente) cambia su topic o descripción
Ejemplo: /renwin @mi_ventana @tu_ventana Nuevo_topic

Ahora que el lector ya conoce todos los comandos que se usan para la creacion/modificacion de ventanas, y antes de entrar en los eventos remotos asociados a las mismas, vamos a hacer un par de ejemplos de crear ventanas, para que se vaya familiarizando con estos comandos y compruebe por usted mismo que no es tan complicado como quizás parece en un principio:

Ejemplo 1: Crear una ventana para dialogar con el usuario

Vamos a crear una ventana que mostrará una informacion al usuario y esperara que éste le responda escribiendo en la editbox de la ventana:

Para ello escribimos en la sección "Aliases" del Editor del mIRC:

```
/vent {
 window -ae @dialogo 50 50 200 100 @dialogo
 aline 12 @dialogo Hola, ¿Cómo te llamas?
}
```

Si prueba ese ejemplo (escribiendo el alias "/vent"), verá que se crea una ventana de nombre @dialogo con un editbox abajo y nos aparece en ella el texto "Hola, ¿Cómo te llamas?". En este momento nos tendremos que detener aqui, más adelante veremos como puede hacer que al teclear algo en la editbox y pulsar 'Enter' , el script le responda adecuadamente.

Ejemplo 2: Ver un listado de archivos y mostrar informacion de los mismos

En este ejemplo crearemos una ventana con una listbox, en la que apareceran los archivos mp3 de nuestro directorio de mp3 (se supondrá que es c:\mp3), y al hacer doble click sobre uno de los archivos se nos mostrara informacion de ese archivo en la ventana principal. Escribiremos el siguiente código en la seccion de "Aliases" en el Editor del mIRC:

```
/mp3 {
   window -l20 @mp3 50 50 600 200 @mp3
   %i = 0
   : comienzo
   inc %i 1
   if (%i > $findfile(c:\mp3,*.mp3,0)) { goto fin }
   else {
         aline -l @mp3 $nopath($findfile(c:\mp3,*.mp3,%i))
         goto comienzo
      }
      : fin
      unset %i
}
```

En el alias /mp3 hemos hecho lo siguiente: Primero creamos la ventana @mp3 con una listbox (de tamaño 20 caracteres) y en la posicion x (50), y (50) y con un tamaño de 400 pixels de largo por 200 de alto.

Despues creamos una variable temporal %i y le damos el valor '0'. Iniciamos un bucle cuya explicacion no forma parte de esta sección así que se explicará solamente en lineas generales lo que hace: incrementa %i en 1 y comprueba si el valor de %i es mayor que la cantidad de mp3s en el directorio c:\mp3, en caso de que %i sea mayor (querrá decir que ya no hay mas mp3) vamos al indicador ": fin" que dejara libre la variable temporal %i , y en caso de que el valor de %i sea menos que la cantidad de archivos mp3 en c:\mp3 (quiere decir que aun hay al menos un mp3 más), se añade una línea a la listbox de la ventana con el nombre del archivo (sin la ruta, de ahi el **$nopath()**)

El resto de funcionabilidad de la ventana, el mostrar informacion del archivo al hacer doble click sobre una de las lineas, la veremos más adelante cuando hayamos visto los eventos remotos.

Identificadores de ventanas

Estos identificadores pueden ser usados en cualquier Alias, Popup o Evento Remoto para recoger informacion sobre una determinada ventana:

$line(@nombre,N,[T])

Devuelve el contenido de la linea N de una ventana. Si N es '0' devuelve el numero total de lineas en la ventana. El parámetro [T] es opcional, y se usa solo cuando nos referimos a una listbox en un lado de la ventana, para ello le daremos el valor '0' a T . Si nos referimos a una ventana sin listbox el parámetro [T] se omite, y si nos referimos a una ventana con listbox, pero queremos información de la ventana principal y no de la listbox, pondremos en lugar de [T] un 1.

Ejemplo: //echo -s $line(@mp3,2,0)
Devolverá el contenido de la segunda linea en la listbox de la ventana @mp3

$sline(@nombre,N)

Devuelve el contenido de la linea seleccionada numero N (solo funciona en una listbox). Si le damos el valor '0' a N, devuelve el numero total de lineas seleccionadas en la listbox.

$sline(@nombre,N).ln

Devuelve el numero de la línea seleccionada numero N (N normalmente se pone '1')
$window(@nombre).<propiedad>

Devuelve las propiedades de una ventana. Los valores posibles de <propiedad> son:

$window(@nombre).x : Devuelve la coordinada x de la posicion de la ventana.

$window(@nombre).y : Devuelve la coordinada y de la posicion de la ventana.

$window(@nombre).w : Devuelve la longitud en pixels de la ventana.

$window(@nombre).h : Devuelve la altura en pixels de la ventana.

$window(@nombre).dx : Devuelve la coordinada x de la ventana con respecto al escritorio (no a la ventana del mIRC).

$window(@nombre).dy : Devuelve la coordinada y de la ventana con respecto al escritorio (no a la ventana del mIRC).

$window(@nombre).state : Devuelve el estado de la ventana (minimized, maximized, normal o hidden).

$window(@nombre).title : Devuelve el texto en la barra de titulo de la ventana.

$window(@nombre).font : Devuelve el nombre del tipo de letra actual en la ventana.

$window(@nombre).fontsize : Devuelve el tamaño del tipo de letra actual en la ventana.

$window(@nombre).fontbold : Devuelve $true si el tipo de letra esta en negrita, o $false si no lo esta.

Popups en ventanas

Para definir el popup de una ventana personalizada sin necesidad de crearlos en un fichero aparte podemos editarlos dentro de la sección Remotes del editor del **mIRC**, la forma de hacerlo es la habitual de los *popup's*. Tan solo es necesario encerrarlos entre corchetes encabezar todo con la clausula MENU <@nombre_ventana>.

```
MENU @nombre {
   Popup1
   .Sub-popup: /accion1
   .Sub-popup2: /accion2
   -
   Popup2: /accion3
   .....
}
```

Eso sí, previamente tendremos que haber especificado en la definición de la ventana que queremos definir en popup en los remotes escribiendo el nombre de la ventana como el parámetro [popup]:

Ejemplo: /window –a @nombre 10 10 100 100 @nombre

También podemos definir el popup de varias ventanas a la vez poniendo:

```
MENU @nombre1,@nombre2,@nombre3 {
   ... popup ...
}
```

Donde he puesto "...popup..." se ha de poner una secuencia de lineas con el formato normal de un Popup. Para que se entienda mejor el uso de esta técnica aquí va un ejemplo que debe ser copiado en la sección “Remotes” del editor del mIRC:

```
alias miventana {
   window -a @mi_ventana 10 10 200 200 @mi_ventana
}
```

Esto anterior tambien se puede hacer dentro te la sección de ALIAS poniendo:

/miventana /window -a @mi_ventana 10 10 200 200 @mi_ventana.

Y ahora ya sí, en remotes necesariamente:

```
menu @mi_ventana {
   Decir Hola
   .En rojo : /aline 4 @mi_ventana Hola!
   .En negro: /aline 1 @mi_ventana Hola!
   Decir Adios
   .En Azul: /aline 12 @mi_ventana Adios!
   .En Verde: /aline 3 @mi_ventana Adios!
   -
   Cerrar Ventana: /window -c @mi_ventana
}
```

El resultado de crear este codigo es que al teclear */miventana* nos aparecerá una ventana personalizada en blanco y que, como le hemos especificado, usará como popup el menú @mi_ventana . Asi pues en el remote hemos declarado el "menu @mi_ventana" y le hemos escrito unos popups de la manera habitual. Si pulsamos el boton derecho sobre la ventana que hemos creado, veremos como aparece ese popup que hemos creado.

Eventos más usados

```
on 1:INPUT:@nombre:{
   comandos
}
```

Este evento solo sirve para las ventanas que creemos que dispongan de una editbox, ya que el evento INPUT "salta" cuando introducimos texto en una editbox y pulsamos 'Enter'.

También podemos usar el evento ON CLOSE:

```
on 1:CLOSE:@nombre:{
   comandos
}
```

Los comandos que especifiquemos se ejecutaran cuando el usuario cierre la ventana, ya sea mediante un popup, o mediante el boton de cerrar (la 'X') de toda ventana de Windows.

Y análogamente tambien disponemos del evento:

```
on 1:OPEN:@nombre:{
   comandos
}
```

Estos comandos se ejecutarán cuando se abra la ventana indicada.

Con lo que hemos visto hasta ahora, y un poco de práctica, el lector debe ser ya capaz de crear sofisticadas ventanas personalizadas, modificar sus atributos, añadirles menús emergentes, y ejecutar comandos cuando suceda algun evento en las mismas (como abrir o cerrar la ventana). A continuacion vamos a repetir, y esta vez a acabar, los dos ejemplos que se expusieron anteriormente, ya que ya se está en condiciones de poder darles toda su funcionabilidad con lo que sabemos hasta ahora:

Ejemplo 1: Crear una ventana para dialogar con el usuario

Primero copiamos el codigo que ya habiamos hecho antes en los Aliases:

```
/ventana {
   window -ae @dialogo 50 50 200 100 @dialogo
   aline 12 @dialogo Hola, ¿Cómo te llamas?
}
```

Ahora nos vamos a la seccion "Remotes" y copiamos el siguiente codigo:

```
menu @dialogo { Salir: window -c @dialogo }

on 1:INPUT:@dialogo:{
   %vari01 = me llamo
   %vari02 = como estás?
   if ( %vari01 isin $1- ) { /aline 12 @dialogo Hola $3 !! }
   elseif ( %vari02 isin $1- ) { /aline 4 @dialogo Estoy bien gracias, y tu? }
   elseif ( estoy isin $1- ) { /aline 3 @dialogo Pues me alegro... }
   elseif ( idiota isin $1- ) { /aline 10 @dialogo Idiota tu !!!! }
   elseif ( adios isin $1- ) { /aline 5 @dialogo Nos vemos! | window -c @dialogo }
   else { /aline 6 @dialogo Lo ke tu digas... }
   halt
}
```

Primero hemos creado la ventana @dialogo con el comando /ventana, a esta ventana le hemos indicado que ha de tener una editbox (-e) y por lo tanto nosotros podremos escribir en ella. Dependiendo de lo que escribamos, el script nos responderá de una forma u otra gracias al evento ON INPUT que controla el texto que escribamos en la ventana especificada. Creo que el código esta bastante claro asi que al lector solo le queda copiarlo en el editor del mIRC, probarlo y modificarlo a su gusto.

Ejemplo 2: Ver un listado de archivos y mostrar informacion de los mismos

Primero copiamos el código que ya teniamos en los "Aliases":

```
/mp3 {
   window -l20 @mp3 50 50 600 200 @mp3
   %i = 0
   : comienzo
   inc %i 1
   if (%i > $findfile(c:\mp3,*.mp3,0)) { goto fin }
   else {
         aline -l @mp3 $nopath($findfile(c:\mp3,*.mp3,%i))
         goto comienzo
      }
      : fin
      unset %i
}
```

Ahora vamos a los "Remotes" y copiamos:

```
MENU @mp3 {
   dclick: {
         aline 4 @mp3 $findfile(c:\mp3,*.mp3,$sline(@mp3,1).ln)
         aline 4 @mp3 $lof($findfile(c:\mp3,*.mp3,$sline(@mp3,1).ln)) bytes
         aline 4 @mp3 $duration($calc($lof($findfile(c:\mp3,*.mp3,$sline(@mp3,1).ln))
         / 16000))
      }
      Recargar mp3: window -c @mp3 | /mp3
      -
      Salir: window -c @mp3
}
```

Supongo que la principal pregunta al leer este codigo es: ¿Que es "dclick"?. Eso se explicará más adelante, cuando veamos las ventanas de imagen (picture windows), de momento ha de saber que el contenido de "dclick" se ejecuta cuando hacemos doble click sobre una ventana de imagen o (como en este caso) sobre una listbox. Es decir que cuando hagamos doble click sobre alguno de los objetos de la listbox (que seran nombres de los mp3 que tengamos en el directorio c:\mp3) nos aparecerá en la ventana principal la ruta completa del archivo en la primera línea, el tamaño de ese archivo (usando el identificador **$lof**) en la segunda línea y la duración aproximada del mp3, que se consigue dividiendo el tamaño del mismo por 16.000, en la tercera. Despues hemos añadido dos opciones al menú popup de la ventana, la primera “Recargar Mp3” consiste en cerrar la ventana, y volverla a abrir (usando el comando que habiamos creado anteriormente de nombre /mp3).

Bien, vistos estos dos ejemplos, usted ya sabe lo suficiente como para hacer ventanas personalizadas muy complejas y útiles, aunque si este es su primer contacto con las ventanas personalizadas quizas sea una buena idea que antes de ponerse a hacerlas, vea mas ejemplos de como se hacen en algún script.

Documento escrito por TeMpEsT · www.relativo.com · tempest@mixmail.com

VENTANAS DE IMAGEN

Introducción

Como se ya se describió en el capítulo "*ventanas personalizadas*" el parámetro **–p** del comando **window** hace que creemos una ventana de imagen. ¿Qué es una ventana de imagen? Pues básicamente es lo mismo que una ventana personalizada normal, con la diferencia de que en ésta en vez de añadir/borrar/modificar líneas de texto, lo que haremos sera dibujar puntos, lineas, figuras o incluso mostrar imágenes con formato **.BMP**. Así pues las *picture windows* no son más que tipo especial de ventanas personalizadas, y este capítulo lo dedicaremos integramente a estudiar su creación, manejo y posibilidades.

Pasando ya a la práctica, un ejemplo de cómo crear una ventana de imagen podria ser el siguente:

```
/window –p @miventana 10 10 200 200 @miventana
```

Estaa línea de código hará que se cree una ventana de imagen en blanco de nombre @miventana (recuerde que las ventanas personalizadas siempre llevan en su nombre el prefijo '@'), en las cooridnadas x (10) y (10), de 200 pixels de largo por 200 de alto, y que además use los menús popup que le especifiquemos bajo la clausula '*menu @miventana*' en la seccion de "*Remotes*" del editor del mIRC. Hasta aquí no hacemos más que repetir lo que se explicó en el capítulo anterior, la diferencia es que esta es una ventana de imagen y nos permitirá aplicar tratamientos y procesos gráficos.

Una vez creada la ventana ahora lo importante, y lo que la diferencia del resto, son los comandos u modificadores que podemos emplear en ella. Con los comandos que a continuación se explicaran el lector sera capaz de dibujar figuras, puntos, y lineas; poner texto en cualquier lugar, y mostrar imágenes **.BMP** tambien en cualquier posicion dentro de la ventana.

Antes de empezar con este tema quiero que quede claro a qué me refiero cuando, más adelante, em`pleando la palabra "rectángulo". La especificación del rectángulo son 4 números que determinan la posición y tamaño de la ventana. Los dos primeros aluden a la posición en que se encontrará su esquina superior izquierda. El primero (x) será la distancia en pixels desde el borde izquierdo de la pantalla, el segundo (y) es la distancia desde el borde superior. Los dos sigueinte números definen el tamaño de nuestra ventana: ancho (w) y alto (h), siempre usando el pixel como unidad de medida, por lo cual las dimensiones reales dependerán del tamaño y definición de su monitor.

Por ejemplo un rectángulo cuya esquina esté en las coordenadas x = 30, y = 5, y mida 100 pixels de largo por 120 de alto lo expresaremos como: 30 5 100 120.

Espero que haya quedado eso claro porque es fundamental para el entendimiento de la siguiente sección...

Comandos para modificar imagenes

Estos comandos, igual que pasaba con los de las ventanas personalizadas "normales" se han de usar despues de haber creado la ventana, y pueden ser insertados en cualquier Alias, Popup o Remote que declaremos.

/drawdot [–hnri] @nombre <color> <tamaño> <x y> [x y ...]
Dibuja un punto del color indicado (del 0-15) con un diametro (tamaño en pixels) y en las coordinadas x y dentro ventana de imagen.

No confundir estas coordenadas x e y con las coordenadas equivalentes que usamos al crear la ventana, en este caso aluden a la posición en que se dibujará el punto dentro de la misma. Corresponden a una escala en pixels que también parte de la esquina superior izquierda de la ventana creada, pero cuyos ejes son ahora el borde superior e izquierdo de la misma (no de la pantalla).

Es decir, que si dibujamos un punto en las coordinadas x(0) y(0) tendremos un punto cuyo centro estara en la misma esquina superior izquierda de la ventana de imagen.

Se pueden especificar multiples coordenadas 'x,y' en la misma orden, esto es, si queremos dibujar varios puntos con el mismo comando.

Los parametros opcionales [-hnri] sirven para lo siguiente:

-h Hará que el icono de la ventana parpadee en el momento de dibujarse el punto si se encuentra minimizada.

-n Hace que la ventana no se actualice inmediatamente. Esto podria ser util si antes de dibujar el punto queremos cambiar el color del fondo o algo similar, aunque realmente eso se conseguiria mas facilmente poniendo el comando que cambiara el color de fondo antes que el que dibujara el punto asi que este parametro rara vez lo usaremos...

-r Indica que el <color> esta especificado en formato RGB (Rojo,Verde,Azul). en caso de que usemos este parametro tendremos que utilizar el identificador:

$rgb(rojo,verde,azul)

por ejemplo: $rgb(0,200,100)

-i Dibujara el punto en modo inverso (color blanco y fondo negro).

Ejemplo:

/drawdot –r @miventana $rgb(255,23,45) 5 10 10 12 10

Este ejemplo dibujara 2 puntos en la ventana @miventana del color definido por el valor $rgb(255,23,45) , con un diametro de 5 pixels.

/drawline [-hnri] @nombre <color> <tamaño> <x y> <x y> [x y...]

Dibuja una línea del color <color>, que tenga un grosor de <tamaño> pixels y que vaya desde las primeras coordinadas <x y> que especifiquemos hasta las segundas <x y>. Se pueden especificar opcionalmente más parametros <x y> para hacer una línea que pase por los puntos definidos. Los parámetros [-hnri] hacen exactamente lo mismo que en el comando **drawdot**, y de hecho comprobaremos que a casi todos los comandos de modificacion de ventanas de imagen se les pueden aplicar estos parámetros

Ejemplo:

/drawline @miventana 4 10 20 0 20 100

Este ejemplo dibujará una linea en @miventana de color rojo (4) y grosor 10 pixels, que irá desde las coordenadas x(20) y(0) hasta x(20) y(100) . Es decir será una linea vertical.

/drawrect [-hnrifec] @nombre <color> <grosor> <x y w h> [x y w h..]

Dibuja un rectángulo del color <color> cuyo borde tenga un grosor de <grosor> pixels, cuya esquina superior izquierda se encuentre en las coordinadas <x y> especificadas, y que mida <w> pixels de largo por <h> de alto. Lógicamente si incluimos un segundo juego de parámetros [x y w h] creará un segundo rectangulo con esas características. Los parámetros [-hrni] una vez más son los mismos que los explicados en el comando **drawdot**, los otros sirven para lo siguiente:

-f Una vez dibujado el rectángulo lo rellena con el color que hayamos especificado para el borde, si no especificamos este parámetro el rectangulo por dentro será del color del fondo.

-e Dibuja una elipse en lugar en un rectángulo... ¿que como dibuja una elipse si lo que le estamos dando son las coordenadas de un rectangulo? Pues simplemente dibuja la elipse que cabría perfectamente dentro de ese rectángulo que especificamos con <x y w h>.

-c Hace que el borde del rectangulo sea del color que le hemos especificado, pero transparente.

Ejemplo:

/drawrect –fr @miventana $rgb(1,2,3) 10 30 30 200 200

Este ejemplo dibujará un rectángulo de color $rgb(1,2,3) , ya que le hemos especificado el parámetro –r , que estará además relleno con ese color, y su esquina superior izquierda estará en la posición x(30) y(30) y medirá 200 pixels de largo por 200 de alto.

/drawfill [–hnris] @nombre <color1> <color2> <x y> [archivo.bmp]

Rellenará el área en el que esté el punto <x y> con el color <color1>. Podemos especificar que en vez de rellenar el área con un color se haga con una imagen **BMP** que tengamos. La imagen ha de ser del tamaño 8x8 pixels, si tiene cualquier otro tamaño no funcionará. Los parámetros [-hnri] cumplen la misma función que en los comandos anteriores.

La funcion del parámetro <color2> depende de si especificamos o no el parámetro [-s]. Si ponemos -s el valor de <color2> indicará el color que deberá ser sustituido por <color1>. Si no incluimos -s el valor que le demos a <color2> será el color ante el cual el relleno deberá parar.

Este comando resulta un poco complicado de explicar, asi que lo mejor será que se fije en el siguiente ejemplo:

Ejemplo:

/drawfill @miventana 1 4 30 30 c:\imagen.bmp

Este ejemplo lo que hará es que usando la imagen imagen.bmp (cuyo tamaño es de 8x8 pixels) y desde la posicion x(30) y(30) pegara multiples copias de esa imagen para rellenar esa zona, parando ante cualquier línea de color rojo, si hubiera alguna.

/drawtext [-hnrpboc] @nombre <color1> [color2] [Tipo_letra] [Tamaño_letra] <x y [w h]> <texto>

Inserta un texto del color <color> en la ventana que especifiquemos y en las coordinadas <x y>. Podemos añadir opcionalmente la longitud y altura del texto (parametros [w h]), esto hará que si el texto que escribimos es demasiado largo y no cabe en el rectángulo que hemos indicado con [w h] aparezca solo el trozo que quepa.

El parámetro [color2] es opcional y sirve para especificar el color del fondo del texto que escribamos. Los parámetros [Tipo letra] y [Tamaño letra] también son opcionales e indican el nombre del tipo de letra a usar (escribirlo todo junto, sin espacios) y su tamaño en puntos. [-hnr] tienen la misma funcion que en anteriores comandos. En cuanto a los otros parámetros:

-p Nos permite el uso de controles de color (ctrl + k) , negrita (ctrl + b) y subrayado (ctrl + u) dentro del texto.

-b Indica que se va a especificar el parámetro [color2] como color de fondo para el texto. Si no usamos este parámetro, [color2] deberá ser omitido.

-o Indica que el tipo de letra elegido debe ser en negrita.

-c Indica que los parámetros opcionales [w h] van a ser especificados.

Ejemplo:

/drawtext –b @miventana 4 9 MSSansSerif 14 30 40 Probando el comando drawtext

Este ejemplo pondrá en pantalla la cadena de caracteres “Probando el comando drawtext” de color rojo (4) y sobre un fondo verde claro (9), con el tipo de letra Ms Sans Serif (recuerde que en el comando se ha de escrbir el nombre del tipo de letra todo junto, sin espacios), de un tamaño 14 puntos.

/drawcopy [-ihnt] @nombre [color] <x y w h> @destino <x y [w h]>

Copia parte de una ventana de imagen a otra parte de la ventana o a otra ventana. Si especificamos los parámetros [w h] la seccion de la imagen que hayamos copiado será ensanchada/estrechada a ese tamaño (el especificado por [w h]) en la ventana de destino. Fíjeses que si lo que quiere es copiar un trozo de una imagen desde una seccion de una ventana a otra sección de la misma ventana, @nombre y @destino serán la misma ventana. Los parámatros [-ihn] son los mismos que explicamos en /drawdot . y en cuanto a **–t** indica que hemos especificado el valor [color] como un valor **$rgb** equivalente al color que queremos que sea transparente en la imagen que hayamos copiado.

Ejemplo:

/drawcopy @miventana 0 0 100 100 @miventana2 10 10

Este ejemplo copiara el contenido de la ventana @miventana contenido desde x(0) y(0) hasta x(100) y(100) a @miventana2 en la posicion x(10) y(10)

/drawsave @nombre <archivo.bmp>

Guarda la imagen de fondo actual de la ventana @nombre como un archivo de nombre <archivo.bmp>

Ejemplo:

/drawsave @miventana ventanita.bmp

Este ejemplo guaradará la imagen de fondo que tengamos en la ventana @miventana en un fichero con el nombre ventanita.bmp.

/drawscroll [-hn] @nombre <x> <y> <x y w h>

Desplaza la región de la ventana comprendida en el rectángulo <x y w h> (recuerde: x – posición x de la esquina superior izquierda, y – posición y de la esquina superior izquierza, w – longitud del rectángulo, h – altitud del rectangulo, todo en pixels) <x> pixels en el eje X y <y> pixels en el eje Y. Los valores de <x> e <y> pueden ser un número negativo si queremos indicar que la región se desplaze hacia la izquierda o hacia abajo respectivamente. [-hn] son los mismos que en el comando **drawdot**

Ejemplo:

/drawscroll @miventana 10 20 50 50 200 200

Este ejemplo desplazará la región de la ventana @miventana contenida en el rectángulo “50 50 200 200” 10 pixels hacia la derecha y 20 hacia arriba

/drawpic [–ihntsc] @nombre [color] <x y [w h]> [x y w h] <archivo.bmp>

Y llegamos sin duda al comando más útil de todos los de las ventanas de imagen, con este comando podremos cargar una imagen cualquiera <archivo.bmp> (con formato .bmp) en una ventana, en las coordinadas que indiquemos <x y> . Si indicamos la longitud y altitud [w h] la imagen sera ensanchada/estrechada a ese tamaño. El parámetro opcional [x y w h] lo usaremos para indicar qué parte del archivo <archivo.bmp> queremos mostrar, útil por si tenemos un bmp grande con varias imágenes y queremos mostrar solo una de ellas. [-ihn] corresponden a los parámetros explicados en **drawdot**. Otros parámetros que acepta este cmando son:

-t Indica que hemos especificado el valor [color] con el fomato $rgb(N,N,N) donde N es un numero comprendido entre 0 y 255, y ese color será tratado como transparente en el archivo que queramos mostrar.

-s Indica que hemos especificado los parámetros [w h] para ensachar/estrechar la imagen.

-c Indica que la imagen debe ser puesta en la memoria caché, esto hace que si en la misma sesión quisieramos volver a hacer uso de esa imagen, el tiempo de carga sería muy inferior. La caché puede almacenar hasta un máximo de 30 imágenes, a partir de ese número empezaran a reemplazarse las que lleven más tiempo sin usarse por las más nuevas.

Ejemplo:

/drawpic –t @miventana $rgb(0,0,0) 0 0 c:\ventanita.bmp

Este ejemplo mostrará la imagen c:\ventanita.bmp en la ventana @miventana en las coordinadas x(0) y(0) y las regiones de la imagen de color $rgb(0,0,0) (negro en este caso) se mostrarán como transparentes (se verá el fondo de la ventana a través de esas regiones).

Con esto acabamos con los comandos para la modificación de ventanas de imagen. Como habrá podido comprobar el dominio de las coordenadas x e y es imprescindible, y también la unidad de medida de tamaño de cualquier gráfico por ordenador, el pixel, para especificar los tamaños de las figuras y/o imágenes. Esto sólo se consigue mediante práctica, y los que anteriormente hayan usado un programa de diseño gráfico como **Corel Draw** o **Photoshop** ya tendrán algo de camino andado en este tema.

Seguidamente veremos los identificadores que nos devolverán información específica de una ventana de imagen.

Identificadores

$mouse.<propiedad>

$mouse.win : Devuelve el nombre de la ventana sobre la que se encuentra el raton.
$mouse.x : Devuelve la posicion x , relativa a la ventana de imagen, del raton.
$mouse.y : Devuelve la posicion y , relativa a la ventana de imagen, del raton.
$mouse.mx : Devuelve la posicion x relativa a la ventana principal del mIRC.
$mouse.my : Devuelve la posicion y relativa a la ventana principal del mIRC.
$mouse.dx : Devuelve la posicion x relativa al escritorio.
$mouse.dy : Devuelve la posicion y relativa al escritorio.

$rgb(nº,nº,nº)

Este identificador ya lo hemos usado en la explicación de los comandos, y sirve para especificar un color, pero con más detalle. Es decir que en vez de usar el color con un número del 0 al 15, los indicaremos suministrando los valores de rojo, verde y azul del color(RGB). Si no controlas el RGB no te preocupes siempre puedes poner el color el el formato habitual (numero del 0 al 15).

$getdot(@nombre,x,y)

Devuelve el valor RGB del color del punto definido por los parámetros ‘x’ e ‘y’.

$inrect(x,y,x2,y2,w,h)

Devuelve **$true** si el punto definido por x,y esta dentro del rectángulo definido por x2,y2,w,h . En caso contrario devuelve el valor **$false**.

$height(texto,tipo_letra,tamaño)

Devuelve la altura en pixels del texto especificado y con un tipo de letra y tamaño. Recuerda que el tipo de letra lo haa de escribir todo junto, por ejemplo: TimesNewRoman.

$pic(archivo.bmp)

Este identificador se puede usar de 3 formas:

$pic(archivo.bmp).size : Devuelve el tamaño del .BMP especificado.
$pic(archivo.bmp).height : Devuelve el tamaño del .BMP especificado.
$pic(archivo.bmp).width : Devuelve el tamaño del .BMP especificado.

$width(texto,tipo_letra,tamaño,B,C)

Devuelve la longitud en pixels del texto especificado y con un tipo de letra y tamaño. Si el parámetro ‘B’ es diferente de ‘0’ se tomará el tipo de letra en negrita. Si el parámetro ‘C’ es diferente de ‘0’ se ignorarán el espacio ocupado por los controles de color, negrita y subrayado.

Eventos y remotes

Con las ventanas de imagen podemos usar los mismo eventos y manejo de remotes que empleabamos en el resto de ventanas personalizadas, como por ejemplo el evento **ON INPUT** (si la ventana contaba con una editbox) o los eventos **ON OPEN** y **ON CLOSE** que se ejecutaban cuando abriamos o cerrábamos la ventana en cuestión.

En lo que se refiere a ventanas de imagen, no existe ningún evento remoto para ellas en especial, seguiremos usando por tanto los ya vistos en el capítulo anterior, pero eso sí, a la hora de definir menús Popup dentro de la sección Remotes para una ventana de imagen, sí que podrá incluir nuevas funciones que ayudarán a sus ventanas a ser un poco más útiles y sofisticadas.

Por ejemplo, usted podrá hacer que al pulsar en cierta región de una imagen pase una cosa, y al pulsar en otra región pase otra cosa. Vayamos por partes, como he dicho antes la nueva funcionalidad de las ventanas de imagen se implementará donde en las ventanas personalizadas normales implementábamos el menu Popup. Por ejemplo si creamos la siguiente ventana:

```
/window –p @otraventana 100 100 100 100 @otraventana
```

Y queremos definir su menu popup, escribiremos en la sección Remotes:

```
menu @otraventana {

}
```

Y aquí empiezan los cambios. Por supuesto, es perfectamente posible especificar el menú popup que queremos para esa ventana dentro de los corchetes pero eso deberá ser puesto “al final”. Y me explico: antes de escribir el menú popup podremos incluir una serie de "gatillos" que saltarán cuando ocurra cierto evento. A continueación se detallan cuales son estos “gatillos” que usted podrá especificar dentro de la clausula “menu @otraventana”:

mouse: Saltará cuando el ratón se mueva por encima de la ventana.

sclick: Saltará cuando el usuario haga click con el botón izquierdo sobre la ventana.

dclick: Saltará cuando el usuario haga click con el botón derecho sobre la ventana.

uclick: Saltará cuando el usuario suelte el botón izquierdo del ratón.

lbclick: Saltará cuando se haga un click sobre un objeto de una listbox.

leave: Saltará cuando el ratón salga de a ventana (se mueva fuera de la ventana). Cuando usemos este gatillo, podemos usar el identificador $leftwin que nos devolverá el nombre de la ventana de la que salió el ratón.

drop: Saltará cuando el usuario haga click con botón izquierdo sobre la ventana, mantenga el botón pulsado, mueva el ratón y después lo suelte otro lugar.

Antes de liarle más, le presentaré un ejemplo de cómo podría usar estos gatillos para que le quede un poco mas claro:

```
menu @otraventana {
   mouse: /echo –s El ratón se ha movido hasta $mouse.x $mouse.y
   sclick: /echo –s Ha hecho click en las coordenadas $mouse.x , $mouse.y
   dclick: /echo –s Ha hecho doble click sobre las coordenadas $mouse.x , $mouse.y
   uclick: /echo –s Ha soltado el boton en las coordenadas $mouse.x , $mouse.y
   leave:{
         echo –s Ha salido de la ventana $leftwin
         window –c $leftwin
      }
      Popup 1
      .sub-popup1: /comando1
      .sub-popup2: /comando2
      Popup2: /comando3
      -
      Popup3: /comando4
}
```

Ahora fíjese bien en el código que acaba de leer. Hay unas cosas importantes que deberían de quedar claras con ese ejemplo:

- Como ha visto los gatillos de una ventana de imagen se especifican dentro de la clausula “menu @otraventana” y siempre ANTES que el menu popup.

- El menu popup de la ventana se especifica, normalmente, de la misma forma que en las ventanas personalizadas normales (después de los "gatillos").

- Se puede hacer uso de los identificadores **$mouse.x** y **$mouse.y** para hallar las coordenadas en las que se encuentra situado el ratón (siempre relativas a la ventana de imagem).

- Se pueden incluir varios comandos para un mismo gatillo usando llaves { } como hemos hecho en el ejemplo del gatillo “leave”.

Aunque le parezca increíble, con lo que se ha explicado hasta ahora ya se puede hacer cualquier cosa que haya visto en cualquier script que tenga que ver con ventanas de imagen. El uso de estas ventanas no es sencillo, y se hace verdaderamente muy pesado, así que sería conveniente que el lector se asegurará de si en realidad va a valer la pena el hacer una ventana de imagen para algo que quizás un simple menu popup podría solucionar. De cualquier forma a continuación se va a exponer y explicar un ejemplo que aunque tenga una escasa utilidad servirá para que pueda ver todos estos comandos e identificadores en acción. En este ejemplo se da por hecho que los conocimientos del lector sobre Aliases, Remotes y popups son suficientes.

Ejemplo 1: Crear una imagen interactiva

Para ello necesitaremos, primeramente, una imagen, usaremos la siguiente:

Esta en concreto tiene unas dimensiones de 100x73 pixels, este valor nos será util más adelante, se supondrá que la imagen se encuentra en c:\pregunta.bmp.

Ahora, como ya habrá adivinado, lo que haremos será transformar esa imagen en una ventana de imagen, y hacer que si el usuario pulsa el ratón sobre “SI”, se cierre el mIRC y, si por el contrario, pulsa sobre “NO”, se cierre la ventana de imagen y que no ocurra nada más. Por lo tanto lo primero será crear un ALIAS que abra la ventana de imagen y cargue en ella pregunta.bmp. Copiaremos lo siguente en la sección “Aliases” del editor del mIRC:

```
/pregunta {
   set %longitud $pic(c:\pregunta.bmp).width
   set %altitud $pic(c:\pregunta.bmp).height
   /window –p +b @pregunta 200 200 %longitud %altitud @pregunta
   drawpic –c @pregunta 0 0 c:\pregunta.bmp
}
```

Hasta aquí lo que hemos hecho es sencillo, declaramos el alias “/pregunta”. Y lo que hará ese alias es guardar la longitud en pixels del la imagen en la variable %longitud , después guardará la altitud en pixels de la imagen en la variable %altitud. Seguidamente se declara la ventana de imagen @pregunta, usando el parámetro **+b** para que no tenga barra de título ni botones de minimizar, maximizar y cerrar. Hemos usado las variables %altitud y %longitud para que la ventana sea exactamente del mismo tamaño que la imagen, de esa forma esta ocupará toda la ventana y quedará bien (sin ningun espacio en blanco).

Despues mediante el comando **drawpic** hemos cargado la imagen en la ventana que hemos creado y hemos metido esa imagen en la cache (mediante el [-c]) para que si la volvieramos a utilizar en la misma sesion se cargara más rapidamente.

Ahora iremos a los remotes para definir el “menu @pregunta” y el gatillo que hará que cuando hagamos un click sobre “SI”, se cierre el mIRC, y que cuando hagamos un click sobre “NO”, se cierre la ventana. Copie lo siguiente en los “Remotes”:

```
menu @pregunta {
   sclick:{
         if ( $mouse.x > 49 && $mouse.x < 73 && $mouse.y > 46 && $mouse.y < 78 ) exit
         elseif ( $mouse.x > 118 && $mouse.x < 163 && $mouse.y > 44 && $mouse.y < 68
         ) {
               window –c @pregunta
            }
            else { /echo –s Haz click sobre “SI” o sobre “NO” ! }
      }

      Información de la imagen
      .¿Cuánto ocupa?: /echo –s La imagen ocupa $pic(c:\pregunta.bmp).size bytes
      -
      Cerrar ventana: /window –c @pregunta
}
```

Esta es la sección más interesante del código, y aquí se ha mostrado como se hará siempre que queramos hacer que ocurran cosas diferentes según en qué la región pulsemos de una imagen. Para ello se ha recurrido al gatillo **“sclick”** que como se explicó antes salta cuando el usuario hace un simple click sobre la ventana. Lo que pasará en este caso es que el script comprobará donde ha hecho el click, y dependiendo de donde sea, ejecutará unos comandos u otros. Pero al **mIRC** no le podemos decir “si el usuario clickea sobre el SI haz esto y lo otro”, al **mIRC** le tendremos que decir “si el usuario hace click en el rectangulo x y w h , entonces ejecuta estos comandos”.

Y eso hemos hecho, primero hemos abierto la imagen en un programa de diseño, el Paint de Windows basta, y moviendo el ratón sobre la imagen nos aparece en la esquina inferior derecha del programa las coordenadas por las que estamos moviendo el raton, así pues apuntamos las coordenadas que definen el rectangulo que contiene a la palabra 'SI'. En este caso el rectangulo tendria su esquina superior izquierda en x(49) y(46) y su esquina inferior derecha en x(73) y(78) . Entonces le hemos dicho al **mIRC**: "si cuando el usuario hace click el raton esta entre las coornidadas x(49) y x(73) y además esta entre las coordinadas y(46) e y(78)" eso necesriamente significa que el usuario ha hecho click sobre la palabra 'SI' y por lo tanto ejecutaremos el comando **exit** , que cierra el **mIRC**, no hace falta que le digamos antes que cierre la ventana de imagen puesto que al cerrarse el **mIRC** se cierran automaticamente todas las ventanas que haya creadas. Análogamente se ha seguido el mismo procedimiento para detectar si el usuario hace click sobre 'NO', y en tal caso hacemos que se cierre la ventana de imagen y que no pase nada más. Por último le hemos dicho que si el click no se produce ni sobre la palabra 'SI' ni sobre la palabra 'NO' que nos salga un mensaje en la ventana de status indicándonos donde tenemos que pulsar.

Visto eso, el resto es sencillamente el menú que aparecerá al pulsar boton derecho sobre la ventana, que se especifica, como ya sabe, despues de el/los gatillos que hayamos empleado.

Hasta aquí este tutorial de ventanas personalizadas. Soy consciente de que al principio parecen muy complicadas, pero en realidad no lo son tanto, lo que sí son es muy pesadas de crear , por eso se recomienda que se usen sólo cuando sea estrictamente necesario, ya que la mayor parte de las veces se podría llevar a cabo la misma tarea, y de una forma más cómoda mediante popups. De cualquier forma, se han comentado con detalles todas las posibilidades de las ventanas personalizadas y de imagen para que también el lector ya experto les pueda sacar el máximo provecho.

Documento escrito por TeMpEsT · www.relativo.com · tempest@mixmail.com

DIALOGS

Introducción

Las ventanas de diálogo o "dialogs" son un nuevo recurso que nos ofrecen las versines de **mIRC** 5.5 y superior para crear auténticas ventanas tipo windows (con botones de radio, campos de texto, etc) de una forma relativamente sencilla, al menos es muchísimo más fácil que hacerlo mediante ventanas de imagen, más adelante veremos por qué. En realidad este tipo de ventanas ya existían antes de la versión 5.5, me refiero a los identificadores **$?**,**$sdir**,**$dir**,... etc. que no eran más que ventanas de diálogo ya configuradas para hacer una tarea concreta. La gran ventaja de los nuevos "dialogs" es que ahorá usted no está limitado al uso de esas ventanas preconfiguradas, sino que podrá crear las suyas propias para todo tipo de funciones. Sin más, y puesto que se supone que usted ya es medianamente experto en el uso de Alias y Remotes, pasemos a ver los comandos que nos permitirán la creación de ventanas de diálogo:

Comando Dialog

Creará una ventana de diálogo totalmente independiente de la tarea que el **mIRC** esté realizando en ese momento, es decir que mientras la ventana de diálogo permanece abierta usted podrá acceder a las otras ventanas del **mIRC** (cosa que no pasaba con los diálogos **$?**, **$dir**,... etc). Para mostrar en pantalla un diálogo, usaremos la sintaxis:

/dialog [–mda] <nombre_dialogo> <nombre_tabla>
Donde <nombre_dialogo> es aquel con el que usted se referirá al mismo, y <nombre_tabla> es el nombre de una tabla de diálogo que usted tendrá que declarar más adelante y en la que diseñará su aspecto, dimensiones y contenido (posición de los botones, botones de radio,... etc).

NOTA: Se puede, y de hecho es recomendable, usar el mismo nombre para el nombre del diálogo y para la tabla, para evitar posteriores confusiones.

A continuación tiene una explicación de la función de los tres parámetros disponibles, tenga en cuenta que el parámetro **-m** es obligatorio y sin él no se creará nuestra ventana de dialogo:

-m: Es el parámetro que usará normalmente, sirve para crear una ventana de diálogo sin ningún atributo, es decir que todos los atributos (título, tamaño...) serán especificados en la tabla del diálogo.
Ejemplo: /dialog –m midialogo mitabla

-ma: Crea una ventana de diálogo sin atributos (como la anterior) pero además usa la ventana activa como "madre", es decir que al cerrar la "ventana madre" se cerrará también la que hemos creado.
Ejemplo: /dialog –ma midialogo mitabla

-md: Crea un diálogo sin atributos (puesto que lleva el parámetro **-m**) y lo abre como ventana de escritorio (será mostrada en la barra de tareas inferior de Windows).
Ejemplo: /dialog –md midialogo mitabla

Una vez creado, podemos volver a usar el comando /dialog, pero esta vez para cambiar alguna de sus propiedades, y con la sintaxis:

/dialog [-tsonkc] <nombre_dialogo> [atributos]

Y dependiendo del parámetro que especifique en [-tsonkc], usaremos unos atributos u otros (o a veces ninguno):

-x: Cierra una ventana de diálogo.
Ejemplo: /dialog –x midialogo

-t: Cambia el título deuna ventana de diálogo.
Ejemplo: /dialog –t midialogo Este es el nuevo título

-s: Cambia el tamaño y la posición del diálogo en base a las coordenadas y dimensiones que se especifiquen:
Ejemplo: /dialog –s midialogo 10 20 300 400

NOTA: Recuerde la definición de "rectángulo" en el tutorial de ventanas de imagen, el ejemplo de arriba pondrá la esquina superior izquierda del diálogo en las coordenadas x(10) y(20) y le dará un tamaño de 300 pixels de largo por 400 de alto.

-o: Pone el diálogo encima de todas las ventanas que tenga abiertas (on top), y sigue estando encima aunque pulse fuera de ella.
Ejemplo: /dialog –o midialogo

-n: Quita el atributo "on top" de un dialogo, es decir que ya no estará permanentemente encima de todas las ventanas.
Ejemplo: /dialog –n midialogo

-k: Provoca el efecto de pulsar el botón "ok" de un diálogo. Por defecto, el efecto de este botón es simplemente el de cerrar el dialogo, pero más adelante veremos que le podrá asignarle más funciones.
Ejemplo: /dialog –k midialogo

-c: Provoca el efecto de pulsar en un boton "cancel" de un diálogo, que por defecto es exactamente el mismo que el de un boton "ok", pero al igual que en el anterior, se podrá modificar.
Ejemplo: /dialog –c midialogo

Vista la creación de ventanas de diálogo, ahora el tema será como crear las ya nombradas "Tablas de diálogo"

Tablas

Como hemos visto anteriormente en la creación de diálogos, es necesario especificar el nombre de una tabla en la que definiremos el título,tamaño y objetos que habrá en el mismo. En está sección usted aprenderá a crear tablas de diálogo revisando todas las posibilidades que nos ofrece el **mIRC**. La declaración de tablas de dialogo se hace en la sección Remotes, y de la siguiente forma:

```
DIALOG <nombre_tabla> {
.........
}
```

Todo lo que hemos hecho es decirle al script que existe una tabla de diálogo de nombre *"nombre_tabla"*, pero para que esta sea válida tendremos que incluir en ella la declaración de 3 cosas imprescindibles: El título del diálogo (que aparecera en la barra de título del mismo), el tamaño y posición del dialogo (en el formato x y w h), y al menos un boton de tipo “ok” o “cancel” para cerrar el diálogo. Para ello escribiremos lo siguiente:

```
dialog mitabla {
 title “Aquí el título”
 size <x y w h>
 button “Texto_del_boton”,<ID>,<x y w h>,<estilo>
}
```

Donde dice x y w h recuerde que ha de poner las coordenadas que definen el rectángulo al que se refiere: posición de la esquina superior izquierda del dialogo (x,y), longitud (w) y altitud (h) en pixels. Donde dice “ID” se refiere a un número que asignaremos a cada objeto que añadamos a la tabla, ese número nos servirá más adelante para referirnos a ese objeto, por lo tanto no se puede repetir la misma ID en dos objetos dentro de una misma tabla. Y por último el estilo, al tratarse de un botón, en este caso será uno de los siguientes:

- **ok**: Crea un boton que cierre el dialogo al pulsarlo.
- **cancel**: Crea un boton que tambien cierra el dialogo al pulsarlo.
- **default**: Crea un boton que no cierra el dialogo, le podremos asignar otra funcion más adelante.

Antes de entrar en el resto de objetos que podrá usar, veamos este pequeño ejemplo para que vaya entendiendo el funcionamiento de las tablas, copie lo siguiente en “Aliases”:

```
/Dialogo { dialog –m midialogo mitabla }
```

Copie esto otro en “Remotes”:

```
dialog mitabla {
 title "Hola Mundo!!"
 size 20 20 120 100
 button "¡¡Adiós!!",1,20 20 80 50,ok
}
```

Si ahora escribe el alias que acaba de crear: "*/dialogo*", verá como le aparece una ventana con el título que ha especificado y con un bóton de tipo “ok” (cierra el dialogo al pulsarlo) , las coordenadas y tamaño de la misma serán: “20 20 100 100”.

A continuación veremos ya la relación de objetos que podemos declarar en una ventana de dialogo:

title “Título”

Como hemos visto en el ejemplo anterior, el texto que pongamos entre “ “ será el texto que salga en la barra de título del dialogo.

size x y w h

Indica la posición y tamaño del diálogo. Si indicamos el valor ‘-1’ para x e y, el diálogo aparecerá centrado en la pantalla.

text “Texto”,ID,x y w h,estilo

Pone el texto que indiquemos en las coordenadas x y w h. Recuerde que a cada objeto que ponga en el diálgo le ha de asignar un número ID, el que usted quiera para despues referirse a ese objeto más adelante. El estilo es opcional, si no se especifica ninguno el texto estará alineado a la izquierda del rectángulo “x y w h”, los estilos que puede utilizar son:

- **right**: para alinear el texto a la derecha.
- **center**: para centrarlo en el rectangulo “x y w h”.

edit “Texto inicial”,ID,x y w h,estilo

Inserta una caja de texto en el lugar indicado, con una posicion y tamaño dados en “x y w h”, y con uno de los estilos siguientes:

<en_blanco> : si no especifica ningun estilo: campo de texto normal con el texto inicial alineado a la izquierda.

- **right**: alinea el texto inicial a la derecha.
- **center**: centra el texto inicial.
- **multi**: crea un campo de texto con multiples líneas, para ello ademas de especificar este estilo tendrá que hacer el campo más alto (aumentar el valor ‘h’).
- **pass**: útil para introducir passwords, trasforma cada carácter que escribamos en el campo en un asterisco (*).
- **read**: para que el campo sea solo para mostrar información al usuario, es decir que el texto que pongamos en ese campo no podrá ser modificado por el usario.
- **hsbar**: pone una barra de desplazamiento horizontal en la parte inferior.
- **vsbar**: pone una barra de desplazamiento vertical a la derecha.
- **autohs**: hace que el campo se desplaze automaticamente en horizontal cuando introduzcamos un texto que ocupe más que el tamaño de la caja de texto.
- **autovs**: hace que el campo se desplaze automaticamente en vertical cuando introduzcamos un texto que ocupe más que el tamaño de la caja.

button “Texto del boton”,ID,x y w h,tipo

Crea un botón en cuyo interior ponga el texto que especifiquemos, y lo posiciona en las coordinadas x,y con un tamaño de ‘w’ pixels de largo por ‘h’ de alto. Los tipos posibles son:

- **ok**: Al pulsar el botón se cerrará el diálogo.
- **cancel**: Al pulsar el botón se cerrará el dialogo.
- **default**: Al pusar el botón no se cierra el diálogo, especificaremos la funcion de este tipo de botones más adelante.

check “Texto”,ID,x y w h,estilo

Crea un botón “checkbox” o botones de selección de opciones (sirven principalmente para activar/desactivar algo) en las coordenadas y con el texto especificado. Los estilos posibles son:

- **right**: el texto se coloca a la derecha del checkbox.
- **left**: el texto se coloca a la izquierda del checkbox.
- **3state**: pone un checkbox de 3 estados (pulsado, no pulsado, e intermedio).
- **push**: pone un tipo especial de checkbox que en vez de con casillas pulsables, se hace mediante botones que permanecen pulsados.

radio “Texto”,ID,x y w h,estilo

Crea un botón de radio (sirven para elegir entre una entre varias opciones) en las coordinadas “x y w h” y con el texto “Texto”. Los estilos posibles son:

- **right**: el texto se coloca a la derecha del boton de radio.
- **left**: el texto se coloca a la izquierda del boton de radio.
- **push**: pone un tipo especial de boton de radio hecho a base de botones estandar.

box “Titulo”,ID,x y w h

Crea un rectángulo con el titulo, posicion y tamaño especificados, se sule usar para enmarcar grupos de objetos, por ejemplo grupos de botones de radio.

list ID,x y w h,estilo

Crea una lista de objetos en las coordinadas especificadas (siempre relativas a la ventana de diálogo) en la que se podrán elegir uno o mas objetos a la vez de la lista dependiendo del estilo. El cómo añadir mas objetos a la lista lo veremos más adelante. Los estilos posibles son:

<en_blanco>: si no especificamos estilo, se crea una lista en la que solo podemos seleccionar un objeto a la vez y el orden de estos será el mismo en que los añadamos mas adelante.

- **sort**: Ordena los objetos de la lista por orden alfabético.
- **extsel**: Permite seleccionar varios obajetos de la lista a la vez.

combo ID,x y w h,estilo

Crea una lista de objetos en la que solo se podrá elegir uno de ellos, en las coordenadas especificadas. Los estilos posibles son:

- **sort**: ordena los objetos de la lista por orden alfabético.
- **edit**: pone una editbox arriba de la lista. Al pulsar en un objeto de la lista éste aparece en la editbox y puede ser editado.
- **drop**: pone un recuadro en el que al pulsar aparecerá la lista de objetos.

icon ID,x y w h,[archivo]

Inserta una imagen en formato .bmp en las coordenadas que indiquemos. Si los valores ‘w’ y ‘h’ no coinciden con el tamaño de la imagen, ésta será estrechada o ensanchada al tamaño que hayamos especificado, el parámetro archivo no hace falta especficarlo ahora, lo podrá hacer más adelante.

Hasta aquí todos los objetos que puede poner en un diálogo. Adicionalmente a los estilos que se han comentado, cualquier objeto puede además contar con los siguientes:

- **disable**: Inabilita en objeto.
- **hide**: Esconde el objeto.
- **group**: Indica el comienzo de un grupo, útil para marcar el comienzo de un grupo de botones de radio, para ello en el primer botón del grupo indicaremos el estilo “group”.
- **result**: Indica que el contenido de este objeto será el resultado que devuelva el dialogo al pulsar el boton “ok”. Solo se usa cuando el diálogo lo abrimos mediante el identificador **$dialog** (explicado má adelante).

Una cosa importante que debe saber es que se pueden especificar VARIOS estilos a la vez con solo ponerlos separados con comas, por ejemplo:

```
edit “Texto”,4,10 10 100 20,autohs,right
```

NOTA: En la declaración de objetos puede usar variables, es decir podrá, por ejemplo, mostrar en un campo de texto el valor de una variable.

Ya hemos visto todos los tipos de objetos que podrá usar a la hora de crear una tabla de diálogo, por supuesto no usará todos estos tipos en el mismo diálogo, aunque podría hacerlo si quisiera... antes de seguir vamos a hacer un ejemplo de una ventana algo más complicada ya que con lo que sabemos hasta ahora es posible crear la interfaz gráfica de cualquier ventana de diálogo imaginable, lo próximo será proporcionar funcionabilidad a cada uno de los objetos... pero antes, lo dicho, vamos a un ejemplo:

Copie lo siguiente en Aliases:

```
/Dialogo2 { dialog –m dialogo2 mitabla2 }
```

Y ahora copie lo siguiente en Remotes:

```
dialog mitabla2 {
 title "Qué información se muestra dónde"
 size 30 30 330 260

 box "Mostrar información",1,10 10 150 100
 radio "En Status",2,20 30 130 20,right,group
 radio "En Ventana aparte",4,20 80 130 20,right

 box "Otras opciones",5,170 10 150 100
 check "Mostrar Nombre",6,180 30 130 20,push
 check "Mostrar E-Mail",7,180 55 130 20,push
 check "Mostrar Web",8,180 80 130 20,push

 box "Datos",9,10 115 310 100
 text "Su Nombre:",10,20 135 100 20
 text "Su E-Mail:",11,20 160 100 20
 text "Su Web:",12,20 185 100 20
 edit "",13,100 135 180 20
 edit "",14,100 160 180 20
 edit "",15,100 185 180 20

 button "Mostrar Información",16,40 225 130 25,ok
 button "Cancelar",17,190 225 90 25,cancel
}
```

No se va a comentar el código línea por línea puesto que sería repetir lo ya explicado, pero una vez leído y probado (usando el alias "*/dialogo2*") se debe haber dado cuenta de unas cosas importantes:

- Cada objeto tiene su ID, en este ejemplo se van numerando de 1 a N siendo 'N' el numero total de objetos, de esa forma más adelante nos será más facil acordarnos de la ID de un objeto.

- Cuando el texto inicial de, por ejemplo, un campo de texto queremos que sea nulo, es decir, que no haya texto inicial, se especifican las comillas vacías "".

- Al pulsar en uno de los botones de radio, el resto quedan sin pulsar, esto se ha conseguido indicando en el primero de ellos el estilo "group". Si hubiera otro grupo de botones de radio en otra seccion del diálogo haríamos lo mismo, indicaríamos en el primer boton el estilo group.

- El uso de "box" da un aspecto más agradable al diálogo, su uso esta muy recomendado para encuadrar objetos del mismo tipo o tema.

Comando DID

Una vez abierta una ventana de diálogo lo normal es que queramos modificar algo, quizas queramos añadir texto a un campo de texto, o poner una variable en éste, para ello usaremos el comando /did que sirve para modificar una ventana de diálogo que ya está abierta. La sintaxis es:

/did [-ftebvhnmcukradiog] <nombre_dialogo> <id> [N] [texto/archivo]
Y a continuación la explicación de los diferentes parámetros que podemos usar con este comando:

-f: Enfoca el objeto <id>.
Ejemplo: /did -f midialogo 20

-t: Hace que el bóton <id> sea el bóton por defecto (su funcion se producirá tambien al pulsar ENTER).
Ejemplo: /did –t midialogo 23

-b: Hace que no se pueda interactuar con el objeto <id>.
Ejemplo: /did –b midialogo 12

-e: Devuelve la interacción al objeto <id>, en caso de que se le hubiera quitado con el parámetro anterior.
Ejemplo: /did –e midialogo 16

-v: Hace que el objeto <id> sea visible.
Ejemplo: /did –v midialogo 10

-h: Esconde el objeto <id> (lo hace invisible).
Ejemplo: /did –h midialogo 10

-m: (Sólo funciona en campos de texto) Hace que el texto NO sea editable.
Ejemplo: /did –m midialogo 30

-n: (Sólo funciona en campos de texto) Deshace el parámetro anterior, es decir el texto ya se puede editar en el campo de texto.
Ejemplo: /did –n midialogo 29

-c: Activa un checkbox o boton de radio (para ello NO se especifica [N]), o selecciona la linea N en una lista de objetos.
Ejemplo: /did –c midialogo 13 5

-u: Desactiva un checkbox o boton de radio (no se especifica [N]), o la linea [N] en una lista de objetos deja de estar seleccionada. Para marcar un checkbox como indeterminado (en caso de que sea de estilo 3state) usaremos los parámetros [-cu] a la vez.
Ejemplo: /did –u midialogo 2

-k: (Sólo funciona en listas de objetos) Al usarse junto con los parámetros [-c] o [-u] hará que se mantenga la selección que ya había y añada o quite a ésta la que acabamos de hacer.
Ejemplo: /did –ck midialogo 4 2

-r: Borra todo el texto del objeto <id> (usado generalmente para campos de texto).
Ejemplo: /did –r midialogo 3

-a: Añade una línea de texto al final de la que ya haya en el objeto <id>.
Ejemplo: /did –a midialogo 6 %variable

-d: Borra la linea [N] de texto de un objeto.
Ejemplo: /did –d midialogo 8 2

-i: Inserta una linea de texto en la posición [N].
Ejemplo: /did –i midialogo 10 2 Insertando entre 2ª y 3ª linea

-o: Sobreescribe la linea [N] con el texto especificado.
Ejemplo: /did –o midialogo 10 1 Sustuimos la linea uno por este texto

-g: Pone una imagen .bmp a un objeto de icono.
Ejemplo: /did –g midialogo 10 c:\imagen.bmp

Vistas ya las formas para crear un a ventana de diálogo y modificarla a nuestro gusto, y antes de entrar a ver el evento ON DIALOG que nos servirá para asignarle ciertas funciones a cada objeto, veamos un par de identificadores propios de los diálogos que nos serán muy útiles para recoger información de las ventanas que creemos.

El identificador $dialog

Este identificador puede ser usado de varias maneras. Si en una tabla de un diálogo especificamos en unos de los objetos el estilo "*result*", y después creamos un diálogo usando este identificador, el valor que nos devoverá el identificador será el mismo que el de ese objeto cuyo estilo es "result". Para crear un diálogo con este identificador usaremos la sintaxis:

$dialog(<nombre_dialogo>,<nombre_tabla>)
Pero por supuesto siendo un identificador no puede usarse independientemente (como si fuera un comando) puesto que devuelve un valor que debemos de recojer, lo tendremos que utilizar para darle un valor a una variable, por ejemplo:

```
%resultado = $dialog(midialogo,mitabla)
```

También podemos usar este identificador sencillamente para obtener el nombre de los diálogos abiertos en este momento, para ello usaremos la sintaxis:

$dialog(<nombre_dialogo>/<N>)

Devolverá el número de diálogo en caso de que especifiquemos un nombre, o el nombre del diálogo abierto numero <N> en caso de que optemos por especificar ese parámetro.

Ejemplo: `%variable = $dialog(3)`

Le da a %variable el valor del nombre del tercer diálogo que tengamos abierto en ese momento.

Y el último uso del identificador **$dialog** es para recoger información general de un diálogo, como su tamaño o su título:

- **$dialog(<nombre>).x**: Devuelve la posición en el eje X del diálogo.
- **$dialog(<nombre>).y**: Devuelve la posición en el eje Y del diálogo.
- **$dialog(<nombre>).w**: Devuelve la longitud en pixels del diálogo.
- **$dialog(<nombre>).h**: Devuelve la altitud en pixels del diálogo.
- **$dialog(<nombre>).title**: Devuelve el título del diálogo.
- **$dialog(<nombre>).modal**: Devuelve $true si el diálogo ha sido creado con el identificador $dialog, o $false si el diálogo ha sido creado con el comando /dialog.
- **$dialog(<nombre>).table**: Devuelve el nombre de la tabla que está usando el diálogo.
- **$dialog(<nombre>).ok**: Devuelve la ID del botón con estilo "ok".
- **$dialog(<nombre>).cancel**: Devuelve la ID del botón con estilo "cancel".

- **$dialog(<nombre>).result**: Devuelve la ID del botón que tenga como estilo “result”.

El identificador $did

Este otro identificador nos servirá para recoger información de los objetos de un diálogo. sus posibles usos son:

- **$did(<nombre>,<id>).text:** Devuelve el texto del objeto <id>.
- **$did(<nombre>,<id>,<N>).len**: Devuelve la lingitud en caracteres de la linea <N> del objeto <id>.
- **$did(<nombre>,<id>).lines**: devuelve el numero total de lineas del objeto <id>.
- **$did(<nombre>,<id>).sel**: Devuelve el número de la linea seleccionada.
- **$did(<nombre>,<id>).state**: Devuelve el estado del objeto <id> (0 = off ; 1 = on; 2 = indeterminado).
- **$did(<nombre>,<id>).next**: Devuelve la id del próximo obejto en el orden del tabulador.
- **$did(<nombre>,<id>).prev**: Devuelve la id del objeto previo en el orden del tabulador.

Una vez aprendidos estos identificadores, aparte de saber crear y modificar ventanas, tan solo nos falta el último paso, el decirle a determindo control que ejecute determinados comandos, decirle a determinado campo de texto que ponga su contenido dentro de determinada variable,... etc. Esto se consigue con un evento remoto, el **ON DIALOG**, que a continuación se expone con detalle.

Evento Dialog

Este es el evento que se usará para controlar la funcionabilidad del dialogo, la sintaxis es la siguiente:

on 1:dialog:<nombre_dialogo>:<evento>:<id>:{ comandos }

En seguida se explicará la sintaxis anterior, antes debe saber que mediante el evento ON DIALOG usted podrá:

- Hacer que se ejcuten los comandos que usted quiera al pulsar sobre cada uno de los botones del diálogo.
- Asignar a las variables mostradas en campos de texto, el valor que el usuario indique en ese campo.
- Inicializar campos de texto u otros objetos con la posibilidad de poder usar variables si no lo ha hecvho al declarar la tabla.
- Asignar funciones a ciertos objetos cuando se produzca un doble click sobre los mismos.
- El identificador **$dname** se puede usar dentro del evento y devolverá el <nombre_diálogo>.

- El identificador **$did** se puede usar dentro del evento y devolverá la ID del objeto que haya causado el evento.

- El identificador **$devent** se puede usar dentro del evento y devolverá el <evento>.

Pasemos a analizar, por tanto, cuales son las variantes del evento ON DIALOG:

on 1:dialog:<nombre_dialogo>:INIT:0:{ ...comandos... }

Se ejcutan los comandos que especifiquemos justo antes de mostrar el dialogo en pantalla, es decir que usaremos este evento para inicializar ciertos objetos, por ejemplo si queremos poner el contenido de una variable en un campo de texto, lo haremos en este evento mediante un **/did –a <nombre_dialogo> <%variable>**.

on 1:dialog:<nombre_dialogo>:SCLICK:<id>:{ ...comandos... }

Se ejecutan los comandos especificados cuando se produce un click sobre el objeto con la id indicada.
Ejemplo: on 1:dialog:midialogo:sclick:3:{ %variable = 0 }

on 1:dialog:<nombre_dialogo>:DCLICK:<id>:{ ...comandos... }

Se ejecutan los comandos especificados cuando el usuario haga un doble click sobre el objeto de la <id> especificada.
Ejemplo: **on 1:dialog:midialogo:dclick:3:{ echo –s Doble click sobre objeto $did }**

on 1:dialog:<nombre_diálogo>:EDIT:<id>:{ ...comandos... }

Se ejecutan los comandos cada vez que el usuario introduce o borra un carácter de una caja de texto.
Ejemplo: on 1:dialog:midialogo:edit:5:{ %variable = $did(5).text

Bien, hasta aquí el manual de ventanas de diálogo, no son realmente complicadas, sobre todo si se compara con la dificultad de hacer una ventana de este tipo mediante ventanas de imagen. Por último vamos a acabar el ejemplo que iniciamos en una de las primeras secciones de este documento, sólo que esta vez podremos acabarlo y darle toda su funcionabilidad con lo que hemos aprendido:

Ejemplo : Creación de un Diálogo con toda su funcionabilidad.

Antes que nada recordemos el código que ya escribimos:

Copie lo siguiente en Aliases:

/Dialogo2 { dialog –m dialogo2 mitabla2 }

Y ahora copie lo siguiente en Remotes:

```
dialog mitabla2 {
 title "Qué información se muestra dónde"
 size 30 30 330 260

 box "Mostrar información",1,10 10 150 100
 radio "En Status",2,20 30 130 20,right,group
 radio "En Ventana aparte",4,20 80 130 20,right

 box "Otras opciones",5,170 10 150 100
 check "Mostrar Nombre",6,180 30 130 20,push
 check "Mostrar E-Mail",7,180 55 130 20,push
 check "Mostrar Web",8,180 80 130 20,push

 box "Datos",9,10 115 310 100
 text "Su Nombre:",10,20 135 100 20
 text "Su E-Mail:",11,20 160 100 20
 text "Su Web:",12,20 185 100 20
 edit "",13,100 135 180 20
 edit "",14,100 160 180 20
 edit "",15,100 185 180 20

 button "Mostrar Información",16,40 225 130 25,ok
 button "Cancelar",17,190 225 90 25,cancel
}
```

Este código ya ha sido visto anteriormente así que doy por hecho que ya lo entiende. Lo siguiente que haremos será declarar en la sección Remotes una serie de eventos ON DIALOG para asingarle nuevos valores a esas variables (según lo desee elusuario) y despues para implementar la función del botón “Mostrar información”. Copiemos lo siguiente en la sección “Remotes”:

```
on 1:dialog:dialogo2:init:0:{
 if (%mostrar.donde == status) { did -c dialogo2 2 }
 if (%mostrar.donde == aparte) { did -c dialogo2 4 }

 if (%mostrar.nombre == si) { did -c dialogo2 6 }
 if (%mostrar.email == si) { did -c dialogo2 7 }
 if (%mostrar.web == si) { did -c dialogo2 8 }
 did -a dialogo2 13 %sunombre
 did -a dialogo2 14 %suemail
 did -a dialogo2 15 %suweb
}
```

En el evento INIT inicializaremos el dialogo, es decir, que si la variable *%mostrar.donde* vale *status*, se activará el botón de radio correspondiente, lo mismo pasa con los checkbox, que son activados por código puesto que las ventanas de dialgoo aparecen con todos sus objetos desactivados a no ser que se especifique lo contrario. Por último se pone el contenido de las variables del nombre, email y web en sus campos de texto correspondientes:

```
on 1:dialog:dialogo2:sclick:2:{ %mostrar.donde = status }
on 1:dialog:dialogo2:sclick:4:{ %mostrar.donde = aparte }
```

Con ese código haremos que según pulse el usuario el un botón de radio u otro la variable *%mostrar.donde* cambie y así podamos saber despuéss donde quiere mostrar el usuario la informacion, puesto que quedará almacenado en dicha variable.

```
on 1:dialog:dialogo2:sclick:6:{ %mostrar.nombre = $iif(%mostrar.nombre == si,no,si) }
on 1:dialog:dialogo2:sclick:7:{ %mostrar.email = $iif(%mostrar.email == si,no,si) }
on 1:dialog:dialogo2:sclick:8:{ %mostrar.web = $iif(%mostrar.web == si,no,si) }
```

Este otro trozo hará que según el usuario clickee sobre los checkbox estos activen o desactiven la variable según su posición anterior, de ahí el uso del identificador **$iif**, que devolverá la primera opción si la condición se cumple, y en caso contrario devolverá la segunda opción siendo **$iif(condicion,1ªopcion,2ªopcion)**.

```
on 1:dialog:dialogo2:sclick:16:{
 if (%mostrar.donde == aparte) {
 window –a @Informacion 100 10 300 200
 if (%mostrar.nombre == si) { aline @Informacion Nombre: %sunombre }
 if (%mostrar.email == si) { aline @Informacion E-Mail: %suemail }
 if (%mostrar.web == si) { aline @Informacion Web: %suweb }
}
else {
 if (%mostrar.nombre == si) { echo –s Nombre: %sunombre }
 if (%mostrar.email == si) { echo –s E-Mail: %suemail }
 if (%mostrar.web == si) { echo –s Web: %suweb }
 }
}
```

En este trozo hemos especificado la conducta que esperamos del boton *“Mostrar Información”* cuando sea pulsado, a pesar de la cantidad de líneas es un código muy simple y se limita a comprobar el valor de la variable *%mostrar.donde* para averiguar donde quiere el usuario mostrar la informacion, y después comprueba si están activadas las variables *%mostrar.nombre*, *%mostrar.web* y *%mostrar.email*: para motrar o no esa información.

```
on 1:dialog:dialogo2:edit:13:{ %sunombre = $did(13).text }
on 1:dialog:dialogo2:edit:14: { %suemail = $did(14).text }
on 1:dialog:dialogo2:edit:15:{ %suweb = $did(15).text }
```

Por último le tenemos que decir al script que cuando el usuario edite los campos de texto 13,14 y 15 asigne el texto de esos campos como valor a sus respectivas variables, para que después puedan ser mostradas.

Bien, si ha llegado hasta aquí, ya debe ser un pequeño experto en este tema, en este tutorial se han tocado detalladamente todos los aspectos de las ventanas de diálogo, espero que haya aprendido de él.

Documento escrito por TeMpEsT · www.relativo.com · tempest@mixmail.com

SOCKETS

Introducción

En muchos aspectos la opciones de los sockets en **mIRC** se quedan muy cortas, pero son muy utiles. Los sockets (o zócalos de conexión) son los puertos lógicos a traves de los cuales los programas de nuestro ordenador se comucican con el resto de máquinas de una red a la que está conectada (caso por ejemplo de Internet), su manejo desde **mIRC** nos permite tener un cierto control sobre estas conexiones, abrirlas, cerrarlas, "escuchar" posibles accesos por cualqiera de ellas... etc. No podemos decir que sea una característica ni muy bien implementada ni muy potente en **mIRC**, pero si que nos permitirá hacer scripts muy interesantes con la posibilidad de crear escaneadores de puertos, detectoresde accesos a nuestra máquina... etc; a medida que se expliquen en este documento podrimos ir viendo algunos ejemplos de todo esto.

Para explicar el uso de los sockets partiremos de la base de que ya se conoce perfectamente el manejo de otras areas programables del **mIRC**, en especial los *alias*, *popups* y el uso de variables, identificadores, eventos y *remotes* en general. Si no es así será necesario que estudie previamente los apartados correspondientes a esos temas en esta misma web.

Es necesario indicar, por último, que los sockets es un recurso que se agota, cada vez que hayas terminado de usar un socket debes cerrarlo para poder usar otro, es decir, no se puede estar trabajando con más de un zócalo de conexión a la vez.

Identificadores

Los siquientes identificadores son los básicos que se han de saber para poder trabajar con los sockets. Hay más que se pueden ver en la ayuda del programa, pero ni son tan importantes ni tienen una explicación tan extensa.

$sock(nombre,numero) [.propiedad]
Retorna información referente a una conexión con sockets que hemos creado usando los comandos correspondientes. Si en "numero" no se especifica nada y se deja con el valor "N", mIRC asumira que es 1. Este identificador estas propiedades:

.name	el nombre de la conexión que utilizamos, para identificarla.
.sent	el numero de bytes **enviados** despues que la conexión se haya terminado.
.rcvd	el numero de bytes **recibidos** despues que la conexión se haya terminado.
.sq/.rq	el numero de bytes de la cola para mandar y recibir buffers respectivamente.
.ls/.lr	el numero de segundos desde el ultimo envio y recibo de información de la conexión.
.mark	el área máxima de almacenamiento del usuario, 512 bytes.
.type	el tipo de conexion socket, TCP o UDP
.saddr	la direccion y el puerto de origen del ultimo paquete UDP.
.sport	exactamente lo mismo que la propiedad anterior (.saddr).

$sockname

Es el nombre que le hemoss dado a una conexion para identificarla. Este identificador puede ser usado en los eventos para saber con que conexion se trabaja.

$sockerr

Sirve para comprobar si se ha realizado la conexión correctamente sin fallos. Si resulta haber algun fallo en ella este identificador devolverá un valor mayor que 1. Si todo esta normal no devolvera ningun valor.

$portfree(Puerto)

Sirve para comprobar si un puerto esta siendo utilizado o no. Devolverá **$true** si el puerto esta libre o **$false** si está siendo usado ya.

Detectando conexiones

En este apartado explicaremos como poner a la escucha un puerto determinado para detectar conexiones entrantes, pero primero vamos a ver los comandos y los eventos que vamos a utilizar:

/socklisten <nombre> [puerto]

Nos permite asignar un nombre a un pueto determinado a fin de identificar la conexión y poderlo usar más adelante.

on 1:socklisten:nombre:comandos

Este evento se activa cuando cualquier usuario intenta conectar por el puerto que estamos "escuchando". Si queremos aceptar esta conexión usaremos el comando /sockaccept, si no la conexión será cerrada.

/sockaccept <nombre>

Este comando acepta la conexión actual que hemos detectado con el evento anterior, y le asigna un nombre para identificarla.

/sockrename <nombre> <nuevo_nombre>

Permite renombrar una conexion existente.

/sockclose <nombre>

Cierra la conexion especificada, si no se indica ninguna se cierran todas.

Vamos a poner un ejemplo y a analizarlo para entender mejor todo esto. crearemos una pequeña rutina para detectar si un usuario intenta conectar por el puerto 12345 (Netßus).

Incluir en aliases o popups:

```
/socklisten Netbus 12345 // ponemos a eschuchar el puerto 12345, con el nombre Netbus
```
Incluir en Events, en remotes:

```
on 1:socklisten:NETBUS:{ // se activará cuando intentan conectar con la conexion Netbus

/sockaccept Netbus69 // acepta la conexion y le pone el nombre de Netbus69
beep // mIRC producira un pequeño pitido de aviso

echo –a detectado una conexion por parte de $sock($sockname,1).ip
// nos informa de la IP del usuario

/sockclose Netbus69 // cerrará la conexión con el usuario que ha intentado conectar
}
```

A partir de aquí se puede crear un script para resolver el nick del usuario haciendo un who con la IP.

Abriendo y cerrando conexiones

Los sockets nos permiten abrir conexiones, por ejemplo para detectar los puertos abiertos de otro usuario. Vamos a ver los comandos y eventos que hemos de saber antes de hacer nada:

/sockopen <nombre> <direccion> <puerto>

Inicia una conexion con el puerto y la direccion IP que especificada.

on 1:sockopen:nombre:comandos

Este evento se activa cuando la conexión ha sido establecida mediante el comando /sockaccept.

on 1:sockclose:nombre:comandos

Este evento se activa cuando la conexion ha sido cerrada por el otro usuario.

Vamos a poner un ejemplo y a analizarlo para entenderlo mejor. Este ejemplo sirve para detectar si un usuario tiene abierto el puerto 12345 (Netßus).

Aliases o popups:

```
/sockopen Netbus 195.77.120.10 12345 // abrimos una conexion por el puerto 12345 con 195.77.120.10
```

Events, en remote:

```
on 1:sockopen:Netbus:{ // la conexion ha sido aceptada
 if ( $sockerr > 1 ) { // el usuario no tiene el puerto 12345 abierto
  echo -a PortScan » $sock($sockname,1).ip no esta infectado por el Netßus }
 else { // si el $sockerr no es >1 es que esta infectado por el Netßus
  echo -a Portscan » $sock($sockname,1).ip esta infectado por el Netßus
 }
 /sockclose Netbus // cierra la conexion Netbus
}
```

Bueno, con este ejemplo ya teneis un scan de Netßus de lo más sencillo, ya que aun se puede mejorar mucho.

Leer y escribir información

/sockwrite [-tn] <nombre> <texto>

Envia información a una conexion ya establecida lo más repatido posible. Los parametros son:

-t aunque el texto comienze por &, este sea tomado como texto y no como una variable binaria

-n inserta una nueva linia de texto en la conexion

on 1:sockread:nombre:comandos

Este evento es llamado cuando el otro usuario recibe la informacion

Observe este ejemplo, cuando un usuario intente entrar por el puerto 12345 le aparecera un mensaje:

En popups o alias:

```
/socklisten TONTIN 12345
```

En events, remotes:

```
on 1:socklisten:TONTIN:{ // Cuando intente la conexion...
 /sockaccept TONTAZO // Acepta la conexion
 /sockwrite -n $sockname Uix, mu listo no eres $sock($sockname,1).ip
// Le manda un mensaje
 /.timer69 1 2 /sockclose $sockname « En 2 segundos se le termina la conexion
 echo -a Detectada una conexion de $sock($sockname,1).ip por el puerto 12345
}
```

Documento escrito por [SaRRiO] · www.sarrio.org · admin@sarrio.org

CÓMO CREAR TUS PROPIOS IDENTIFICADORES

Es muy, pero muy senzillo. Sólo debes crear un alias normalmente (en alias o a traves de remote) y el nombre del alias sera el nombre del identificador, si el alias se llama "sucubus" el identificador sera $sucubus.

Para identicar parametros usa $identificador(parametro1,parametro2...), de form que en el alias parametro1 seria $1 y parametro2 seria $2. Esto es muy util y realmente en el código fuente queda muy bien usar identificadores propios.

Puedes usarlos por ejemplo para comprobar bases de datos (archivos txt por ejemplo). Puedes usar las respuestas estandares de mIRC, $true y $false por ejemplo, pero puede devolver lo que quiera.

Perdon, perdon, me olvidaba una parte fundamental, para que devuelva un valor debes usar el comando /return seguido del texto/variables etc...

Ejemplo:

```
alias logea {
  if ($1 == $null) echo -s Faltan parametros. Usar $logea(nick/canal)
  else {
    set %logea.texto $readini mirc.ini logging $1
    if (%logea.texto == $null) set %logea.resultado off
    elseif (%logea.texto == off) set %logea.resultado off
    else { set %logea.resultado on }
    .timer 1 1 unset %logea.resultado %logea.texto
    return %logea.resultado
  }
}
```

Para usar el identificador que hemos creado ($logea) usalo asi: $logea(nick/canal). Lo que hace esto es mirar en el archivo mirc.ini si estas guardando los mensajes del canal o nick especificados. Si no especificas parametros sale un mensaje de error en status.

Yo creo que es fácil de entender, pero con lo bien que explico seguro que hay dudas, pero tranquilos respondo todos los mails que me envia la gente y más si es sobre algun trabajo o documento mio!

Documento escrito por [SaRRiO] · www.sarrio.org · admin@sarrio.org

Como linkar servidores irc en P10

Introducción

Aquí se explica como linkar un servidor IRC creado con mIRC scripting al ircu2.10.x. **ESTE SERVIDOR NO SIRVE PARA QUE OTROS USUARIOS ACCEDAN A LA RED DONDE LO TENGAS LINKADO**, sino que sirve para conectar los bots a la red, ya sean bots de juegos, noticias, debates, servicios de la red, etc. Tu servidor aparecerá como un servidor más a la lista de Links, recibirás toda la información que reciban los bots que conectes por el y toda la información relacionada con la red, Jupes, Splits, WallOps, etc.

En los ejemplos se usan servidores inventados, excepto el servidor a linkar que se uso servidores de la de Univers para que se entienda mejor y de pasada hacer un poco de publicidad (tampoco les hace falta).

KaT fue (almenos eso tengo entendido) el primer español a usar los sockets en mIRC con la finalidad de linkar un servidor IRC a una red.

Recomiendo visitar www.blackcode.com/irc dónde podrás encontrar addons/scripts donde linkan servidores a la mayoría de ircds de la redes más conocidas, estos scripts también son bots de servicio de la red, registro de nicks, canales, mensajería, etc. La versión del ircd ircu2.10.x lo usan la mayoría de redes españolas y otras internacionales como pueden ser: GlobalChat, IRC Hispano, UnderNet, etc.

Trios en P10

En p10, cuando un usuario entra a través de un servidor IRC, este lo asocia y anuncia al resto de la red con un trío de letras que le identifica, por ejemplo ATX, en este trío, la primera letra (A) siempre es la letra que identifica el servidor IRC por donde conecta el usuario, las otras dos letras del trío son elegidas al azar. Cuando los usuarios realicen alguna acción y recibas información de un usuario, en casos lo recibirás como trío de letras o como nick directamente.

Al usar p10, debes de crear una base de datos donde relaciones los tríos con los nicks de los usuarios de la red, cuando linkes en servidor IRC te mandara esta información, pero lo que ya es más difícil es actualizar la base de datos ya que la has de modificar cuando des/conecte un usuario, cuando se realice un gline/kill...

Por ejemplo, cuando el servidor linke aparecerián así los datos de los usuarios:

```
A NICK sarrio 1 948033194 NoD 127.0.0.1 +oiw B]AAAB AAA :mirc551

A NICK Dinamick 1 948033634 oOIIIIIIOo 127.0.0.1 B]AAAB AAB :Resplandor

A NICK pakito 1 948035117 sarrio 195.77.120.10 DDTXgK AAE :InTuDeR ScRiPt UsEr
```

Glosario de la segunda linia (los marcados con un * son los mas importantes):

- A = Letra del servidor por donde esta el usuario
- Dinamick = Nick del usuario (*)
- 948033634 = Fecha del dia en formato $ctime
- oOIIIIIIOo = Identd del usuario
- 127.0.0.1 = Host del usuario
- Resplandor = Full Name
- AAB = trío de letras p10 (*)

Linkando el servidor IRC

Para poder linkar tu servidor a otro necesitas el permiso de este otro. Si el te deja linkar, debe añadir estas lineas en su ircu2.10.x o el ircd que uses:

```
H:*:*:nombre del servidor de bots::
U:*:*:nombre del servidor de bots::
N:ip del servidor de bots:claveremota:nombre del servidor de bots::
C:ip del servidor de bots:claveremota:nombre del servidor de bots::clase
```

Por ejemplo, si lo linkas a un ircd que esta en tu ordenador:

```
H:*:*:Grawkid.Univers.Org::
U:*:*:Grawkid.Univers.Org::
N:127.0.0.1:OpenAccess:Grawkid.Univers.Org::
C:127.0.0.1:OpenAccess:Grawkid.Univers.Org::2
```

Una vez el administrador del otro servidor haya añadido estas lineas a su ircd, podrás linkar sin problemas. Dónde se indica el servidor IRC, siempre va el nombre de tu servidor (siempre el mismo nombre).

Primero de todo debes tener muy claro el tema de los tríos en p10. Debes configurar unas opciones en algunas variables (no hace falta usar variables, pero así lo veras más claro en el ejemplo) para poder linkar el servidor correctamente:

1. %server.name = Nombre del servidor IRC, Grawkid.Univers.Org
2. %server.start = El $ctime de cuando iniciaste el mIRC
3. %server.letra = Letra para tu servidor IRC, T
4. %server.info = Descripción/utilidad del servidor, Servidor de pruebas
5. %server.password = Password remoto de acceso al servidor, OpenAccess

Linkar el servidor sólo consta de dos lineas, PASS y SERVER. Primero debes conectar con el servidor mediante el comando SockOpen, luego debes esperar que verifique la conexión (On SockOpen) y cuando conecte enviar información con el comando SockWrite. Una vez enviada la información (PASS y SERVER), pueden pasar 2 o 3 minutos sin recibir información alguna, luego puede pasar que linkes perfectamente o no.

```
/sockopen servpersonal Xonix.Univers.Org 6667

on 1:SockOpen:servpersonal:{
  if ($sockerr > 0) { echo -s no se pudo conectar :( | return }
  echo -s conectado... abriendo conexión server-server...
  sockwrite -nt servpersonal PASS %server.password
  sockwrite -nt SERVER %server.name 1 %server.start $ctime P10 %server.letra $+ D] :
$+ %server.info
}
```

Conectando Bots

A través del servidor que has linkado pueden conectar bots por ejemplo, para hacerlo utiliza este comando con estos parámetros (los modos de usuario no son necesarios):

```
/sockwrite -tn servpersonal T N Karawicx 1 $ctime Serv Kara.Wicx +okid DDNSca TAS
:Servicios de la red
```

Glosario:

- T = Letra del servidor (ya la configuraste al linkar el servidor)
- Karawicx = Nick del Bot

- Serv = Identd del bot
- Kara.Wicx = Host del bot (Mask: Serv@Kara.Wicx)
- +okid = Modos iniciales del bot
    - +o = IrcOp, sin password ni nada :)
    - +d = No pueden echarlo de ningún canal (Usuario protegido)
- TAS = trío de letras, la primera letra (servidor) es fija, las otras puedes elegirlas
- :Servicios de la red = Full Name del Bot

Enviando y recibiendo mensajes por querys

Los bots pueden recibir mensajes por query, por lo tanto debemos poderlos leer y procesarlos, por eso los mensajes los recibiremos con el comando SockRead hasta aquí bien, luego debemos comprobar si la información que recibimos es un mensaje por query (PRIVMSG) y a que bot lo ha recibido. Si recibimos un mensaje por query aparecería una cosa así:

```
:sarrio PRIVMSG KAS :estas visitando http://sockets.univers.org
```

Glosario:

- :sarrio = Nick del usuario que manda el mensaje
- :TAS = El trío de letras del bot que ha recibido el mensaje
- :estas visitando http://sockets.univers.org = El mensaje que ha recibido
- PRIVMSG = Mensaje por query

Hasta aquí es fácil, pero ahora quizás el bot deba responder con algún mensaje al usuario con información o con algún mensaje de error. Para poder mandar un mensaje al usuario debes hacerlo a través del trío de letras el cual desconoces, pero como sabes su nick puedes buscar en tu base de datos el trío de letras. Una vez hayas averiguado el trío de letras debes enviar el mensaje así:

```
/sockwrite -tn servpersonal TAS PRIVMSG AAA :Error >> Comando desconocido!
```

Glosario:

- TAS = trío de letras del bot que enviara el mensaje
- AAA = trío de letras del usuario a enviar el mensaje
- :Error >> Comando desconocido! = Mensaje a enviar

Información del servidor

Los usuarios pueden solicitar información sobre un servidor IRC cuando quiera con los comandos version, admin, etc. lo interesante es responder a estos eventos raw tal y como haría un servidor IRC real. Sólo has de comprobar la información que recibes con el evento On SockRead y procesar la información. Mira este ejemplo:

```
on 1:SockRead:servpersonal:{
  sockread %servpersonal
  reading %servpersonal
}

alias reading {
  if ($2 == ADMIN) {
    /sockwrite -tn servpersonal T 256 TAQ :Administración de Info.Univers.Org
    /sockwrite -tn servpersonal T 257 TAQ :Información de Info.Univers.Org
    /sockwrite -tn servpersonal T 258 TAQ :Servidor IRC de pruebas
    /sockwrite -tn servpersonal T 259 TAQ :F. Sarrió <sarrio@univers.org>
  }
  elseif ($2 == VERSION) {
    /sockwrite -tn servpersonal T 351 TAQ :Ejemplo de servidor IRC de
http://sockets.univers.org - v1.0
  }
  elseif ($2 == TIME) { /sockwrite -tn servpersonal T 391 TAQ : $+ $fulldate }
}
```

Glosario:

- TAQ = El trío de letras del nick que solicito ADMIN/VERSION
- T = La letra de tu servidor IRC

NOTAS: Has de averiguar el trío de letras del nick que solicito la información buscando en tu base de datos. El número que hay después de "T", es el número raw que devuelve la información

Ejecutando comandos

Puedes ejecutar comandos con el servidor, como cambiar modos de un canal, etc...

```
da op: /sockwrite -tn servpersonal T M #univers +o AAB

banea: /sockwrite -tn servpersonal T M #ayuda_scripting +b *!*@vilallonga.telefonica.com

envía wallops: /sockwrite -tn servpersonal T WA :Mensaje WallOps a toda la red
```

Y con los bots también claro...

```
topic: /sockwrite -tn servpersonal TAS T #canal :Visita http://sockets.univers.org

cambiar modos: /sockwrite -tn servpersonal TAS M #canal +msintp
```

Documento escrito por [SaRRiO] · www.sarrio.org · admin@sarrio.org

Basado en los addons y scripts de KaT · www.sarrio.org/kat · kat@blackcode.com

---

Uso de los TABs en Dialogs

Introducción

Antes de empezar a leer esto, debes tener ciertos conociminetos sobre dialogs, si no los tienes visita el apartado curso de scripting de www.ayuda-irc.net, donde encontraras dos documentos muy completos sobre los dialogs escritos por EsTePaRiO y TeMpEsT.

Los TAB, son las pestañas por las cuales puedes ver diferentes ventanas de configuración. En una sola ventana puedes llegar a tener muchismas opciones gracias a los TAB.

Tab Control

Para añadir diferentes pestañas TAB, debes meter esto en la tabla del dialog:

Sintaxis: tab "texto", **id**, x y w h

Ejemplo: tab "m", 1, 5 5 100 90

- texto: el texto que aparece en la pestaña
- id: el número de ID del TAB (no puede repetirse el id)
- x: distancia desde la izquierda del monitor (en pixels)
- y: distancia desde la parte superior del monitor (en pixels)
- w: tamaño horizontal (en pixels)
- h: tamaño vertical (en pixels)

Cuando ya hayas introducido un TAB, en los que vengan a continuación no hace falta que especifiques el tamaño y las coordenadas de los TAB. Por ejemplo:

```
tab "m", 1, 5 5 100 90
tab "I", 2
tab "R", 3
tab "C", 4
```

Cada pestaña TAB, tendra sus opciones, por eso al introducir un "edit" o cualquier opción debes especificar el número de TAB donde debe ir, en caso contrario apareceria en todos los TAB. Mira este ejemplo:

```
button "m is for ... ;)", 11, 30 50 50 24, ok tab 1
button "I is for Internet", 12, 30 50 50 24, tab 2
button "R is for Relay", 13, 30 50 50 24, tab 3
button "C is for Chat", 14, 30 50 50 24, tab 4
```

Puedes seleccionar un TAB usando el comando did, de forma que pulsando en un menu o un boton del mismo dialog, puedas saltar a otra pestaña TAB. Este es el comando con sus parametros:

Sintaxis: /did -fu dialog id-tab

Ejemplo: /did -fu dialog69 22

Con el identificador $dialog puedes saber que TAB esta utilizando en ese momento:

Sintaxis: $dialog(dialog).tab

Ejemplos: $dialog(dialog69).tab

Este documento es muy corto y explica muy poca cosa, pero sólo va destinado a explicar las novedades en los dialogs a los usuarios que dominan los dialogs y aun no conocen esta funcion tan util.

Documento escrito por [SaRRiO] · www.sarrio.org · admin@sarrio.org

PUERTOS TCP INTERESANTES

21	FTP
23	Telnet
25	SMTp
53	DNS
59	DCC IP
79	Finger
80	HTTP
110	POP3
113	Identd
119	News
135	Win NT

110	POP3
139	NetBios
1080	WinGate
6667	IRC Server
8080	Proxy HTTP
12076	GJamer
12345	NetBus
21554	GirlFriend
20034	NetBus 2
40426	Paradise

Puertos recopilados por [SaRRiO] · www.sarrio.org · admin@sarrio.org

SCAN DE CLONES MEDIANTE IAL

Primero vamos con conceptos basicos. El klon es na menos ke un usuario con dos conexiones a un server o una red de servidores, usea dos usuarios con la misma direccion. Podemos usar muchas tecnicas, yo prefiero usar la $ial porke es lo mas rapido al momento de aplicarlo.

Tambien podemos usar el comando /who pa hacer el scan de clones

Pero es algo lento, asi ke vamos por $ial, la sintaxis es:

$ial(mask , numero_de_usuario) [.nick / .user / .host / .addr]

Antes ke eso... la ial se encarga de almacenar todas las direcciones de los usuarios ke esten en los canales ke tu te encuentras (Internal Address List). Bueno, esto lo **debemos actualizar cada ves ke entremos a un canal usando el comando /who**. Con eso ya tendremos almacenadas en el mirc las direcciones, y de ahi en adelante se seguira actualizando a medida ke entren o salgan los usuarios

su uso...

$ial(mask , numero_de_usuario) [.nick / .user / .host / .addr]

- numero_de_usuario : nos referimos a la lista de nicks, siendo de esta manera el numero uno, el ke se encuentre al principio de esta lista. Como ejemplo, usare esta direccion.. pepito!userid@127.0.0.1. Un ejemplo entonces de $ial, seria:

  Si usamos esto: $ial(*!*@127.0.0.1,1)

  Nos devolvera como respuesta: nick!userid@127.0.0.1

Asi mismo aplicamos las propiedades

Si usamos $ial(*!*@127.0.0.1,1).nick... nos devolvera el nick del usuario

Otro $identidicador ke deberemos usar es el $nick. $nick, trabaja de la siguiente manera... $nick(#canal , numero_en_la_nicklist). Ejemplo, si escribimos... echo -a $nick(#scripting,1) , nos devolvera CEK, y asi seguidamente...

Y tambien usaremos el identificador $address. El $address traßaja de la siguiente manera: $address(nick , tipo). Esta claro no? el tipo se refiere a como nos mostrara la direccion. Por ejemplo, si escribimos echo -a $address(nick,2) nos devolvera *!*@j1.arrakis.es.

Tanto como al identificador $ial, como al identificador $nick... si en el numero_de_la_nicklist le pones un 0... este devolvera el valor total de los usuarios ke estemos buscando. Por ejemplo...

Si usais echo -a $nick(#scripting,0) tendremos el numero total de usuarios dentro de este canal, en este caso hay 10 usuarios en el canal #scripting. Ok, entonces, teniendo estos conceptos claros, nos vamos de lleno al scaneo.

Para hacer un scaneo en canales, podemos usar tanto $ial como $ialchan.

Cual es la dieferencia?

$ialchan se refiere a un canal especifico en cambio $ial, nos guarda informacion global de todas las direcciones puede ser ke haya otro ususario con un clon en otro canal y de casualidad tu tambien estes en ese canal...

Entonces tendras problemas con el who, conviene ponerlo al momento ke entras al canal. En un on 1:join y haces un group... donde lo activas... y le metes un halt al raw correspondiente pa ke no te salga la lista enera de nicks en el status claros los identificadores?

Bueno, hasta aki es la primera parte...

Lo primero ke haremos es tener nuestras varialbles en cero y hacemos un bucle para iniciar el scan

```
klon {
 say ßuscando clones en #canal
 unset %clones.*
 ...
```

Nos aseguramos de tener las limpias las variaßles para ke no hayan errores en la buskeda. Ok, ahi tenemos nuestras variables limpias, luego debemos asegurarnos ke el nick ke vamos a procesar exista.

```
if ($nick(#,%clones.num) == $null)
```

%clones.num sera el contador de la nicklist

El ejemplo anterior se traduce a if ($nick(#scripting,1) == $null)

Entonces...

```
if ($nick(#,%clones.num) == $null) { say fin de la ßuskeda | unset %clones.* | halt }
```

Si el nick ya no existe se parara la busqueda... caso contrario... seguimos ahora.. debemos asegurarnos ke el nick ke estamos procesando, no lo hayamos visto anteriormte (su ip), para esto .. al final del scaneo almacenaremos la ip... eso lo vemos despues

```
if ($address($nick(#,%clones.num),2) isin %clones.ip) { goto start }
```

Entonces, si el nick, ke estamos viendo, esta en lo ke ya teniamos... volvera el scan al principio... para pasar al siguiente nick.

Ok, ahora viene el uso de $ialchan. Como saßemos, este tiene todas las direcciones almacenadas... de manera ke si escribimos `$ialchan(123.123.123.123,#canal,0)` nos devolvera el numero de todos los usuarios ke esten conectados con la misma ip ejemplo:

```
if ($ialchan($address($nick(#,%clones.num),2),#,0) > 1) {
```

Si escribimos esto... `echo -a $ialchan($address($nick(#scripting,1),2),#,0)` nos dira ke solo hay un usuario con esa direccion... Bueno en el ejemplo lo traducimos... ke si el valor de la ip botado por $ialchan es mayor a “1”, entonces tenemos ke el usuario tiene clones.

A continuacion... debemos iniciar un bucle para la ip ke estabamos chekeando para eso, usaremos dos %variables... ke deberemos volverlas a cero, cada ves ke usemos este pekeño bucle... ya ke se lo usaremos cada ves ke nos topemos un usuario con clones. Entonces seguimos...

```
:clon
inc %clones.user.num
if ($ialchan($address($nick(#,%clones.num),2),#,%clones.user.num) != $null) {
```

Recordemos ke usamos esta linea para saßer si tenia mas clones el usuario...

```
$ialchan($address($nick(#,%clones.num),2),#,0)
```

Ahora ya no usaremos el ultimo cero, sino ke iremos aumentando para saßer la direccion especifica

`%clones.user.num` sera la variable ke llevara el numero ke tenga asignado el usuario de acuerdo a la nicklist

```
if ($ialchan($address($nick(#,%clones.num),2),#,%clones.user.num) != $null) {
```

Si el valor, osea si el usuario no es $null, osea ke existe… entonces, almacenamos el nick en una variable

```
%clones.nick = %clones.nick
$ialchan($address($nick(#,%clones.num),2),#,%clones.user.num).nick
```

Recordemos ke si a `$ialchan($address($nick(#,%clones.num),2),#,%clones.user.num)` le poniamos `.nick` al final nos devolvia el nick... (una de la propiedades de $ialchan) entonces, estaremos almacenando el nick del usuario clonado en la variable %clones.nick

```
goto clon
```

Volvemos nuevamente al punto :clon para seguir viendo si eske el usuario tiene mas clones

```
}
```

Logicamente, si es ke el usuario no tenia mas clones, se saltara esas lineas y seguimos con el siguiente comando

```
say %clones.nick ( $+ $remove($address($nick(#,%clones.num),2),*!*@) $+ )
```

Aqui entonces se avisara en este caso a la ventana actica los clones del usuario con su ip entre parentesis la direccion ke bota es en este formato logicamente *!*@127.0.0.1. Asi ke como no me interesa el nick, ni el user.. lo kito usando $remove.

Ok, como ya hemos chekeado esta ip, con todos sus clones ke pueda tener... debemos almacenarla en alguna variaßle para ke despues no vallamos a chekiar la misma ip...

```
%clones.ip = %clones.ip $address($nick(#,%clones.num),2)
```

Cerramos el bucle con un `}` y luego nos vamos al principio para seguir con el siguiente nick de la nicklist... `goto :start` y si es que el nick ke sigue ya lo hemos procesado junto con sus demas clones... lo sabremos porque su ip estara almacenada en la variable %clones.ip.

Como vemos, el scan, es un gran bucle, ke ira procesando nick por nick, de acuerdo a sus ips... se ven complicados los identificadores... pero no lo es... es logica...

Documento escrito por S|k^ri0 (Zyker) · http://krakatau.cjb.net · zyker@usa.net

CÓDIGOS RAW

Introducción

Para kien no este habituado al uso de los valores RAW, aclaro ke para usar estos hay ke seguir la misma estructura de programación ke cuando programamos cualkier tipo de remotes, eso sí, es conveniente hacerse un archivo INI especifico para los RAW, por ejemplo RAW.INI.

¿Qué son los valores RAW?

Los códigos RAW sirven para identificar las respuestas ke nos manda el servidor ante un comando ejecutado, por ejemplo, cuando ponemos /whois <nick> , si resulta ke ese tal <nick> no esta en este momento conectado al irc-servidor, el servidor nos devuelve un mensaje tal ke "NO SUCK NICK NAME", pues bien, ese mensaje ekivale al numero RAW 401, y sabiéndolo podremos modificar ese mensaje para ponerlo a nuestra forma y añadirle comandos ke mejoren nuestro script.

¿Cómo se usan?

Una utilización bastante común ke se le da a los RAW es para traducir los mensajes ke el servidor nos devuelve en ingles al castellano, ahora veremos como hacerlo. Hay ciertos RAW ke los trataremos por el contenido, es decir nos devolverá un valor por ejemplo una palabra o un numero.

Y también hay otro tipo de RAW ke los trataremos de forma más simple, es decir sencillamente SI o NO, si el servidor nos manda ese código RAW significa ke es cierto, si no, será ke no hay esa información o es falsa, un ejemplo de este ultimo tipo es el código RAW 313 ke nos manda el servidor cuando hacemos un whois a un IRCop, si nos lo devuelve damos por hecho ke es un IRCop, si no, es ke no.

La estructura para manejar estos comandos es bastante sencilla:

```
raw 314:*: {
 <COMANDOS>
}
```

Ejemplo práctico

Ya ke menciono ese código RAW de IRCop aprovecho para explicar como podemos hacernos un comprobador de Scytale, es decir, como comprobar ke el ke tiene el nick Scytale es el autentico y no un manga-claves.

Algunos pensaran ke valdría con comprobar ke al hacer un whois a Scytale nos devuelve el código RAW 313 ke ekivale a status de IRCop, sin embargo, aunke no fuese el autentico Scytale, nos devolvería ese código igualmente, pero hay otra forma de comprobarlo, y es con el mismo whois comprobando ke el host del BOT es el ke tiene ke ser, por ejemplo, el host de Scytale es theilax.arrakis.es y de esta manera podemos asegurar ke no le estamos entregando nuestras claves de autentificación a un extraño.

un ejemplo basico de esto seria:

```
[ALIAS]
ComproScy { set %comproscy on | whois Scytale }
```

```
[RAW]

raw 311:*: {
 if (%comproscy == on) {
   if ($4 == theilax.arrakis.es) { echo 4 -s scytale es autentico }
   else { echo 4 -s Scytale es Falso }
   }
}

raw 318:*: { unset %comproscy }

raw 319:*: { if (%comproscy == on) { halt } }
```

Tácticas para hacer un buen uso

Ha kedado claro ke con los RAW podemos hacer muchos cosas ke son fundamentales para un script, hay ke distingir dos funciones ke nos facilita el uso de los RAW, por una parte lo ke mencionaba antes sobre traducir los mensajes en ingles del server al castellano, esto es muy sencillo, tan solo hay ke ocultar los mensajes del server con el comando HALT y poner nuestro propio mensaje con el comando ECHO, por ejemplo, si ponemos en la sección Remotes-RAW.INI:

```
RAW 306:*: { echo -s A partir de ahora estas AWAY }
```

cuando nos pongamos en modo Away, en vez de salirnos el mensaje: "You have been marked as being away" nos pondra "A partir de ahora estas away".

Pero ese es el uso más simple ke podemos darle a los RAW, como hemos comprobado con el ejemplo anterior tiene utilidades más interesantes.

Para no caer en ciertas trampas comunes, hay ke hacer una visión general de la relación comando-respuesta de los RAW, por ejemplo, si usas Variables para definir los RAW, tal y como he hecho yo en el ejemplo de comprobación del bot, no hay ke olvidarse de borrar las variables, no seria correcto borrar la variable en el mismo momento ke nos devuelve el primer RAW (311) puesto ke en el caso de ke por casualidad el amigo Scy se hubiese caido, se nos kedaria la variable activada, y no es muy agradable ke cada vez ke hagas un simple WHOIS te aparezca el mensaje "Scytale es autentico".

La clave esta en relacionar bien todos los RAW ke pertenecen a una misma respuesta, y de esta forma ir encadenando uno con otro para conseguir el efecto deseado.

Otro error ke se da a veces es al usar las variables, los parámetros ke nos devuelve el servidor y ke usamos en forma de $1 $2 $3 etc.. o $1-, ten en cuenta ke cada $1 ekivale a una sola palabra, con ke si alguna respuesta consta de varias palabras y la ponemos como $3 solo nos saldría la primera, para ke salgan todas las palabras a partir del tercer parámetro usaríamos simplemente: $3-

Muchas veces el servidor nos devuelve parámetros de fecha, y lo hace en un formato un tanto raro, para convertirlo a un formato de fecha entendible, muchas veces tendremos ke usar el comando $ctime y $asctime, por ejemplo, cuando vemos el topic de un canal y nos dice la fecha en ke fue puesto podemos configurarlo de la siguiente manera:

```
raw 333:*: { echo -a Lo puso $3, fecha: $asctime( $4 ) | halt }
```

Algunos códigos RAW

BIENVENIDA SERVIDOR

```
001   Respuesta Bienvenida Servidor - "Wellcome to..."
002   Respuesta Bienvenida Host - "Tu host es..."
003   Respuesta Bienvenida Fecha - "this server was created..."
004   Respuesta Bienvenida
```

AWAY

```
301   Respuesta Away - "Nick esta away - <motivo>"
305   Respuesta Away - "Dejas de estar Away"
306   Respuesta Away - "A partir de ahora estas Away"
```

WHOIS

```
311   Respuesta User
            $2 ---> nick
            $3 ---> user ident
            $4 ---> host
            $6 ---> Nombre "Real" (1ª palabra)
            $7 ---> Nombre "Real" (2ª palabra)
            etc...

312   Respuesta Servidor
            $3 ---> Su servidor IRC

313     Respuesta IRCop
            $1- por defecto: "<nick> :is an IRC operator"

317     Respuesta idle
            $3 --> Tiempo de idle
            $4 --> Tiempo de Conexion al server

401   Respuesta Whois - "NO SUCK NICK..."
            Indica ke el nick no se encuentra

318   Fin del Whois – END OF WHOIS

319     Canales en los ke esta el individuo
            $3 ---> primer canal
            $4 ---> segundo canal
            etc... (usa $3- para englobarlos todos)
```

WHOWAS

```
314   Respuesta Whowas
            $2 ---> Nick
            $3 ---> User Ident
            $4 ---> Host
            $6- --> Nombre "Real"

369   Respuesta Fin del Whowas
```

WHO

```
352   Respuesta del who
            $3 ---> User Ident
            $4 ---> Host
            $6 ---> Nick

315   Fin del Who
```

TOPIC

```
331   El canal .. no tiene ningun topic
            $2 ---> Canal

332     El topic del canal .. es ..
            $2 ---> Canal
            $3- --> Topic

333   Kien y cuando se puso el topic
            $3 ---> Kien lo puso
            $4 ---> Cuando lo puso
```

OTROS

```
351   Version del servidor
            $4
            $2

421   El comando no existe
            $2 ---> comando erroneo ejecutado

481   No eres IRCop

482   No eres OP

474   No puedes pasar al canal porke estas baneado
            $2 ---> canal donde te pusieron el ban
```

Documento escrito por HOWE · www.lanzadera.com/dinamick · howe21@hotmail.com

IDENTIFICADORES DE SIMBOLO - TOKENS

¿Qué son los Tokens?

Pues son unos identificadores de simbolos mas comunmente llamados Tokens. El uso de símbolos en scripting puede ser el grupo de identificadores mas dificil de el archivo de ayuda, pero sin duda una de las mejores herramientas para el scripting.

Si quieres aprender a entender estos identificadores esta charla te ayudará en tus problemas o dudas que tienes sobre ellos. En algún tiempo, Espero que esto te ayude y te venga la inspiracion para que comencen a tener sentido los tokens para ti.

Un TokEn es tan importante como un $parm. Cada palabra de cada frase en este documento podría considerarse un Token de la frase. Cada ToKen es separado por un espacio. Por supuesto, cada frase podría considerarse un ToKen y separada por los espacios.

Básicamente, un ToKen es cualquier cosa en un conjunto de datos que puede encontrar alguna marca con la que separar algo, muchos como los espacios y los períodos separan los datos en las frases de sentencía y párrafos.

Los Token tienen como base los separadores, como dije antes,estas palabras son separadas por un espacio. Para contar los identificares Tokens que queremos usar como separador, usamos el $asc () del separator.

Se pueden determinar usando //$asc(.) por ejemplo:

El $asc() de "." es 46.
El $asc() de "!" es 33.
El $asc() de "espacio" es 32.
El $asc() de "@"es 64.

Estos se usan frecuentemente en mIRC codificando con símbolos.Verás algunos de estos en los ejemplos siguientes:

Otra cosa que nosotros debemos contar el identificador Token es qué la posición Token con la que nosotros queremos trabajar. Esta simplemente se denota en la forma de número, como 2 o 4. Verás esto también en los ejemplos siguientes.

```
$gettok(texto,N,C)
```

Da el símbolo N en el texto.

```
$gettok(a.b.c.d.e,3,46) da c
$gettok(a.b.c.d.e,9,46) da $null
```

Puedes también especificar el rango de símbolos:

```
$gettok(a.b.c.d.e,2-,46)      da el segundo símbolo entre b.c.d.e
$gettok(a.b.c.d.e,2-4,46)     da el símbolo 2 a través de 4 b.c.d
```

El ToKen más usado es $gettok. Uso $gettok bastante un monton de veces en mi codigo, como me permite a" tirón "datos específicos que quiero desde un grupo más grande de datos. Hace exactamente lo qué su nombre implica, consigue un símbolo.

En este ejemplo, como en todos los demas, se usa $fulladdress como la cosa nosotros conseguiremos símbolos. Como ejemplo, $fulladdress suponemos que sale "Foo! bar@ppp.sum.com". Este va a ser el usuario de IRC en mis ejemplos. Su nick es Foo, su ident es la bar y su dirección es ppp.sum.com.

Ahora aplicamos un $gettok.

Suponga desde esto hacer creer dirección, Busqué para algunos de razón para que se evaluaran la sum.com parte de esta dirección.

Podríamos querer hacer algo así como esto:

```
set %addy $fulladdress
set %site $gettok(%addy,2,64) --- en esto sale ppp.sum.com
%domain = $gettok(%site,2,46) $+ . $+ $gettok(%site,3,46) --- en esto sale sum.com
```

Vamos a ver que realiza el ejemplo anterior ya que puede que no haya tenido ejemplo pera tí. La 1ª cosa que hice era el set %addy. Este es simplemente el $fulladdress de Foo!bar@ppp.sum.com. Luego usé esta variable como los datos para conseguir símbolos desde en mi próximo comando.

Pongo un $gettok sobre %addy, pido el segundo (el 2) símbolo que es separado por "@". La $asc() para "@"es 64, y la razón la declaración termina con 64. Todo junto lo mira como esto: $gettok (%addy, 2,64) . Esto se lee como: consigue el segundo Token desde %addy que es separado por @. Si miras el ejemplo de la dirección, verás que los segundos datos después del @ es desde luego ppp.sum.com, tan %site es el set a ppp.sum.com. Este está cerca de lo que he querido hacer, pero no lo bastante, tanto que hago otro $gettok sobre ello. Este tiempo pido el segundo y tercero ToKeN desde el %site.

El segundo ToKeN es "sum" y el tercero es "com". Produzco un dominio agregando en el "." rendir sum.com. Por supuesto, 46 es el $asc() de "." y la razón el $gettoks termina con 46.

Una nota final aquí ...

Si trabajas con el texto "mIRC is an excellent IRC client" y lo único que quieres coger es la 3ª 4ª y 5ª palabra, debes usar $gettok($1-,3-5,32), que devuelve "an excellent IRC".

O, si trabajas con "Foo!bar@ppp.sum.com" todavía, entonces para recobrar simplemente "Foo!bar@ppp.sum" deberías usar $gettok ($fulladdress,1-2,46).

Si te confundes, trata de releerlo una vez más .Si no, entonces lee algo mas sobre los ToKeNs y deja confundirte, intentalo con otro ToKen.

```
$remtok(texto,símbolo,N,C)
```

Borra el símbolo N comparado en el texto.

```
$remtok(a.b.c.d,b,1,46) devuelve  a.c.d
$remtok(a.b.c.d,e,1,46) devuelve  a.b.c.d
$remtok(a.c.c.d,c,1,46) devuelve  a.c.d
```

$remtok trabaja mucho como $gettok, excepto se usa para quitar datos seguros desde su input. Tan, en el lugar de un ToKeN posicion como 2 o 4, contamos $remtok el texto real que queremos quitar. A excepción de esta distinción, es idéntico en la sintaxis. El dejar de aplicar el ejemplo de dirección. Quitaremos desde Foo!bar@ppp.sum.com el ToKeN "sum".

Podría mirar este ejemplo:

```
set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %modify $remtok(%addy,$gettok(%addy,2,46),46) --- saldría Foo!bar@ppp.com
```

Veamos que hizo el ejemplo. Coloqué un $fulladdress a %addy. Entonces apliqué una $remtok a ello, queriendo quitar la parte "sum" desde ello. %addy es el aporte a $remtok. Usé un $gettok para conseguir la palabra "sum" para el contar $remtok que quise quitar, y entonces contó $remtok el ToKeN para quitar era separado por periodos (46).

$instok(texto,símbolo,N,C)

Inserta símbolo en la posición N en el texto, incluso si existe en el texto.

$instok(a.b.d,c,3,46) da a.b.c.d
$instok(a.b.d,c,9,46) da a.b.d.c

El ToKeN que tengo aún para encontrar un uso es $instok, pero su uso es simple suficiente. Mete símbolos en la misma manera que $remtok los quita. Veamos a un ejemplo. Nosotros meteremos el símbolo "test" en el ejemplo entre "sum" y "com".

Nuestra dirección falsa entonces será Foo!bar@ppp.sum.test.com

set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %oddaddy $instok(%addy,test,3,46) --- saldría Foo!bar@ppp.sum.test.com

Notará un parámetro extra al $instok proceso. Yo doy lo el aporte de %addy, la prueba para agregar de "test" , la posición quise que agregara a de 3, y los separadores de los ToKeNs de 46 (período). $instok necesita toda esta que información para que trabaje correctamente: agregue a blah, qué, donde, y seperado por qué (la idea básica).

$addtok(texto,símbolo,C)

Añade un símbolo al final del texto pero solo si no está.

$addtok(a.b.c,d,46)   devuelve a.b.c.d
$addtok(a.b.c.d,c,46) devuelve a.b.c.d

$addtok es un ToKeN .Puede ser útil en crear una variable que es una lista de cosas que crece como inputs se gana (quizá nicks etc). Tiene la propiedad bonita de comprobar para ver si qué se está agregando y si existe lo que se esta agregadando(ningunas duplicaciones!). Si no existe ya, el símbolo se agrega al final del ToKeN anterior. Doy el ejemplo de agregar "Test" a nuestro ejemplo usando $addtok.

set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %edit $addtok(%addy,test,46) --- saldría Foo!bar@ppp.sum.com.test

(como un ejemplo adicional que muestra que no esta siendo duplicando su función, considerando este ejemplo)

set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %edit $addtok(%addy,sum,46) --- saldría Foo!bar@ppp.sum.com

$findtok(texto,símbolo,N,C)

Da la posición N comparando el símbolo en el texto.

$findtok(a.b.c.d,c,1,46)   da 3
$findtok(a.b.c.d,e,1,46)   da $null

Si especificas cero para N, da el número total de símbolos comparados. Otro ToKeN,nunca tenido que usar el $findtok, aunque su uso esté parecido al resto y más bien fácil de comprender. Ubica la posición de un ToKeN en datos, y devuelve su

posición numérica (ie 3 o 5). Dejar ubicar la posición de "com" es nuestro ejemplo de dirección.

```
set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %check $findtok(%addy,$gettok(%addy,3,46),46) --- saldría 3
```

Yo usé un $gettok que devuelve "com" en el lugar de simplemente agregando la palabra "com" para recordar que estos pueden combinarse cuando su trabajando con el aporte que es desconocido. Nosotros sabemos qué la dirección es y donde cosas son, pero en IRC codificando, nosotros no tenemos ese lujo. Frecuentemente toma muchas cosas juntas para producir los resultados nosotros queremos.

```
$reptok(texto,símbolo,nuevo,N,C)
```

Reemplaza el símbolo N comparado con un nuevo símbolo.

```
$reptok(a.b.c.d,b,e,1,46)     devuelve  a.e.c.d
$reptok(a.b.c.d,f,e,1,46)     devuelve  a.b.c.d
$reptok(a.b.a.c,a,e,2,46)     devuelve  a.b.e.c
```

$reptok es de los últimos ToKeNs disponibles para manipular datos.Este reemplaza un ToKeN con otro ToKeN diferente. No es usado ampliamente en más scripting, pero explico simplemente para que sea completa esta charla.Usa la misma sintaxis. Reemplazaremos el ToKeN "sum" con el ToKeN "mine".

```
set %addy $fulladdress --- saldría Foo!bar@ppp.sum.com
set %find $gettok(%addy,2,46) --- saldría sum
set %new $reptok(%addy,%find,mine,46) --- saldría Foo!bar@ppp.mine.com
```

Acontinuacion muestro otros ToKeNs que son de la misma sintaxis de los cuales os dejo a vosotros plena eleccion y pruebas para que veais lo que habeis aprendido.

```
$deltok(texto,N,C)
```

Borra N del texto.

```
$deltok(a.b.c.d,3,46)   da a.b.d
$matchtok(símbolo,cadena,N,C)
```

Da el símbolo contenido en la cadena especificada.

```
$matchtok(one two three, e, 0, 32)    devuelve 2
$matchtok(one two three, e, 2, 32)    devuelve three
```

Si especificas cero para N, da el número total de símbolos comparados.

```
$puttok(texto,símbolo,N,C)
```

Sobreescribe el símbolo N en el texto con un nuevo símbolo.

```
$puttok(a.b.c.d,e,2,46) devuelve  a.e.c.d
$wildtok(símbolos,*cadena,N,C)
```

Da el símbolo N comparando comodines en las cadenas.

```
$wildtok(one two three, t*, 0, 32)       da 2
$wildtok(one two three, t*e, 1, 32)    da three
```

Si especificas cero para N, da el número total de símbolos comparados.

Documento escrito por Flubbers para el canal #Scripting

MAS INFORMACIÓN

Este largisimo documento es una recopilación de los documentos de la web del canal #Ayuda_IRC de iRC-HiSPANO y las charlas del canal #Ayuda_Scripting de la misma red. Otros han sido extraidos de la web dedicada a los Sockets en mIRC.

Encontraras más información sobre mIRC Scripting en...

- #Ayuda_IRC – www.ayuda-irc.net
- Sockets en mIRC – www.sarrio.org/sockets
- #Ayuda_Scripting – www.scripting.es.org
- #100Scripts – http://100scripts.islaweb.com
- Chevalier’s World – www.guitarra.net/irc
- uKBoT & web de KaT – www.blackcode.com/irc

En ingles...

- Hawkee – www.hawkee.com
- Mirc.net – www.mirc.net
- X-Calibre – www.xcalibre.com
- mIRCX – www.mircx.com
- mIRC Scripts – www.mircscripts.com

Estos documentos han sido recopilados por [SaRRiO]

· admin@sarrio.org · sarrio@altecom.es · www.sarrio.org ·