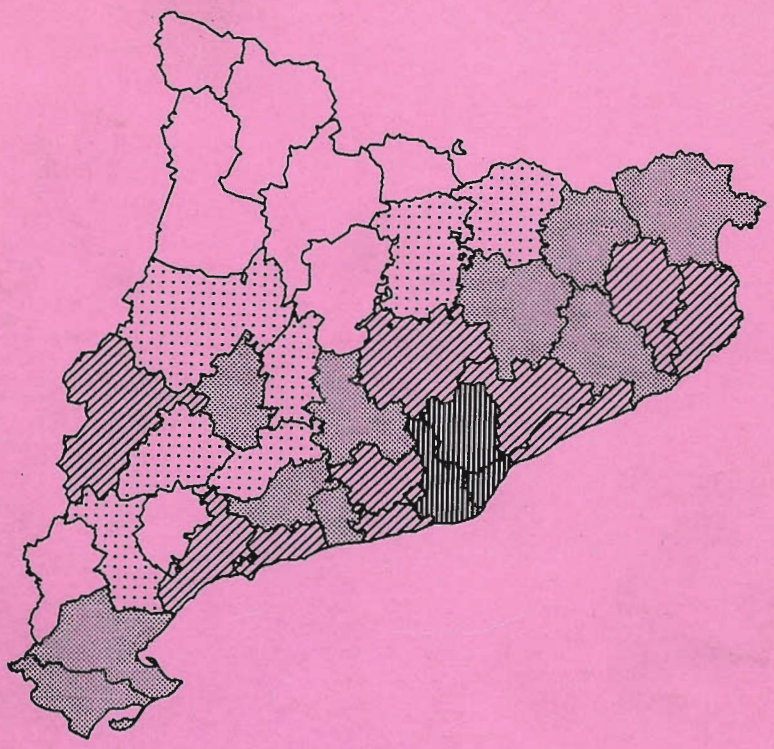
Núm. 6. Març 1987



informàtica UB

Revista d'Informàtica Universitària
Centre d'Informàtica
Universitat de Barcelona

informàtica UB

Revista d'Informàtica Universitària
Número 6. Març 1987

UNIVERSITAT DE BARCELONA
CENTRE D'INFORMÀTICA

ÍNDEX

la coberta 2

el CIUB informa 3

informàtica i universitat 13

CURSOS PER A PROFESSORS D'UNIVERSITAT (MARÇ-JUNY 87)

EARN news and notes 15

Resumen de actividades recientes 15
Reunión de Otoño de EARN-España 16
Datos dobre el uso de la red en la Universidad de Barcelona 17

contribucions 19

COMPUTACCIÓ SIMBÒLICA: REDUCE 19
ESTAD- PC: UN PAQUETE DE ESTADISTICA DE LA UNIVERSIDAD DE BARCEONA PARA MICROORDENADORES IBM-PC O COMPATIBLES 37
COMPUTER NETWORKING FOR SCIENTISTS 40
APLICACIÓ DE LA INFORMÀTICA A LA CARTOGRAFIA 55

món informàtic 59

bibliografia 61

Qué pasa cuando se introduce un comando en la terminal

Josep Maria Blasco Comellas
Centre d'Informàtica

Introducción

Es interesante saber exactamente qué pasa cuando se introduce una instrucción en la terminal. Interesante por dos razones: como información general y porque, al utilizar EXECs escritos en REXX, por defecto se envían las instrucciones como si se escribieran por pantalla. Muchos execs están escritos así y saber el funcionamiento exacto puede ayudar a descubrir errores.

Las dos listas de parámetros de CMS

Cuando se introduce una instrucción para CMS en la terminal, CMS la analiza (para determinar qué programa debe ser ejecutado) y forma dos listas de parámetros separadas:

La ***'tokenized parameter list'***, en la que la instrucción se pasa a mayúsculas y se separa en grupos de 8 caracteres. La separación se realiza atendiendo a los límites entre palabras y a los paréntesis. Para CMS, es una palabra cualquier cosa separada por blancos o paréntesis que no contenga paréntesis ni blancos. Esta lista se pasa al programa apuntada por el registro 1, cada parámetro ocupando 8 bytes y terminada por 8 bytes que contienen 255 hexadecimal (x'FF'). Por ejemplo, la instrucción

```
CopyFile Data File A1 = = c2 ( Rep OldDate
```

se pasaría del siguiente modo:

```
R1 ----> 'COPYFILEDATA FILE A1  =  =  C2'||,
   '(  REP  OLDDATE'
```

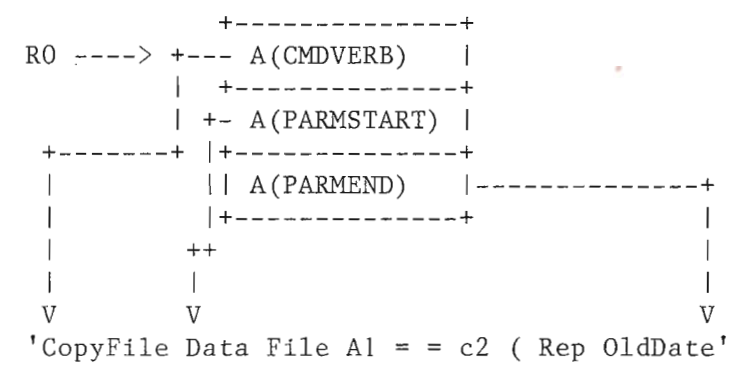
Esto es muy cómodo para escribir programas en assembler (que es como están escritos la mayoría de comandos de CMS), pero tiene inconvenientes graves:

- Todo se pasa a mayúsculas.
- Las palabras mayores de 8 letras se truncan a 8 letras.
- La longitud del comando en su forma interna crece bastante y queda limitada en función del número de paréntesis.

La ***'untokenized parameter list'***, en la que la instrucción se pasa tal cual al programa, sin ningún tipo de transformación. La dirección de tal lista de parámetros se pasa en el registro general 0; mejor, lo que se pasa en el registro 0 es la dirección de un bloque de control o descriptor de la lista de parámetros: para la misma instrucción de antes,

```
CopyFile Data File A1 = = c2 ( Rep OldDate
```

se tendría



este método es absolutamente general y muy superior al anterior, pero sólo existe desde VM/SP release 3. La lista 'tokenized' se mantiene por razones de compatibilidad. Algunos programas utilizan sólo la lista 'tokenized', otros sólo la 'untokenized' y otros las dos. Naturalmente, utilizar la 'untokenized' es más complicado, pero da más flexibilidad.

Cómo determina CMS qué programa se invoca al recibir una instrucción

Cuando se escribe una instrucción en la terminal, CMS realiza una serie de pasos para determinar qué tipo de programa (EXEC o MODULE, por ejemplo) debe invocar. Los pasos que sigue al hacer esa búsqueda dependen de una serie de opciones especificables por el usuario. Describiremos primero esas opciones y las instrucciones de CMS que las controlan para poder exponer después con más facilidad los pasos tomados por CMS (la llamada 'resolución de instrucciones de CMS').

Comandos que alteran la resolución de instrucciones de CMS:

SET IMPCP (ON|OFF) determina si una instrucción que CMS no reconoce debe ser pasada a CP para que intente ejecutarla o no. Si está en ON, se pasan los comandos a CP después de pasarlos a mayúsculas; por defecto está siempre en ON: es la razón de que si se introduce una instrucción desconocida conteste 'UNKNOWN CP/CMS COMMAND'; de estar en OFF contestaría 'UNKNOWN CMS COMMAND'. Si se pone a OFF, una instrucción de CP escrita tal cual da justamente ese último mensaje y no se reconoce; para utilizar una instrucción de CP debe prefijarse tal instrucción con la instrucción de CMS 'CP'. Por ejemplo, CP Q R ALL.
Otro método es entrar en CP con la tecla AP1 y escribir entonces la instrucción de CP. Para volver a CMS debe utilizarse la instrucción de CP 'Begin' (abreviable B) o volver a pulsar PA1.

Desde un programa REXX se puede saber si se está operando en modo SET IMPCP ON o OFF con la función incorporada en REXX/CMS CmsFlag('IMPCP'), que devuelve 1 si IMPCP está en ON y 0 si es OFF.
SET IMPEX (ON|OFF) determina si CMS intentará buscar un EXEC con el nombre de la instrucción tecleada o no. Por defecto está siempre en ON; si se pone a OFF, debe escribirse EXec (abreviable EX) antes de llamar a un EXEC, si no, CMS intenta llamar a un MODULE con el mismo nombre (o una extensión de núcleo, como se verá más abajo). Si está en ON, CMS intenta encontrar primero un EXEC con el mismo nombre que la instrucción; si no lo encuentra, busca un MODULE. Por ejemplo, con SET IMPEX OFF la instrucción SENDFILE deberá utilizarse como EXEC SENDFILE fn ft fm names.
Desde un programa REXX se puede saber si se está operando en modo SET IMPEX ON o OFF con la función incorporada en REXX/CMS CmsFlag('IMPEX'), que devuelve 1 si IMPEX está en ON y 0 si es OFF.

SET ABBREV (ON|OFF) determina si CMS acepta abreviaciones para las instrucciones tecleadas o no, independientemente de como estén SET IMPCP y IMPEX. Naturalmente, esto no afecta a las abreviaciones de las instrucciones de CP. Muchas instrucciones se suelen utilizar de forma abreviada sin saberlo; por ejemplo, COPY es una posible abreviación de COPYFILE y SF es una posible abreviación de SENDFILE. Si ABBREV está en OFF, COPY no se reconoce y debe teclearse COPYFILE; lo mismo sucede con SF y SENDFILE, si está en ON, que es lo que sucede por defecto, se aceptan los sinónimos. Se notará que se puede tener, por ejemplo, ABBREV ON y IMPEX OFF, con lo cual funcionaría COPY pero no SF, ya que SF se traduciría a SENDFILE, pero SENDFILE sería incorrecto debido a IMPEX OFF.
Las abreviaciones se definen mediante la instrucción SYNONYM, que se explica a continuación, y algunas ya están definidas por el sistema. Las abreviaciones estándar del Centre d'Informàtica se pueden encontrar en el fichero CCBSYN SYNONYM P; las estándar del sistema se obtienen mediante QUERY SYNONYM SYSTEM.
Desde un programa REXX se puede saber si se está operando en modo SET ABBREV ON o OFF con la función incorporada en REXX/CMS CmsFlag('ABBREV'), que devuelve 1 si ABBREV está en ON y 0 si es OFF.
SYNONYM filename [SYNONYM [filemode]] define sinónimos adicionales además de los definidos por el sistema. También define truncaciones mínimas o abreviaciones, que vienen controladas por SET ABBREV, explicado anteriormente. Cuando CMS encuentra una instrucción que no es capaz de procesar (teniendo en cuenta el valor de IMPEX), busca en una tabla de sinónimos que asocia a cada instrucción del sistema o otro nombre o el mismo, con posibilidad de abreviaciones. El fichero definido en la instrucción SYNONYM debe contener líneas con el siguiente formato (puede consultarse como ejemplo el fichero CCMSYN SYNONYM P):

instrucción sinónimo [número mínimo de caracteres]
Por ejemplo,

```
SENDFILE ENVIA
```

permite escribir 'ENVIA fn ft fm names' en vez de 'SENDFILE fn ft fm names'.
Un parámetro adicional de longitud permite establecer abreviaciones:

```
SENDFILE ENVIA 2
```

permite usar 'ENVIA', 'ENVI', 'ENV' y 'EN' como sinónimos de 'SENDFILE', siempre que se tenga SET ABBREV ON.
Dado un fichero de sinónimos (p. ej., el CCBSYN mencionado), la instrucción SYNONYM se invoca en la forma

```
SYNONYM CCBSYN SYNONYM *
```

admitiendo la opción '(NOSTD', que desactiva las abreviaciones estándar de CMS de modo que si, por ejemplo, se desea crear un conjunto de sinónimos en la propia lengua que puedan llegar a conflictuar con las abreviaciones estándar de CMS, pueden desactivarse éstas mediante este mecanismo.
Un defecto de la instrucción SYNONYM es que no tiene ninguna opción para que su funcionamiento sea acumulativo, es decir, que al escribir SYNONYM A y luego SYNONYM B se pierden las definiciones contenidas en SYNONYM A. Aunque la opción STACK de QUERY SYNONYM puede ayudar a escribir execs para añadir sinónimos, puede ser muy práctico utilizar el comando EQU/UNEQU, escrito por Barry D. Gates <GATES@MAINE> y ofrecido por los servers CSNEWS@MAINE y SERVER@UOGUELPH, del que se dispone en el Centro:

```
EQU instrucción sinónimo [longitud mínima]
```

añade dinámicamente un sinónimo de usuario al conjunto de los sinónimos activos;

```
UNEQU instrucción
```

quita un sinónimo, sin modificar los demás. El uso de esta instrucción en los perfiles de usuario es altamente recomendable.

Tipos de programas en CMS:

Establecidos los comandos que afectan a la resolución de instrucciones de CMS, pasamos a comentar los distintos tipos de programas que pueden ser invocados utilizando sólo su nombre desde CMS:

MODULEs: son programas en imagen de memoria (es decir, casi iguales, comparado con otros formatos como el utilizado en los ficheros TEXT, al contenido de la memoria principal en el momento en que son cargados para la ejecución). Bastantes instrucciones del sistema y la mayoría de las programadas en lenguajes de programación tradicionales (PASCAL o FORTRAN) por el usuario se ejecutan en forma de MODULEs. Los MODULEs se generan mediante la instrucción GENMOD utilizada inmediatamente después de LOAD: p. ej.,

```
LOAD programa
GENMOD
```

crea un 'programa MODULE A' que puede ejecutarse cada vez que se desee tecleando 'programa'.

Los MODULEs son muy eficientes en cuanto a tiempo de carga (ya que la resolución de subrutinas incorporadas y de soporte del lenguaje utilizado está ya hecha por LOAD) pero ocupan bastante espacio en disco.
Algunos MODULEs escritos en lenguaje ensamblador que no realizan operaciones de entrada y salida están generados mediante la opción ORIGIN de la instrucción LOAD:

```
LOAD programa ( ORIGIN TRANS
GENMOD
```

Tales programas deben ser reusables (es decir, su copia en memoria debe poder ser llamada sin problemas varias veces seguidas) y, de ser utilizados repetidamente, ofrecen, respecto a los normales, la ventaja de que sólo se cargan una vez en un área especial de memoria (la llamada 'transient area') localizada en la posición X'E000' de memoria. Esto permite invocarlos desde otro programa sin mayores problemas.
La mayoría de MODULEs se generan para ser cargados en direcciones fijas de memoria (X'20000' -- 128 Kbytes por defecto; o x'E000' para las transients). Si el módulo es 'relocation-free' (libre de reubicación, o sea, si no contiene direcciones absolutas de su propio código o datos) o bien si ha sido generado mediante la opción RLDSAVE de la instrucción LOAD, como en

```
LOAD programa ( RLDSAVE
GENMOD
```

el programa puede ser cargado como 'extensión de núcleo' mediante la instrucción NUCXLOAD:

```
NUCXLOAD programa [(SYSTEM]
```

Una extensión de núcleo es un programa que reside permanentemente en memoria y, por tanto, es mucho más eficiente, pues no necesita cargarse cada vez que se utiliza. Si se genera con la opción SYSTEM, permanece en memoria hasta el siguiente IPL; si se genera sin ella, hasta el primer HX o terminación anormal. Algunos programas (p. ej. EXECIO, IDENTIFY o WAKEUP) son extensiones de núcleo. Se puede saber qué extensiones de núcleo están activas mediante la instrucción NUCXMAP; y puede usarse NUCXDROP para suprimir una extensión de núcleo. Si se utiliza NUCXMAP con el parámetro ALL se pueden observar también las pseudo-extensiones de núcleo o 'look-aside entries' que definen los puntos de entrada en memoria al núcleo de CMS para la mayoría de comandos estándar de CMS.
CMS intenta optimizar el uso de extensiones de núcleo y 'look-aside entries' del siguiente modo: mantiene constantemente una lista lineal, en cuya cabeza se encuentra el último comando utilizado; si inmediatamente se vuelve a utilizar el mismo comando, se encuentra en seguida, porque está en la cabeza de la lista; si son unos pocos los comandos que se invocan de forma repetida, la ventaja también es evidente.

EXECs: pueden estar escritos en EXEC, EXEC2 o REXX. No tiene sentido escribirlos en EXEC, y EXEC2 proporciona muy pocas cosas que no proporcione REXX, por tanto se escribirán siempre en REXX.
Un programa en REXX puede ser ejecutado de forma algo más eficiente si previamente se somete a algún mecanismo de compresión. Tales mecanismos suelen

consistir en suprimir espacios en blanco y comentarios y, posiblemente, en concatenar líneas. El programa COMPREXX, público en el disco M, realiza estas funciones. La compresión es especialmente interesante al utilizar los execs residentes en memoria, que se explican a continuación. [Nota: si un EXEC hace uso de sus propios comentarios para ofrecer algún tipo de ayuda interactiva NO puede ser comprimido.]
Los 'storage-resident EXECs' (EXECs residentes en memoria) son programas escritos en EXEC2 o REXX que se hallan fijos en memoria (se colocan ahí mediante la instrucción EXECLOAD), a imagen de las extensiones de núcleo; al igual que estas últimas, pueden ser totalmente fijos (atributo SYSTEM) o borrarse con el primer error o HX. El uso de EXECs residentes en memoria puede aumentar mucho la eficiencia en el caso de procesos iterativos que utilicen subrutinas externas en REXX o EXEC2. Dado que un EXEC en memoria se almacena en su forma textual (es decir, no se realiza ningún tipo de compilación previa), es muy importante reducir su tamaño mediante algún mecanismo de compresión.
La instrucción EXECLOAD se utiliza en la forma

```
EXECLOAD fn ft (p. ej. EXECLOAD PROFILE XEDIT)
```

También puede modificarse el filename y/o el filetype al cargar el programa: por ejemplo, la instrucción CIUBSET MSGSERV ON utiliza una instrucción

```
EXECLOAD MSGSERV RXXEDIT * MSGSERV XEDIT
```

para copiar la macro de editor MSGSERV RXXEDIT en memoria como MSGSERV XEDIT. Se notará que la versión en disco con filetype RXXEDIT NO puede ser utilizada directamente (debería tener filetype XEDIT), con lo que se consigue el efecto de opcionalidad deseado.
Para saber qué execs están residentes en memoria, puede utilizarse la instrucción EXECMAP. Para suprimir un exec en memoria, EXECDROP. Para saber cuál es el estado de un exec (esto es, en memoria, en disco o inexistente), EXECSTAT.

El orden de resolución de CMS:

Podemos ya describir el orden de resolución de instrucciones de CMS: cuando se teclea una instrucción, los primeros 8 bytes de la 'tokenized parameter list' se utilizan para determinar el programa a invocar del siguiente modo:
Si se tiene SET IMPEX ON,

- Se busca un exec residente en memoria con ese nombre y se ejecuta si se encuentra.
- Si no se encuentra, se busca en todos los discos accedidos (de la A a la Z) un exec con ese nombre y se ejecuta si se encuentra.
- Si no se encuentra y se tiene SET SYNONYM ON, se substituye por el sinónimo correspondiente y se repiten los pasos anteriores.

Si los tres pasos fallan o se tiene SET IMPEX OFF,

- Si hay una extensión de núcleo o entrada 'look-aside' con ese nombre, se ejecuta.
- Si no, se busca un programa en el área de 'transients' con ese nombre y se ejecuta si se encuentra.

- Si no se encuentra, se busca un fichero MODULE en cualquiera de los discos accedidos (de la A a la Z) y se ejecuta si se encuentra.
- Si todos los pasos anteriores fallan, se busca en la tabla de sinónimos para ver si el nombre es sinónimo de otro nombre. Sólo si se tiene SET ABBREV ON, se consideran abreviaciones. Si se encuentra que el nombre es un sinónimo, posiblemente abreviado, se repite el proceso anterior entero, UNA SOLA VEZ (esto es, no vale que un sinónimo lo sea de otro sinónimo).
- Finalmente, si todos los pasos anteriores fallan y se tiene SET IMPCP ON, se pasa la instrucción original en mayúsculas a CP.
- Si ninguno de estos pasos da resultado, se produce un mensaje de error (UNKNOWN [CP/]CMS COMMAND) y se termina con return code -3.

*****

Instal.lació al CIUB de l'SPAD Versió 1985.

L'Spad és un paquet general de mètodes estadístics descriptius i multidimensionals aplicables a grans taules dades quantitatives, qualitatives o textuals.
A més de les facilitats pròpies d'un paquet de programes (selecció de variables i casos, recodificació de dades, utilització de fitxers de treball auxiliars, ús d'etiquetes per a identificació de les variables, tabulació i graficació dels resultats, etc.), aquest paquet incorpora tot un ventall d'anàlisis exploratòries multidimensionals basades en les tècniques desenvolupades a partir de dels treballs de Benzecri.
L'estructura del paquet és modular i va ser dissenyat establint ETAPES (conjunt d'instruccions que permetran llegir les dades, fer els càlculs d'una anàlisi determinada, etc.). De forma general, els mètodes que presenta el paquet es poden catalogar en els següents grans grups:

Tasques més generals que permet realitzar el paquet

- Entrada de dades i codis: (DONNE, DPLUM, LILEX, ECRIT).
- Gestió i utilitats de l'Spad: (ECRIT, LISTP, CODAJ, ARCHI, TRANS, VALDA).
- Recodificació i Transformacions per les etapes (CODAJ, CREAT, CROIS, FREGA, REDRE, COMRO).
- Selecció de dades (LILEX, ARCHI).
- Estadístics clàssics: Histogrames (TRIHI), Tabulacions (TABUL), Mitjanes, Desviacions tipus i Correlacions (COMPL), Taules de correspondències i Perfils (MULTCT), Gràfiques per a dues coordenades (REPLA), Anàlisi discriminant en dos grups (DIS2G), Regressió i Anàlisi de la variança (MCREG) i Millors Regressions i Discriminants (FUWIL).
- Classificacions. Mitjana per a dades nominals brutes (ECLAT), Mixta per coordenades factorials (SEMIS), Jeràrquica per coordenades factorials (RECIP), Caracterització de classes (TAMIS), Qualitat d'una partició (INERC) i per a Magatzematge d'una Partició (ARCHI)