

Notions intermédiaires en mIRC scripting

par [Soufiane Hassou](#)

Date de publication : 17/10/2006

Dernière mise à jour : 14/11/2006

Quelques connaissances de base en mIRC scripting !

Remerciements

I - Introduction

II - Premiers pas

II-A - Aliases

II-B - Remotes

II-B-1 - Gestion des évènements

Évènements déclenchés par une personne

Évènements spéciaux

II-B-2 - Instructions de contrôle

II-B-3 - Les groupes

II-C - Popups

II-C-1 - Popups statiques

II-C-2 - Popups dynamiques

II-D - Variables

II-D-1 - Variables globales

II-D-2 - Variables locales

II-D-3 - Opérations sur les variables

La fonction \$var

Incrémenter, décrémenter

Opérations arithmétiques

II-E - Niveaux d'accès

III - Les tokens

III-A - Définition

III-B - Fonctions de traitement des tokens

III-B-1 - \$gettok(texte,N1-N2,S)

III-B-2 - \$findtok(texte,token,N,S)

III-B-3 - \$addtok(texte,token,S)

III-B-4 - \$deltok(texte,N-N1,S)

III-B-5 - \$istok(texte,token,S)

III-C - Conversion d'un texte en tokens

IV - Fichiers

IV-A - Fichiers séquentiels

IV-A-1 - Lire dans un fichier séquentiel

IV-A-2 - Ecrire dans un fichier séquentiel

IV-A-3 - Application (cas de l'auto-greetings)

IV-B - Fichiers d'initialisation

IV-B-1 - Lire dans un fichier d'initialisation

IV-B-2 - Ecrire dans un fichier d'initialisation

V - Conclusion

VI - Pré-requis

Remerciments

Merci à Mavina et Swoog pour leur aide.

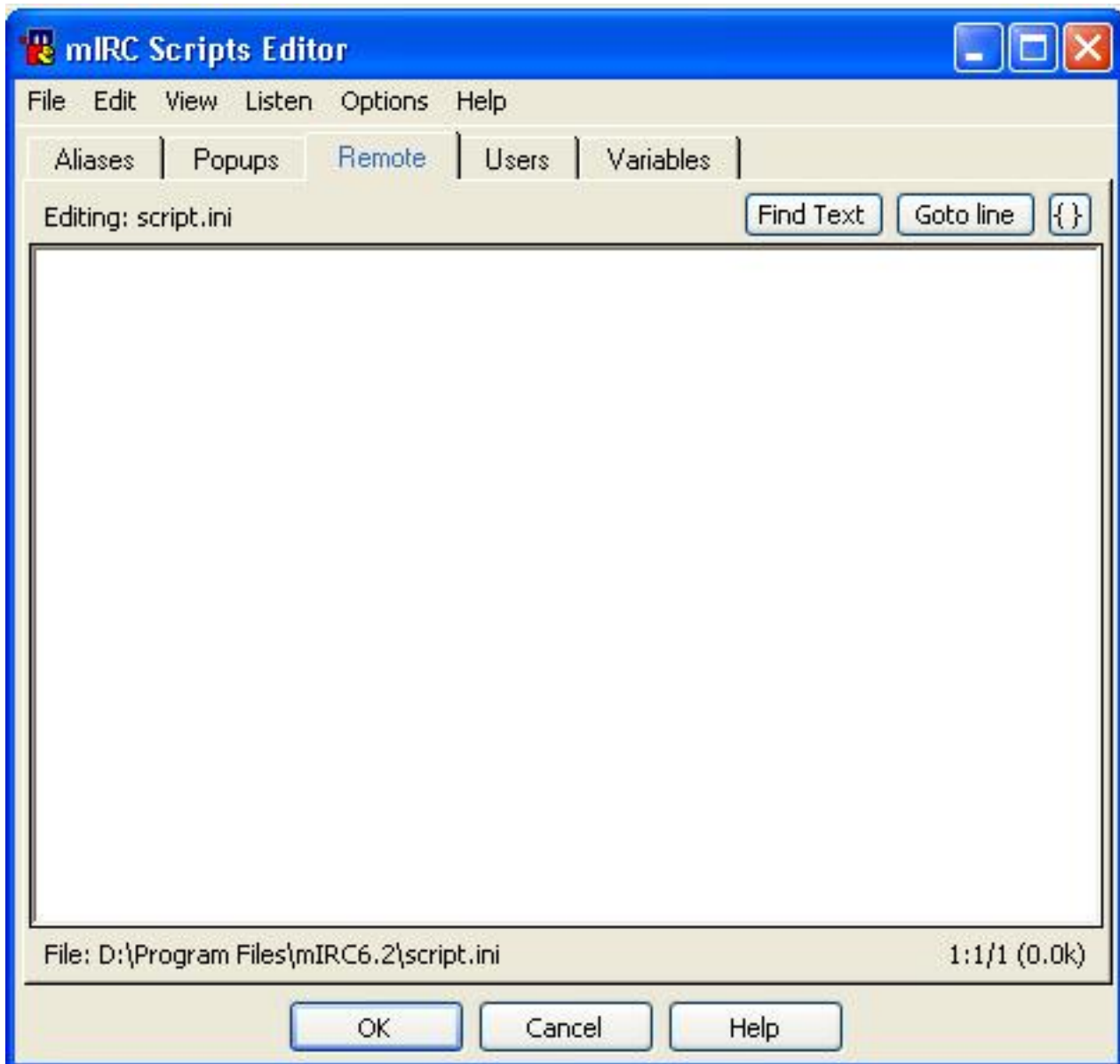
Merci à wichtounet pour sa correction orthographique et à Woufeil pour ses suggestions.

I - Introduction

La compréhension de ce tutoriel repose essentiellement sur vos connaissances de bases en mIRC scripting.

Pour acquérir ces bases, des liens sont mis à votre disposition en fin du tutoriel.

mIRC dispose de son propre éditeur de texte donc pour concevoir vos scripts vous aurez le choix entre cet éditeur et votre éditeur favori, pour y accéder déroulez le menu tools/script editor (alt+R).



- **Aliases** : Un alias est une sorte de raccourci d'une ou plusieurs lignes de code.
- **Popups** : Les popups sont les menus contextuels de mIRC (click bouton droit).

- **Remotes** : C'est la partie la plus importante et la plus avancée du mIRC scripting, c'est ici qu'on écrira nos scripts.
- **Users** : Ici on définira les niveaux des utilisateurs...
- **Variables** : Logique..., ici réside la liste des variables définies.

II - Premiers pas

On va traiter le même exemple et tout en l'améliorant on abordera les différentes notions. J'ai choisi comme exemple l'auto-greetings, en gros, on va concevoir un script qui salue toute personne entrant dans un canal.

II-A - Aliases

Un alias est un raccourci d'une ou plusieurs lignes de code.

Exemple d'un alias sur une seule ligne :

```
/salut /say salut toi ! Bienvenue sur #Developpez.com
```

/say affiche le message passé dans la fenêtre active.

Il est préférable d'utiliser /msg qui permet de définir la fenêtre où afficher le message. (\$chan, \$active, #chan, ...)

Pour un alias sur plusieurs lignes, il est préférable d'utiliser les accolades (au lieu de pipes) pour plus de clarté :

```
/salut {
/say salut toi!
/say Bienvenue sur #Developpez.com
}
```

Pour l'instant, nous n'avons fait qu'un alias qui nous permet d'afficher manuellement la phrase de salutations **générale**.

Maintenant, nous allons essayer de personnaliser la phrase pour un pseudo donné ! Rien de plus simple, on prend 3 pseudos : AjJi, mavina et swoog

```
/salut1 /msg #Developpez.com Salut AjJi ! Bienvenue sur #Developpez.com !
/salut2 /msg #Developpez.com Salut mavina ! Bienvenue sur #Developpez.com !
/salut3 /msg #Developpez.com Salut swoog ! Bienvenue sur #Developpez.com !
```

/salut1 pour saluer AjJi, /salut2 pour mavina et /salut3 pour swoog et on laisse /salut pour le reste.

Et si Jack rentre dans le canal, on fait comment ?

Une solution serait de le saluer comme les autres, avec un "salut toi!", sauf que nous on veut un "salut Jack!".

C'est simple, il suffit d'utiliser un [alias paramétré](#), un alias qui prend des arguments.

Notre alias devient ainsi :

```
/salut /say Salut $$1 ! Bienvenue sur #Developpez.com !
```

On peut aussi mettre le canal comme paramètre :

```
/salut /say Salut $$1 ! Bienvenue sur $$2 !
```

L'utilisation du \$\$ à la place de \$ est obligatoire dans notre cas (voir [la différence entre \\$1 et \\$\\$1](#))

II-B - Remotes

Comme ça a été dit plus haut, les remotes est la notion la plus importante du mIRC scripting, au point que tout peut être défini dans les remotes (Aliases, popups, ...).

II-B-1 - Gestion des évènements

Évènements déclenchés par une personne

Les remotes offrent de grandes possibilités, entre autres, celle de la gestion des évènements. Un évènement peut être une personne qui rentre dans un canal (**JOIN**), qui change de pseudo (**NICK**) ou même qui prononce une phrase (**TEXT**).

Il faut noter qu'il existe d'autres types d'évènements (du côté serveur en l'occurrence) que nous n'allons pas traiter dans ce cours.

Pour intercepter l'évènement, ainsi exécuter votre script, on utilise la syntaxe suivante :

```
ON Niveau:évènement:Fenetre:Bloc de commandes
```

- **Niveau** est le niveau de l'utilisateur concerné par cette action, on y reviendra plus tard.
- **évènement** est l'évènement (ou l'action) qui vient de se produire.
- **Fenêtre** correspond à l'endroit où l'évènement s'est produit (canal(#), privé(?), fenêtre dcc chat(=), ...).
- **Bloc de commandes** est la suite des instructions à exécuter en cas de déclenchement de l'évènement.

Après le déclenchement d'un évènement, mIRC positionne un certain nombre de paramètres selon le type de l'évènement, voici la liste des évènements les plus utilisés :

Évènement	Se déclenche lorsque une personne	Fenêtres possibles	Paramètres positionnés
JOIN	joint un canal	#	\$nick : Pseudo déclencheur \$chan : Canal du déclenchement
PART	part d'un canal	#	\$nick : Pseudo déclencheur \$chan : Canal du déclenchement
TEXT	parle	Toutes les fenêtres	\$nick : Pseudo déclencheur \$chan : Canal du déclenchement
QUIT	quitte IRC	ne nécessite pas le paramètre fenêtre	\$nick : Pseudo déclencheur

Evènement	Se déclenche lorsque une personne	Fenêtres possibles	Paramètres positionnés
			\$1- : Message de quit
OP	se fait opper (+o)	#	\$nick : Pseudo de la personne qui a oppé \$chan : Canal du déclenchement \$opnick : Pseudo de la personne oppée
DEOP	se fait déopper (-o)	#	\$nick : Pseudo de la personne qui a deoppé \$chan : Canal du déclenchement \$opnick : Pseudo de la personne deoppée
VOICE	se fait voicer (+v)	#	\$nick : Pseudo de la personne qui a voicé \$chan : Canal du déclenchement \$vnick : Pseudo de la personne voicée
DEVOICE	se fait devoicer (-v)	#	\$nick : Pseudo de la personne qui a devoicée \$chan : Canal du déclenchement \$vnick : Pseudo de la personne devoicée
NICK	change de pseudo	ne nécessite pas le paramètre fenêtre	\$nick : Ancien pseudo \$newnick : Nouveau pseudo
BAN	se fait bannir (+b)	#	\$nick : Pseudo de la personne qui a banni \$chan : Canal du déclenchement \$bnick : Pseudo de la personne bannie \$banmask : Le masque banni (pseudo!ident@host)
UNBAN	se fait débannir (-b)	#	\$nick : Pseudo de la personne qui a débanni

Évènement	Se déclenche lorsque une personne	Fenêtres possibles	Paramètres positionnés
			\$chan : Canal du déclenchement \$nick : Pseudo de la personne débannie (si le banmask contient un pseudo)
KICK	se fait kicker	#	\$nick : Pseudo de la personne qui a kické \$chan : Canal du déclenchement \$knick : Pseudo de la personne kickée
ACTION	prononce un /me	Toutes les fenêtres	\$nick : Pseudo déclencheur \$chan : Canal du déclenchement
INPUT	se déclenche lorsque vous écrivez un message.	Toutes les fenêtres	\$1, \$2, \$1-, ...

Revenons à nos moutons, je rappelle que notre but est de concevoir l'auto-greetings, donc l'évènement qui nous intéresse dans ce cas est le **JOIN**:

```
ON *:JOIN:#:/msg $chan Salut $nick ! Bienvenue sur $chan !
```

Évènements spéciaux

L'évènement **START** qui se déclenche lors de l'ouverture de mIRC, cela peut être utile pour l'affichage d'un message de bienvenue sur mIRC.

Exemple :

```
ON *:START:{
  /echo Bienvenue sur mon mIRC
}
```

L'évènement **CONNECT** qui se déclenche lors de la connexion à un serveur IRC, efficace pour gérer ses propres auto-joins (joindre automatiquement certains canaux lors de la connexion).

Exemple :

```
ON *:CONNECT:{
  /join #developpez.com
  /join #informatique
  /join #mIRC
}
```

Il en existe beaucoup du même type mais on va se limiter à ces deux évènements.

II-B-2 - Instructions de contrôle

Comme tout autre langage, mIRC scripting dispose de ses propres structures de contrôle, **if** (associée à **elsif** et **else**) et **while** (qu'on verra plus tard).

Syntaxe de l'instruction **if**:

```
if (Exp1) { Premier bloc de commandes }
elseif (Exp2) { Deuxième bloc de commandes }
else { Troisième bloc de commandes }
```

On traduit par :

Si *Exp1* est vraie alors on exécute le premier bloc de commandes sinon si *Exp2* est vraie alors on exécute le deuxième bloc de commandes, sinon on exécute le troisième.

L'expression booléenne peut contenir des opérateurs (*v1 opérateur v2*) :

Opérateur	Signification
==	égal à
===	égal à (sensible à la casse)
!=	différent de
<	inférieur à
>	supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
//	v2 est multiple de v1
\\	v2 n'est pas multiple de v1
&&	ET logique
	OU logique
&	comparaison bit à bit
isin	v1 est dans v2
iswm	v1 correspond à v2
isnum	v1 est un nombre dans v2 (v2 étant un intervalle optionnel)
isletter	v1 est une lettre dans v2 (optionnel)
ison	pseudo v1 est dans canal v2
isop	pseudo v1 est op dans canal v2
isvoice	pseudo v1 est voice dans canal v2

Pour notre programme, on va éviter de nous auto-saluer quand on rentre dans un canal, puisque dans ce cas nous sommes la personne qui a déclenché l'évènement, pour cela, on doit comparer le pseudo de la personne qui vient d'entrer avec le notre.

Sachant que mIRC met notre pseudo courant dans la variable **\$me** et que **\$nick** est le pseudo de la personne déclencheuse de l'évènement, on fait :

```
ON *:JOIN:#:{
if ( $nick != $me ) {
```

```

/msg $chan Salut $nick ! Bienvenue sur $chan !
}

```

II-B-3 - Les groupes

Le fait de définir un groupe est de rendre une portion du code activable/désactivable sur demande ! Simple non? Syntaxe de définition d'un groupe :

```

#Nom_du_groupe état_défaut
votre code ici qui peut tenir sur plusieurs lignes
#Nom_du_groupe end

```

état_défaut est l'état par défaut du groupe il peut être soit **on** soit **off**.

Pour activer le groupe :

```

/enable #Nom_du_groupe

```

Pour le désactiver :

```

/disable #Nom_du_groupe

```

Adaptons à notre cas ! Voilà le code :

```

#greetings off
if ( $nick != $me ) {
/msg $chan Salut $nick ! Bienvenue sur $chan !
}
#greetings end

```

Dès maintenant, vous pouvez activer/désactiver l'auto-greetings quand vous voulez avec des simples /enable #greetings et /disable #greetings.

II-C - Popups

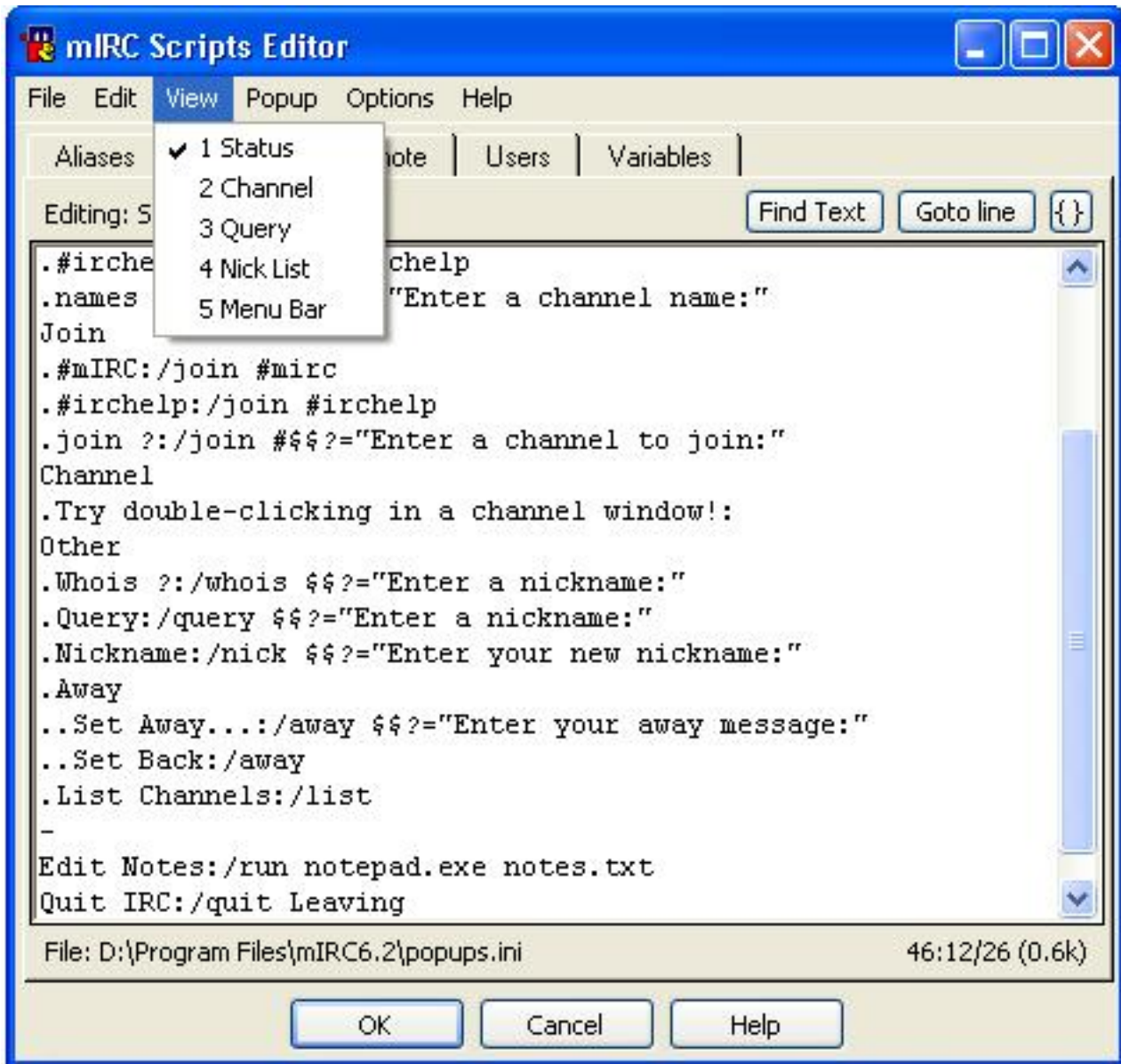
II-C-1 - Popups statiques

Comme dit dans la définition au-dessus, les popups sont les menus contextuels sous mIRC, sur chaque "type" de fenêtre on accède à un menu différent.

On distingue 5 fenêtres :

- La fenêtre status (**Status**)
- Les fenêtres canaux (**Channel**)
- Les fenêtres privées (dcc chat inclus) (**Query**)
- La fenêtre des listes des pseudos (à droite du canal) (**Nicklist**)
- La fenêtre "principale" (**Menubar**)

Pour modifier ces menus, script editor (alt+r) > onglet popups > menu view et hop !



Avant de continuer, j'aimerais apporter votre attention sur un aspect de variables qu'on n'a toujours pas abordé, je profite de l'occasion pour prendre comme exemple une portion du code que vous voyez sur l'image.

```

Join
.#mIRC:/join #mirc
.#irchelp:/join #irchelp
.join ?:/join #$$$?="Enter a channel to join:"

```

Cette syntaxe est un peu nouvelle, je l'admets, mais oublions le "." et ":" pour l'instant et regardons de plus près cette ligne :

```
.join ?:/join #$$?="Enter a channel to join:"
```

Ce code permet tout simplement de joindre le canal de votre choix et ce dans une boîte de dialogue avec un champ texte pour écrire le nom du canal.

Pas la peine de mettre le # vu qu'il déjà mentionné. Notez que si vous mettez le # ça ne posera pas de problème.

Si vous ne voulez pas mettre un # automatiquement, il suffit de l'enlever de l'expression.

Ce genre de variables vous sera très utile surtout pour la conception des popups.

Pour définir un champ dans un menu :

```
Nom_du_champ : commande
```

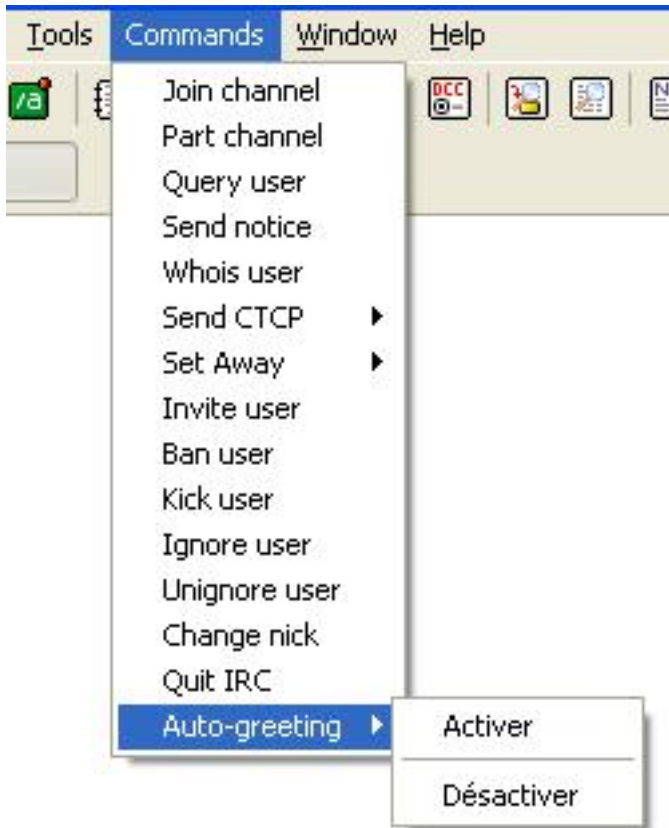
Sans oublier qu'il faut ajouter un certain nombre de points au début de la ligne, selon le menu dans lequel est situé le champ.

Pour un champ qui est dans le menu principal, pas de points, pour un champ dans un sous-menu un point, pour un sous sous-menu deux points, ainsi de suite.. Comme exemple, on va essayer d'intégrer la possibilité d'activer/désactiver notre auto-greeting depuis le menu principal de mIRC. Dans **view -> menubar**, on ajoute cela à la fin du code (vous pouvez supprimer le code, faites comme vous voulez):

```
Auto-greeting  
.Activer:/enable #greetings  
.-  
.Désactiver:/disable #greetings
```

.- permet d'ajouter un séparateur entre les champs, pratique !

Aperçu :



II-C-2 - Popups dynamiques

Vous avez sûrement remarqué qu'on pouvait appuyer sur désactiver (ou activer) même si le groupe est déjà désactivé (ou activé).

L'utilisation des popups dynamiques va nous permettre d'activer/désactiver l'appui sur un champ quand on veut.

On va tester si le groupe est activé, si oui, on désactive l'appui sur "Activer", sinon on désactive l'appui sur "Désactiver".

La fonction **\$group(#nom_du_group)** vaut **on** si le groupe est activé et Bien sûr **off** s'il est désactivé.

Pour cela, on utilisera la fonction **\$iif** :

```
$iif(expression,valeur si vrai,valeur si faux)
```

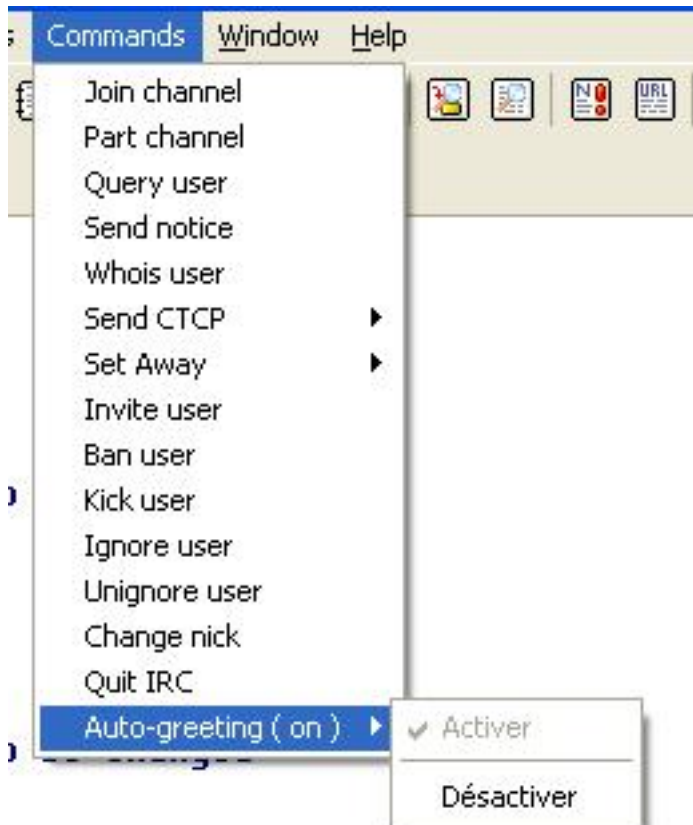
Pour activer/désactiver les champs, on aura besoin de la fonction **\$style(N)** qui définit le style du champ.

- **N = 1** champ coché
- **N = 2** Appui sur le champ désactivé
- **N = 3** les deux

Voici le code modifié :

```
Auto-greeting ( $group(#greetings) )
.$iif($group(#greetings) == on,$style(3)) Activer:/enable #greetings
.-
.$iif($group(#greetings) == off,$style(3)) Désactiver:/disable #greetings
```

Ce qui donne :



II-D - Variables

Une **variable** s'écrit de la façon suivante :

```
%Nom_de_la_variable
```

La valeur d'une variable peut être utilisée dans tous vos scripts (**Variable globale**) comme elle peut n'être exploitable que dans un seul script (**Variable locale**).

II-D-1 - Variables globales

Une variable globale peut être utilisée dans tous vos scripts et ne sera pas supprimée tant que vous ne l'aurez pas fait.

Pour définir une variable globale (en l'occurrence lui affecter une valeur):

```
/set [-snzueN] %nom_de_la_variable valeur
```

Expliquons les options :

- **-uN**: efface la variable après N secondes.
- **-z**: décrémente la variable jusqu'à ce qu'elle soit à 0 puis l'efface.
- **-s**: affiche un message lors de l'affectation.
- **-n**: traite la valeur comme un texte normal.
- **-e** : efface la variable lorsque mIRC est fermé.

Pour effacer une variable, il suffit de faire un :

```
/unset -s %nom_de_la_variable
```

*Il est possible d'utiliser des **caractères génériques** dans votre unset, (*) remplace zéro, un ou plusieurs caractères, tandis que, (?) remplace un caractère quelconque.*

Pour effacer toutes les variables définies :

```
/unsetall
```

Bien sûr, les variables définies sont consultables dans l'onglet "variables" du ScriptEditor.

II-D-2 - Variables locales

Une variable locale est une variable qui ne peut être utilisée que dans le script dans lequel elle a été définie, donc elle s'efface automatiquement après la fin du script en question.

Pour définir une variable locale :

```
/var %nom_de_la_variable = valeur
```

II-D-3 - Opérations sur les variables

La fonction \$var

```
$var(%nom_de_la_variable,N)
```

Retourne le nom de la Nième variable correspondant à notre "requête" en cherchant dans les variables locales ET globales.

Exemple:

Supposons que nous avons définis les variables suivantes

```
%a 22
%b dvp
%c 2006
%c2 Ajji
%cx oui.
```

\$var(%a,1) retourne %a

\$var(%c,1) retourne %c

\$var(%c,2) ne retourne rien

alors que **\$var(%c*,2)** retourne %c2 et **\$var(%c*,3)** retourne %cx

En effet, %c* correspond à %c %c2 %cx ce qui explique le retour du \$var.

si N=0, cela retourne le nombre de variables correspondantes, **\$var(%c*,0)** retourne 3.

Incrémenter, décrémenter

Pour incrémenter une variable avec une certaine valeur :

```
/inc [-cszeuN] %Nom_de_la_variable valeur
```

- **-uN**: Incrémente la variable par la valeur une seule fois et l'efface après N secondes
- **-z**: Incrémente la variable par la valeur une seule fois et la décrémente jusqu'à ce qu'elle soit à 0 puis l'efface.
- **-c**: Incrémente la variable par la valeur puis l'incrémente par 1 une fois par seconde
- **-s**: affiche un message lors de l'incrémentation.
- **-e** : Incrémente la variable par la valeur une seule fois et l'efface lors de la fermeture de mIRC.

Pour decrementer une variable avec une certaine valeur :

```
/dec [-cszeuN] %Nom_de_la_variable valeur
```

- **-uN**: décrémente la variable par la valeur une seule fois et l'efface après N secondes
- **-z**: décrémente la variable par la valeur une seule fois et la décrémente jusqu'à ce qu'elle soit à 0 puis l'efface.
- **-c**: décrémente la variable par la valeur puis l'incrémente par 1 une fois par seconde
- **-s**: affiche un message lors de la décrémentation.
- **-e** : décrémente la variable par la valeur une seule fois et l'efface lors de la fermeture de mIRC.

Opérations arithmétiques

Sachez que vous pouvez faire autant d'opérations sur les variables que vous voulez, affectation (=), addition (+), soustraction (-), multiplication (*), division (/), reste de division (%), puissance (^), ...

```
%nombre = 1
%a = %nombre + 3
```

```
%b = %a / 5  
%c = %b ^ $1
```

Pour faire des calculs un peu plus compliqué mieux vaut utiliser la fonction **\$calc()**

```
//echo -a $calc(6.11 * ((%a / 10) - %nombre) ^ %c
```

II-E - Niveaux d'accès

Les niveaux d'accès d'un utilisateur sont en relations avec les évènements.

Rappelez-vous la syntaxe pour l'interception d'un évènement :

```
ON Niveau:évènement:Fenetre:Bloc de commandes
```

Un utilisateur ne pourra déclencher un évènement que s'il a le niveau requis pour.

Les niveaux des utilisateurs sont définis dans l'onglet "Users" de la façon suivante :

```
<niveau1,niveau2,...>:<adresse de l'usager>
```

Le premier niveau est un niveau générale, et les autres sont des niveaux spécifiques, en d'autres mots, un utilisateur fera réagir tous les évènements qui nécessitent un niveau inférieur ou égale à son niveau général et égal à ses niveaux spécifiques.

Exemple :

```
3,5,11:AjJi!mirc@user.dvp.com
```

L'utilisateur aura accès aux évènements dont le niveau est soit 2, 3, 5, 11 (1 étant le niveau par défaut pour tous les utilisateurs et donc les évènements de niveau 1 seront accessibles à tous les utilisateurs). Il existe un autre type de niveaux à part les niveaux numériques, les niveaux nommés.

Exemple :

```
5,friend:*!*@*.dvp.com
```

(Pour les (*), référez-vous au chapitre sur les caractères génériques).

Pour obliger mIRC à traiter le premier niveau comme un niveau spécifique, il suffit d'ajouter un signe = avant la liste des niveaux.

Vous pouvez ajouter ces accès manuellement comme vous pouvez les ajouter via des commandes, il y en a plusieurs, parmi elles :

```
/guser [-a] niveaux pseudo [type] [info]
```

Donne les accès mentionnés à l'utilisateur, si l'option **-a** est spécifié, l'utilisateur sera créé dans la liste s'il n'existe pas.

```
/ruser [niveaux] pseudo|adresse [type]
```

Si les niveaux ne sont pas mentionnés, la commande supprime l'utilisateur, sinon elle supprime les niveaux mentionnés de la liste des niveaux de l'utilisateur, si tous les niveaux sont supprimés l'utilisateur le sera aussi.

Bravo ! Vous venez de finir la partie la plus importante du scripting IRC.

Toutes les bases ne sont pas mentionnées ci-dessus Bien sûr mais une très grande partie y est !

Vous n'êtes pas obligé de lire les chapitres suivants successivement, bonne lecture.

III - Les tokens

III-A - Définition

Un Token (ou jeton) est un ensemble de caractères (pas forcément un mot).

Pour parler de Tokens, on parle avant tout de séparateur (un caractère [ASCII](#)) qui sépare un ensemble de caractères en plusieurs sous-ensembles (tokens).

Exemple:

Soit la ligne suivante :

```
123;abc:xy;z;2006
```

Si on prend le point-virgule comme séparateur :

```
123;abc:xy;z;2006
```

Notre premier Token sera **123**, le deuxième **abc:xy:z** et le troisième **2006** Alors que si on prend les deux-points comme séparateur, le premier Token sera **123;abc**, le deuxième **xy** et le troisième **z;2006**.

Une application simple des Tokens pourrait être un script qui isole le jour, mois et année d'une date.

Bien sûr mIRC offre une multitude de fonctions pour gérer les Tokens, c'est ce qu'on va voir dans ce chapitre.

III-B - Fonctions de traitement des tokens

III-B-1 - \$gettok(texte,N1-N2,S)

Cette fonction permet de chercher un token dans une chaîne de caractères.

- **texte** - texte contenant les tokens
- **N1-N2** - N1 étant la position du token dans le texte (N2 optionnel servant à définir un intervalle ex: 3-5)
- **S** - la valeur ASCII du caractère séparateur

Exemple:

```
$gettok(1;2;3,2,59) retourne 2  
$gettok(a.b.c.d,3-,46) retourne c.d  
$gettok(a.b.c.d,5,46) retourne $null  
$gettok(1.2.3.4,-2,46) retourne 3
```

- Une valeur négative de N1 retourne la position du Token en partant de la fin.
- Les séparateurs au début et à la fin d'une chaîne de caractère sont tout simplement omis.
- Si un séparateur est en double (;;) il est tout considéré comme un seul séparateur (;).
- Si vous utilisez comme séparateur un caractère non existant dans la chaîne, **\$gettok(texte,1,S)** retournera toujours texte.
- **\$gettok(texte,0,S)** retourne le nombre de tokens trouvés dans la chaîne ce qui peut être intéressant pour

parcourir celle-ci et afficher les tokens un par un.

III-B-2 - \$findtok(texte,token,N,S)

Cette fonction retourne la position du token donné en paramètre.

- **texte** - texte contenant les tokens
- **token**- token recherché
- **N**- N est le Nième token trouvé dans la chaîne (N = 0 retourne le nombre total trouvé)
- **S**- la valeur ASCII du caractère séparateur

Exemple:

```
$findtok(1;2;3,x,1,59) retourne $null
$findtok(a.b.c.d,c,1,46) retourne 3
$findtok(a.b.c.c.d.c,c,2,46) retourne 4
```

III-B-3 - \$addtok(texte,token,S)

Cette fonction ajoute un token à la liste des tokens s'il n'y est pas déjà, il retourne le texte avec le token ajouté si l'opération a eu lieu.

- **texte** - texte contenant les tokens
- **token**- token à ajouter
- **S**- la valeur ASCII du caractère séparateur

Exemple:

```
$addtok(1;2;3,59) retourne 1;2;3
$addtok(a.b.c,c,46) retourne a.b.c
$addtok(a.b.c,f,59) retourne a.b.c;f
```

III-B-4 - \$deltok(texte,N-N1,S)

Cette fonction efface le token dont la position est N et retourne le texte modifié.

- **texte** - texte contenant les tokens
- **N-N1**- N étant le numéro du token dans le texte (N1 optionnel servant à définir un intervalle ex: 3-5)
- **S**- la valeur ASCII du caractère séparateur

Exemple:

```
$deltok(1;2;3,59) retourne $null
$deltok(a.b.c,1,46) retourne b.c
```

III-B-5 - \$istok(texte,token,S)

Cette fonction retourne \$true si le token est dans le texte sinon il retourne \$false.

- **texte** - texte contenant les tokens
- **token**- token à comparer
- **S**- la valeur ASCII du caractère séparateur

Exemple:

```
$istok(1;2,3,59) retourne $false  
$istok(a.b.c,a,46) retourne $true
```

Il existe plusieurs fonctions pour les tokens, je vous invite fortement à les consulter directement sur l'aide de mIRC.

III-C - Conversion d'un texte en tokens

```
/tokenize S texte
```

- **S** - la valeur ASCII du caractère séparateur
- **texte**- texte contenant les tokens

La commande **/tokenize** convertit un texte en tokens numérotés pour faciliter leur utilisation.

```
/tokenize 45 t1-t2-t3-t4
```

convertira t1 en \$1, t2 en \$2, t3 en \$3 et t4 en \$4.

\$0 est le nombre de tokens.

Pour avoir le code ASCII d'un caractère :

```
//echo -a $asc(caractère)
```

IV - Fichiers

mIRC permet de gérer deux types de fichiers :

- Les fichiers séquentiels.
- Les fichiers d'initialisation (**ini**).

Comme pour les tokens, mIRC met en votre disposition un ensemble de fonctions, pour écrire, lire, ... dans un fichier.

IV-A - Fichiers séquentiels

Un **fichier** séquentiel permet uniquement d'accéder aux données dans l'ordre de leur écriture.

Pour accéder au Nième caractère, on est obligé de passer par les N-1 caractères d'avant.

IV-A-1 - Lire dans un fichier séquentiel

La commande **/play** permet de **jouer** un fichier texte ou une partie de celui-là à un utilisateur ou un canal.

En plus simple, elle permet de lire une partie du fichier et l'afficher.

```
/play [-aescpbn q# m# f# rl# t#] [alias] [canal/pseudo/stop] nom_du_fichier [délai]
```

- **nom_du_fichier** : le nom du fichier que vous voulez lire.
- **-a** : /play utilisera l'alias mentionné au lieu de /msg ou /notice.
- **-e** : le texte sera affiché en /echo.
- **-s** : la commande /play sera faite dans la fenêtre status en étant déconnecté. Donc si vous êtes déconnecté, vous êtes obligé de mentionner **-s** avec /play pour qu'elle marche.
- **-c** : le texte sera évalué.
- **-n** : /play utilisera /notice (au lieu de /msg).
- **-p** : cette demande sera prioritaire aux autres demandes de play (valable qu'au cas où plusieurs demandes play sont en cours).
- **-q#** : spécifie le nombre maximum de requêtes possibles en liste d'attente, si le nombre est supérieur ou égal à la valeur donnée, cette requête sera ignorée.
- **-m#** : limite le nombre maximum de requêtes possible en liste d'attente, si le nombre est supérieur ou égal à la valeur donnée, cette requête sera ignorée.
- **-r** : une ligne au hasard du fichier sera jouée.
- **-f#** : joue le fichier de la ligne spécifiée à la dernière ligne.
- **délai** : le délai d'attente entre la lecture/affichage de chaque ligne.

*Les paramètres **-q#** et **-m#** ne s'appliquent qu'à des /play exécutés par un déclenchement d'un évènement.*

*Pour arrêter le déroulement du /play et vider la liste, vous faites **/play stop***

IV-A-2 - Ecrire dans un fichier séquentiel

Pour cela, on a à notre disposition la commande **/write**.

```
/write [-acdin l# s# w# r#] Nom_de_fichier Texte
```

Cette commande écrit le **Texte** dans le fichier **Nom_de_fichier**, elle accepte plusieurs paramètres:

- **-a** : le texte est ajouté au texte de la ligne spécifiée.
- **-c** : efface le fichier complètement avant d'écrire.
- **-d** : efface une ligne du fichier, si aucune ligne n'est spécifiée (cf paramètre -l#) c'est la dernière ligne du fichier qui sera effacée.
- **-i** : Insère le texte spécifié dans la ligne spécifiée, si aucun texte, une ligne vide sera insérée, si aucune ligne spécifiée, l'insertion sera dans la fin du fichier.
- **-n** : empêche l'ajout d'un **\$CrLf** à la fin du texte.
- **-l#** : spécifie le numéro (#) de la ligne où l'action sera faite (ajout, suppression, ..)
- **-s#** : recherche une ligne commençant par le texte spécifié (#) et exécute les opérations sur cette ligne (les opérations étant les autres paramètres).
- **-w#** : même comportement que le **-s#** sauf que le texte ici peut contenir des caractères génériques.

\$CrLf (carriage return/linefeed): retourne le caractère de fin de ligne suivi du caractère de nouvelle ligne.

Exemple :

```
/write -dstest exemple.txt
```

Recherche une ligne dans le fichier exemple.txt qui commence par "test" et l'efface (-d).

IV-A-3 - Application (cas de l'auto-greetings)

Revenons à notre petit script, on va essayer d'exploiter **/play** et **/write** pour faire un auto-greetings plus évolué.

Le code de notre auto-greetings, pour l'instant, ressemble à :

Remote

```
#greetings off
if ( $nick != $me ) {
/msg $chan Salut $nick ! Bienvenue sur $chan !
}
}
#greetings end
```

Popup

```
Auto-greeting ( $group(#greetings) )
.$iif($group(#greetings) == on,$style(3)) Activer:/enable #greetings
.-
.$iif($group(#greetings) == off,$style(3)) Désactiver:/disable #greetings
```

A faire :

- Rendre le message de bienvenue aléatoire parmi les messages contenus dans un fichier **hello.txt**.
- Modification du popup, ajout de champs pour ajouter/supprimer des messages de bienvenue et pour modifier le fichier des messages.

Commençons par la remote, au lieu d'un message figé, on demande à mIRC de lire un message aléatoire du fichier **hello.txt**

hello.txt

```
salut $nick ! bienvenue sur $chan !
Oh ! salut $nick
lut $nick
hello $nick $+ , comment ca va ? :)
```

Notre script deviendra :

```
#greetings off
ON *:JOIN:#:{
  if ( $nick != $me ) {
    //play -r $chan %fichier
  }
}
#greetings end
```

-r permet de jouer une ligne aléatoire.

La variable **%fichier** sera la variable qui contiendra le chemin vers le fichier cible (le fichier des messages).

Regardons les popups maintenant, examinez le code suivant :

```
Auto-greeting ( $group(#salut) )
.Changer le fichier ( $+ %fichier $+ ):/set %fichier $$?="Chemin du fichier"
.Ajouter un message:/write -a %fichier $$?="Entrez le message :"
.Effacer un message:/write -dl $+ $$?="Numéro de la ligne à supprimer" %fichier
.-
.$iif($group(#salut) == on,$style(3)) Activer:/enable #salut
.$iif($group(#salut) == off,$style(3)) Désactiver:/disable #salut
```

*Essayez de définir la variable **%fichier** (**changer le fichier**) avant de faire tout autre opération pour éviter l'erreur de manque d'arguments du /write (fichier non spécifié).*

IV-B - Fichiers d'initialisation

Les fichiers d'initialisation (d'où l'extension **.ini**) contiennent plusieurs éléments, chaque élément est caractérisé par :

- **Sections** : les sections commencent par un '[' et se terminent par un ']', c'est la plus grande entité dans le fichier.
- **Paramètres** : les paramètres commencent par une clé, suivie de '=', puis la valeur de la clé.
- **Commentaires** : Toutes lignes qui commencent par un ';'.

exemple.ini

```
[section1]
clé1=val1
clé2=val2
[section2]
clé1=val1
clé2=val2
```

IV-B-1 - Lire dans un fichier d'initialisation

\$readini permet de lire dans un fichier séquentiel.

```
$readini(nom_de_fichier, (np), section, paramètres)
```

- **n** : la ligne lue sera traitée comme du texte et ne sera pas évaluée.
- **p** : le séparateur de commandes '|' sera traité en tant que tel et non en tant que texte normal.
- **Section** : la section où vous voulez lire.
- **paramètre** : le paramètre que vous voulez lire dans la section.

Exemple:

```
//echo $readini(exemple.ini, section2, clé1)
```

Affichera **val1**.

IV-B-2 - Ecrire dans un fichier d'initialisation

```
/writeini -n fichierINI section paramètre valeur
```

- **fichierINI** : le fichier dans lequel vous voulez écrire.
- **Section** : la section que vous voulez créer (en cas d'inexistence), où modifier (modifier la valeur d'un paramètre en cas d'existence).
- **Paramètre** : le paramètre à modifier ou à ajouter.
- **valeur** : la nouvelle valeur du paramètre entré.

Vous êtes obligé de donner tous les arguments sinon la commande ne s'exécutera pas.

Évitez d'utiliser cette commande sur un fichier utilisé par mIRC.

*Vous pouvez effacer des sections et/ou paramètres à l'aide de **/remini**.*

V - Conclusion

Cet article couvre en gros les aspects les plus importants à savoir pour mieux assimiler les notions avancées en mIRC scripting (regex, dialogs, ...).

Je vous invite à utiliser l'aide du mIRC (F1) le plus souvent possible, c'est LA référence.

Bon scripting !

VI - Pré-requis

- [Scripting mIRC : prise en main](#) : Première approche du mIRC scripting par des exemples simples.
- [Le Protocole IRC](#) : Connaissances de base concernant le protocole IRC.