SCAT

manuel de l’utilisateur

version 1.12

Introduction

Scat est un programme destiné à tester le programme enfoui dans une carte à puce. Il permet envoyer des commandes via un lecteur PC/SC.

Scat peut s’utiliser en mode interactif ou par l’envoi d’un lot de commandes depuis un fichier. Il lit des commandes sur l’entrée standard et affiche un résultat.

Le programme peut aussi être invoqué avec une liste d’arguments qui représente une liste de fichiers. Ces fichiers sont alors lus dans l’ordre de leur apparition dans la ligne de commande. Le programme s’arrête dès la lecture d’une première commande *exit*.

Lors de son lancement sans argument, le programme vérifie si un ou plusieurs lecteurs sont branchés et en affiche la liste. Si aucun lecteur n’est présent, le programme s’arrête. Si au moins un lecteur est présent, le programme affiche une invite de commande « `*` ». Chaque lecteur est repéré par un numéro.

```
Scat version 1.10
2 lecteur-s connecté-s
0 : SCM Microsystems Inc. SCR 355 [CCID Interface] 00 00
1 : Gemalto PC Twin Reader 01 00
*
```

L’utilisateur est alors invité à introduire une commande.

Généralités sur le langage

Une ligne de commande peut prendre deux formes :

1. forme du nom d’une commande suivi par ses paramètres.

   ```
   COMMANDE A1 ... An
   ```

2. un lot de données envoyés à la carte.

   ```
   d1 d2 ... dn
   ```

Le caractère # est un marqueur de début de commentaire. Tous les caractères qui suivent jusqu’à la fin de la ligne sont ignorés. L’interprétation redevient active au début de la ligne suivante.

Les paramètres des commandes et les lots de données transmises à la carte sont des listes d’octets. Ces listes être lus ou affichés selon les modes suivants :

1. le mode liste d'octets au format hexadécimal, par exemple : `62 6f 6e 6a 6f 75 72`
2. le mode chaîne de caractères ascii, par exemple : `"bonjour"`
3. le mode entier en numération décimale `%d32211726873245 6802`, ou hexadécimale `72756f6a6e6f62`. Les octets d'une table peuvent être interprétés selon la convention *little endian*, c'est-à-dire en considérant la liste des octets du poids faible vers le poids fort, ou selon la convention *big endian*, qui considère la liste des octets du poids fort vers le poids faible. Deux commandes permettent d'assigner l'une des deux conventions.

Quel que soit le format de lecture des expressions, c'est la table d'octets qui lui correspond qui est envoyé à la carte.

Des attributs permettent de tronquer, d'extraire, de dupliquer et de compléter les données avec des zéros.

Les alias

Il est possible de donner de nom symboliques à des expressions. Ces noms symboliques sont appelés *alias*.

Un alias a une taille fixe. Il est déclaré par la commande

```
alias index       # pour déclarer un alias d'un octet de nom "index"
```

ou

```
alias tableau[8]  # pour déclarer un alias de 8 octets
```

Une fois un alias déclaré, on ne peut plus modifier sa taille.

Un alias se traite comme une expression. Pour l'invoquer, son nom doit être précédé par le symbole $.

Grammaire des expressions

Une expression définit une liste d'octets. La grammaire de définition des expressions est décrite ci-après.

Une *expression* est une *expression simple*, ou bien une opération de comparaison appliquée à deux expressions simples :

expression ::== *expression_simple, expression_simple* {==, >=, <=, !=, >, <} *expression_simple*

Une *expression simple* est une liste de composantes :

expression_simple ::== *composante* [*composante*]*

Une *composante* est constituée d'un ou plusieurs termes reliés par les opérateurs + (addition), - (soustraction) | (ou logique) ou ^ (ou exclusif)

composante::== *terme* [{ **+**, **–**, **|**, **^}** *terme*]*

Un terme est constitué d'un ou plusieurs facteurs reliés par les opérateurs * (multiplication) ou & (et logique)

terme ::== *facteur* [{**`*`**, **`&`**} *facteur*]*

Un facteur est une expression entre parenthèses, ou l'application d'une fonction prédéfinie à une expression, ou une donnée. Un facteur est éventuellement suivi d'une liste d'attributs qui permettent d'accéder aux différents octets d'une expression.

facteur_simple ::=
{(*expression*), ***length*** (*expressoin*), **sizeof**(*expression*), *constante*} *liste_attributs*,

liste_attributs ::= { [*expression*], **`:`** *facteur*, **`;`** *facteur*, **`.`** *facteur* }*

facteur { [{ **`*`**, **`;`**, **`:`** } *facteur*] , [**`[`** *expression* **`]`**] }

Une constante est un octet exprimé en notation hexadécimale, ou un entier, en numération décimale ou hexadécimale, ou une chaîne de caractères entre doubles quotes, ou un nom symbolique appelé *alias*.

constante::= *octet* , **`%`** {**`d`** , **`x`**} *chiffres* , **`"`** *caractères* **`"`** , **`$`** *alias*

Cette grammaire définit l'ordre de priorité des opérateurs. Ainsi, l'attribut d'un facteur a la priorité la plus haute, suivi par les opérateurs multiplicatifs **`*`** et **`&`**. Ensuite viennent les opérateurs additifs **`+`**, **`–`**, **`|`**, et **`^`**, puis finalement la concaténation.

Les résultats des opérateurs arithmétiques dépendent de la convention *big_endian* ou *little_endian.*

Ainsi, en mode *little_endian*, l'expression `1 + (1 2 3)` a pour valeur `02 02 03`, alors qu'en mode *big_endian*, elle a pour valeur `01 02 04`.

Les expressions suivantes sont des constantes :

- octet : `3b`
- nombre en numération décimale : `%d1789`
- nombre en numération hexadécimale : `%x6b3daa`
- une chaîne de caractère : `"bonjour\"monsieur !\n"`
- un alias : `$header`

Pour accéder au contenu d'un alias, il faut faire précéder son nom par le caractère `$`.

Il existe cinq alias prédéfinis :

- `reply` : Cet alias de 258 octets contient la réponse de la carte après l'exécution d'une commande.
- `sw`, `sw1` et `sw2`, respectivement de taille 2, 1 et 1 octets contiennent le *status word*, c'est-à-dire les deux derniers octets de la réponse.

- `time`, de taille 4 contient le temps, en millisecondes, entre le dernier *reset* de la carte et l'exécution de la dernière commande apdu.

En mode *little endian*, la constante `%xabcdef` renvoie la liste d'octets `ef cd ab`.

En mode *big endian*, la constante `%xabcdef` renvoie la liste d'octets `ab cd ef`.

Le mode par défaut au démarrage est le mode *little endian*.

La fonction `sizeof` opère sur une expression et renvoie son nombre d'octets dans le lot. Le résultat est un lot d'un seul octet.

La fonction `length` opère sur une expression et renvoie son nombre d'octets jusqu'au premier zéro. Il est destiné à opérer sur des lots d'octets dont l'interprétation est une chaîne de caractères ascii avec zéro terminal.

- `length("bonjour")` renvoie la valeur 7, constituée d'un octet égal à 07.

Les caractères utf-8 multi octets comptent pour autant d'octets qu'ils contiennent.

- `length("été")` renvoie la valeur 5, le caractère unicode `é` étant codé sur deux octets.

Pour opérer sur un lot d'octet, il faut introduire des parenthèses :

- `sizeof(ab ac 12 44)` renvoie la valeur 4
- `length(ab cd 0 33)` renvoie la valeur 2, le zéro étant considéré comme terminal.
- `length(ab) ac a2 44` renvoie la valeur `01 ac a2 44`

Les attributs d'un facteurs sont facultatifs, et leur nombre est indifférent. Chaque attribut opère sur la donnée qui le précède.

L'attribut `.` duplique la donnée autant de fois que l'indique le second opérande :

- "`abc".3` renvoie `"abcabcabc"`
- `"abc".1` renvoie `"abc"`
- `"abc".0` renvoie la valeur 0, constituée d'un octet nul.

L'attribut : (deux points) recopie par la droite et remplit éventuellement avec des zéros sur la gauche.

Lorsque le premier terme a une taille inférieure à la valeur donnée par le second terme, des zéros sont ajoutés sur la gauche.

- L'expression `(0a 0b 0c) : 5` produit les octets `00 00 0a 0b 0c`

L'attribut ; (point-virgule) recopie par la gauche et remplit éventuellement avec des zéros sur la droite :

- L'expression `(0a 0b 0c 0d) ; 3` produit les octets `0a 0b 0c`

Lorsque le premier terme a une taille inférieure à la valeur donnée par le second terme, des zéros sont ajoutés sur la droite :

- L'expression `(0a 0b 0c) ; 5` produit les octets `0a 0b 0c 00 00`

L'attribut [*n*] renvoie la liste des octets à partir du *n*-ième. En mode *little endian*, l'expression `%x1234abcedf[2]` renvoie `ab 34 12`

Les attributs peuvent se suivre consécutivement, ainsi :

- `%x1234abcdef[2]:2*.4` renvoie la liste `34 12 34 12 34 12 34 12`

Finalement, dans une expression, tous les termes sont concaténés les uns aux autres :

- l'expression `%d100 34 ab "toto"` renvoie la liste d'octets `64 34 ab 74 6f 74 6f`

Lors de la lecture d'une chaîne de caractère, les caractères d'échappement suivants sont reconnus avec leur valeur décimale :

`\a`	7	bip
`\b`	9	Retour arrière
`\e`	27	échappement
`\f`	12	Saut de page
`\n`	10	Saut de ligne
`\r`	13	Retour chariot
`\t`	8	Tabulation horizontale
`\v`	11	Tabulation verticale

Pour tous les autres caractères, le code d'échappement est le caractère lui-même, par exemple `\"` est le caractère `"`, `\c` est le caractère ascii `c`, etc.

exemples

Expression	Résultat
`(0a 0b 0c 0d):3`	`0a 0b 0c`
`(0a 0b 0c) : 5`	`00 00 0a 0b 0c`
`5221.5`	`52 21 21 21 21 21`
`(5221).5`	`52 21 52 21 52 21 52 21 52 21`
`"\eab\"\n"`	`1b 61 62 22 10`

Liste des commandes

La syntaxe pour invoquer un alias comme destination d'une expression est la suivante :

`<nom>[offset];longueur`

L’affectation a lieu à partir de l’octet n° *offset* sur un nombre d’octets égal à *longueur.* Les octets d’un alias sont numérotés à partir de zéro.

Exemple :

```
* alias buffer[10]          # définit un alias de 16 octets
* set buffer[a];3 ff.3      # affecte ff aux octets 10, 11 et 12
* say $buffer               # affichage du contenu de "buffer"
 00 00 00 00 00 00 00 00 00 00 ff ff ff 00 00 00
```

alias <nom> ou alias <nom>[n]

Déclare un nouvel alias d’une taille spécifiée. Sans indication, la taille par défaut est 1.

```
* alias index[1]      # définit un alias de 1 octet
* alias one_byte      # définit un autre alias de 1 octet
```

big_endian

Cette commande commute la convention d’ordre des octets pour représenter les entiers en mode *big endian*. Avec cette convention les octets aux adresses basses sont les octets de poids fort. Le mode par défaut au démarrage du programme est le mode *little endian*.

call <nom de macro>

Cette commande appelle une macro en exécutant l’ensemble de ses commandes.

end

Commande de fin de définition de macro.

exit

Termine l’exécution et quitte le programme.

if (<expression>) <commande>

Cette commande teste l’expression, si cette expression a une valeur non nulle, la commande est exécutée. Dans le cas contraire, la commande est ignorée. Dans les deux cas, l’exécution se poursuit sur la ligne suivante.

input <nom de fichier>

Branche la fonction d’entrée sur le fichier qui porte le nom spécifié. Cette fonction permet d’inclure les commandes inscrites sur le fichier.

little_endian

Commute la convention d'ordre des octets pour représenter les entiers en mode *little endian*. Avec cette convention, les octets aux adresses basses sont les octets de poids faible. Le mode par défaut au démarrage est le mode *little endian.*

macro <nom de macro>

Définit une macro avec le nom spécifié. Une macro est définie par une liste de commande qui sont exécutée consécutivement lors de l'invocation de la macro. La définition de la macro s'arrête lors de la rencontre de la commande *end*. L'invocation de la macro se fait avec la commande *call*.

random <alias>

Affecte à l'alias des octets aléatoires.

Exemple

```
* alias alea[%d10]         # déclare un alias de 10 octets
* random alea              # remplit alea avec 10 octets aléatoires
* say $alea                # affichage
 67 c6 69 73 51 ff 4a ec 29 cd
```

reader <n>

Connecte le lecteur numéro *n*. L'expression *n* doit représenter un entier compris entre 0 et le numéro maximal d'un lecteur connecté. Une carte doit être insérée dans le lecteur.

reset

Effectue un reset de la carte sur le lecteur connecté.

say [-d | -x |-s] <expression>

Affiche l'expression à l'écran selon le format spécifié.

- `-d` affiche l'expression comme un entier en numération décimale. Ce mode utilise la convention *little endian* (par défaut) ou *big endian* spécifiée.
- `-x` affiche l'expression comme un entier en numération hexadécimale. Ce mode utilise la convention *little endian* (par défaut) ou *big endian* spécifiée.
- `-s` affiche l'expression comme une chaîne de caractères ascii. Les caractères d'échappement reconnus sont affichés comme tel. Les caractères ascii non affichable, c'est-à-dire non compris entre 32 et 127 sont affichés sous forme octale \nnn.
- En l'absence d'indication de format, l'expression s'affiche comme une liste d'octets en numération hexadécimale, séparés par un espace.

Exemples

```
* alias x[2]          # déclare un alias de 10 octets
* set x 45 23         # affecte à x le lot d'octets 45 23
*                     # la convention par défaut est little endian
* say -s $x           # affiche "E#"
* say -d $x           # écriture décimale 9029
* say -x $x           # écriture hexadécimale 2345
* big_endian          # commute en mode big endian
* say -d $x           # affiche 17699
* say -x $x           # affiche 4523
* set x 8 2
* say -s x            # affiche "\b\002"
```

set <alias> <expression>

Affecte une expression à un alias ou à une partie d'un alias. Les champs qui définissent l'offset et la longueur sont optionnels.

Signale un débordement lorsque la taille de l'expression dépasse celle de l'alias ou de la taille où les données doivent être écrites.

Si la taille de l'expression est inférieure à celle de l'alias ou de la taille d'écriture, des valeurs supplémentaires nulles ont ajoutées aux poids forts selon la convention big endian ou little endian courante.

```
* set buffer[3];1 ab       # affecte ab en position 3 de l'alias buffer
                           # les autres octets sont inchangés
```

Exemples

```
Reader 0             # déclare actif le lecteur numéro 0
alias nom[32]        # déclare un alias de 32 octets
set nom "sarah"      # affecte une chaîne de 5 octets suivie de 0
# envoi d'une commande de personnalisation
# p3 doit contenir la longueur du nom
# et les données entrantes limitées au nombre de caractères
80 01 00 00 length($nom) $nom;length($nom)
80 c0 00 00 00
80 c0 00 00 $sw2
exit
```