UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE CIENCIAS FISICO MATEMATICAS

S C L

(Lenguaje de comandos para CYBER NOS/VE)

T E S I S

QUE PARA OBTENER EL TITULO DE
LICENCIADO EN CIENCIAS COMPUTACIONALES

PRESENTA:

GILBERTO CHAPA RODRIGUEZ

SAN NICOLAS DE LOS GARZA, N. L.
NOVIEMBRE DE 1989


1080171544

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE CIENCIAS FISICO MATEMATICAS

S C L

(Lenguaje de comandos para CYBER NOS/VE)

T E S I S

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN CIENCIAS COMPUTACIONALES

PRESENTA:

GILBERTO CHAPA RODRIGUEZ

SAN NICOLAS DE LOS GARZA, N. L.

NOVIEMBRE DE 1989

AGRADECIMIENTOS

A DIOS Nuestro Señor:

Te doy las gracias por haberme dado la oportunidad de vivir en este mundo, y lograr ver realizada una etapa más en esta vida.

Gracias porque nunca me has abandonado y mucho menos me has apartado o privado de ti.

Gracias por haberme cargado más de una vez en tus hombros Señor, cuando el camino era rudo, agotador y asperoso.

Gracias por haberme dado la paciencia suficiente y demostrarme con la vida misma que para todo evento siempre hay un momento.

A Mis Padres:

- Les doy las gracias primero por haberme dado la vida, por haberme cuidado, vestido y alimentado durante todos estos largos años.

Gracias por ayudarme a quitar las molestas piedras del camino, de las cuales fueron objeto ustedes.

Gracias por prepararme para tener una mejor vida con menos sufrimientos penas y carencias que las suyas.

A Mis Hermanas:

Porque gracias a ustedes soy
lo que ahora soy y porque a ustedes
debo mi profesión.

Gracias por haber siempre estado a
mi lado en las buenas y en las
malas, sin nunca haber perdido la
fe o la esperanza en mi.

A Mis Maestros:

Por haber compratido conmigo
la sabiduría que de ellos emana.

Gracias por no haberme dado el
pescado en la boca sino haberme
enseñado a pescar.

A Mi Asesor:
ING. EDUARDO MOLINA CORDOVA

Con mucha admiración y respeto
por ser tan buen maestro y
por su atenta colaboración
para el desarrollo de este
tema de tesis.

A Mis Amigos:

Por compartir tan agradables
y gratos momentos.

Por ayudarme a superar las etapas
difíciles y sobre todo por haber
sembrado una magnífica amistad.

A TODOS MUCHAS GRACIAS

I N D I C E

INTRODUCCION

INTRODUCCION A SCL

ELEMENTOS DE LENGUAJE

EXPRESIONES

PARAMETROS Y VALORES

COMANDOS Y ESTATUTOS

CREACION DE UN PROCEDIMIENTO EN SCL

FUNCIONES

CONCLUSIONES

INTRODUCCION

Hoy en día sabemos muy bien que la comunicación es una parte muy esencial en la vida cotidiana de cualquier ser humano, inclusive hasta en los animales.

Al igual que los seres vivientes necesitan de una comunicación para tener una adecuada armonía, entendimiento, las computadoras también tienen su propio lenguaje de comunicación con el mundo exterior. Este mundo exterior somos nosotros, y nosotros mismos diseñamos a las computadoras y una gran gama de lenguajes de computadoras, cada uno de estos lenguajes aplicados a la estructura, ambiente, diseño y funcionalidad de cada computadora.

Entonces decimos que cada computadora tiene un lenguaje para que el usuario tenga interacción con ella. Probablemente halla un mismo lenguaje para diferentes tipos de máquinas, quizás halla otros lenguajes que sean parcialmente diferentes pero muy similares entre si, o simplemente los lenguajes sean totalmente opuestos.

Sabemos de que cada computadora tiene su propio lenguaje, es decir, la VAX, la IBM, la HP, la CYBER y otras tienen su propio lenguaje con el cual hay una estrecha comunicación entre usuario-máquina.

Este tema de tesis enfocará exclusivamente a uno de estos lenguajes usuario-máquina, sus bases, expresiones, comandos, funciones, procedimientos, etc., serán tratadas aquí.

Este lenguaje se conoce como SYSTEM COMMAND LANGUAGE (SCL) y es aplicado para el nuevo equipo de la CONTROL DATA (CYBER) conocido como equipo de ambiente virtual.

1.- INTRODUCCION A SCL

¿QUE ES SCL? 1-1

USO DE ARCHIVO 1-1

ESTATUTOS Y PROCESAMIENTO DE ESTATUTOS 1-1
Tipos de Estatutos 1-1
- o Comandos 1-2
- o Utilerías de Comandos 1-2
- o Estatutos de Control 1-3
- o Estatutos de Asignación 1-3
- o Ejemplos de Estatutos 1-4

FUNCIONES 1-4

PROCEDIMIENTOS 1-5

ESTRUCTURA DE BLOCK 1-5

PROCESAMIENTO DE CONDICION 1-5

REPRESENTACION DE DATOS 1-6
Constantes 1-6
Variables 1-6

1.- INTRODUCCION A SCL

¿QUE ES SCL?

El Lenguaje de Comandos para Sistema (SCL) es un lenguaje con estructura de block que realiza dos papeles básicos:

- Interpreta todos los estatutos de un Sistema Operativo de Red para Ambiente Virtual (NOS/VE) como llamadas a compiladores, comandos de ejecución de programas, comandos del sistema, comandos de operador y subcomandos de utilerías. Al hacer esto, SCL examina cada estatuto en una serie de comando para la corrección de sintaxis y evaluación de los parametros del estatuto. Si el estatuto contiene algún error de sintaxis o parámetro inválido, o no es reconocido como válido, SCL se encarga de reportarte los errores y no intenta ejecutar dicho estatuto.

- Como lenguaje de programación, SCL proporciona el uso de variables, funciones, expresiones, y la agrupación de estatutos de SCL conocidos como procedimientos.

Aquí se describirá la sintaxis de SCL y el como utilizar SCL como lenguaje de programación.

USO DE ARCHIVO

Un archivo es la máxima unidad de almacenamiento de datos y recuperación en NOS/VE. NOS/VE proporciona un manejo de archivos flexible y una jerarquía de catálogo que nos permite eficientemente organizar los programa texto, códigos de objetos, procedimientos de SCL, y otros datos.

Se puede crear o accesar un archivo en cualquier lugar dentro de un job. También se puede usar archivos para almacenamiento permanente o temporal

ESTATUTOS Y PROCESAMIENTO DE ESTATUTOS

SCL impone una sintaxis estricta para todos los estatutos. Los parametros pasados a un comando o a una función deben ser de una clase predefinida de datos por ejemplificar: entero, booleano, carácter, etc.

Tipos de estatutos

Los estatutos de SCL realizan operaciones en datos y controlan el flujo de una serie de comandos SCL proporciona 4 tipos de estatutos:

- COMANDOS para inicializar una operación específica, como la de crear o borrar un archivo.
- UTILERIAS DE COMANDO (iniciado por comandos seleccionados) - que proporcionan sus propios subcomandos.
- ESTATUTOS DE CONTROL para estructurar una serie de comandos y controlar el flujo de la serie de comando.
- ESTATUTOS DE ASIGNACION para establecer y cambiar variables.

o Comandos

Bajo NOS/VE, un comando inicia una operación específica como - - crear o borrar un archivo. Interactivamente, se captan comandos de SCL después de un slash (/) que es el prompt de NOS/VE.

Un nombre de comando es reconocido por el interprete de SCL si - aparece como entrada en la LISTA DE COMANDO (una lista de comandos que actualmente estan disponibles para ser usados).

Cuando uste se da de alta en NOS/VE, la LISTA DE COMANDO consiste inicialmente de todos los comandos que el sistema proporciona. Usted puede borrar o reemplazar cualquier entrada de la lista de comando (incluyendo comandos del sistema NOS/VE) con sus - propias entradas. Cualquier cambio que se haga a la lista de -- comando se aplicará únicamente al job que actualmente esta en -- proceso. Inclusive se puede controlar el orden en el cual se hará la búsqueda en la lista de comando para el reconocimiento o - no del comando ejecutado.

o Utilerías de Comandos

Las utilerías de comandos son comandos especiales que permiten - hacer disponibles un conjunto de subcomandos. Los subcomandos - realizan las operaciones actuales de la utilería. Cuando se capta el comando para iniciar la utilería, el sistema se encarga de agregar todos los subcomandos asociados con la utilería a la lista de comando.

Por ejemplo, la utilería CREATE_OBJECT_LIBRARY es iniciada y todos sus subcomandos se hacen disponibles para ser usados cuando se capte el siguiente comando:

/create_object_library

NOS/VE despliega COL/ como un prompt para notificar que ya se encuentra dentro de la utilería.

Cuando la utilería se encuentra en uso, se puede captar tanto comandos de SCL como los subcomandos de la utilería. Por ejemplo, cuando proporciona el comando CREATE_OBJECT_LIBRARY en la terminal, también esta habilitado a usar subcomandos de la utilería - CREATE_OBJECT_LIBRARY. Una vez que finaliza la sesión de la utilería (usualmente con el comando QUIT), los subcomandos de la -- utilería dejan de ser habilitados para su uso.

Si usted coloca el comando de CREATE_OBJECT_LIBRARY en un archivo de procedimiento, puede continuar con subcomandos adicionales en el mismo procedimiento, como el siguiente ejemplo:

```
PROC creol_demo
  create_object_library
  add_modules 1=mi_archivo
  generate_library_ 1=mi_librería
  quit
PROCEND creol_demo
```

o Estatutos de Control

Un estatuto de control estructura y controla el flujo de una serie de comando. Con los estatutos de control, se puede estructurar una serie de comandos en blocks de estatutos llamados listas de estatutos. Estas listas de estatutos pueden ser ejecutadas - hasta que ya sea ocurra una condición requerida, una cuenta de - repetición es alcanzada, o una salida específica es ejecutada -- desde el block.

SCL proporciona los siguientes tipos de estatutos de control:

- Estatutos Estructurados usados para estructurar una serie de comandos en grupos de comandos ya sea, procesados como entidades separadas o hasta que una condición o estatuto de terminación es encontrado.

- Estatutos de Control de Flujo usado para controlar el flujo - de un estatuto estructurado.

- Estatutos de Ejecución Condicional usados para controlar la - ejecución de estatutos basados en la evaluación de una expresión booleana asociada.

- Estatutos de Manejo de Condición usados para ejecutar grupos de estatutos cuando ocurre alguna de las condic-ones anormales específicas.

o Estatutos de Asignación

Los estatutos de asignación asignan valores a variables. La forma general de un estatuto de asignación es como sigue:

Variable=Expresión

El valor de la expresión en el lado derecho del signo de igualdad debe corresponder a la clase de variable de la derecha.

Se puede crear variables implícitamente asignándoles un valor en un estatuto de asignación, y también se puede crear o borrar variables explícitamente por un comando de SCL. Si se crea una variable explícitamente, entonces se especifica la clase de la va-

riable. Si se crea una variable implícitamente, la clase de la variable es la misma que la de la expresión a la cual es equivalente.

o Ejemplos de Estatutos

Los siguientes ejemplos contienen estatutos de asignación, un comando, y un estatuto de control.

i=10	Estatuto de asignación
for j=1 to 1 by 1+1 do	Cláusula de inicio del estatuto de control FOR
k=j*j	Estatuto de asignación
display_value k	Comando DISPLAY_VALUE
forend	Cláusula de fin del estatuto de control FOR

En el ejemplo, las siguientes actividades toman lugar:

1. La variable I, es creada (las variables pueden ser referenciadas ya sea en minúsculas o mayúsculas). La variable I es usada como una interación límite para el subsequente estatuto de control FOR.

2. La variable J es creada implícitamente por el estatuto FOR. J es inicializada con el valor de 1 y es incrementada en 2 (1+1) cada vez que el estatuto FOR es ejecutado.

3. La variable K es creada implícitamente y posteriormente se le asigna el valor de J veces J.

4. El comando DISPLAY_VALUE es usado para desplegar el valor de K.

La ejecución de los estatutos anteriores produce estos resultados:

```
 1
 9
25
49
81
```

FUNCIONES

SCL proporciona un conjunto de funciones que realizan operaciones tales como conversión de datos, interrogación del medio ambiente actual, accesar la fecha y hora del sistema. Ambas, las funciones generales de SCL y las funciones usadas para referenciar parame-

tros dentro de un procedimiento serán descritas detalladamente más adelante.

Las utilerías de comandos pueden proporcionar funciones que realizan operaciones específicas para la utilería.

PROCEDIMIENTOS

Un procedimiento es una lista de comandos mandada llamar por el nombre con sus respectivos parametros que son pasados opcionalmente. La sintaxis de un llamado de procedimiento es idéntica a la sintaxis de comandos proporcionados por NOS/VE. Por lo tanto, es posible usar procedimientos ya sea para crear tus propios comandos o para reemplazar los comandos de NOS/VE con su propia versión.

ESTRUCTURA DE BLOCK

La programación con estructura de Block con SCL nos permite organizar una serie de comandos en un block que puede ser procesado como sigue:

- En una manera secuencial, condicional o repetitiva
- En un orden fijo y predeterminado

El flujo de un programa con estructura de block es generalmente fácil de seguir, y por lo tanto el programa resulta fácil de rastrear en casos de error. Se puede controlar el flujo de una serie de comandos incluyendo estatutos de control que realizan interaciones y se ejecutan condicionalmente, dependiendo de las variables proporcionadas o de ciertas condiciones del sistema.

Un job en NOS/VE es organizado típicamente como varios blocks de SCL. Como mínimo, cada job tiene un block llamado job block. El job block contiene todos los estatutos SCL desde el inicio del job hasta su terminación (incluyendo el procesamiento del prólogo y del epílogo iniciado por estos comandos).

Los blocks creados dentro de un block estan subordinados al block inicial. Se crean blocks usando varios estatutos estructurados.

PROCESAMIENTO DE CONDICION

SCL permite especificar la acción que el sistema tomará cuando un error u otra condición inesperada ocurra. Cada comando de SCL contiene un parámetro de STATUS que se puede utilizar para reportar la condición de terminación del comando. Una serie de comandos puede checar este status y tomar cualquier acción apropiada.

Alternativamente, puede usar manejo de condición ya sea cuando una condición específica ocurre para un comando en el cual el parámetro STATUS no es especificado o cuando un error de sintaxis en un

comando es encontrado.

REPRESENTACION DE DATOS

Cada job de NOS/VE consiste de datos y estatutos que realizan algunas operaciones sobre los datos. Además de la información contenida dentro de los archivos, las constantes y las variables pueden -- también representar información en un job de NOS/VE.

Constantes

Una constante específica un valor de dato fijo para el interprete - de SCL. Se pueden usar constantes en cualquier parte dentro de un job. Las constantes pueden ser de alguna de las siguientes clases de datos de SCL: entero, real, carácter o booleanos.

Además, SCL proporciona varias funciones que regresan valores constantes; estas funciones son particularmente útiles para quienes escriben procedimientos. Por ejemplo, la función $MAX_INTEGER regresa el entero positivo mas grande que el sistema maneja.

Variables

Una variable es un nombre que se refiere a un dato elemental cuyo - valor puede estar cambiando durante la ejecución de el job. Se pueden usar las variables en cualquier parte del job pero son generalmente reconocidas únicamente dentro del block en el cual son creados.

Se puede crear una variable ya sea explícitamente con el comando de SCL CREATE_VARIABLE o implícitamente con un estatuto que asigne un valor a una variable previamente sin definir.

Las variables pueden ser de cinco clases de datos en SCL: entera, - real, carácter, booleano, o status. También se puede definir variables de arreglos uni-dimensionales. Se puede referenciar a los elementos de un arreglo individualmente o referenciar a todo el arreglo como una sola entidad.

Una clase de datos especial, llamada de estado, permite a los comandos de SCL reportar la condición de terminación de un comando.

2.- ELEMENTOS DE LENGUAJE

ENTRADA A SCL 2-1
- Propoporcionando Más de un Estatuto en una Línea de Entrada 2-2
- Proporcionando Una Línea de Entrada en Más de una Línea Física 2-2

PROPORCIONANDO COMENTARIOS 2-2

COMO CONTESTA SCL 2-3
- Contestación para Líneas de Entrada 2-3
- Contestación para la Entrada de un Programa 2-3
- Contestación dentro de una Utilería 2-4

USANDO ESPACIOS 2-4

CREANDO NOMBRES EN SCL 2-5
- Caracteres que se pueden usar 2-5
- Usando Letras Mayúsculas y Minúsculas 2-6
- Ejemplos de Nombres Válidos e Inválidos 2-6

PROPORCIONANDO NOMBRES DE JOB QUE EL SISTEMA OFRECE ... 2-6

ESPECIFICACION DE CONSTANTES 2-7
- Constantes Enteras 2-8
  - Especificación de Constantes Enteras Excepto - Hexadecimales 2-8
  - Especificación de Constantes Enteras Hexadecimales 2-8
  - Especificación de Números Base 2-8
  - Especificación de Signo 2-9
  - Usando Espacios y Delimitadores 2-9
  - Ejemplos de Constantes Enteras Válidas e Inválidas 2-9
- constantes Numéricas Reales 2-10
  - Especificación de Constantes Numéricas Reales. 2-10
  - Ejemplos de Constantes Reales Válidas e Inválidas 2-10
- Constantes Tipo String 2-11
- Constantes Tipo Booleano 2-11

CREACION Y REFERENCIA A VARIABLES 2-12
- Variables Enteras 2-13
- Variables Reales 2-13
- Variables Tipo String 2-13
- Variables Tipo Booleano 2-13
- Variables Tipo Estado 2-13
- Referencias a Variables 2-14

ARCHIVOS 2-16

2.- ELEMENTOS DE LENGUAJE

Los elementos del lenguaje SCL son parte de todos los estatutos -- que SCL interpreta, como llamados a compiladores, ejecución de programas, comandos de operador, comandos del sistema, y subcomandos de utilería.

ENTRADA A SCL

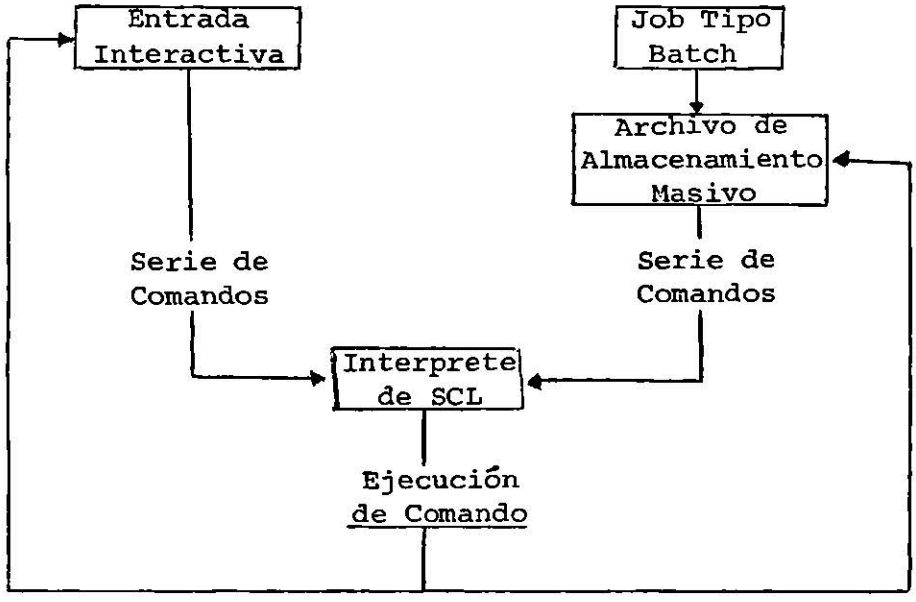
El interprete de SCL procesa todos los estatutos a NOS/VE. El proceso varía, dependiendo de si se envía jobs como archivos de almacenamiento masivo o se proporcionan comandos de SCL a través de -- una terminal interactiva.

- Cuando se envía un job, los estatutos de SCL son primeramente puestos en un archivo de almacenamiento masivo y posteriormente procesados por el interprete de SCL.

- Cuando se encuentra en una terminal interactiva y ahí se -- proporcionan los estatutos, dichos estatutos de SCL son procesados directamente por el interprete SCL.

Los estatutos que el interprete de SCL recibe son llamados la serie de comandos. Una serie de comandos esta compuesta de una lista de líneas de entrada de SCL.

La siguiente figura ilustra el flujo de datos al interprete de - - SCL:



Se puede redireccionar una serie de comandos ya sea usando comandos como INCLUDE_FILE, INCLUDE_LINE, e INCLUDE_COMMAND, o ejecutando un procedimiento en SCL.

Proporcionando más de un Estatuto en una Línea de Entrada:

Se puede captar más de un estatuto de SCL en una línea de entrada. Para separar a los estatutos uno del otro, se usa el punto y coma. Un punto y coma no es requerido como indicador de terminación de un comando al finalizar la línea de entrada. El siguiente ejemplo contiene dos estatutos en una sola línea de entrada:

```
string1='String 1 es una constante caracter.'; display_catalog
```

Proporcionando una Línea de Entrada en más de una Línea Física:

Un estatuto no se encuentra restringido por el ancho de una pantalla de terminal o dispositivo de entrada, ya que se puede especificar una línea de entrada en más de una línea física.

Al finalizar una línea es necesario colocar una elipsis (dos o más puntos consecutivos) para poder expresar el comando en dos líneas físicas. El siguiente ejemplo muestra el uso de la elipsis:

```
string2-' Este es un string muy extenso el cual se encuentra conte-..
nido en dos líneas físicas'
```

El siguiente ejemplo muestra tres estatutos en dos líneas físicas:

```
realizado=true; copy_file input=..
archivo1 output=arvhivo2; display_catalog
```

Para incluir una elipsis como parte actual de la línea (por ejemplo, formando parte del string), se pone inmediatamente de la elipsis un caracter en lugar de un punto o un espacio.

Una línea de entrada no puede exceder 65,535 caracteres (la elipsis usada para la continuación de un comando no es incluida en el límite de 65,535 caracteres). SCL empieza a procesar una línea de entrada con caracteres de continuación únicamente después de que todas las líneas físicas han sido captadas. Luego mezcla las líneas físicas en una sola línea de entrada, interpreta la sintaxis de las líneas, y evalua los parametros y expresiones.

PROPORCIONANDO COMENTARIOS

Para proporcionar un comentario en SCL, delimite el texto con el símbolo de doble comilla como se muestra en el siguiente ejemplo:

```
"Esto es un Comentario."
```

En el texto del comentario, se puede incluir cualquier caracter ASCII excepto la misma doble comilla.

Si se indica un comentario al finalizar una línea de entrada, no es necesario entonces indicar la doble comilla de terminación, como se muestra en el siguiente ejemplo:

"Este comentario es finalizado por el fin de la línea.

Para continuar un comentario en líneas adicionales, se coloca una elipsis (dos o más puntos consecutivos) al final de cada línea a ser continuada, como se muestra en el siguiente ejemplo:

```
"Este comentario es demasiado extenso para alimentarse..
 en una única línea."
```

COMO CONTESTA SCL

Durante la realización de un job interactivo, SCL puede responder que esta listo para captar una instrucción de diferentes maneras.

Contestación para Líneas de Entrada:

En un block de job, SCL responde estar listo para la siguiente línea de entrada como sigue:

Este prompt indica que NOS/VE ha procesado cualquier entrada captada con anterioridad.

Si se continua una línea más alla de una línea física, SCL responde en las siguientes líneas como:

```
/copy-file input-archivos1..
../output=archivo2
```

Dentro de un block de control, SCL responde para cada línea antecediendo al slash (/) con ya sea el nombre del estatuto o su abreviación. En el siguiente ejemplo, SCL intercambia de la forma de responder en un block de job a la forma de responder en un estatuto LOOP:

```
/loop"Inicio de un Estatuto LOOP."
loop/display_value..
loop../value=i
loop/exit
loop/loopend
```

Contestación para la Entrada de un Programa:

Generalmente SCL responde para cuando espera sea captada la entrada a un programa en ejecución con el caracter de interrogación, como se muestra en el siguiente ejemplo:

```
/Igo "Ejecución de un programa que lee de un archivo de entrada."
?
```

Sin embargo, el sistema proporciona la opción de poder cambiar el - caracter de contestación por algún string seleccionado por el usuario, pero esto no será tratado en este libro. Para más información referirse al manual de SCL "Interfase con el Sistema".

Contestación dentro de una Utilería:

Dentro de una Utilería, SCL generalmente responde con el caracter - slash precedido por una abreviación del nombre de la utilería. En el siguiente ejemplo se muestra como responde SCL dentro de la utilería CREATE_OBJECT_LIBRARY:

```
/create_object_library
COL/
```

El comando CREATE_OBJECT_LIBRARY es el que inicia la sesión a dicha utilería generando la nueva contestación de SCL. El subcomando - - QUIT finaliza la sesión a la utilería. Cuando la sesión a esta utilería ha finalizado, SCL despliega el caracter de contestación del - block de un job (/).

```
COL/quit
```

USANDO ESPACIOS

SCL reconoce las siguientes entradas como un simple espacio:

- Uno o más caracteres de espacio en ASCII (SP)
- Uno o más caracteres tab horizontales (HT)
- Un comentario
- Cualquier combinación de las anteriores

En los comando, es recomendable usar el espacio para separar los - elementos de SCL el uno del otro y para mejorar su legibilidad. -- Por ejemplo, en el siguiente comando de LOGIN, los espacios delimitan los parametros del comando y mejoran la legibilidad del comando:

```
login   user = smith     password = jones
```

Desde el momento en que el sistema reconoce los espacios como separadores, no es posible usarlos dentro de nombres. Por ejemplo, -- los siguientes formatos de el comando LOGIN son inválidos.

COMANDO INVALIDO	RAZON
log in user=user_123 password=pass_456	Un espacio aparece dentro del nombre del comando LOGIN.

login user=user _123 password=pass_456	Un espacio aparece dentro del nombre del usuario -- USER_123.
login user=user_123 pass word=pas_456	Un espacio aparece dentro del nombre de parámetro - PASSWORD.

Además, no se puede usar espacios antes o después de los operadores en un valor de comando.

La definición de cada elemento de SCL ofrece más información de como SCL procesa los espacios.

CREANDO NOMBRES EN SCL

En la creación de nombres de SCL, se puede usar de 1 a 31 caracteres, incluyendo caracteres alfabéticos y numéricos, además de ciertos caracteres especiales (listados posteriormente en esta sección). Además, cada nombre proporcionado a SCL debe de iniciar con un caracter alfabético.

Caracteres que se pueden Usar:

En un nombre de SCL, se puede usar cualquiera de los siguientes caracteres ASCII:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

Cuando se forman nombres en SCL, se pueden usar los siguientes caracteres especiales ASCII:

CARACTER	NOMBRE
_	Underline (subrayador)
$	Signo de pesos[1]
#	Signo de número
@	Comercial at
[	Corchete abierto[2]
]	Corchete cerrado[2]
\	Slash invertido[2]
^	Circumflex[2]
`	Acento grave[2]
{	Llave abierta[2]
}	Llave cerrada[2]

¦	Línea vertical[2]
~	Tilde[2]

1. Un nombre de función, nombre de variable, o nombre de archivo definido por NOS/VE contiene un signo de pesos para distinguirlo de los nombres definidos por el usuario. Los nombres para las variables en SCL no deben empezar con el caracter de signo de pesos. En general, se prohibe definir nombres que contengan el caracter de signo de pesos en cualquier posición.

2. Los caracteres especiales ([,], \, ^, `, {, }, ¦, ~) se aplican a lenguajes que requieren de caracteres adicionales. A menos el lenguaje que usted utilice estos caracteres, es recomendable evitar el uso de estos caracteres.

Usando Letras Mayúsculas y Minúsculas:

El sistema convierte las letras minúsculas en nombres con su equivalente a mayúsculas. Como resultado, los siguientes nombres son equivalentes:

NOMBRE_con_MAYUSCULAS_y_minúsculas
nombre_CON_mayúsculas_Y_MINUSCULAS

Ejemplos de Nombres Válidos e Inválidos:

Los siguientes son ejemplos de nombres válidos:

nuevo_archivo_de_datos
#3
Usuario_123
localidad@349_231
número_parte_804284_472383

Los siguientes son ejemplos de nombres inválidos:

NOMBRE INVALIDO	RAZON
123_3456A	Inicia con dígito
abc.345	Contiene un caracter inválido (un punto).
este_nombre_va_ha_ser_demasiado_grande_de_representar	Contiene más de 31 caracteres

PROPORCIONANDO NOMBRES DE JOB QUE EL SISTEMA OFRECE

El sistema ofrece para cada job interactivo y batch un nombre de job único de 19 caracteres en el siguiente formato:

$modelo_serie_contadoralfanumerico_contadorentero

Los componentes de este nombre son los siguientes:

modelo	Número de modelo de la máquina con cuatro dígitos.
serie	Número de serie de la máquina con cuatro dígitos.
contador alfanumérico	Contador alfanumérico de tres caracteres. El contador inicia con AAA.
contador	Contador entero de cuatro caracteres. El contador inicia con 0001.

El siguiente es un nombre típico de job proporcionado por el sistema:

$0860_0452_AAA_4000

Se puede abreviar este nombre cuando se trate de identificar un job al sistema. El sistema asume que la información emitida se aplica a la máquina en la cual el job se encuentra ejecutando. Las siguientes son abreviaciones válidas:

- $contadoralfanumerico_contadorentero
  Por ejemplo, se puede abreviar $0860_0452_AAA_4000 como $AAA_4000

- $contadorentero
  Por ejemplo, se puede abreviar $0860_0452_AAA_4000 como $4000

  Cuando se usa esta abreviación, NOS/VE proporciona el contador alfanumérico:

  - Si el valor del contador entero actual es menor o igual que el contador entero anterior, NOS/VE proporciona el contador alfanumérico actual.

  - Si el valor del contador entero actual es mayor que el contador entero anterior, NOS/VE proporciona el contador alfanumérico anterior.

ESPECIFICACION DE CONSTANTES

SCL reconoce las siguientes formas de constantes:

- Constantes Enteras
- Constantes Reales
- Constantes String
- Constantes Booleanas

Constantes Enteras:

Una constante entera representa un valor entero binario, octal, decimal o hexadecimal (para una lista de valores enteros máximo y mínimo, se tienen las funciones $MAX_INTEGER y $MIN_INTEGER que se verán más adelante).

- Especificación de Constantes Enteras Excepto Hexadecimales

  Para la especificación de cualquier constante entera en una base dada excepto la hexadecimal, se usa la combinación de los caracteres numéricos.

- Especificación de Constantes Enteras Hexadecimales

  Para especificar una constante hexadecimal, se usa la combinación de los caracteres numéricos y las siguientes letras:

  A B C D E F
  a b c d e f

  SCL no hace alguna distinción entre letras mayúsculas y minúsculas en las constantes enteras hexadecimales.

  Se debe iniciar una constante hexadecimal con un dígito. Cuando se especifica una constante hexadecimal que inicia con un caracter alfabético, se debe agregar un cero al inicio. Esto es necesario para prohibir la ambigüedad que surge en ciertas representaciones. Por ejemplo, el 16to elemento en un arreglo cuyo nombre es A1 es indicado como sigue:

  A1(16)

  Este es también la representación hexadecimal del valor decimal 161. Para evitar esta ambigüedad, se debe proporcionar la representación hexadecimal siguiente:

  0A1(16)

- Especificación de Números Base

  Para especificar una constante entera en una base que no sea decimal, se debe incluir una raíz. Si la raíz se omite, entonces se asume que es decimal. Las raíces válidas son las siguientes:

RAIZ	SIGNIFICADO
2	Constante Binaria
8	Constante Octal
10	Constante Decimal. Esta es la raíz por omisión
16	Constante Hexadecimal

Cuando se indica una raíz, debe ir delimitada por paréntesis - - abierto y cerrado, como se muestra en los ejemplos:

1011010(2)
1fff(16)

- Especificación de Signo

Opcionalmente, se puede preceder una constante entera con un signo (los caracteres de mas o menos en ASII). Si se omite el signo se asume que es positivo. Ejemplos:

+100	Número Positivo
-1af 2(16)	Número Negativo
700(8)	Número Positivo

- Usando Espacios y Delimitadores

Las siguientes reglas y principios se aplican para como utilizar espacios y delimitadores cuando se proporcionan constante enteras:

- o Preceder y continuar una constante entera con ya sea un delimitador o un operador gráfico.
- o No especificar un espacio entre los dígitos y la especificación de la raíz.
- o Se pueden colocar espacios después de abrir un paréntesis y antes de cerrarlo en la especificación de la raíz.
- o Prohibido usar espacios entre el signo y dígitos numéricos, especialmente cuando sean constantes reales en expresiones para valores de parametros.

- Ejemplos de Constantes Enteras Válidas e Inválidas

Las siguientes son constantes enteras válidas:

123456789	+704264(8)
ffff(16)	-63(10)
-101(2)	10101010101(2 .)

Las siguientes son constantes enteras inválidas:

ENTERO INVALIDO	RAZON
100200(2)	Contiene un dígito que es inválido en una constante binaria.
fff(16)	No inicia con un dígito, aunque es una constante hexadecimal.
345(17)	Contiene una raíz inválida (17).
fff	No contiene una raíz indicando que se trata de una constante hexadecimal.

123 (8)	Contiene un espacio entre los dígitos y la raíz.

Constantes Numéricas Reales:

Una constante numérica real consiste de los siguientes elementos:

- o Mantissa, la cual es una secuencia de dígitos conteniendo un punto decimal (punto).
- o Exponente (opcional) con signo opcional.
  Actualmente, únicamente constantes reales no-signadas estan habilitadas.

- Especificación de Constantes Numéricas Reales

  Se específica una constante numérica real de la siguiente forma:

  - o Incluir al menos un dígito a cada lado del punto decimal.
  - o Usar únicamente números decimales.
  - o Separar la mantissa del exponente con ya sea la letra E o D (en minúscula o mayúscula).
  - o Delimitar la constante real en ambos extremos.
  - o No poner un espacio en la mantissa, en el exponente, o entre la mantissa y el exponente.
  - o Prohibido usar espacios entre el signo y los dígitos numéricos, especialmente cuando sea una constante real en expresiones para los valores de los parametros.
  - o Crear números reales en un rango de 4.8E-1234 a 3.2E+1232

  El valor del número real es la mantissa multiplicada por 10 elevado a la potencia del exponente. SCL mantiene números reales en formato normalizado de doble-precisión y punto-flotante.

  Se pueden usar números reales como variables creados implícitamente, constantes y parametros. Las operaciones aritméticas -- con números reales no estan habilitadas, ni se puede indicar un número real cuando una variable sea creada explícitamente.

- Ejemplos de Constantes Reales Válidas e Inválidas

  Las siguientes son constantes reales válidas:

  12.26E2 34.0e5
  3.0d-3 0.003

  Las siguientes son constantes reales inválidas:

CONSTANTE REAL INVALIDA	RAZON
.33e-5	No tiene al menos un dígito a cada lado del punto decimal.
5.6(8)	Incluye un número base.
-3.2 d-4	Contiene un espacio en medio y esta precedido por un signo.

Constantes Tipo String:

Una constante tipo string es cualquier secuencia de caracteres ASCII delimitada por apóstrofes (la máxima longitud de un string es de 256). El siguiente es un ejemplo de una constante tipo string:

```
'Este es un String'
```

Se puede indicar un string conteniendo únicamente un espacio, como sigue:

```
' '
```

Para denotar un string vacio, se indican dos apóstrofes consecutivos, como en el siguiente ejemplo:

```
''
```

En un string los apóstrofes sirven como delimitadores y no forman parte del mismo string. Para agregar un apóstrofe a un string, se representa con datos apóstrofes consecutivos, como se muestra:

```
'You can''t use a single apostrophe within a string.'
```

Las siguientes son constantes tipo string válidas:

STRING VALIDO	EXPLICACION
'A. B. Smith'	String Ordinario
''''	String de un apóstrofe
'Este string.. muy grande.. es continuo'	String con continuación

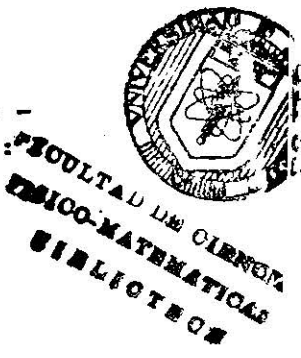
Los siguientes son strings inválidos:

STRING INVALIDO	RAZON
'Este string no termina.	No se indica el apóstrofe final.
'You can't have a single.. apostrophe.'	Dentro de un string, un apóstrofe debe ser representado por dos apóstrofes consecutivos.
'Este string demasiado grande no esta bien continuado.'	La elipsis (..) no se indicó al finalizar las primeras dos líneas.

Constantes Tipo Booleano:

Una constante booleana un valor ya sea TRUE (cierto) o FALSE (falso). Se puede representar una constante booleana con valor TRUE con cualquiera de los siguientes nombres (en mayúsculas o minúsculas):

```
TRUE
YES
ON
```



Se puede representar una constante booleana con valor FALSE con - - cualquiera de los siguientes nombres (en mayúsculas o minúsculas):

FALSE
NO
OFF

Las siguientes son asignaciones booleanas correctas:

```
realizado=false
modo_debug=ON
impresion_de_listado=no
```

Las siguientes son asignaciones booleanas inválidas:

ASIGNACION BOOLEANA INVALIDA	RAZON
modo_debug= correcto	Especificación impropia para el - valor TRUE.
realizado=.f.	Especificación impropia para el - valor FALSE.

CREACION Y REFERENCIA A VARIABLES

Con SCL se puede crear, borrar, referenciar, y cambiar valores de - variables que contienen datos. La creación de variables es un proceso ya sea explícito o implícito.

SCL reconoce las siguientes clases de variables:

Entera
Real
String
Booleana
Status

Cada variable es referenciada por un nombre de variable que es únidentro de un block. Un nombre de variable es cualquier nombre válido en SCL (excluyendo los nombres TRUE, YES, ON, FALSE, NO, OFF y - los nombres que inician con signo de pesos). También se puede - - crear una variable consistiendo de un arreglo de valores.

Para referencia un elemento de una variable tipo arreglo, especificando el subindice con el siguiente formato:

```
nombre_de_variable(subindice)
```

El subindice puede ser cualquier expresión entera.

Los siguientes son ejemplos de variables:

```
última
arreglo(8)
dato_elemental(4)
siguiente_localidad(última+1)
```

Variables Enteras:

Se pueden crear variables enteras dentro del rango de $MIN_INTEGER al $MAX_INTEGER como sigue:

- o $MIN_INTEGER representa el valor entero mínimo
- o $MAX_INTEGER representa el valor entero positivo máximo

Variables Reales:

Se pueden crear variables numéricas reales dentro del rango de - - 4.8E-1234 a 3.2E+1232.

Se pueden usar números reales para crear variables implícitamente. Las operaciones aritméticas reales no estan aún habilitadas, ni -- tampoco se puede especificar un número real cuando se crea una variable explícitamente.

Variables Tipo String:

Se pueden crear variables tipo string con una longitud máxima de - 256 caracteres. Si una variable tipo string es creada implícita-- mente, o no se le indica la longitud del string cuando la variable es creada implícitamente, el sistema asigna una longitud máxima -- por omisión de 256 caracteres.

En cualquier momento, una variable string tiene una longitud ac- - tual igual a la longitud del string anteriormente asignado a la variable. Si a un string se le asigna un valor que es más grande -- que su longitud máxima, la longitud actual del string es su longitud máxima. Los caracteres excedidos son truncados.

Variables Tipo Booleano:

Se pueden crear variables tipo booleano con los valores TRUE o FALSE, como se describió anteriormente en las constantes tipo booleano.

Variables Tipo Estado:

Una variable tipo estado es un registro que contiene el estado completo del comando. Cada registro de una variable tipo estado, consiste de tres campos. Estos campos se describen en la tabla 2-1.

Tabla 2-1. Camos de una Variable Tipo Estado

NOMBRE DEL CAMPO	CLASE DE CAMPO	DESCRIPCION
Normal	Booleano	Indica si el comando terminó sin errores (TRUE) o con errores (FALSE).
Condition	Entero	Si el comando tuvo error, este entero indica el código de condición del mensaje de diagnóstico para un comando abortado. Este campo también contiene una identificación producto de 2 caracteres.

Text	String de Longitud 256	Este string contiene los parametros del mensaje que son sustituidos en el mensaje de diagnóstico básico correspondiente a la condición encontrada.

Si el comando termina normalmente, el campo NORMAL es puesto en TRUE y todos los demás campos son indefinidos. Si el comando no pudo ser completado, el campo NORMAL es puesto en FALSE; los otros campos regresan condiciones y los parametros de diagnóstico del mensaje describiendo el error que ocurrió.

Las referencias a una variable tipo estado, puede incluir un nombre de campo. Para referenciar un campo específico de una variable tipo estado, se usa el siguiente formato:

```
nombre_de_variable.nombre_de_campo
```

El siguiente ejemplo referencia el campo de condición de la variable tipo estado con nombre ESTADO_LOCAL:

```
estado_local.condition
```

También se puede referenciar un arreglo de variables tipo estado. El siguiente ejemplo referencia el campo TEXT del quinto elemento de un arreglo de variables tipo estado con nombre ESTADO_JOB:

```
estado_job(5).text
```

Si una variable es un arreglo de valores tipo estado, entonces se debe incluir un subindice cuando se especifique un nombre de campo. Por ejemplo, si un arreglo de variables de estado llamado ESTADO_ACTUAL contiene 10 elementos, se debe indicar la siguiente información para referenciar el campo NORMAL del tercer elemento de la variable ESTADO_ACTUAL:

```
fallo=estado_actual(3).normal
```

Referencias a Variables:

Para referenciar a una variable depende de como esta fue creada. Se puede crear variables en ambas formas explícitamente (con el comando CREATE_VARIABLE) e implícitamente.

Las variables creadas con el comando CREATE_VARIABLE puede contener subindices y nombres de campos de variables tipo estado. Los espacios no estan permitidos entre el campo y los componentes del subindice de una referencia de variable.

Las siguientes son referencias correctas a variables creadas explícitamente:

REFERENCIA A VARIABLE	DESCRIPCION
lista_de_datos_1(10)	Referencia al décimo elemento de la variable tipo arreglo LISTA_DE_DATOS_1

estado.condition	Referencia al campo CONDITION de la variable tipo estato llamado ESTADO.
estado_arreglo(4).normal	Referencia al campo NORMAL de la variable tipo estado llamada ESTADO_ARREGLO.

Las siguientes son referencias incorrectamente a variables.

REFERENCIA INVALIDA	RAZON
estado.siguiente	Especificación inválida del campo (SIGUIENTE) para la variable de estado.
lista (4)	Existe un espacio entre el nombre de la variable y el subindice, lo cual no es permitido.
estado . normal	Los espacios que rodean al punto no estan permitidos en la especificación del campo.

Cuando una variable es creada implícitamente (en un estatuto de asignación o en un estatuto FOR), se aplican los siguientes convenios.

- Las variables creadas con un estatuto de asignación no pueden contener un subindice o un nombre de campo para variables tipo estado. Como resultado, se pueden crear variables tipo estado implícitamente únicamente cuando se le asigne un valor de otra variable tipo estado (o con la función $STATUS descrita más adelante). En este caso, los campos de ambas variables tipo estado contienen los mismos valores.
- Si se omite el subindice de una referencia a una variable arreglo, la referencia es para la variable en su totalidad.
- Si se omite el nombre de campo de una referencia a una variable tipo estado, la referencia es al valor completo del estado (todos los campos).

Por ejemplo, si una variable tipo estado con el nombre de ESTADO_JOB existe actualmente, el siguiente estatuto crea una variable tipo estado con el nombre NUEVO_ESTADO_JOB que es idéntico en valor a la variable JOB_STATUS:

```
nuevo_estado_job=estado_job
```

Como variables creadas implícitamente, las siguientes son referencias inválidas a variables:

```
lista_nueva(10)=lista vieja
Estado.normal=estado_normal
```

El siguiente ejemplo crea una variable tipo arreglo llamada LISTA - con 10 elementos, y posteriormente crea una variable llamada NUEVA_ LISTA, la cual tiene la misma clase y los mismos valores que la variable LISTA:

```
create_variable name=lista dimensión=10 kind=string
nueva_lista=lista
```

Como LISTA(1) es una variable string, LISTA_NUEVA(1) es también una variable string.

El siguiente ejemplo crea una variable tipo estado llamada NUEVO_ESTADO, y posteriormente crea una variable tipo estado llamada VIEJO_ ESTADO, el cual contiene todos los campos de NUEVO_ESTADO:

```
create_variable name=nuevo_estado kind=status
viejo_estado=nuevo estado
```

El siguiente estatuto crea una variable booleana llamada FALLA con el valor contenido en el campo NORMAL de la variable tipo estado -- VIEJO_ESTADO

```
falla=viejo_estado.normal
```

ARCHIVOS

Un elemento básico en los estatutos de SCL es el archivo, el cual - almacena programas o datos temporal o permanentemente. Como en los nombres de SCL, los nombres de archivo que se crean consisten de un máximo de 31 caracteres alfabéticos, numéricos, y caracteres espe-- ciales e inician con un caracter alfabético. Se hace referencia a un archivo ya sea indicando su nombre o indicando uno o mas elementos en la ruta de acceso del archivo (path).

3.- EXPRESIONES

EXPRESIONES ENTERAS 3-1

EXPRESIONES DE TIPO STRING 3-2

EXPRESIONES LOGICAS 3-3

EXPRESIONES RELACIONALES 3-4

COMBINACION DE EXPRESIONES 3-5

DESPLEGAR EL VALOR DE UNA EXPRESION 3-7

3.- EXPRESIONES

Una expresión es una representación de una o más operaciones realizadas para calcular un valor. Una constante o variable apareciendo a solas también constituye una expresión. Este capítulo se enfoca a expresiones en las cuales las constantes y variables estan combinadas por operadores.

NOTA:

Operaciones Aritméticas con números reales no estan habilitadas.

Las expresiones son usadas para realizar los siguientes tipos de -- cálculos:

- Suma Entera
- Resta Entera
- Multiplicación Entera
- División Entera
- Exponenciación Entera
- Concatenación de Strings
- Diferencia Lógica y Suma
- Complemento Lógico
- Producto Lógico
- Comparaciones Relacionales

En una expresión, el sistema evalua primero los operandos dentro de paréntesis, iniciando con los paréntesis mas anidados en la expre-- sión.

Cuando el sistema encuentra una constante definida en SCL o una fun ción en una expresión, evalua la constante o función (y cualquier - argumento indicado) y usa el valor resultante en la expresión.

Una expresión puede contener cualquier número de datos elementales, pero después de ser evaluada, representa un solo valor.

En las expresiones usadas en estatutos de asignación, puede haber - espacios en los extremos del operador; en una expresión usada en un parámetro, no se puede indicar espacios en los extremos de los operadores.

EXPRESIONES ENTERAS

Una expresión entera es aquella que se evalua como un valor entero. En las expresiones enteras, se pueden indicar los siguientes operadores:

OPERADOR	USO
+	Indica que dos operandos van ha ser suma-- dos.
-	Indica que al operador de la izquierda se le restará el operando derecho.

*	Indica que dos operandos enteros van ha -- ser multiplicados.
/	Indica que el operando izquierdo será divi dido por el operando derecho. Aquí se usa la división entera estandar: el resultado es el cociente entero más grande con cualquier residuo descartado.
**	Indica que el entero a la izquierda del -- operador será exponenciado por el entero - que esta a la derecha del operador.

A excepción de cuando se use una expresión entera como valor de un parámetro, se puede anteponer y/o continuar con espacios al opera-- dor.

Los siguientes son ejemplos de estatutos de asignación (cada uno -- afectando al siguiente) usando expresiones enteras.

ESTATUTO	SIGNIFICADO
i=1	Asigna la constante 1 a la variable I.
j=i+2	Asigna la suma de I+2 a la variable J. Por lo tanto, J es igual a 3.
k=2**j	Asigna el resultado de elevar 2 a la J-ésima potencia a la variable K. Por - lo tanto, K es igual a 8.
k=k/3	Asigna el cociente de la variable K di vidida por 3 a la variable K. Por lo tanto, K es igual a 2.

EXPRESIONES TIPO STRING

Una expresión tipo string es aquella que se evalua como un string. En las expresiones tipo string, se puede usar el siguiente operador de concatenación:

//

Este operador une los dos operandos string para formar un solo - - string. Excepto cuando se use una expresión como valor de un parámetro, se puede anteponer y/o continuar con espacios al operador.

En el siguiente ejemplo se incluye al operador de concatenación en estatutos de asignación.

ESTATUTO	SIGNIFICADO
string_1='Primera parte'	Asigna la constante string 'Primera - parte' a la variable tipo string - - STRING_1.

string_2' Ultima parte'	Asigna la constante string ' Ultima - parte' a la variable tipo string - - STRING_2.
s=string_1//string_2	Concatena los valores de las varia- - bles tipo string STRING_1 y STRING_2 y coloca el resultado en la variable tipo string S. El valor de S es si-- guiente: 'Primera parte Ultima parte'

EXPRESIONES LOGICAS

Una expresión lógica es aquella que es evaluada como falsa o verdadera. En una expresión lógica, todos los operandos deben ser valores booleanos. El resultado de una expresión lógica es siempre un valor booleano.

En expresiones lógicas, se pueden usar los siguientes operadores:

OPERADOR	USO
OR	Realiza una suma lógica (OR inclusive) de - dos operandos. Si un operando es TRUE, la expresión es evaluada como TRUE; de otro modo, es evaluada como FALSE. Si el operando de la izquierda es TRUE, el operando dere-- cho no es evaluado.
XOR	Realiza una diferencia lógica (OR exclusi-- ve) entre dos operandos. Si un operando es TRUE, pero no ambos, la expresión es evaluada como TRUE; de otro modo, es evaluada como FALSE.
AND	Realiza un producto lógico de dos operan- - dos. Si ambos operando son TRUE, la expresión es evaluada como TRUE; de otro modo, - es evaluado como FALSE. Si el operando de la izquierda es FALSE, el operando derecho no es evaluado.
NOT	Realiza un complemento lógico de un operando. Si el operando es TRUE, la expresión - es evaluada como FALSE. Si el operando es FALSE, la expresión es evaluada como TRUE.

Se debe de preceder y continuar con uno o mas espacios a un operador lógico. La siguiente es una expresión lógica válida:

```
(string='a') or (string='b')
```

La siguiente es una expresión lógica inválida:

```
(string='a')or(string='b')
```

Los siguientes estatutos de asignación son ejemplos del uso de ex-- presiones lógicas:

ESTATUTO	SIGNIFICADO
y=true;z=false	Creación de variables booleanas Y y Z.
x= y OR z	Si Y ó Z es TRUE, X es True. Si Y es - TRUE, X es TRUE.
x= NOT y	X es el complemento de Y. Si Y es - - TRUE, entonces X es FALSE.
x= y AND z	X es TRUE solo si Y y Z son TRUE. Si Z es FALSE, X es FALSE.
x= y OR z	X es TRUE porque Y es TRUE y Z es FAL-- SE (un operando es TRUE, pero no ambos).

EXPRESIONES RELACIONALES

Una expresión relacional es aquella que expresa una relación entre valores de una misma clase (por ejemplo, dos operandos tipos string ó dos operandos enteros). El resultado de una expresión relacio- - nal es siempre un valor booleano.

En las expresiones relacionales, se pueden usar los siguientes operadores:

OPERADOR	RESULTADO
>	Cuando el operando de la izquierda de este operador es mayor que el operando de - la derecha, la expresión es evaluada como TRUE; de otro modo, es evaluada como FALSE.
<	Cuando el operando de la izquierda de este operador es menor que el operando de - la derecha, la expresión es evaluada como TRUE; de otro modo, es evaluada como FALSE.
>=	Cuando el operando de la izquierda de este operador es mayor o igual al operando de la derecha, la expresión es evaluada - como TRUE; de otro modo, es evaluada FALSE.
<=	Cuando el operando de la izquierda de este operador es menor o igual al operando de la derecha, la expresión es evaluada - como TRUE; de otro modo, es evaluada como FALSE.

=	Cuando el operando de la izquierda de este operador es igual al operando de la derecha, la expresión es evaluada como TRUE; de otro modo; es evaluada como FALSE.
<>	Cuando el operando de la izquierda de este operador es diferente al operando de la derecha, la expresión es evaluada como TRUE; de otro modo, es evaluada como FALSE.

A excepción de cuando se use una expresión relacional como valor de un parámetro, se puede anteponer o continuar con espacio al operador relacional.

Las siguientes reglas adicionales se aplican a expresiones relacionales:

- Un valor FALSE es evaluado como menor que un valor TRUE.
- Las comparaciones se hacen de acuerdo a la secuencia ordenada de ASCII. Todos los caracteres ASCII son ordenados de acuerdo a sus códigos ASCII. Por ejemplo, el caracter # tiene código 23 (hexadecimal) en ASCHII el cual es mayor en la secuencia de orden que el caracter $, el cual tiene código 24 (hexadecimal) en ASCII. Las mayúsculas y minúsculas no son equivalentes en una secuencia ordenada ASCII, y por lo tanto, no son iguales - en comparaciones de strings.
- Cuando el sistema compara dos strings con longitud diferente, considera al string pequeño como si tuviera espacios en la parte derecha donde finaliza el string pequeño, completandolo hasta la longitud del string grande.

Las siguientes son series de estatutos que ilustran como utilizar -- las expresiones relacionales:

ESTATUTO	SIGNIFICADO
i=5 j=1 s='abc' t='abcde' y=true z=false	Inicializa las variables enteras I y J, las variables tipo string S y T, y las variables booleanas Y y Z.
i>j	I(5) es mayor que J(1). Por lo tanto, la expresión es evaluada como TRUE.
i<j	I(5) no es menor que J(1). Por lo tanto, la expresión es evaluada como FALSE.
s=t	Antes de la comparación, el string S - es extendido a la derecha con espacios de tal manera que su longitud iguale - la longitud del string T. S('abc' seguida de un espacio adicional) no es -

	igual a T('abcde'). Por lo tanto, la expresión es evaluada como FALSE.
y <> z	Y (TRUE) es diferente a Z (FALSE). - Por lo tanto, la expresión es evaluada como TRUE.
i >=j	I(5) es mayor que J(1). Por lo tan-- to, la expresión es evaluada como - - TRUE.
s <=t	Los primeros 3 caracteres de ambos -- S('abc ') y T('abcde') son iguales -- (abc). Sin embargo, para S, el cuar- caracter es un espacio, donde para T el cuarto caracter es d. El código - ASCII para un espacio es 20 (hexadeci mal), donde el código ASCII para d es 64 (hexadecimal). Por lo tanto, S es menor que T, y la expresión es evalua da como TRUE.

COMBINACION DE EXPRESIONES

Se puede combinar únicamente ciertas clases de operandos con un ope rador particular. Las combinaciones válidas y las clases de resul- tados se muestran en la Tabla 3-1.

Cuando no se especifican paréntesis en las operaciones, el sistema realiza las operaciones en el orden mostrado en la tabla. Por - - ejemplo, ** es evaluado antes que * en la siguiente expresión:

```
a*b**c
```

Esta expresión es idéntica a la siguiente expresión:

```
a*(b**c)
```

Cuando el orden de precedencia es equivalente, la evaluación se -- realiza de izquierda a derecha. Por ejemplo, el resultado de la - siguiente expresión es 4:

```
8/4*2
```

Si se invierte el orden de los elementos de la expresión anterior, el resultado sería 1.

```
2*4/8
```

Tabla 3-1. Operandos y Combinaciones de Operadores

OPERANDO IZQUIERDO	OPERADOR	OPERANDO DERECHO	RESULTADO	ORDEN DE PRECEDENCIA
Entero	**	Entero	Entero	1
Entero	* ó /	Entero	Entero	2
Entero	+ ó -	Entero	Entero	3
Ninguno	+ ó -	Entero	Entero	3
String	//	String	String	4
Entero	Relacional	Entero	Booleano	5
String	Relacional	String	Booleano	5
Booleano	Relacional	Booleano	Booleano	5
Ninguno	NOT	Booleano	Booleano	6
Booleano	AND	Booleano	Booleano	7
Booleano	OR ó XOR	Booleano	Booleano	8

1. Cuando el orden de precedencia es equivalente, los operadores, son evaluados de izquierda a derecha.

DESPLEGAR EL VALOR DE UNA EXPRESION

Para desplegar el valor de una expresión, se utiliza el comando - - DISPLAY_VALUE.

Formato

```
DISPLAY_VALUE o
DISPLAY_VALUES o
DISV
   VALUE = lista de expresión
   OUTPUT = archivo
   STATUS = variable de estado
```

Parametros

VALUE
Una o más expresiones. Cuando se indica más de un valor en la lista, el sistema despliega cada - valor en una línea separada. Este parámetro es requerido.

OUTPUT
Archivo en el cual la información será escrita.

STATUS
Condición de Terminación del comando

Comentarios

- Cuando el sistema regresa un entero con la -- raíz, la raíz concuerda con la raíz del pri-- mer operando de la expresión.
- Cuando el archivo indicado en el parámetro -- OUTPUT no esta concectado a la terminal (como en un job batch), ponga cuidado especial al - utilizar el comando DISPLAY_VALUE. Cuando se crea un archivo usando el parámetro OUTPUT de

este comando, y el archivo es asignado a almacenamiento masivo, el atributo de archivo - PAGE-FORMAT (formato de página) es indicado - como BURSTABLE. Cuando el archivo es asignado a la terminal, el archivo es creado con el atributo de PAGE_FOR_MAT igual a CONTINUOUS. Así, para archivos de almacenamiento masivo, la ejecución de DISPLAY_VALUE puede resultar en una impresora en línea como salto de página y título de página.

Para evitar saltos de página y títulos de página, colocar el atributo PAGE_FORMAT a - - CONTINUOUS y el atributo FILE_CONTENTS a LEGIBLE. Hacer esto, incluyendo el siguiente comando antes de usar el comando DISPLAY_VALUE en un archivo de almacenamiento masivo (asumiendo un nombre de archivo ARCHIVO_LISTA para el parámetro OUTPUT):

```
/set_file_attribute archivo_lista..
../page_format=continuous..
../file_contents=legible
```

Ejemplos

- Ejemplos del uso del comando DISPLAY_VALUE -- aparecen en toda esta tesis. Los siguientes ejemplos demuestran el uso del comando con varias clases de valores:

```
/display_value 2**7-4
124

/create_variable name=estado  kind=status..
../value=$status(false,'US',2,'Contenido de la..
../Variable de Estado')
/display_value estado
 --ERROR-- CC=US 2 TEXT=? Contenido de la Variable Estado.

/display_value ('Linea 1', 'Linea 2', 'Linea 3')
Linea 1
Linea 2
Linea 3

/display_value  $variable(estado, kind)
STATUS

/display_value ($max_integer,$min_integer)
281474976710655
-281474976710655
```

- Además se puede usar el comando DISPLAY_VALUE para determinar el equivalente en hexadecimal, octal, o decimal. En los siguientes tres ejemplos, el primero despliega el equivalente deci-

mal de un número hexadecimal, el segundo despliega el equivalente hexadecimal de un número decimal, y el tercero despliega el equivalente octal de un número decimal.

```
/display_value  1*  FFFFFF(16)
16777215

/display_value  1(16)*16777215
  FFFFFF(16)

/display_value  1(8)*16777215
77777777(8)
```

Como una alternativa para la conversión de enteros, ver la descripción de la función $STRREP en el capítulo 7.

4.- PARAMETROS Y VALORES

ESPECIFICACION DE PARAMETROS 4-1
Los Parametros son Dependientes de un Orden..... 4-1
Abreviación de Parametros 4-2
Indicando Lista de Valores 4-2
- Indicando un Valor Simple 4-3
- Indicando un Valor como un Rango 4-3
- Indicando Conjuntos de Valores 4-4
- Representación de un valor como Expresión 4-5
Espacios en Valores de Parametros 4-5

CLASES DE VALORES 4-6

4.- PARAMETROS Y VALORES

Este capítulo describe como especificar los parametros de los comandos y los valores que les corresponden.

Cada comando de NOS/VE puede tener una lista de parametros. Esta lista de parametros es una serie de parametros separados por comas o espacios. Para cada parametro, el comando define una clase de valor. Cuando se indica un parametro, debe corresponder con la clase de valor definido. Posteriormente se verán la definición y listas de clases de valores de SCL en la sección de clases de Valores.

ESPECIFICACION DE PARAMETROS

Se puede indicar un parámetro de un comando ya sea como un nombre de parámetro igualado a una lista de valores (llamada lista de valores) o solamente con un valor de la lista.

```
nombre_de_parámetro = lista_de_parametros
                      ó
valor_de_la_lista
```

El siguiente ejemplo llama al comando CREATE_VARIABLE para crear una variable. Dos nombres de parametros son igualados a listas de valor.

```
/create_variable name = ciclos  kind=integer
```

Se puede indicar el mismo comando en incluir únicamente las listas de valor como sigue:

```
/create_variable  ciclos  integer
```

Los siguientes comandos realizan la operación idénticamente:

```
/create_variable  name = ciclos integer
/create_variable  ciclos kind = integer
```

Los nombres de parametros siguen las mismas reglas como los otros nombres de SCL.

Los Parametros son Dependientes de un Orden:

El orden en el cual se indican los parametros depende en si se indica también los nombres de los parametros.

- Cuando se indican los nombres de los parametros, se pueden listar los parametros en cualquier orden, como se muestra en los siguientes ejemplos:

```
/create_variable  kind = integer  name = ciclos
/create_variable  name = ciclos  kind = integer
```

Cada descripción de comando define el orden de los parametros. - Cuando se indica un nombre de parámetro, el sistema ubica al parámetro en la posición definida por el.

o Cuando se omiten los nombres de parametros, se debe indicar los parametros posicionalmente. Esto es, se deben de indicar en el orden mostrado en el formato de la descripción del comando. En el siguiente ejemplo, el segundo parámetro esta listado sin un - nombre. Debido a que el comando define su aparición en segunda posición de la lista de parametros, por lo que debe de indicarse en esa posición.

```
/create_variable  name = ciclos integer
```

Para posicionarse en un parámetro particular, se debe de incluir el número apropiado de comas. Por ejemplo, para indicar el segundo y quinto parámetro, proporcionar el comando con el siguiente formato:

```
comando,,parámetro,,,parámetro
```

Abreviación de Parametros:

Se pueden abreviar los nombres de parametros tomando el primer caracter de cada palabra en el nombre del parámetro. Por ejemplo, la abreviación para el parámetro COMMAND es C, y la abreviación para - el parámetro OUTPUT_DESTINATION_USAGE es ODU.

Las siguientes son excepciones a este estandar:

- STATUS no tiene abreviación
- PASSWORD es abreviado como PW
- OUTPUT_DESTINATION es abreviado como ODE
- OUTPUT_DISPOSITION es abreviado como ODI
- Cualquier parámetro iniciado con MAX o MIN (como MAXIMUN_WORKING_SET y MINIMUM_WORDING_SET) conservan esos tres caracteres como parte de su abreviación (MAXWS Y MINWS). En esta forma, la unicidad de estos parametros es conservada.

Indicando Listas de Valores:

Una lista de valores indica uno o más valores que pueden ser procesados por un parámetro. Cada parámetro define la manera en la cual se pueden indicar los valores.

Se puede indicar un valor de cualquiera de las siguientes formas:

- Valor Simple
- Rango de Valores
- Conjunto de Valores

- Indicando un Valor Simple

Para indicar un valor simple para un parámetro, use una de las siguientes formas:

```
nombre_del_parámetro = valor
nombre_del_parametro = lista de valores
valor
lista de valores
```

Por ejemplo, vamos a asumir que hay un comando llamado DISPLAY_NUMBER y que despliega el valor decimal de una expresión entera. El comando acepta un solo parámetro llamado NUMBER, el cual debe ser un valor entero. Se puede indicar este comando y su parámetro de la siguiente forma:

```
/display_number  number=  a(16)
10
```

Para indicar más de un valor, encerrar el grupo de valores en paréntesis y delimitar cada valor con comas o espacios.

```
(valor, valor,...,valor)
```

Es opcional poner un espacio después del paréntesis abierto o antes de un paréntesis cerrado.

Por ejemplo, el siguiente comando despliega tres números:

```
/display_number (4+1,2*3,8-1)
 5
 6
 7
```

- Indicando un Valor como un Rango

Se puede indicar un valor de parámetro como un rango mostrándolo como dos valores de la misma clase separados por una elipsis - - (dos puntos consecutivos). No debe haber espacios antes o después de la elipsis. Los siguientes son formatos válidos:

```
nombre_del_parámetro = valor..valor
nombre_del_parámetro = lista de valores..valor
valor..valor
lista de valores..valor
```

El siguiente ejemplo es un valor cuyo rango esta entre los enteros 2 y 4:

```
2..4
```

Si el parámetro NUMBER del comando DISPLAY_NUMBER acepta un rango de valores, el siguiente comando es válido:

```
/display_number 2..4
2
3
4
```

Para indicar una lista de valores tipo rango, encerrar el grupo - de valores en paréntesis y delimitado cada valor con comas o espa_ cios.

```
/display_number (2..2**2, 100(8)..42(16) )
 2
 3
 4
64
65
66
```

Es opcional poner un espacio después del paréntesis abierto o antes de un paréntesis cerrado.

- Indicando Conjuntos de Valores

Un valor simple y un valor rango son cada uno compuesto de un solo valor elemental. SCL proporciona otro tipo de valor llamado - conjunto de valores. Este tipo de valor acepta mas de un elemento y es definido como sigue:

(valor elemental, valor elemental,...,valor elemental)

Si una lista de valores contiene un conjunto de valores, hay que encerrar la lista en paréntesis y delimitar cada conjunto de valo_ res con comas o espacios.

((Conjunto de valores) (conjunto de valores)...(conjunto de - valores))

También se puede incluir valores simples y valores rango en la -- misma lista.

Por ejemplo, asumamos que el comando DISPLAY_NUMBER acepta arriba de 2 valores elementales por conjunto de valores como sigue:

- El primer valor elemental indica una expresión entera a ser desplegada.
- El segundo valor elemental indica la base en la cual cada - entero será desplegado. (Si se omite el segundo valor elemental, una base decimal es asumida).

La siguiente forma del comando es válida y produce la salida que se muestra:

```
/display_number ( (5,2) (100,16) 143(8) )
 101
  64
  99
```

El primer conjunto de valores es como sigue:

```
(5,2)
```

Contiene los valores elementales 5 y 2. El comando DISPLAY_NUMBER desplega el equivalente binario del número 5.

El segundo conjunto de valores es como sigue:

```
(100,16)
```

Contiene los valores elementales 100 y 6. El comando DISPLAY_NUMBER desplega el equivalente hexadecimal del número 100 (hexadecimal). Alternativamente, se puede cambiar este conjunto de valores como sigue y obtener el mismo resultado:

```
(64(16),10)
```

El tercer valor en la lista es un valor simple:

```
143(8)
```

El comando DISPLAY_NUMBER desplega el equivalente decimal del número 143 (octal).

- Representación de un Valor como Expresión

Se puede representar cada valor o valor elemental como una expresión. Por ejemplo, el parámetro NUMBER espera un valor entero expresado en cualquiera de las siguientes formas:

```
10

(i=j)

(((i-3)/(j*2)))
```

Espacios en Valores de Parametros:

En contraste con los estatutos de asignación, las expresiones para parametros no pueden contener espacios en los extremos de los operadores.

Los ejemplos son basado en el siguiente procedimiento:

```
PROC  muestra (
   número    :   integer=$optional
   string    :   string=$optional
   estado    ;   var of status=$optional
   )

   IF  $specified (número) THEN
       display_value  $value (número)
   ELSELF  $specified (string) THEN
       display_value  $value (string)
   IFEND

PROCEND  muestra
```

Los siguientes son ejemplos de expresiones válidas de parametros:

```
muestra número=(3+5)
muestra string='a'//'b'
```

Los siguientes son ejemplos de expresiones inválidas para parámetros:

EXPRESION INVALIDA	RAZON
muestra número=(3 +5)	Contiene un espacio antes del operador +.
muestra string='a' //'b'	Contiene un espacio antes y después del operador //.

CLASES DE VALORES

Para cada parámetro, la descripción del comando define cierta clase de valor. Si el valor que se indica no es de la clase definida para ese parámetro, el sistema termina el comando y emite un mensaje de diagnóstico.

SCL reconoce las siguientes clases de valores, cada una de las cuales puede ser representada como una expresión:

CLASE DE VALOR	DESCRIPCION
File	Nombre de un archivo de SCL. Un nombre de archivo consiste de un máximo de 31 caracteres (alfabéticos, numéricos y especiales) y debe de empezar con un caracter alfabético).
Name	Combinación de 1 a 31 caracteres (alfabéticos, numéricos y especiales). Un nombre de SCL inicia con caracter alfabético. Ver nombres de SCL en capítulo 2.
String	Combinacion de hasta 256 caracteres (alfabéticos, numéricos y especiales) delimitados por una comilla simple. Para mas información, ver constantes tipo string en el capítulo 2.
Integer	Valores enteros binario, octal, decimal o hexadecimal. Los caracteres numéricos expresan valores enteros. Una combinación de caracteres numéricos y caracteres alfabéticos A.B.C.D.E. y F (mayúsculas y minúsculas) expresan enteros hexadecimales. Para más información, ver constantes enteras en capítulo 2.

BOOlean	Valor que es verdadero o falso. Las constantes booleanas son TRUE, YES, ON, FALSE, NO, y OFF. Además, se pueden crear variables booleanas de acuerdo a las necesidades. Para más información ver constantes booleanas en el capítulo 2.
Real	Secuencia de dígitos conteniendo un punto decimal y un exponente opcional con signo también opcional. Para más información, ver constantes de números reales en el capítulo 2.
Status	Registro que contiene 3 campos (NORMAL, CONDITION, Y TEXT) que contienen en el estado completo del comando. Para más información, ver variables tipo estado en el capítulo 2 y Procesamiento de Condición en el capítulo 5.
Any	Cualquier expresión. Ciertos parametros permiten valores que no son de una clase específica o que puede ser de una de varias clases diferentes. Por ejemplo ver Desplegar el Valor de una Expresión en el capítulo 3.
Application Value	Valor cuya sintaxis y significado esta definida por una aplicación o utilería.
Keyword	Nombre que tiene un significado especial en el contexto de un parámetro particular. En general, usarlo cuando se desea que un parámetro reconozca únicamente un conjunto de valores predefinidos. Las keywords son usadas además para una de las otras clases de valores para un parámetro con nombre COUNT podrá normalmente esperar valores enteros pero puede ser definido para aceptar el valor de keyword ALL. Si subsecuentemente se indica COUNT =ALL, el sistema repite una operación específica hasta que algún límite es alcanzado (por ejemplo, hasta que todas las marcas de una cinta hallan sido saltadas).

5.- COMANDOS Y ESTATUTOS

PANORAMICA GENERAL 5-1

USO DE COMANDOS Y ESTATUTOS DE CONTROL 5-1
Creación, Eliminación y Desplegado de Variables 5-1
o Creación de Variables (CREATE_VARIABLE) 5-2
o Alcance de Variables en una Estructura Tipo Block.. 5-4
o Eliminación de Variables (DELETE_VARIABLE) 5-5
o Desplegado de una Lista de Variables (DISPLAY_VARIABLE_LIST) 5-6
Estructurando un Grupo de Comandos 5-6
o Agrupación de Estatutos en un Block (BLOCK/BLOCK-END)..... 5-7
o Causando la Repetición Ilimitada de una Lista de Estatutos (LOOP/LOOPEND) 5-7
o Causando la Repetición Precondicional de una Lista de Estatutos (WHILE/WHILEND) 5-8
o Causando la Repetición Postcondicional de una Lista de Estatutos (REPEAT/UNTIL) 5-9
o Causando la Repetición Controlada de una Lista de Estatutos (FOR/FOREND) 5-9
Controlando el Flujo de un Grupo de Comandos 5-10
o Causando la siguiente Iteración de un Estatuto Repetitivo (CYCLE) 5-10
o Saliendo de un Estatuto Estructurado (EXIT)..... 5-11
o Saliendo de un Procedimiento o Utilería (EXIT) 5-12
Ejecutando Listas de Estatutos Condicionalmente (IF/END) 5-13
Procesamiento Condicionado 5-13
o Procesamiento de Error 5-13
. Registro de la Variable Tipo Estado 5-14
. Creación de una Variable Tipo Estado 5-14
. Haciendo Pruebas para Condiciones Específicas de Error 5-15
o Manipulación Condicionada 5-15
. Establecimiento de un Manipulador Condicionado (WHEN/WHENEND) 5-16
. Saliendo de un Block WHEN (CONTINUE) 5-18
. Cancelación de una Condición (CANCEL) 5-19
Lectura de Líneas de un Archivo y Escritura de Líneas a un archivo 5-19
o Lectura de Líneas de un Archivo (ACCEPT_LINE) 5-19
o Escritura de Líneas a un Archivo (PUT_LINE)..... 5-20
Inserción de Archivos o String en un Grupo de Comandos 5-20
o Inserción de Archivos (INCLUDE_FILE) 5-21
o Inserción de Líneas (INCLUDE_LINE)..... 5-21
o Inserción de Comandos (INCLUDE_COMMAND) 5-22

Suspensión del Procesamiento de Comandos (WAIT) 5-22
Cambiando el Medio Ambiente del Sistema 5-23
o Establecimiento de un Nuevo Medio Ambiente (PUSH).. 5-23
o Restauración del Medio Ambiente Anterior (POP) 5-23

DESCRIPCIONES DE FORMATOS DE COMANDOS Y ESTATUTOS 5-23
ACCEPT_LINE 5-24
BLOCK/BLOCKEND 5-25
CANCEL 5-26
CONTINUE 5-26
CREATE_VARIABLE 5-27
CYCLE 5-29
DELETE_VARIABLE 5-30
DISPLAY_VARIABLE_LIST 5-31
EXIT 5-31
EXIT_PROC 5-32
FOR/FOREND 5-33
IF/IFEND 5-34
INCLUDE_COMMAND 5-35
INCLUDE_FILE 5-36
INCLUDE_LINE 5-37
LOOP/LOOPEND 5-38
POP 5-38
PUSH 5-38
PUT_LINE 5-39
REPEAT_UNTIL 5-40
WAIT 5-40
WHEN/WHENEND 5-41
WHILE/WHILEND 5-42

5.- COMANDOS Y ESTATUTOS

Este capítulo describe los comandos y estatutos usados para estructurar y controlar un grupo de comandos de SCL.

PANORAMICA GENERAL

SCL proporciona varios comandos y estatutos que controlan y estructuran un grupo de comandos. Estos comandos y estatutos son usados para realizar las siguientes tareas descritas en este capítulo:

- Creación, eliminación y desplegado de variables.
- Estructuración de un Grupo de Comandos en Blocks.
- Control del Flujo de un Grupo de Comandos.
- Ejecución de listas de estatutos condicionalmente.
- Procesamiento de Condiciones.
- Lectura de Líneas de un archivo y escritura de líneas a un archivo.
- Insersión de archivos o strings en un grupo de comandos.
- Suspensión del procesamiento de un comando.
- Cambiando el Medio Ambiente del Sistema.

Este capítulo también contiene una sección de referencia describiendo los formatos y parametros de los comandos y estatutos.

Las siguientes tareas son descritas en el manual de SCL "System -- Interface".

- Establecimiento del catalogo de trabajo (SET_WORKING_CATALOG)
- Delimitación de una tarea (TASK/TASKEND).
- Delimitando una secuencia de estatutos (JOB/JOBEND).
- Creación de archivos texto (COLLECT_TEXT).

USO DE COMANDOS Y ESTATUTOS DE CONTROL

Esta sección describe los usos comunes de los comandos seleccionados de SCL y de los estatutos de control. Los ejemplos acompañan cada descripción.

Creación, Eliminación y Desplegado de Variables:

Las variables de SCL existen en el block en el cual son creadas; cuando el block cesa de existir, son borradas automáticamente. Antes de que las variables sean referenciadas dentro de un block, deben ser creadas (ya sea explícita o implícitamente).

El ambiente en el cual las variables son conocidas se llama el alcance de la variable. A menos que otra cosa sea especificada, el alcance de una variable es local al block en el cual es creada; esto es, las variables creadas locales para un block son desconocidas para otros blocks.

Se pueden crear variables en las siguientes clases de blocks:

- Block de Job. El block de un job existe una vez que el job inicia e incluye todos los estatutos de un comando LOGIN hasta dar un comando LOGOUT, incluyendo cualquiera de los comandos prologo o epilogo.
- Block de Utilería. Un block de utilería se establece por un programa que procesa subcomandos de una utilería; cuando la utilería termina, el block deja de estar definido.
- Block de Procedimiento. Un block de procedimiento es creado cuando se llama a un procedimiento en SCL; cuando termina el procedimiento, el block deja de estar definido. Un procedimiento puede ser llamado de cualquier parte dentro de un job pero es considerado como un block separado en el momento en el cual es llamado. Por lo tanto, cualquier variable local creada dentro del procedimiento son conocidas unicamente para el procedimiento.
- Block WHEN. Un block When es creado cuando una condición ocurre para la cual el estatuto WHEN ha sido definido. Ver Procesamiento Condicionado de este mismo capítulo.

Esta sección describe lo siguiente:

- Como crear una variable
- Alcance de variables
- Como eliminar una variable
- Como desplegar variables accesibles

NOTA:

Para información de como desplegar valores, ver Desplegar el Valor de una Expresión en el Capítulo 3 (Comando DISPLAY_VALUE).

---

o Creación de Variables (CREATE_VARIABLE)

Se pueden crear variables de dos formas:

- Explícitamente, usando el comando CREATE_VARIABLE (descrita más después en este capítulo).
- Implícitamente, cuando son usados en un estatuto de asignación o como variable de control para un estatuto FOR. (El estatuto de asignación es descrito en el capítulo 1; el estatuto FOR es descrito posteriormente en este capítulo).

Los ejemplos en esta sección muestran como accesar una variable con un alcance de JOB desde dentro de un procedimiento. Los ejemplos son basados en el siguiente procedimiento (almacenado en el archivo $LOCAL.MI_PROC):

```
PROC mi_proc  ( )

 display_value  $variable (g declared)
 create_variable  name=g  scope=xref
 display_value  $variable (g declared)
```

```
PROCEDN mi_proc
```

Indicando el siguiente comando, se crea la variable G con un alce de JOB:

```
/create_variable  name=g  scope=j
```

Ejecutando el procedimiento produce los siguientes resultados:

```
/mi_proc
 NONLOCAL
 LOCAL
/
```

Al procesarse el primer DISPLAY_VALUE en el procedimiento, antes de ejecutarse el comando CREATE_VARIABLE, el sistema regresa el atributo de NONLOCAL. Este atributo indica que la variable G -- fue declarada en un block existiendo fuera del procedimiento. -- Cuando el segundo comando DISPLAY_VALUE en el procedimiento es - ejecutado, el sistema regresa el atributo de LOCAL. Este atributo indica que la variable G fue declarada dentro del procedimiento (por el comando CREATE_VARIABLE).

El siguiente ejemplo crea una variable llamada ESTADO_GLOBAL, de clase tipo ESTADO (Status), y con alcance de JOB:

```
/create_variable  estado_global  kind=status  scope=job
```

La variable ESTADO_GLOBAL puede ser referenciada externamente -- desde cualquier block (con el parámetro SCOPE=XREF).

El siguiente ejemplo crea una variable llamada REALIZADO de clase booleana. Su valor inicial es establecido en FALSE, y se le proporciona un alcance de XDCL. Otros blocks pueden referenciar a la variable REALIZADO si se crea la misma variable con una alcance de XREF.

```
/create_variable  name=realizado  kind=boolean..
../value=false  scope=xdcl
```

El siguiente ejemplo crea una variable llamada STRING_CORTO de - clase string. Su valor inicial es 'US,' con un máximo de longitud de 2 caracteres. Como el parámetro SCOPE es omitido, se asume un alcance LOCAL.

```
/create_variable  string_corto , value='US'  kind (string, 2)
```

El siguiente ejemplo crea un arreglo de enteros (ARREGLO_1) de - 10 elementos que puede referenciarse del 1 al 10:

```
/create_variable  arreglo_1  dimensión=1..10
```

El siguiente ejemplo crea un arreglo tipo string (ARREGLO_2) de 10 elementos que puede referenciarse del -5 al 4.

```
/create_variable  arreglo_2  dimension=-5..4
```

En el siguiente ejemplo, el parámetro VALUE establece una variable booleana llamada bandera en TRUE:

```
/create_variable  bandera  kind=boolean  value=true
```

o Alcance de Variables en una Estructura tipo Block

La figura5-1 ilustra un block de job. Las siguientes consideraciones se aplican a esta ilustración:

- Cada block anidado es un block de procedimiento creado por el llamado a un procedimiento.
- El block de job abarca todo desde el LOGIN hasta el LOGOUT.
- El block de job contiene dos blocks subordinados creados por la ejecución de los procedimientos A y D.
- Las variables creadas en el block de job y fuera de los blocks A y D con alcance XDCL son accesibles dentro del block de job. Otros blocks pueden accesar estas variables si se crean las mismas variables con alcance XREF.
- Las variables locales creadas dentro del block A son conocidas unicamente en el block A; las variables locales creadas dentro del block D son conocidas unicamente en el block D.
- Una vez que los blocks A y D dejan de existir (esto es, una vez que el procedimiento termina), las variables locales dentro de su alcance también dejan de existir.
- Los blocks A y D, cada uno contiene un block de procedimiento subordinado. El block C es subordinado al block B, y el block F es subordinado al block E.
- Las reglas para el alcance de las variables en los blocks B, C, E y F son las mismas a las descritas para los blocks A y D.

TABLA 5-1. Alcance de Variables

Variable	Definida en el Block	Accesible al Block
i	JOB	JOB, A,B,C,D,E,F
a	A	A,B,C
b	B	B,C
c	C	C
j	JOB	JOB, D,E,F
d	D	D,E,F
e	E	E,F
f	F	F
k	JOB	JOB
l	A	A
m	B	B
n	D	D
o	E	E

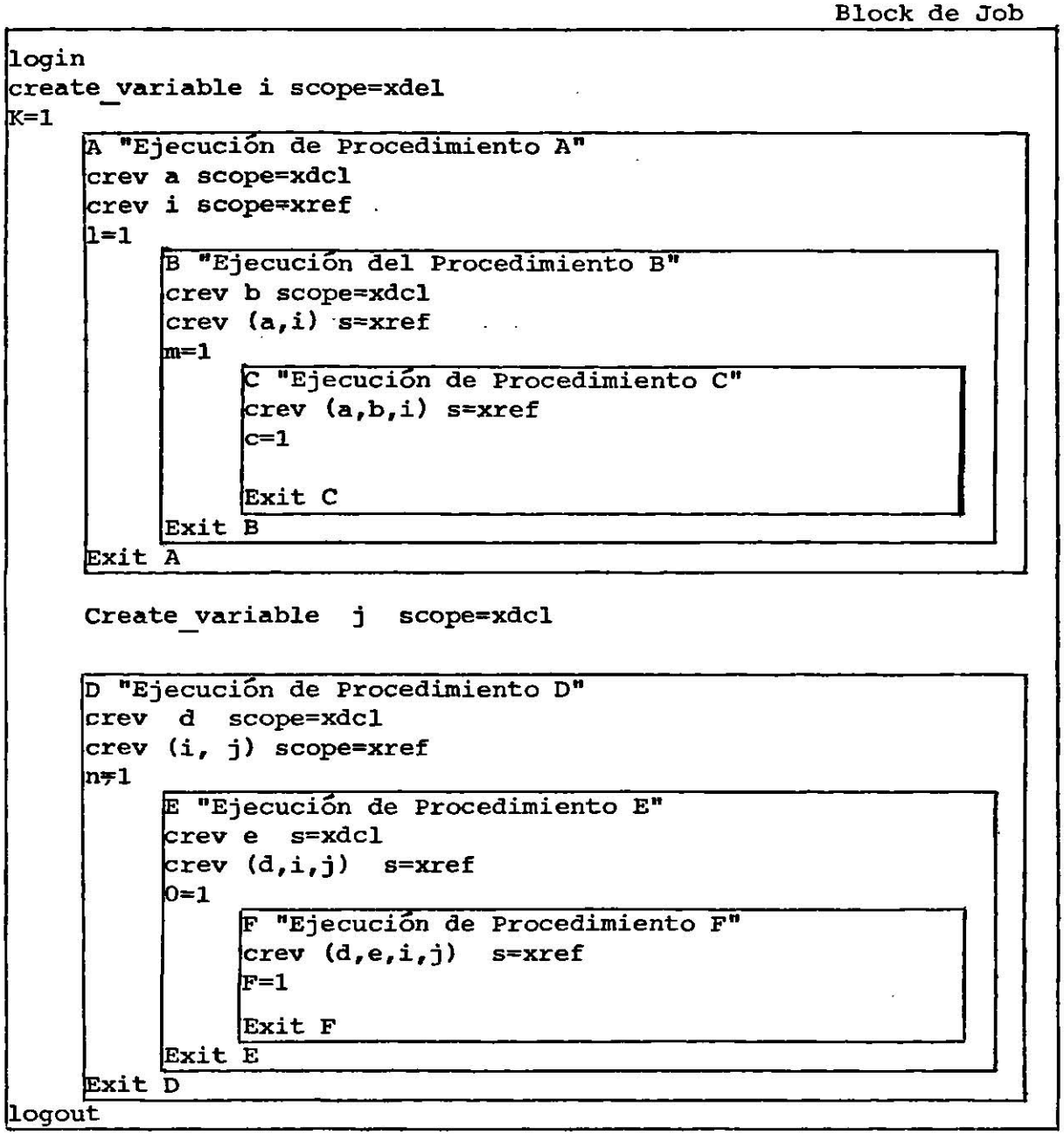


Figura 5-1 Estructura de un Block de JOB

o Eliminación de Variables (DELETE_VARIABLE)

Se puede eliminar variables en las siguientes formas:

- Explícitamente, usando el comando DELETE_VARIABLE unicamente desde el block actual. (El comando DELETE_VARIABLE es descrito posteriormente en este capítulo).
- Implícitamente, cuando el block en el cual las variables fueron creadas termina.

El siguiente ejemplo utiliza el comando DELETE_VARIALBE para borrar las variables con nombre CONTADOR Y CICLOS:

```
/delete_variable  (contador, ciclos)
```

o Desplegado de una Lista de Variables (DISPLAY_VARIABLE_LIST)

Para desplegar una lista de variables para el job actual, se usa el comando DISPLAY_VARIABLE_LIST. El sistema despliega los nombres de variables empezando con la variable creada mas recientemente.

El siguiente es un ejemplo de desplegado de variables. VARIABLE_1 fue creada primero y VARIABLE_4 fue la última en ser creada.

```
/display_variable_list
 LOCAL VARIABLES IN JOB

  Variable_4      Variable_3
  Variable_2      Variable_1
 OSV$status
```

OSV$STATUS es una variable tipo estado creada por el sistema en un block de job al inicio del job.

Estructurando un Grupo de Comandos:

SCL proporciona estatutos (llamados estatutos estructurados) que pueden ser utilizados para estructurar un grupo de comandos en grupos de estatutos. Los grupos son procesados como entidades separadas hasta que una condición de terminación específica o un estatuto es encontrado.

Cada estatuto estructurado tiene el siguiente formato:

```
Etiqueta: CLAUSULA DE INICIO DEL ESTATUTO ESTRUCTURADO
          Lista de estatutos
CLAUSULA DE TERMINO DEL ESTATUTO ESTRUCTURADO etiqueta
```

La etiqueta opcional es cualquier nombre de SCL; es una forma conveniente de referirse a un estatuto estructurado. Los estatutos de control de flujo (descritos posteriormente en este capítulo) pueden transferir el control dentro o fuera de un estatuto estructurado al referenciar la etiqueta del estatuto.

Una etiqueta antecede a un estatuto estructurado al cual lo identifica y esta separado del estatuto estructurado por el delimitador (:). A excepción del estatuto REPEAT, una etiqueta puede opcionalmente continuar con el estatuto estructurado.

El siguiente ejemplo ilustra un estatuto LOOP con etiqueta

```
entrada_multiple:  LOOP
        .
        .
    "Lista de estatutos para entrada_multiple"
        .
        .
LOOPEND entrada_multiple
```

Las siguientes condiciones se aplican al formato de estatutos etiquetados:

- Las etiquetas de inicio y de fin deben ser identicas.
- Una etiqueta no puede ser referenciada por procedimientos llamados desde dentro de un estatuto estructurado o por listas

de estatutos introducidos por los comandos INCLUDE FILE ó - - INCLUDE LINE (descritos posteriormente en este capítulo).
- Ningún espacio debe haber antes de los dos puntos (:).
- Es opcional poner un espacio después de los dos puntos (:).
- El alcance de una etiqueta de estatuto esta restringida al estatuto que no es etiquetado. Esto es, no es posible referenciar una etiqueta en un estatuto estructurado desde fuera de este estatuto.

Los siguientes estatutos estructurados son descritos en esta sec- - ción:

- BLOCK
- LOOP
- WHILE
- REPEAT
- FOR

o Agrupación de Estatutos en un Block (BLOCK/BLOCKEND)

Para agrupar una secuencia de estatutos en un block, se usa el - estatuto BLOCK. El estatuto BLOCKEND marca el fin del block. - Para salirse desde el block puede hacerse cuando el último estatuto en la lista de estatutos es ejecutado o a través de una - - transferencia de control con el estatuto EXIT. Los estatutos de control de flujo serán descritos posteriormente en este capítu-- lo.

NOTA:

El tipo de block referenciado en conección con el estatuto BLOCK difiere del block mencionado anteriormente en este capítulo en - Creación, Eliminación y Desplegado de Variables.

El siguiente ejemplo crea un block llamado PRINCIPAL. La lista de estatutos incluye al estatuto de control de flujo EXIT.que -- ocasiona una salida del block PRINCIPAL cuando la condición de - prueba es TRUE. Si la condición de prueba es FALSE, el estatuto EXIT es inhibido.

```
principal:  BLOCK
              .
              .
              .
        exit when not status normal
              .
              .
              .
BLOCKEND principal
```

o Causando la Repetición Ilimitada de una Lista de Estatutos (LOOP /LOOPEND)

Para ocasionar que una lista de estatutos sea repetida en un número ilimitado de veces, se usa el estatuto LOOP. Este estatuto es útil, por ejemplo, cuando se desea leer un número indetermina do de línea de un archivo.

El estatuto LOOPEND marca el fin de la lista de estatutos ha ser repetidos. Para salir de un estatuto LOOP solo es posible con un estatuto EXIT.

El siguiente ejemplo usa un estatuto LOOP para leer líneas del archivo INPUT. Cada línea leída es almacenada en una variable tipo string STRING_ENTRADA. Cuando una línea nula sea indicada (al -- dar un simple carriage return) la ejecución del LOOP finaliza; de otro modo la variable tipo string es escrita al archivo $OUTPUT.

```
string_entrada="
lee_input: LOOP
    accept_line  string_entrada  input
    Exit  lee_input  WHEN $strlen (string_entrada)=Ø
    display_value  string_entrada
LOOPEND lee_input
```

Cuando se ejecuta este ciclo, la siguiente interacción toma lu- - gar:

```
PROPORCIONA STRING_ENTRADA  prueba
 prueba
PROPORCIONA STRING_ENTRADA
/
```

Presionando la tecla RETURN después del prompt PROPORCIONA STRING _ENTRADA da como resultado que la variable STRING_ENTRADA tiene - una longitud de Ø. Esto ocasiona que la condición del estatuto - EXIT sea TRUE. Así termina el ciclo delimitado por LOOP/LOOPEND.

o Causando la Repetición Precondicional de una Lista de Estatutos (WHILE/WHILEND)

Para realizar una repetición condicional de una lista de estatu-- tos, se usa el estatuto WHILE. Antes de cada iteración de la lis ta de estatutos, la expresión booleana en el estatuto WHILE es -- evaluada. Si la expresión es TRUE, la lista de estatutos es ejecutada; si la expresión es FALSE, el control se transfiere al siguiente estatuto de la clausula WHILEND.

El estatuto WHILEND marca el fin de la lista de estatutos.

El siguiente ejemplo calcula el factorial de una variable llamada FACTORIAL_DE:

```
/factorial_de=5          "Calcula el factorial de 5."
/ultimo_valor=1          "Valor para el primer ciclo."
/factorial:  while  factorial_de>1 do
while/ultimo_valor= factorial_de * ultimo_valor
while/factorial_de= factorial_de - 1
while/whilend factorial
/display_value  ultimo_valor
120
/
```

- Causando la Repetición Postcondicional de una Lista de Estatutos (REPEAT/UNTIL)

  Para realizar una repetición condicional de una lista de estatutos, se usa el estatuto REPEAT. Después de cada iteración de la lista de estatutos, la expresión booleana dentro de la clausula UNTIL es evaluada. Si esta expresión es evaluada como FALSE, la lista de estatutos se vuelve a ejecutar. Si la expresión es evaluada como TRUE, el control se transfiere al siguiente estatuto de la clausula.UNTIL. Por lo tanto, un estatuto REPEAT siempre se ejecuta mínimo una vez.

  El siguiente ejemplo lee líneas del archivo INPUT hasta que una línea nula sea encontrada:

```
/línea="
/repeat
repeat/accept_line  v=linea  i=input
repeat/display_value  linea
repeat/until  linea="
PROPORCIONA LINEA  Linea 1
Linea 1
PROPORCIONA LINEA

/
```

  Presionando la tecla RETURN después del prompt PROPORCIONA LINEA, señala el fin de la captura del ciclo.

- Causando la Repetición Controlada de una Lista de Estatutos (FOR/FOREND)

  Para controlar la repetición de una lista de estatutos, se usa el estatuto FOR.

  El estatuto FOREND marca el fin de la lista de estatutos a ser repetidos.

  El siguiente ejemplo despliega cada caracter de un string en líneas separadas:

```
/string='down hil'
/blancos=' '
/for i=1 to $strlen (string) do
for/display_value $substr(blancos, 1, i,)//..
for../$substr(string, i, 1)
for/forend
 d
  o
   w
    n

       h
        i
         l
          l
```

o Controlando el Flujo de un Grupo de Comandos

Los siguientes estatutos de control de flujo controlan el flujo de un estatuto estructurado:

- CYCLE; estatuto que provoca la ejecución de la siguiente iteración en un estatuto repetitivo.
- EXIT; estatuto que provoca transferir el control a uno de los siguientes:
  - Al siguiente estatuto después del estatuto estructurado.
  - Al llamador de un procedimiento de SCL o utilería.

o Causando la siguiente Iteración de un Estatuto Repetitivo (CYCLE).

Para provocar que la siguiente iteración de un estatuto repetitivo sea ejecutado, se usa el estatuto CYCLE. El estatuto CYCLE puede usarse dentro de una lista de estatutos de los siguientes estatutos estructurados:

- Estatuto LOOP: La ocurrencia del estatuto CYCLE provoca que el control se transfiera al primer estatuto de la lista de estatutos del estatuto LOOP.
- Estatuto WHILE: La expresión booleana que condicionalmente repite al estatuto se evalua, y el estatuto WHILE termina o continua desde ese punto.
- Estatuto REPEAT: La expresión booleana que condicionalmente repite al estatuto se evalua, y el estatuto REPEAT termina o continua desde ese punto.
- Estatuto FOR: El estatuto FOR termina con la última iteración o continua con la siguiente iteración.

La figura 5-2 ilustra funcionamiento del estatuto CYCLE. Las siguientes consideraciones se aplican a esta ilustración:

- Asume que hay estatutos condicionales que provocarían que el control se pasara al loop 2 antes del estatuto CYCLE.
- Si el primer CYCLE se ejecuta, el estatuto LOOP con etiqueta LOOP1 vuelve a ejecutarse con su siguiente iteración.
- Si el segundo CYCLE se ejecuta, el estatuto LOOP con etiqueta LOOP1 es también vuelto a ejecutar en su siguiente iteración. Por lo tanto, LOOP1 y LOOP2 son efectivamente ciclados.
- Si el tercer estatuto CYCLE se ejecuta, el estatuto LOOP con etiqueta LOOP2 se vuelve a ejecutar y el control permanece en LOOP2.

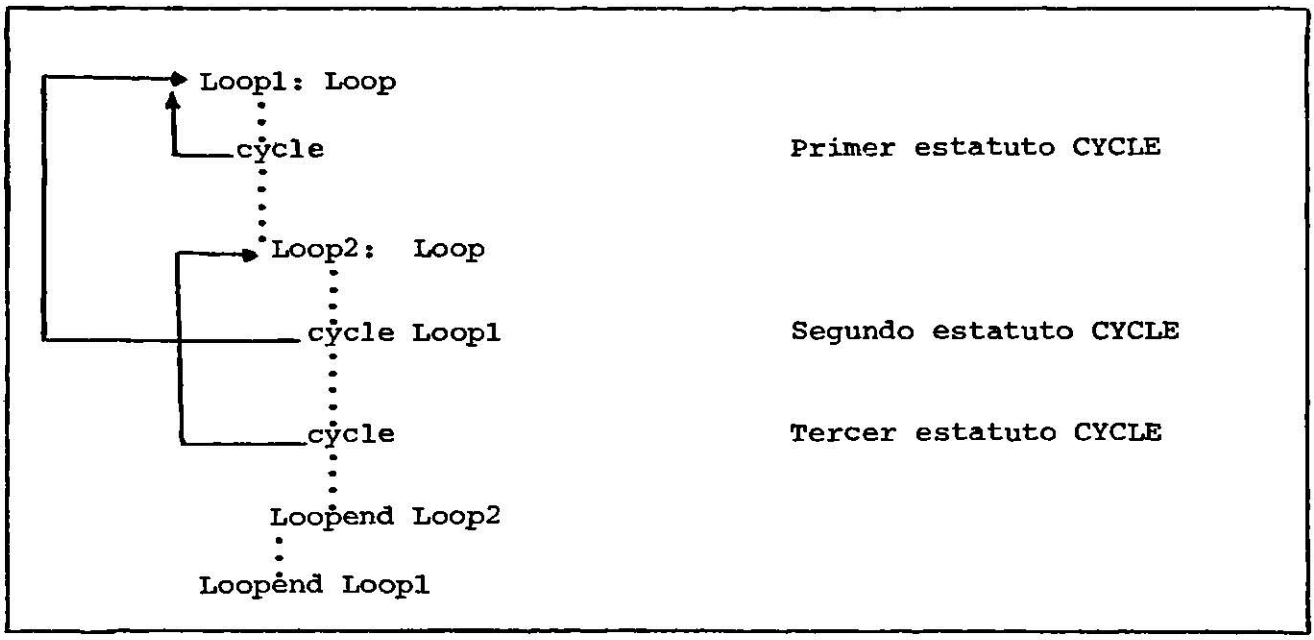


Figura 5-2. Uso del Estatuto CYCLE

- Saliendo de un Estatuto Estructurado (EXIT)

  Para provocar que un Job transfiera el control fuera de un estatuto estructurado, se usa el estatuto EXIT. La ejecución continua con el estatuto posterior al estatuto estructurado.

  La figura 5-3 ilustra el funcionamiento del estatuto EXIT. Las siguientes consideraciones se aplican a esta ilustración:

  - Se asume que, antes del estatuto EXIT, hay estatutos condicionales que provocarían que el control pasaría al BLOCK 2.
  - Si el primer EXIT es ejecutado, el control se transfiere al siguiente estatuto después del estatuto LOOPEND con etiqueta BLOCK 1, y el BLOCK 1 se termina.
  - Si se ejecuta el segundo EXIT, el control se transfiere al siguiente estatuto después del estatuto LOOPEND con etiqueta -- BLOCK1. Por lo tanto, ambos BLOCK 2 y BLOCK 1 son terminados.
  - Si el tercer estatuto EXIT es ejecutado, el control se transfiere al siguiente estatuto después del estatuto LOOPEND con etiqueta BLOCK 2, y el BLOCK 1 permanece activo.

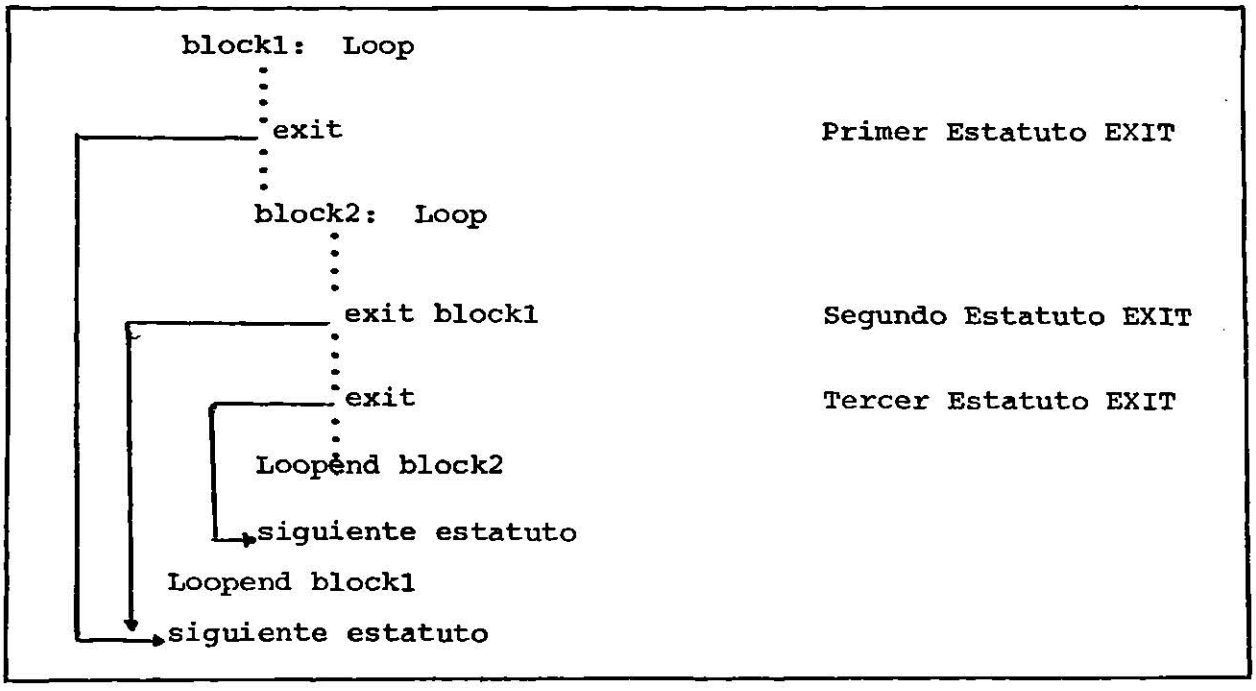


Figura 5-3. Uso del Estatuto EXIT

En el siguiente ejemplo, el estatuto FOR termina cuando el sistema encuentra un status anormal:

```
proceso:  FOR i=1  TO 10 DO
    .
    .
    .
  EXIT proceso WHEN NOT status.normal
    .
    .
    .
FOREND proceso
```

o Saliendo de un Procedimiento o Utilería (EXIT)

Para regresar el control al llamador de un procedimiento de SCL o a una utilería, se usa el estatuto EXIT. La ejecución se reanuda en el estatuto posterior al llamado del procedimiento o -- o utilería.

El siguiente ejemplo sale de un procedimiento con un estado de anormal:

```
PROC  ejemplo  (status)
    .
    .
 EXIT ejemplo WITH..
  $status (false, 'US', 1,' Fallas en el Procedimiento EJEMPLO')
    .
    .
PROCEND ejemplo
```

Si el estatuto EXIT se ejecuta, aparece entonces la siguiente salida:

```
/ ejemplo
  -- ERROR -- CC=US 1  TEXT=? Fallas e n el Procedimiento - -
  EJEMPLO
```

Ejecutando Listas de Estatutos Condicionalmente (IF/IFEND):

Para ejecutar condicionalmente una o varias listas de estatutos, se usa el estatuto IF en conjunción con las clausulas.ELSEIF y ELSE. La ejecución del estatuto esta basada en la evaluación de expresiones booleanas asociadas.

El estatuto IFEND marca el termino de la lista de estatutos.

El siguiente es un ejemplo de un estatuto IF dentro de un procedi-- miento de SCL (referirse al capítulo 7 para las descripciones de -- las funciones $PARAMETER Y $VALUE). Asumamos que un parametro entero de nombre INT es pasado al procedimiento. El siguiente estatuto IF provoca que el procedimiento reporte si el entero es negativo, - cero, o positivo:

```
PROC  pos_o_neg  (int: integer)
   IF $VALUE (int)= Ø  THEN
      display_value  $parameter (int)//' es Cero'
   ELSEIF $VALUE (int) > o  THEN
      display_value  $parameter (int)//' es Positivo'
   ELSE
      display_value  $parameter (int)//' es Negativo'
   IFEND
PROCEND pos_o_neg
```

Procesamiento Condicionado:

SCL proporciona los siguientes mecanismos básicos para la especificación de la acción a tomar cuando ocurra una condición anormal:

- Procesamiento de Error. Este mecanismo esta disponible cuando - un parametro STATUS es incluido en un comando de SCL y el comando termina con un estado de anormal (como errores de Sintáxis en el comando). Como un ejemplo de como son procesados los errores de sintáxis, ver la descripción del comando INCLUDE_LINE más adelante en este capítulo.
- Manipulación Condicionada. Este mecanismo puede implementarse - cuando ocurre una condición anormal para un comando bajo las siguientes circunstancias:
  - El parametro STATUS no es indicado en el comando
  - Errores de sintáxis encontrados en el comando
  - Se ha alcanzado el límite de recursos para un job
  - Cuando se indique una pausa

o Procesamiento de Error

Todos los comandos en NOS/VE tienen un parametro opcional llamado STATUS. Si se incluye en el comando, el interprete de SCL -- prosigue con el siguiente comando a pesar de encontrarse con una condición anormal. Por lo tanto, la mayoría de los comandos no

informan de un posible error si el parametro STATUS es incluido. La ausencia de el parametro status ocasiona que el interprete - de SCL salte los comandos sucesivos en el block actual.

Cuando se especifica el parametro STATUS, una variable de tipo status (estado) declarada previamente debe proporcionarse como su valor. Esta variable es usada por el interprete de SCL para mantener el estado completo del comando.

Examinando los contenidos de una variable status indicada, los comandos sucesivos pueden alterar el flujo de estatutos, basado en la ocurrencia de condiciones anormales.

. Registro de Variable Tipo Estado
Un registro de una variable tipo estado contiene los siguien tes campos:

NORMAL
Un valor booleano, el cual es FALSE si el requerimiento no - pudo ser procesado correctamente (estado anormal) y TRUE si ha sido procesado correctamente (estado normal).

CONDITION
Un código único entero indicando el error específico que fue detectado. Este campo contiene un producto identificador de dos caracteres. Este campo es indefinido si el campo NORMAL es TRUE. Para verificar el código de condición, usar la fun ción $CONDITION_NAME descrita en el capítulo 7.

TEXT
Un string con longitud máxima de 256, consistiendo de parame tros de mensaje usados para sustituir texto en el mensaje de error asociado con la condición. Este campo es indefinido - si el campo NORAML es TRUE.

Creación de una Variable Tipo Estado
El siguiente ejemplo crea una variable tipo estado; llama a un comando de SCL; y, dependiendo del resultado del comando, condicionalmente ejecuta un conjunto de estatutos:

```
create_variable  name=status_local  kind=status
attach_file  file=$USER. libreria_datos  status=status_local
IF NOT status_local.normal THEN
    .
    .
  "Realiza procesamiento de error."
    .
    .
IFEND
```

En el ejemplo, la siguiente actividad toma lugar:
1. El comando CREATE_VARIABLE crea una variable de estado - llamada STATUS_LOCAL.
2. Un comando ATTACH_FILE produce un requerimiento de aso-- ciación para el archivo LIBRERIA_DATOS en el catalogo -- maestro.
3. Debido a que se incluyó el parametro STATUS, no se noti- ficará de algún error al procesar el comando ATTACH_FILE. Sin embargo, se puede examinar el valor del campo NORMAL

de la variable de estado STATUS_LOCAL. Este examen produce la siguiente información:

- Si el valor es TRUE, el comando fue ejecutado propiamente y el estatuto IF no se ejecutaría.
- Si el valor es FALSE, ocurrió un error al intentar -- procesar el comando ATTACH_FILE y el estatuto IF se - ejecutaría.

. Haciendo Pruebas para Condiciones Especificas de Error También se puede controlar el procesamiento de error, -- probando para condiciones especificas de error. El campo CONDITION de la variable de estado puede contener un código de condición único para cada diagnóstico que - - NOS/VE produce. Además, cada código de condición es asociado con un nombre identificador de condición única.

Usando la función $CONDITION_NAME descrita en el capítulo 7, se puede comparar una representación de string de un nombre de condición con cualquier nombre de condición específica y controlar el procesamiento de error, basado en el resultado de la función. Los códigos de condición y los nombres de condición son descritos en el manual de NOS/VE Mensajes de Diagnóstico.

NOTA:

El procesamiento de error no ocurre cuando un error se - localiza durante la evaluación de los parametros de co-- mandos. Para captar el estado de un comando con errores de parametros, usar el comando INCLUDE_COMMAND especificando un parametro STATUS. Como ejemplo, ver el comando INCLUDE_COMMAND posteriormente en este capítulo.

o Manipulación Condicionada

La manipulación condicionada se activa cuando existe una condición para un comando que no contiene el parametro STATUS. En - este caso, uno de los siguientes eventos ocurre:

- Un job batch que no tiene definido un manipulador condi-- cionado para la condición que se presente es terminado.
- Un job interactivo que no tiene definido un manipulador condicionado para la condición que se presente termina el comando con un mensaje y da un prompt esperando el siguiente comando.
- Un manipulador condicionado que halla sido definido para la condición es activado para procesar la condición.

La severidad de una condición determina si el manipulador condicionado sea activado. La manipulación condicionada toma lugar como sigue:

Nivel de Severidad	Manipulación Condicionada
INFORMATIVE	La manipulación condicionada no se activa, el comando que produce la condición termina sin abortar, aunque SCL escribe el mensaje en el archivo $RESPONSE.
WARNING	Igual que para INFORMATIVE.
NONSTANDARD	Igual que para INFORMATIVE.
DEPENDENT	Igual que para INFORMATIVE.
ERROR	La manipulación condicionada es activada.
FATAL	Igual que para ERROR.
CATASTROPHIC	Igual que para ERROR.

Para más información acerca de niveles de severidad, ver el manual de NOS/VE MENSAJES DE DIAGNOSTICO.

Las condiciones que pueden ser probadas son definidas por categorías de condición. Cada categoría es identificada por un nombre de condición como sigue:

Nombre de la Condición	Descripción de la Condición
PROGRAMA_FAULT	Ejecución de Programas que terminaron -- con error. Esto incluye a los progra-- mas provistos por el sistema (por ejem-- plo COBOL o SORT) o sus programas.
LIMIT_FAULT	Cuando se alcanza el límite de recursos de un job.
COMMAND_FAULT	El interprete de SCL o un comando de utilería halla detectado un error de sintaxis o semántica en un comando o estatuto.
ANY_FAULT	Incluye cualquiera de las condiciones de fallas PROGRAMA_FAULT, LIMIT_FAULT, - - COMMAND_FAULT.
INTERRUPT	Pausa (interupción desde la terminal) indicada.

Establecimiento de un Manipulador Condicionado (WHEN/WHENEND)
Un manipulador condicionado consiste de los siguientes componentes:

- Estatuto WHEN que establece la condición o condiciones.
- Lista de estatutos que son procesados cada vez que ocu- - rren las condiciones especificas.

Si las condiciones indicadas en el estatuto WHEN ocurre, la lista subsecuente de estatutos es ejecutada. Si se establece mas de una manipulación condicionada, en aquella del block

actual o la mas reciente ocurrencia toma precedencia. Cuando un block cesa de existir, cualquier estatuto WHEN en ese - - block se convierte en indefinida.

Cuando una de las condiciones ocurren, dos variables son inicializados:
OSV$STATUS y OSV$COMMAND_NAME

Variable	Descripción
OSV$STATUS	Contiene una copia del registro de estado establecido por el programa que determina la condición.
OSV$COMMAND_ NAME	Contiene una variable string cuyo valor es el nombre de el comando que fue procesado cuando la condición ocurrió.

El estatuto WHEN se activa, y la lista de estatutos pueden entonces interrogar esta variable para determinar la condición de error e iniciar la acción apropiada.

Si alguna de las condiciones indicadas en el estatuto WHEN -- ocurren cuando la lista de estatutos en el WHEN son procesa-- dos, el block que define al estatuto WHEN es terminado con un estado de anormal.

Las variables creadas dentro del block WHEN son locales al -- block y son liberadas cuando el control regresa del estatuto WHEN.

Para salir del block WHEN se hace por medio del estatuto CONTINUE. Si no se encuentra algún estatuto CONTINUE, la salida ocurre después de que el último estatuto en el block WHEN es procesado.

Lo que sucede cuando se sale de un block WHEN depende de los siguientes factores:

- La condición que ocasionó al estatuto WHEN ser activado.
- Si la opción RETRY se especifica en el estatuto CONTINUE. Para mayor información, leer Saliendo de un Block WHEN en la siguiente sección.

El siguiente ejemplo ilustra cómo establecer un manipulador condicionado. En este caso, los estatutos posteriores a la - clausula WHEN son ejecutadas si se detecta una condición de - tiempo límite.

```
WHEN limit-fault DO
   put-Line  'Incr    ıtando el Tiempo Límite en 60 C.P. --
             segu     '.
   change_job_limit  name=cp_time..
   resource_limit= ($JOB_limit(cp_time, acumulator)+60)
CONTINUE RETRY
WHENEND
```

Saliendo de un Block WHEN (CONTINUE)
El estatuto CONTINUE permite la salida de un block WHEN en - el cual aparece.

Si la expresión booleana en la clausula WHEN es TRUE, el - - block WHEN finaliza. Si la expresión es FALSE, el block - - WHEN no es finalizado. Si se omite la clausula WHEN, el - - block es finalizado.

Lo que suceda durante la salida depende de los siguientes -- factores:
. La condición que ocasionó al estatuto WHEN ser activado.
. Si se especifico RETRY en el estatuto CONTINUE.

Las siguientes descripciones ilustran el proceso de salida - cuando se especifica la opción RETRY en el estatuto CONTINUE; también se aplican a la acción tomada en la ausencia del estatuto CONTINUE:

COMMAND_FAULT o PROGRAM_FAULT, sin RETRY.
El proceso continua en el estatuto posterior al que provocó ser activado el estatuto WHEN.
COMMAND_FAULT o PROGRAMA_FAULT, con RETRY.
El estatuto que provocó al estatuto WHEN ser activado vuelve a reprocesar.
INTERRUPT o LIMIT_FAULT, sin RETRY.
El proceso continua en el punto de la interrupción.
INTERRUPT o LIMIT_FAULT, con RETRY
Los resultados son indefinidos.

Como un ejemplo del uso del estatuto WHEN, ver la sección anterior (Establecimiento de un Manipulador Condicionado). En ese ejemplo, el comando CHANGE_JOB_LIMIT es ejecutado cada - vez que el comando provoque una condición de LIMIT_FAULT.

El siguiente ejemplo establece un manipulador condicionado - para una interrupción de terminal (pausa). Al usuario le -- aparece un prompt para varias opciones cuando ocurra una condición de INTERRUPT.

```
WHEN  interrupt DO
   create_variable  name=response  king=string
   LOOP
       accept_line  variable=response..
       prompt='Teclee RETRY, CONTINUE, o comando: '..
       input=input
       IF $translate (itu, response)= 'RETRY' THEN,
          CONTINUE, RETRY
       ELSEIF $string ($name(response))= 'CONTINUE' THEN
          CONTINUE
       ELSE
          include_line  variable=response
       IFEND
   LOOPEND
WHENEND
```

Cancelación de una Condición (CANCEL)
Para cancelar una o más selecciones de condición hechas con un estatuto WHEN, se usa el estatuto CANCEL. El WHEN más reciente con la selección de condición es cancelada.

El siguiente ejemplo cancela la condición de LIMIT_FAULT establecida en el WHEN anterior:

```
/ cancel  limit_fault
```

Varias condiciones pueden cancelarse al mismo tiempo, como se muestra en el siguiente ejemplo:

```
/ cancel  command_fault  program_fault
```

Lectura de Líneas de Archivo y Escritura de Líneas a un Archivo:

Los comando estan provistos para leer una línea de un archivo y dejarla en una variable string y para escribir una variable string en un archivo.

- Lectura de Líneas de un Archivo (ACCEPT_LINE)
  Para leer líneas de un archivo y dejarlas en una variable string, se usa el comando ACCEPT_LINE. Se debe crear la variable string antes de indicar el comando ACCEPT_LINE.

  Para garantizar que un comando ACCEPT_LINE lee de una terminal, no usar el archivo estandar (como $INPUT) para el parametro IN-- PUT. En su lugar, usar un archivo tal como el archivo de job -- INPUT. Esto garantiza que la lectura ocurre de una terminal en lugar de otro archivo en blanco conectado al archivo $INPUT. -- También asegura que el prompt es escrito en la terminal.

  En el ejemplo siguiente, el procedimiento WRITE_FILE_TO_OUTPUT - lee cada línea del archivo INFILE dejandola en una variable - - string llamada INPUT_STRING hasta que se alcance el fin de información.

  Por omisión, el ACCEPT_LINE realiza una asociación automática antes de leer una línea y una desasociación automática después de leer una línea (llamado asociación y desasociación automática). Sin embargo, en el ejemplo, el procedimiento WRITE_FILE_TO_OUT-- PUT elimina la necesidad de esto al usar ATTACH_FILE para aso- - ciar al archivo explícitamente. La posición de archivo $ASIS -- mantiene la posición correcta para cada ejecución del comando -- ACCEPT_LINE. Después de que el archivo es leído, es explícita-- mente desasociado con el comando DETACH_FILE.

  Se asume que el archivo INFILE contiene lo siguiente:

```
123
  456
    789
```

El archivo WRITE_FILE_TO_OUTPUT tiene el siguiente contenido:

```
PROC write_file_to_output (
   input, i       :    file=$required
   output, o      :    file: output
   status         :    var of status = $optional
  )

   create_variable  input_string  kind=string
   create_variable  input_count  kind=integer
   attach_file  $value (input)
   LOOP
      accept_line  variable=input_string..
      input=$fname ($string($value(input))//'.$asis')..
      prompt='  >>>?'line_count=input_count
      EXIT WHEN  input_count=Ø
      display_value  input_string
   LOOPEND
   detach_file  $value (input)
PROCEND write_file_to_output
```

Cuando se ejecuta el procedimiento, se produce la siguiente salida:

```
123
  456
    789
```

o Escritura de Líneas a un Archivo (PUT_LINE)
Para escribir líneas de un string en un archivo, se usa el comando PUT_LINE.

El siguiente ejemplo escribe un mensaje de tres líneas al archivo OUTPUT.

```
/put_lines  lines=(..
../' Dia de hoy: '//$date(month)..
../' Hora actual: '//$time(ampm)..
../' Bienvenidos a NOS/VE.')
```

```
Dia de Hoy:    March28, 1989
Hora actual:   2:45 PM
Bienvenidos a NOS/VE.
```

Inserción de Archivos o String en un Grupo de Comandos:

SCL proporciona comandos que insertan (incluyen) un archivo o string en un grupo de comandos. Los siguientes comandos son descritos en esta sección.

```
INCLUDE_FILE
INCLUDE_LINE
INCLUDE_COMMAND
```

- Inserción de Archivos (INCLUDE_FILE)
  Para hacer que un archivo texto que contiene estatutos de SCL -- sea lógicamente reemplazado en el mismo comando, se usa el comando INCLUDE_FILE. Los comandos en el archivo indicado son procesados en el mismo contexto que en el comando INCLUDE_FILE. Esto es, el comando tiene acceso a las mismas variables y condiciones, así como también la sustitución de parametros.

  Para comprender el siguiente ejemplo, asumamos que el archivo -- PROLOG contiene la siguiente lista de estatutos:

```
display_value 'Inicio de Sesión a las' //$time(ampm)//'de'//..
$date(month)//'.'
```

  Al dar el siguiente comando INCLUDE_FILE se genera el resultado indicado:

```
/
Inicio de Sesión a las 3:21 PM de March 28, 1989.
```

- Inserción de Líneas (INCLUDE_LINE)
  Para hacer que un texto de un string sea lógicamente reemplazado en el mismo comando, se usa el comando INCLUDE_LINE. Los comandos en el string indicado son procesados en el mismo contexto que en el comando INCLUDE_LINE. Esto es, el string tiene acceso a las mismas variables, condiciones y sustitución de parametros como el comando INCLUDE_LINE.

  El comando permite construir una línea de uno o mas estatutos mediante la manipulación del string y posteriormente procesarla.

  El siguiente ejemplo utilica la concatenación de strings para -- crear una línea y luego procesar la línea con el comando INCLUDE_LINE:

```
/hora='$time(ampm)'
/fecha='$date(month)'
/lista_de_comandos='display_value ('//hora//' '//fecha//')'
/include_line  lista_de_comandos
 3:26 PM
 March 28, 1989
```

  Se pude usar el comando INCLUDE_LINE para controlar cualquier -- error de sintaxis encontrado durante la evaluación de los parametros del comando. Por ejemplo, si se indican los siguientes comandos, se despliega el error mostrado:

```
/create_variable  n=s1  k=status
/display_catalog  c=3  status=s1
--ERROR-- Expecting FILE value, Found integer for parameter C.
Enter correct value for parameter CATALOG:
```

  Al indicar 3 para el parametro C en el comando DISPLAY_CATALOG - se genera un error de sintaxis. Como el error ocurre antes de - que el comando sea ejecutado, la variable de estado no se establece.

El siguiente ejemplo demuestra el uso del comando INCLUDE_LINE - para controlar los errores de sintaxis:

```
/create_variable  n=incl_status  k=status
/include_line 'display_catalog  c=3'  status=incl_status
/display_value ('Ocurrio el siguiente error:' incl_status)
Ocurrio el siguiente error:
--ERROR-- Expecting FILE value, found INTEGER por parameter C.
```

- Inserción de Comandos (INCLUDE_COMMAND)

  Para construir un comando a través de la manipulación de strings y luego procesar el comando, usar el comando INCLUDE_COMMAND. El comando causa que el texto del string indicado sea logicamente - reemplazado en la ocurrencia del comando INCLUDE_COMMAND. El comando en el string indicado es procesado en el mismo contexto -- que en el comando INCLUDE_COMMAND. Esto es, el comando en el -- string tiene acceso a las mismas variables, condiciones y sustitución de parametros como en el comando INCLUDE_COMMAND.

  Tomar conciencia de las siguientes distinciones entre el comando INCLUDE_LINE y el comando INCLUDE_COMMAND:

  . El comando INCLUDE_LINE permite procesar múltiples comandos - y estatutos de control. Sin embargo, el comando INCLUDE_CO-- MMAND acepta unicamente un solo comando.
  . El comando INCLUDE_LINE crea un block de control de entrada; esto es, trata a la lista de estatutos como si fuera una lista de estatutos de un estatuto BLOCK sin etiqueta. El comando INCLUDE_COMMAND no lo hace.

  El siguiente ejemplo crea un string que será interpretado como - un comando. Luego, procesa el comando con el comando INCLUDE_ COMMAND.

```
/hora='$time(ampm)'
/comando='display_value ('//hora//')'
/include_command  command=comando
11:41 AM
```

Suspensión del Procesamiento de Comandos (WAIT)

Para suspender el procesamiento de comandos dentro de un job hasta que halla transcurrido un periodo de tiempo, o hasta que uno o más eventos indicados hallan tomado lugar, utilizar el comando WAIT.

Por ejemplo, para hacer que un job espere por 20,000 milisegundos, indicar la siguiente forma del comando:

```
/wait 20000
```

Alternativamente, se puede llevar a cabo el mismo resultado indi-- cando la siguiente forma del comando WAIT:

```
/wait  20*1000
```

Cambiando el Medio Ambiente del Sistema:

SCL contiene estatutos que permiten cambiar termporalmente ciertos ambientes del sistema dentro de procedimientos. Estos son los estatutos PUSH y POP.

- Establecimiento de un Nuevo Medio Ambiente (PUSH)
  Para establecer una nueva versión del medio ambiente del sistema, se usa el estatuto PUSH. El medio ambiente más recientemente establecido permanece en efecto hasta que sea removido con un estatuto POP o cuando se termine el block en el que el estatuto PUSH fue ejecutado.

  Cuando un objeto del medio ambiente es afectado por el estatuto PUSH, su valor es copiado de la versión previa. Se puede revisar o interrogar con comandos de SCL, estatutos, y/o funciones específicas para ese objeto del medio ambiente. Unicamente la versión ma recientemente proporcionada (push) del medio ambiente del sistema puede ser referenciada o cambiada.

  El siguiente ejemplo ilustra un procedimiento que cambia la lista de comandos del medio ambiente. Este procedimiento utiliza la librería de comandos. AJL.COMMAND_LIBRARY únicamente durante su ejecución. Después de terminado el procedimiento, ya no se tendrá la librería de comandos en la lista de comandos.

```
PROC  cambia_medio_ambiente
    PUSH  command_list
    set_command_list  a=.ajl.command_library
      :
      :
     "Ejecución de Comandos localizados en .ajl.command_li--
      brary."
      :
      :
PROCEND
```

- Restauración del Medio Ambiente Anterior (POP)
  Para borrar la versión actual del medio ambiente del sistema y restaurar los cambios del medio ambiente a su valor anterior, usar el estatuto POP.

  Esta operación de eliminación es realizada automáticamente cuando el programa termina el block en el cual el estatuto PUSH correspondiente fue ejecutado (ver el ejemplo en la sección anterior).

DESCRIPCIONES DE FORMATOS DE COMANDOS Y ESTATUTOS.

Esta sección describe el formato y parametros de cada comando y estatuto cuyo uso es descrito anteriormente, en este capítulo. Los comandos y estatutos son presentados alfabéticamente como sigue:

ACCEPT_LINE
BLOCK/BLOCKEND
CANCEL
CONTINUE
CREATE_VARIABLE
CYCLE

DELETE_VARIABLE
DISPLAY_VARIABLE
DISPLAY_VARIABLE_LIST
EXIT
EXIT_PROC
FOR_FOREND
IF/IFEND
INCLUDE_COMMAND
INCLUDE_FILE
INCLUDE_LINE
LOOP/LOOPEND
POP
PUSH
PUT_LINE
REPEAT_UNTIL
WAIT
WHEN/WHENEND
WHILE/WHILEND

ACCEPT_LINE:

Propósito: Lee una línea de un archivo en una variable - -

Formato:
ACCEPT_LINE ó
ACCEPT_LINES ó
ACCL
VARIABLE=variable string
INPUT=archivo
PROMPT=string
LINE_COUNT=variable entera
STATUS=variable de estado

Parametros:

VARIABLE (I)
Variable string en la cual la línea será leída y su valor estará contenido en esta variable. Lo variable puede ser un arreglo de strings. -- Este parámetro es requerido.

INPUT (I)
Archivo del cual será leída la línea. Este parámetro es requerido.

PROMPT (P)
String que se usa para desplegarse en la terminal. El prompt por omisión es la palabra SUPPLY, seguido por el nombre de la variable que contendrá la línea. Si el archivo indicado por el parámetro INPUT no esta asignado a la terminal, - entonces no se usa el prompt.

LINE_COUNT (LC)
Variable entera que recibe un contador de líneas leídas por el comando ACCEPT_LINE.

STATUS
Condición de terminación del comando.

Comentarios:

- Para garantizar que el comando ACCEPT_LINE lea desde la terminal, no utilizar el archivo standard (como $INPUT) para el parámetro INPUT. En su lugar, usar un archivo como - el archivo de job INPUT. Esto garantiza -- que la lectura ocurra desde la terminal en lugar de un archivo conectado al archivo -- $INPUT. También asegura que el prompt sea escrito a la terminal.
- Para usar el comando ACCEPT_LINE y leer líneas sucesivas de un archivo:
  - Asociar el archivo usando el comando - - ATTACH_FILE.
  - Especificar la posición de abierto $ASIS en la referencia del archivo para el parámetro INPUT del comando ACCEPT_LINE.
  - Utilizar el parámetro LINE_COUNT para determinar cuando el fin de archivo sea alcanzado.
  - Desasociar el archivo usando el comando DETACH_FILE.

  Si no se asocia explícitamente el archivo, la posición de abierto $ASIS es ignorada y el archivo es accesado desde el principio. Ver los siguientes ejemplos.
  El siguiente ejemplo lee líneas sucesivas de un archivo:

```
create_variable n=contador_lineas k=integer
create_variable n=lineas k=string d=1..100
attach_file f=$user.algun_archivo
REPEAT
 accept_line v=lineas i=$user.algun_archivo.$asis..
   Ic=contador_lineas
 FOR i=1 to contador_lineas DO
  "procesar la línea en la variable lineas(i)"
 FOREND
UNTIL contador_lineas  $variable(lineas,frontera_mayor)
detach_file f=$user.algun_archivo
```

  Usando una variable arreglo para leer información es más eficiente que utilizar una variable elemental, desde que el archivo es - abierto un número menor de veces.

BLOCK/BLOCKEND

Propósito: Agrupar una secuencia de estatutos en un block.

Formato:

```
etiqueta: BLOCK
              Lista de estatutos
          BLOCKEND etiqueta
```

Parametros:

etiqueta
Nombre del block. La etiqueta puede ser usada por los estatutos EXIT dentro de un block.

lista de estatutos
Toda una lista de estatutos que residen en un - block.

Comentarios: Para salir de un block se lleva a cabo ya sea - completando la ejecución del último estatuto de la lista de estatutos ó a través de la transferencia de control explícita por medio del estatuto EXIT.

CANCEL:

Propósito: Cancelar la condición más recientemente selec-- cionada.

Formato:

```
CANCEL
   Condición
```

Parametros:

condición
Una ó más selecciones de condición a ser canceladas. Los múltiples nombres de condición son separados por coma o un espacio. Este paráme-- tro es requerido. Los siguientes son nombres - de condición válidos:

```
PROGRAM_FAULT
LIMIT_FAULT
INTERRUPT
COMMAND_FAULT
ANY_FAULT
```

Comentarios: SCL ignora cualquier intento de cancelar una -- condición no seleccionada.

CONTINUE:

Propósito: Salir del estatuto WHEN actual.

Formato:

```
CONTINUE
   RETRY
   WHEN expresión booleana
```

Las siguientes son formas válidas del estatuto CONTINUE:

```
CONTINUE
CONTINUE RETRY
CONTINUE WHEN expresión booleana
CONTINUE RETRY WHEN expresión booleana
```

Parametros:

RETRY
Especificación para instruir a SCL de regresar el control al estatuto que provocó la condi- - ción. Si este parámetro se omite, el control regresa al estatuto siguiente al estatuto que ocasionó al estatuto WHEN.

expresión booleana
Especificación que indica si la salida se llevará a cabo. Si la expresión es TRUE, la sali da se lleva a cabo. Si la expresión es FALSE,

el proceso continua en el siguiente estatuto. Si se omite la clausula, la siguiente salida - se lleva a cabo.

Comentarios: Las siguientes descripciones ilustran el proceso de salir con un estatuto RETRY; también se aplican a la acción tomada en la ausencia del estatuto CONTINUE:

COMMAND_FAULT o PROGRAM_FAULT, sin RETRY
El proceso continua en el estatuto siguiente - al que ocasionó que se activará el estatuto -- WHEN.

COMMAND_FAUTL o PROGRAM_FAULT, con RETRY
El estatuto que ocasionó al estatuto WHEN ser activado es reprocesado.

INTERRUPT o LIMIT_FAULT, sin RETRY
El proceso continua en el punto de la interrupción.

INTERRUPT o LIMIT_FAULT, con RETRY
Estos resultados son indefinidos.

CREATE_VARIABLE:

Propósito: Crear variables de SCL

Formato:
```
CREATE_VARIABLE ó
CREATE_VARIABLES ó
CREV
   NAME=lista de nombres
   KIND=keyword o (STRING, integer)
   DIMENSION=rango de enteros
   VALUE=expresión de parametros
   SCOPE=keyword o name
   STATUS=variable de estado
```

Parametros:

NAME (NAMES o N)
Nombre(s) de la(s) variable(s) a ser creada(s) El parámetro es requerido.

KIND (K)
Tipo o clase de variable a ser creada. Las - siguientes son clases válidas para el parámetro KIND:

INTEGER
Crea una variable entera

BOOLEAN
Crea una variable bolleano

STRING
Crea una variable string

STATUS
Crea una variable de estado.

Si no se indica un valor de clase para este parámetro, se asume que es de clase INTEGER.
El elemento entero del parámetro KIND especifica la longitud máxima para la variable string; no puede ser indicada para otras clases de variables. Si el elemento entero es omitido para una variable string, la longitd máxima se asume de 256.

DIMENSION (D)
Los límites inferior y superior de un arreglo. Este parámetro es usado unicamente cuando la variable creada es un arreglo. Se puede indicar un rango desde -2,147,483,647 a 2,147,483,647. Si se omite el parámetro DIMENSION, el sistema asume que la variable no es un arreglo (esto es, una dimensión de 1..1 es asumido).

VALUE (V)
El valor inicial de la variable a ser creada. - Se puede inicializar una variable de estado con el parámetro VALUE mediante la función $STATUS o igualando a otra variable de estado.
Si se omite el parámetro VALUE, se asignan los siguientes valores por omisión:

Clase de Variable	Valor por Omisión
Integer	Ø
String	Espacios (string con longitud de cero)
Boolean	FALSE
Status	Campo Normal es TRUE Campo IDENTIFIER es indefinido Campo CONDITION es indefinido Campo TEXT es indefinido

SCOPE (S)
Alcance de la variable. El alcance de una variable define los blocks en los cuales la variable puede ser accesada. Los siguientes son valores válidos para el parámetro SCOPE:

LOCAL
Hace que las variables sean local al block actual. Estas variables no pueden ser referenciadas por otros blocks. Este es el valor por omisión.

XDCL
Hace que las variables sean externamente declaradas. (XDCL). Una variable con este alcance - puede referenciarse por otros blocks si una variable con idénticos parametros KIND y DIMENSION es creada con alcance de XREF. Los atributos -

de una variable KIND y DIMENSION pueden determinarse a través de la función $VARIABLE.

XREF
Afirma que las variables son externamente referenciadas (XREF). Una variable con este alcance tuvo que ser creada en un block de fuera con alcance de XDCL y con los parametros KIND y DIMENSION idénticos a aquellos proporcionados por este comando. Los atributos de variables KIND y DIMENSION pueden determinarse a traves de la función $VARIABLE.

JOB
Hace que las variables sean creadas en un block de job con alcance XDCL. Si el block actual no es el block del job, se hace una declaración XREF en el block actual. Una variable con alcance de JOB es implícitamente accesible desde comandos de SCL al mismo nivel, desde dentro de un ambiente de utilería, desde dentro de un block, y desde dentro de otra tarea (task). Sin embargo, un alcance de JOB no permite que una variable sea accesada implícitamente desde dentro de un procedimiento de SCL. Para accesar una variable con alcance de job desde un procedimiento de SCL, se debe crear una variable dentro del procedimiento con alcance XREF.

name(nombre)
Hace que las variables sean creadas en el block de utilería indicados por el nombre con alcance de XDCL. Si el block actual no es el block de utilería, se hace una declaración XREF en el block actual.

STATUS
Condición de terminación del comando.

Comentarios: Normalmente, una variable es local al block en el cual es creada. Esto es, otros blocks pueden accesar la variable únicamente si es pasada a ellos como parámetro. Se puede dar una variable con alcance XDCL, la cual permitiría a otros blocks creados dentro del block en el cual la variable existe para accesar la variable. El block que accesa la variable debe crear la variable con un alcance de XREF.

CYCLE:

Propósito: Permite la ejecución de la siguiente iteración (si es que hay alguna) de un estatuto repetitivo.

Formato:

```
CYCLE
   etiqueta
   WHEN expresión booleana
```

Las siguientes son formas válidas del estatuto CYCLE:

```
CYCLE
CYCLE etiqueta
CYCLE WHEN expresión booleana
CYCLE etiqueta WHEN expresión booleana
```

Parametros:

etiqueta
Etiqueta asociada con el estatuto repetitivo de terminación ha ser vuelto a procesar. Si se -- omite la etiqueta, el estatuto repetitivo mas - anidado es el que vuelve a procesar.

expresión booleana
Especificación que indica si el estatuto asigna do debería ser vuelto a procesar. Si la expresión es TRUE, se ejecuta el siguiente ciclo. Si la expresión es FALSE, o si es omitida, el siguiente ciclo no se ejecuta.

Comentarios:

- En caso de un estatuto LOOP, el control se - le da al primer estatuto de la lista de estatutos del LOOP.
- En caso de los estatutos WHILE o REPEAT, se evalua la expresión booleana que controla el ciclo.
- En el caso del estatuto FOR, el proceso realizado entre el FOR y FOREND se ejecuta.
- La clausula WHEN es evaluada en el punto de referencia en lugar de continuamente, como - se realiza por el manipulador condicionado - WHEN.

DELETE_VARIABLE:

Propósito:

Borrar las variables declaradas desde el block actual.

Formato:

```
DELETE_VARIABLE ó
DELETE_VARIABLES ó
DELV
   NAME=lista de nombres
   STATUS=variable de estado
```

Parametros:

NAME(N)
Nombre(s) de la(s) variable(s) cuya(s) declaracion(es) serán removidas. Este parámetro es - requerido.

STATUS
Condición de Terminación del comando.

DISPLAY_VARIABLE_LIST:

Propósito: Desplegar las variables que actualmente pueden ser accesibles al job.

Formato:
```
DISPLAY_VARIABLE_LIST ó
DISVL
   OUTPUT=archivo
   STATUS=variable de estado
```

Parametros:

OUTPUT (O)
Nombre del archivo que contendrá la lista de variables. El archivo por omisión es el $OUTPUT.

STATUS
Condición de terminación del comando.

Comentarios: La variable más recientemente creada aparece en la cima de la lista.

EXIT:

Propósito: Transferir el control fuera del estatuto estructurado, procedimiento de comandos, o utilería de comandos.

Formato:
```
EXIT
   designador
   WHEN expresión booleana
   WITH expresión de estado
```

Los siguientes son formatos válidos del estatuto EXIT:

```
EXIT
EXIT designador
EXIT WHEN expresión booleana
EXIT designador WHEN expresión booleana
EXIT designador WITH expresión de estado.
EXIT WHEN expresión booleana, WITH expresión de estado
EXIT designador WHEN expresión booleana - -
WITH expresión de estado
```

Parametros;

designador
Designa las etiquetas de estatutos estructurados, procedimiento de comandos, o utilería de comandos del cual saldrá (terminar ejecución).

etiqueta
Etiqueta asociada con el estatuto estructurado. Si se indica una etiqueta, el estatuto estructurado con dicha etiqueta deja de procesarse (se da EXIT). Si se omite la etiqueta, el estatuto estructurado mas anidado es al que se da EXIT.
No se puede usar la clausula WITH con estatutos estructurados.

nombre de procedimiento
Nombre de un procedimiento activo. Este nombre debe ser el primer nombre en la lista de nombres en el encabezado del procedimiento. - Para salir de un procedimiento usar las palabras especiales PROCEDURE o PROC.

nombre de utilería
Nombre de una utilería activa. Para salir de la utilería activa mas anidada, usar la palabra UTILITY.

expresión booleana
Especificación que indica si el estatuto será terminado (dar EXIT). Si la expresión es - - TRUE, el estatuto es terminado. Si la expresión es FALSE, o si es omitida, el estatuto - no se termina.

expresión de estado
Especificación que indica la condición de estado, bajo la cual se llevará a cabo el EXIT. Esta expresión debe ser un valor de estado -- del procedimiento. Este estado se convierte en el estado de terminación del procedimiento indicado. Si se omite la clausula WITH al salir de un procedimiento o utilería, ocurre -- una terminación con estado normal.

Comentarios:

- La clausula WHEN es evaluada en el punto de - referencia. Esto difiere de la continua evaluación por el manipulador condicionado WHEN.
- Se pueden usar las clausulas WHEN y WITH ya - sea juntos o separadamente. Si se usan juntos, se puede listar cualquier clausula primero.
- Unicamente los estatutos estructurados que se contienen a si mismo se les puede dar EXIT. - Por ejemplo, unicamente estatutos estructurados sin referencia de archivos fuera de lo común se les puede dar EXIT.
- Cualquier procedimiento de comandos o utilería activos se les puede dar EXIT.

EXIT_PROC:

Propósito: Salir de un procedimiento. La ejecución continua en el siguiente estatuto después del llamado al procedimiento.

Formato:

```
EXIT_PROC
    WITH expresión de estado (status)
    WHEN expresión booleana
```

Los siguientes son formatos válidos del estatuto EXIT_PROC:

```
EXIT_PROC
EXIT_PROC WITH expresión de estado (status)
EXIT_PROC WHEN expresión booleana
```

Parametros:

expresión de estado (status)
Estado o resultado que será regresado al block que mando llamar al procedimiento. El valor -- por omisión (default) es un estado normal.

expresión booleana
Expresión que indica si terminará o no la ejecución de un procedimiento determinado. Si la expresión es TRUE o se omite, la salida o termino del procedimiento toma efecto. Si la expresión es FALSE, la salida o termino del procedimiento no toma efecto.

Comentarios:

- Para usar la clausula WITH, no es necesario o requerido un parámetro de estado en la definición del procedimiento.
- La cláusula WHEN es evaluada en el punto de referencia mas bien que continuamente, como se hace en el manipulador condicionado WHEN.

FOR/FOREND:

Propósito:

Proporciona una repetición controlada de una -- lista de estatutos.

Formato:

```
etiqueta: FOR
   variable=inicial TO final BY step DO
   lista de estatutos
   FOREND etiqueta
```

Parametros:

etiqueta
Nombre con el que se reconoce el block FOR. La etiqueta puede ser utilizada por los estatutos CYCLE o EXIT dentro del block.

variable
Variable de control del ciclo FOR. Si aun no ha sido declarado, esta variable es creada por el estatuto FOR. En este caso, la variable no puede contener un indice o campo calificador. Este parámetro es requerido.

inicial
Número entero que indica el valor inicial de la variable de control. Este parámetro es requerido.

final
Expresión entera que indica el valor final de - la variable de control. Este parámetro es re-- querido.

step
Expresión entera que indica el incremento de -- la variable de control para cada ciclo. El incremento por omisión es 1. Si se omite el elemento step, el estatuto FOR tiene el siguiente formato:

```
etiqueta: FOR vairable= inicial TO final DO
```

lista de estatutos
Conjunto de estatutos que residen en el block.

Comentarios:

- Cualquier alteración a la variable de con-- trol, elemento inicial, elemento final o al step dentro de la lista de estatutos no tiene efecto en el número de iteraciones del estatuto FOR.
- Si el valor inicial es menor al valor final, el step debe ser positivo, de otro modo el - estatuto FOR no se ejecuta.
- Si el valor inicial es mayor al valor final, el step debe ser negativo; de otro modo el - estatuto FOR no se ejecuta.
- Si el valor inicial es igual al valor final o step es igual a cero, la lista de estatu-- tos se ejecuta una vez..
- Si la variable de control es alterada y luego referenciada dentro de la lista de esta-- tutos, el valor alterado es obtenido (la variable de control, sin embargo, es reestablecida a su valor correcto en la siguiente iteración del estatuto FOR).
  Después tiene el últimov valor que alcanzó - (ya sea a través de la iteración del estatuto FOR o por alteración).

IF/IFEND:

Propósito: Proporciona la ejecución condicionada de una o más listas de estatutos.

Formato:

```
IF expresión booleana THEN
   lista de estatutos
IFEND
```

El formato para el estatuto IF puede ser expandido como sigue para proporcionar la ejecución de una lista de estatutos según la alternativa, si la expresión booleana es FALSE:

```
IF expresión booleana THEN
   lista de estatutos

ELSE
   lista de estatutos

IFEND
```

En el formato anterior, la primera lista de estatutos es ejecutada si la expresión booleana es evaluada como TRUE. La lista de estatutos posterior a la clausula ELSE es ejecutada si la expresión booleana es evaluada como FALSE.

Una u otras más listas de estatutos pueden considerarse para su ejecución con el siguiente formato:

```
IF expresión booleana THEN
   lista de estatutos
ELSEIF expresión booleana THEN
   lista de estatutos
ELSE
   lista de estatutos
IFEND
```

En el formato anterior, cada expresión booleana es evaluada en su turno y la lista de estatutos después de la primera expresión booleana a ser evaluada se ejecuta si es TRUE. Cualquier número de clausulas ELSEIF pueden aparecer en el estatuto IF. Una clausula ELSE es permitida. Si no se indica la clausula ELSE y ninguna de las expresiones booleanas son TRUE, todas las listas de estatutos son saltados.

Parametros:

expresión booleana
Condición que debe ser TRUE para que la lista de estatutos correspondiente sea ejecutada. Este parámetro es requerido.

lista de estatutos
conjunto de estatutos a ser ejecutados.

INCLUDE_COMMAND:

Propósito:

Hace que el texto de un string específico sea reemplazado en la ocurrencia del comando INCLUDE_COMMAND por lo tanto permite construir un comando a través de la manipulación de string para su posterior ejecución.

Formato:

```
INCLUDE_COMMAND ó
INCC
   COMMAND=string
   ENABLE ECHOING=booleano
   STATUS=variable de estado (status)
```

Parametros:

COMMAND (C)
String a ser interpretado y procesado como un comando. Este parámetro es requerido.

ENABLE_ECHOING (ENABLE_ECHO o EE)
Entrada que indica si el comando será escrito en el archivo $ECHO. Si este parámetro es FALSE, el comando no se refleja en la terminal.

El valor por omisión es TRUE.

STATUS
Condición de terminación del comando.

Comentarios:

- El comando construido tiene acceso a las mismas variables, condiciones y sustitución de parametro como el comando INCLUDE_COMMAND.
- Este comando acepta unicamente un comando (a diferencia del comando INCLUDE_LINE).
- Este comando no crea un block de control de entrada (a diferencia del comando INCLUDE_LINE). Esto es, el comando incluido no es tratado como si estuviera en la lista de estatutos de un estatuto BLOCK sin etiqueta.

INCLUDE_FILE:

Propósito: Insertar un archivo texto conteniendo estatutos de SCL, en un grupo de comandos.

Formato:

```
INCLUDE_FILE o INCF
   FILE=file
   PROMPT=string
   UTILITY=name o keyword
   STATUS=variable de estado (status)
```

Parametros:

FILE (F)
Archivo tipo texto que contiene estatutos de SCL para ser incluidos. Este parámetro es requerido.

PROMPT (P)
Mensaje a usar si el archivo esta asignado a una terminal interactiva. Si usa el parámetro UTILITY para asociar una utilería con el comando INCLUDE_FILE, el prompt de la utilería es el utilizado.
Si se omite este p arámetro y no se indica una asociación con utilerías, el prompt incf se utiliza.

UTILITY (U)
Indica el nombre de la utilería a ser asociada con el comando INCLUDE_FILE. Para evitar la asociación con alguna utilería, se usa la clave NONE. Si se omite el parámetro, se asume NONE.

STATUS
Condición de terminación del comando.

Comentarios:

- El texto insertado se reemplaza logicamente en el comando INCLUDE_FILE.
- Se puede usar el comando ya sea con o sin utilería de comandos (para iniciar el procesamiento de comandos para la utilería).

- La lista de estatutos insertado es considera da como si fuera una lista de estatutos de - un estatuto BLOCK sin etiqueta. Cuando fina liza el archivo inclúido, el block deja de - existir. Consecuentemente, si se llama a -- una utilería de comandos dentro de la lista de estatutos, y se finaliza el archivo inclu ido, la utilería es terminada, aunque ningun comando de terminación explicito sea locali- zado.

INCLUDE_LINE:

Propósito: Insertar un string que contiene estatutos de -- SCL en un grupo de comandos.

Formato:
INCLUDE_LINE or INCL
STATEMENT_LIST=string
ENABLE_ECHOING=booleano
UTILITY= name o keyword
STATUS= variable de estado (status)

Parametros:

STATEMENT_LIST
String ha ser procesado como lista de estatutos. Este parámetro es requerido.

ENABLE_ECHOING (EE)
Indica si el comando será reflejado (YES) o si no lc hará (NO).
Si se omite el parámetro, se asume entonces - - YES.

UTILITY (U)
Indica el nombre de la utilería que será asocia da con el comando INCLUDE_LINE. Para evitar la asociación con una utilería, usar la clave NONE.

STAUTS
Condición de terminación del comando.

Comentarios:

- La lista de estatutos indicada se reemplaza logicamente en el comando INCLUDE_LINE.
- Se puede usar este comando ya sea con o sin utilería de comandos (para iniciar el proce- samiento de comandos para la utilería.
- La lista de estatutos insertada es considera da como si fuera una lista de estatutos de - un estatuto BLOCK sin etiqueta. Cuando fina liza el archivo incluido el block deja de -- existir. Consecuentemente, si se llama a -- una utilería de comandos dentro de la lista de estatutos, y se finaliza el archivo inclu ido, la utilería es terminada, aunque ningún comando de terminación explicita sea locali- zado.

LOOP/LOOPEND:

Propósito: Provoca la repetición ilimitada de una lista de estatutos.

Formato:
```
etiqueta: LOOP
   lista de estatutos
LOOPEND etiqueta
```

Parametros:

etiqueta
Nombre del block del LOOP. Esta etiqueta puede usarse con los estatutos CYCLE y EXIT dentro del block.

lista de estatutos
Conjunto de estatutos que residen en el block.

Comentarios: Salir de un estatuto LOOP es posible unicamente por medio de una transferencia de control explicita (esto es, por medio del estatuto EXIT).

POP:

Propósito: Borra la versión actual del objeto del medio ambiente del sistema en un procedimiento de SCL, y reestablece el objeto del medio ambiente del sistema a su valor inicial.

Formato:
```
POP
  lista de claves  (keyword)
```

Parametros:

Keyword
Uno o más componentes del medio ambiente del sistema (objetos del medio ambiente) para ser borrados. Los componentes deben estar separados por espacios o comas. Ocurre un error si el objeto del medio ambiente no fue previamente establecido (push). Este parámetro es requerido. Seleccione de la siguiente lista de objetos del medio ambiente del sistema:

```
COMMAND_LIST
FILE_CONNECTIONS
INTERACTION_STYLE
MESSAGE_LEVEL
NATURAL_LANGUAGE
PROGRAMA_ATTRIBUTES
WORKING_CATALOG
```

PUSH:

Propósito: Cambiar temporalmente cierto medio ambiente del sistema en procedimientos de SCL.

Formato:
```
PUSH
  lista de claves (keyword)
```

Parametros:

Keyword
Uno o mas componentes del medio ambiente del -- sistema (objetos del medio ambiente) ha ser cambiados (push). Los componentes deben estar separados por espacios o comas. Este parámetro - es requerido. Seleccione de la siguiente lista de objetos del medio ambiente del sistema:

COMMAND_LIST
FILE_CONNECTIONS
INTERACTION_STYLE
MESSAGE_LEVEL
NATURAL_LANGUAGE
PROGRAM_ATTRIBUTES
WORKING_CATALOG

Comentarios:

- Unicamente la versión más reciente del medio ambiente del sistema puede ser referenciada o cambiada.
- Un objeto específico puede cambiarse unica-- mente una vez en el procedimiento.

PUT_LINE:

Propósito:

Escribir líneas en un archivo.

Formato:

PUT_LINE ó
PUT_LINES ó
PUTL
LINE=lista de string
OUTPUT=file
STATUS=variable de estado (status)

Parametros:

LINE (L)
Líneas para escribirse en el archivo de salida. Este parámetro es requerido.

OUTPUT (O)
Archivo de salida en el que serán escritas las líneas. El archivo de salida por omisión es - el $OUTPUT.

STATUS
Condición de terminación del comando.

Comentarios:

- Este comando nunca agrega títulos de página o indicadores de formato. El sistema asume que el primer caracter de cada línea es un indicador de formato.
- Si se desea escribir mas de una línea en -- forma consecutiva a un archivo, usar el comando COLLECT_TEXT para una respuesta más - rápida. Este comando solo abre una sola -- vez el archivo de salida para múltiples líneas, mientras que el comando PUT_LINE abre el archivo para cada línea. Además, el comando COLLECT_TEXT permite sustituir los --

valores de las variables y expresiones tipo - string en el archivo de salida.

REPEAT/UNTIL:

Propósito: Proporcionar una repetición condicionada de una lista de estatutos.

Formato:
```
etiqueta:REPEAT
   lista de estatuto
UNTIL expresión booleana
```

Parametros:

etiqueta
Nombre del block REPEAT. Esta etiqueta puede -- usarse por los estatutos CYCLE y EXIT dentro del block.

lista de estatutos
lista de estatutos que residen en el block.

expresión booleana
Condición de terminación del estatuto REPEAT. Este parámetro es requerido.

Comentarios: Una etiqueta de termino no es permitida para el estatuto REPEAT.

WAIT:

Propósito: Suspende el procesamiento de comandos hasta que ya sea un número indicado de milisegundos hayan transcurrido, u otro evento indicado se lleve a cabo.

Formato:
```
WAIT
   TIME=integer
   TASK_NAMES= list of name
   QUEUE_NAMES= list of name
   UNTIL= keyword
   STATUS= variable de estado (status)
```

Parametros:

TIME (T)
Número de milisegundos deseados para suspender el procesamiento de comandos antes de que la se cuencia de comandos después del comando WAIT -- sean elegibles para continuar el procesamiento. Si se omite el parámetro, no ocurre suspensión.

TASK_NAME (TASK_NAMES o TN)
Nombre(s) de la(s) task(s) que deben completarse antes de que el procesamiento de comandos -- pueda continuar. La(s) task(s) indicada(s) han sido iniciadas por el requerimiento de una ta-- rea (task).

QUEUE_NAME (QUEUE_NAMES o QN)
Nombre(s) de las colas de job(s) de los cuales debe recibirse un mensaje antes de continuar el procesamiento de comandos.

UNTIL (U)
Parámetro que indica si algún o todos los eventos deben ocurrir antes de que termine el comando. Este parámetro puede continuarse inmediatamente ya sea por:

ANY
Estipula que cualquier evento indicado debe ocurrir. Este es el valor por omisión (default).

ALL
Estipula que todos los eventos indicados deben ocurrir.

STATUS
Condición de terminación del comando.

Comentarios: Este comando afecta unicamente tareas en las cuales se utilice el comando.

WHEN/WHENEND:

Propósito: Delimita la secuencia de estatutos de SCL que serán ejecutados cuando ocurra una condición específica.

Formato:
```
WHEN nombre de condición DO
   lista de estatutos
WHENEND
```

Parametros:

nombre de condición
Uno o más nombres indicando condiciones para las cuales serán procesadas una secuencia de estatutos. Múltiples nombres de condición son separados por espacios o comas. Este parámetro es requerido. Los siguientes son nombres de condición válidos:

PROGRAM_FAULT
LIMIT_FAULT
INTERRUPT
COMMAND_FAULT
ANY_FAULT

lista de estatutos
Lista de estatutos que residen en el block WHEN.

Comentarios:

- Se usa el estatuto CONTINUE para salir de un block WHEN.
- Las siguientes variables son disponibles para determinar más información acerca de la condición.

OSV $STATUS
Variable de estado inicializado por el programa que determino la condición.

OSV COMMAND_LINE
Variable string inicializada como el nombre del comando que esta siendo procesado.

WHILE/WHILEND:

Propósito: Proporciona una repetición condicionada de una lista de estatutos.

Formato:

```
etiqueta:WHILE
   expresión booleana DO
   lista de estatutos
WHILEND
   etiqueta
```

Parametros:

etiqueta
Nombre del block WHILE. Esta etiqueta puede ser usada por los estatutos CYCLE y EXIT dentro del block.

expresión booleana
Condición que debe ser TRUE para que se ejecute la siguiente lista de estatutos. Este parámetro es requerido.

lista de estatutos
Lista de estatutos que residen en el block WHILE.

Comentarios: La evaluación de la condición booleana toma lugar antes de que cada iteración de la lista de estatutos.

6.- CREACION DE PROCEDIMIENTOS EN SCL

CREACION DE PROCEDIMIENTOS 6-1

LLAMADO DE PROCEDIMIENTOS 6-2
Indicando el Nombre del Archivo 6-2
Indicando el Nombre del Procedimiento 6-3
Ambientes para el llamado de Procedimientos..... 6-3

ESCRIBIENDO A UN ARCHIVO DESDE UN PROCEDIMIENTO 6-4

EJEMPLO DE UN PROCEDIMIENTO 6-4

FORMATO DE PROCEDIMIENTO 6-6
Formato para el Encabezado del Procedimiento 6-6
Formato de la Lista de Estatutos 6-7
Formato de Terminación del Procedimiento 6-7

DEFINICION DE LOS PARAMETROS DE UN PROCEDIMIENTO 6-8
Nombre del Parámetro 6-9
Indicación del Valor 6-9
Clases de Valores del Parámetro 6-10
. Formatos para Clases de Valores del Parámetro 6-10
. Clases de Valores 6-10
. Uso de la Clase de Valor Name 6-10
. Uso de la Clase de Valor String..... 6-11
. Uso de la Clase de Valor Integer 6-11
. Uso de la Clase de Valor KEY o una clausula KEY .. 6-11
. Uso de las Clausulas VAR OF y ARRAY OF 6-11
Indicación por Omisión (DEFAULT) 6-12
. Indicación $REQUIRED 6-12
. Indicación $OPTIONAL 6-12
. Indicación de una Lista de Valores 6-12

ASIGNACION DE VALORES A PARAMETROS 6-13

COMANDOS UTILIZADOS DENTRO DE PROCEDIMIENTOS 6-14

COMO DAR FORMATO A LOS PROCEDIMIENTOS DE SCL 6-15
Ejemplo de Como dar Formato 6-15
Comando FORMAT_SCL_PROC 6-16
Entrada que Recibe la Rutina que da Formatos a Procedimientos 6-18
Control del Proceso de Formateo 6-18

6.- CREACION DE PROCEDIMIENTOS EN SCL

Un procedimiento de SCL consiste de una lista de estatutos que son procesados cuando se ejecuta (call) el procedimiento con un simple nombre.

La finalización de un procedimiento de usuario no tiene diferencia entre un llamado de procedimiento y el llamado de cualquier otro comando de SCL. Esta capacidad hace fácil crear un nuevo comando o redefinir cualquier comando existente. En los procedimientos de usuarios no se necesita saber si el comando es implementado como un procedimiento SCL o como un programa ejecutable.

Se pueden definir procedimientos para aceptar parametros. Estos parametros pueden ser probados para ejecutar ciertos comandos o estatutos, o pueden ser sustituidos en comandos o estatutos que son ejecutados dentro del procedimiento.

CREACION DE PROCEDIMIENTOS

Para crear un procedimiento, se tiene que hacer los siguientes pasos (se asume que se esta trabajando en el catalogo $USER):

1.- Usar la utilería EDIT_FILE para crear un archivo que contendrá el procedimiento. Por ejemplo, el siguiente comando crea el archivo $LOCAL.EJEMPLO_PROC para contener el procedimiento:

```
/ edif $Local.ejemplo_proc
```

2.- Crear el Procedimiento. Más adelante se verán los detalles del formato de procedimientos. El siguiente es un ejemplo de procedimiento:

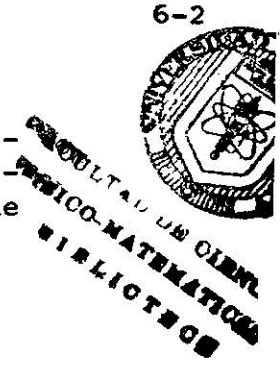
```
proc imprime_reporte, impr(
    archivo, a:  file=$required
    numero, n:   integer 1..1o=5
    banner, b:   string='REPORTE DE ESTADO MENSUAL'
    status)

    print_file  archivo=$value(archivo) copies=$value(número)..
                routing_banner=$value (banner)..
                comment_banner=$strrep($value(número))//'copias impresas

procend imprime_reporte
```

3.- Darle formato al procedimiento usando el formateador de SCL. Este es un paso opcional que se puede realizar para dar mayor legibilidad a su lectura. Por ejemplo, el siguiente comando da formato al procedimiento contenido en el archivo. $LOCAL. EJEMPLO_PROC:

```
/ format_scl_proc  input=$local.ejemplo_proc  output=impresión
```



Debido a que se escribe en un archivo permanente (localizado en el catalogo de trabajo $USER, el cual contiene unicamente archivos permanentes), el archivo $USER.IMPRESION que contiene el procedimiento es ahora permanente.

El procedimiento ya con formato aparece como sigue:

```
PROC  imprime_reporte, impr (
      archivo, a     :  file=$requiered
      numero, n      :  integer 1..10=5
      banner, b      :  string='REPORTE DE ESTADO MENSUAL'
      status         :  var of status=$optional
      )

      print_file archivo=$value(archivo) copies=$value(número)..
                 routing_banner=$value(banner)..
                 comment_banner=$strrep($value(número))//'copias impresas'

PROCEND imprime_reporte
```

LLAMADO DE PROCEDIMIENTOS

Para llamar a un procedimiento, hay dos métodos disponibles:

- Indicar el nombre del archivo que contiene al procedimiento.
- Indicar el nombre del procedimiento, el cual reside en una biblioteca de objetos.

Indicando el Nombre del Archivo:

El método más simple para llamar a un procedimiento es indicando el nombre del archivo que contiene al procedimiento. Cuando se usa este método, se aplican las siguientes condiciones.

- El archivo deben contener únicamente el procedimiento deseado para ser llamado; no podrá contener alguna otra información.
- El llamado al procedimiento debe indicar el nombre del archivo y también el nombre de cualquier archivo que el procedimiento manipulará.
- Si el catalogo que contiene al archivo es parte de la lista de comandos (command list), se necesita únicamente indicar el nombre del archivo que contiene al procedimiento. No importa cual sea el catalogo de trabajo actual. Por omisión (default) el catalogo $LOCAL pertenece a la lista de comandos (command list).
- Si el catalogo o catalogos que contienen al archivo no son parte de la lista de comandos (command list), se necesita indicar aquellos nombres de ctalogos junto con el nombre del archivo que contenga al procedimiento.

Por ejemplo, asumamos que se escribió el procedimiento anterior (IMPRIME_REPORTE) al archivo permanente IMPRESION y que al archivo manipulará a LIST_FILE. El llamado al procedimiento aparece como sigue (se asume que $USER no forma parte de la lista de comandos):

```
/ $user.impresion archivo=list_file
```

Si el archivo IMPRESION es parte del subcatalogo STATS del catalogo $USER, se llama al procedimiento como sigue (se asume que el catalogo de trabajo es $USER):

```
/ stats.impresion  archivo=list_file
```

Si el catalogo de trabajo es $LOCAL, se debe llamar al anterior procedimiento indicando explícitamente al catalogo $USER (aunque $USER sea parte de la lista de comandos):

```
/ $user.stats.print  archivo=list_file
```

Si el subcatalogo STATS es indicado en la lista de comandos, se puede llamar al procedimiento simplemente por el nombre del archivo.

```
/ print
```

o Indicando el Nombre del Procedimiento

Se puede llamar a un procedimiento indicando el mismo nombre del procedimiento. Sin embargo, antes de llamar al procedimiento de esta forma, se deben completar los siguientes pasos.

1. Poner al procedimiento en una bibilioteca de objetos usando el comando CREATE_OBJECT_LIBRARY.
2. Agregar la bibilioteca de objetos que contiene al procedimiento en la lista de comandos usando el comando SET_COMMAND_LIST.

El siguiente ejemplo pone al archivo IMPRESION (conteniendo al procedimientos IMPRIME_REPORTE) en la biblioteca de objetos y luego agregar la biblioteca de objetos a la lista de comandos:

```
/create_object_library
COL/add_modules  library=$user.impresion
COL/generate_library  library=$user.biblioteca_impresion
COL/quit
/set_command_list  add=$user.biblioteca_impresion
```

Con estos pasos, se puede llamar al procedimiento IMPRIME_REPORTE como sigue:

```
/ imprime_reporte  archivo=list_file
```

o Ambientes para el Llamado de Procedimientos

Los procedimientos de SCL pueden ser llamados desde los siguientes ambientes:

- Desde fuera de un procedimiento
- Desde dentro de otro procedimiento de SCL. Esto es conocido como llamados anidados.
- Desde dentro de si mismo. Esto es conocido como llamados recursivos.

Los procedimientos de SCL no pueden ser definidos dentro de otros procedimientos (definiciones anidadas). Por ejemplo, la siguiente estructura no es valida:

```
PROC  a
   :
   PROC  b
     :
     :
   PROCEND  b
   :
PROCEND  a
```

ESCRIBIENDO A UN ARCHIVO DESDE UN PROCEDIMIENTO

Cuando se escribe en un archivo desde dentro de un procedimiento, - se puede usar uno de los siguientes comandos:

```
COLLECT_TEXT
PUT_LINE
```

Si se desea escribir mas de una línea en forma sucesiva a un archivo, utilizar el comando COLLECT_TEXT para una rápida respuesta. Este comando abre el archivo de salida una sola vez para múltiples -- líneas, mientras que el comando PUT_LINE abre el archivo para cada línea. Además el comando COLLECT_TEXT permite sustituir los valores de las variables y expresiones string en el archivo de salida.

EJEMPLO DE UN PROCEDIMIENTO

La figura 6-1 contiene un procedimiento que acepta parametros. Estos parametros:

- Identificar el listado de la impresión del archivo usando el parámetro ROUTING_BANNER.
- Indicar el número de copias para impresión.

El procedimiento luego imprime un comentario en la página del ba- - nner indicando cuantas copias fueron requeridas.

Para llamar al procedimiento, indicar el nombre del procedimiento - (IMPRIME_REPORTE), seguido por los parametros indicados en la definición del procedimiento. (Este paso asume que ya se integró el -- nombre del procedimiento en la lista de comandos mediante una bi- - blioteca de objetos descrita anteriormente).

La siguiente llamada instruye a SCL para imprimir cuatro copias del archivo ESTADO_DE_CUENTAS con encabezado REPORTE DE ESTADO SEMANAL el cual será impreso en la primera página de cada listado.

```
/imprime_reporte  archivo=estado_de_cuentas  number=4..
../banner='REPORTE DE ESTADO SEMANAL'
```

```
PROC  imprime_reporte, impr(
      archivo, a     :   file=$required
      numero, n      :   integer 1..10=5
      banner, b      :   string='REPORTE DE ESTADO MENSUAL'
      status         :   var of status= $optional
     )

      print_file  file=$value(archivo) copies=$value(numero)..
                  routing_banner=$value(banner)..
                  comment_banner=$strrep($value(numero))//'copias impresas'.

PROCEND imprime_reporte
```

Figura 6-1. Ejemplo de un Procedimiento de SCL.

La siguiente secuencia describe que ocurre en el procedimiento de ejemplo:

1. El encabezado del procedimiento (estatuto PROC) identifica el nombre del procedimiento IMPRIME_REPORTE y su alias IMPR. El paréntesis abierto (el cual debe estar en la misma línea que el nombre del procedimiento) introduce la lista de parametros.

2. Los parametros son definidos en líneas separadas.

   - El parámetro ARCHIVO indica el archivo que será impreso. Cuando se indique, este parámetro puede ser abreviado con una A. Este es un parámetro requerido.

   - El parámetro NUMERO indica el número de copias a ser impresas. SCL solo acepta valores ente ros entre 1 y 10 para este parámetro. Si no se indica este parámetro cuando se llama al procedimiento, SCL asume que se desean cinco copias. El parámetro es opcional.

   - El parámetro BANNER indica el texto del comentario deseado para su impresión en la primera página de cada listado. SCL acepta solo un valor de string para este parámetro, el cual es opcional. Si no se indica este parámetro cuando se llama al procedimiento, se usa el valor REPORTE DE ESTADO MENSUAL.

3. El paréntesis cerrado termina con la definición de parametros.

4. El comando PRINT_FILE esta configurado de manera tal que puede aceptar los parametros indicados en el llamado al procedimiento. Cuando se llama al procedimiento, este comando es ejecutado con los parametros designados en efecto.

   - El parámetro FILE acepta el valor contenido en el parámetro ARCHIVO el cual debe acompañar al llamado del procedimiento. La función $VALUE realiza la referencia necesaria.

   - El parámetro COPIES acepta el valor contenido en el parámetro opcional NUMERO el cual acompaña al llamado del procedimiento. La función $VALUE realiza la referencia necesaria.

- El parámetro ROUTING_BANNER acepta el valor contenido en el parámetro opcional BANNER el cual acompaña al llamado del procedimiento. La función $VALUE realiza la referencia necesaria.
- La expresión para el parámetro COMMENT_BANNER hace tres cosas:
  - Acepta el valor contenido en el parámetro opcional NUMERO el cual acompaña al llamado del procedimiento.
  - Convierte ese valor en su representación en string.
  - Concatena esa representación en string con el string 'copias impresas'.

  La función $VALUE accesa los valores de los parametros del procedimiento, y la función $STRREP realiza la conversión necesaria.

5. El estatuto PROCEND termina el procedimiento.

FORMATO DE PROCEDIMIENTO

Un procedimiento de SCL tiene el siguiente formato general:

```
encabezado del procedimiento
  lista de estatutos
termino del procedimiento
```

El encabezado del procedimiento es requerido y siempre empieza con el estatuto PROC. El termino del procedimiento es requerido y siempre empieza con el estatuto PROCEND. La lista de estatutos es opcional.

Por ejemplo, para crear un procedimiento vacío llamado DESPLEGA_NUMERO en el archivo llamado DESPLEGA_EL_NUMERO, se indican los siguientes estatutos:

```
/collect_text  output=desplega_el_numero
ct?proc desplega_numero
ct?procend
ct?**
/
```

Para ejecutar el procedimiento del ejemplo anterior, indicar el nombre del archivo en el cual reside el procedimiento. Se asume que el catalogo de trabajo esta en la lista de comandos.

```
/desplega_el_numero
```

El encabezado del procedimiento, lista de estatutos, y termino del procedimiento son descritos en mayor detalle en los siguientes parrafos.

Formato para el Encabezado del Procedimiento:

El encabezado del procedimiento debe conformar con las siguientes directivas:

- Mínimamente, debe incluir el estatuto PROC con algún nombre de procedimiento. El nombre del procedimiento es cualquier nombre válido en SCL.
- El estatuto PROC debe estar separado del nombre del procedimiento por un espacio.

El encabezado del procedimiento más simple es como sigue:

```
PROC  nombre del procedimiento
```

El encabezado del procedimiento puede además incluir alias para un nombre de procedimiento. Se indican las alias en una lista en la cual cada nombre esta separado por un espacio o una coma:

```
PROC  nombre del procedimiento, alias,..., alias
```

El procedimiento DESPLEGA_NUMERO contiene algunos alias como DESPLEga-numero o DESN:

```
PROC  desplega_numero, desplega_numeros, desn
      .
      .
      .
     lista de estatutos
      .
      .
      .
PROCEND  desplega_numero
```

Con esta definición, cualquiera de las siguientes llamadas son validas:

```
desplega_numero
desplega_numeros
desn
```

Formato de la Lista de Estatutos:

Una lista de estatutos constituye el cuerpo de un procedimiento. El siguiente ejemplo muestra una lista de estatutos agregada al procedimiento DESPLEGA_NUMERO.

```
PROC  desplega_numero, desplega_numeros, desn
     "Este procedimiento despliega el número 3
     display_value 3

PROCEND  desplega_numero
```

Para llamar a este procedimiento, indicar el nombre del archivo en el cual reside el procedimiento, como en el siguiente ejemplo:

```
/$user.desplega_el_numero
3
```

Para ser capaces de llamar al procedimiento del ejemplo anterior por su nombre, se necesita poner DESPLEGA_NUMERO en una bibiliioteca de objetos y agregar la librería en la lista de comandos.

Formato de Terminación del Procedimiento:

Opcionalmente, se puede indicar el nombre del procedimiento en el termino del procedimiento que es con el estatuto PROCEND como se indica:

```
PROCEND  nombre del procedimiento
```

Si se indica el nombre del procedimiento, debe concordar con el primer nombre de la lista en el encabezado del procedimiento como en el siguiente ejemplo:

```
PROC  desplega_numero
      .
      .
      .
PROCEND  desplega_numero
```

En general, se incluye el nombre del procedimiento en el estatuto -- PROCEND para propositos de documentación.

DEFINICION DE LOS PARAMETROS DE UN PROCEDIMIENTO

Para acomodar la definición de parametros opcionales, podemos expander la definición del procedimiento básico como sigue:

```
PROC  nombres de procedimiento (definición de parametros)
    lista de estatutos
PROCEND  nombre de procedimiento
```

Las siguientes directivas se aplican a este formato de procedimiento expandido:

- Los paréntesis de abrir y cerrar son requeridos
- El paréntesis abierto debe estar en la misma línea que los nombres del procedimiento.
- Cada definición de parametros puede expresarse en una de las siguientes formas:
  - En líneas separadas
    ```
    definición de parámetro
    definición de parámetro
    ```
  - Mas de uno en una línea, separados por punto y coma:
    ```
    definición de parámetro; definición de parámetro
    ```
  - Dividido en dos o mas líneas. En este caso, una ellipsis (..) debe aparecer al final de cada línea a ser continuada:
    ```
    parámetro..
    definición
    ```

El siguiente es un ejemplo del formato de un encabezado de procedimiento con su definición de parametros:

```
PROC  nombres de procedimientos (
      definición de parametros
        .
        .
        .
      definición de parámetro)
```

Cada definición de parámetro tiene el siguiente formato:

```
nombres del parámetro:  indicación del valor=indicación por omi--
                        sión
```

nombres del parametro

Nombre o nombres con los cuales se puede referenciar al parámetro. Para cada parámetro definido se debe indicar al menos un nombre.

indicación del valor
Clase de valor y si se puede representar como lista y/o rango. Se pueden definir listas para favorecer el permitir un número específico de un conjunto de valores. Además, se puede definir cada conjunto de valores para que contengan un número específico de valores elementales. La indicación del valor es opcional.

indicación por omisión
Indica si el parámetro es opcional. Si el parámetro es opcional, se puede indicar su valor por omisión. La indicación por omisión es opcional.

En la definición del procedimiento, el formato completo de la definición del parámetro aparece como sigue:

```
PROC  nombres del procedimiento (
      nombres del parametro: indicación de valor=indicación por omisión)
      lista de estatutos
PROCEND nombre del procedimiento
```

Nombre del Parámetro:

El nombre de un parámetro permite referenciar un parámetro particular en el encabezado del procedimiento. Para cada parámetro definido, se debe incluir al menos un nombre. El siguiente ejemplo ilustra como se puede escribir el procedimiento DESPLEGA_NUMERO para aceptar un número:

```
PROC  desplega_numero, desplega_numeros, desn(
      numero, numeros, n: integer)
```

El parámetro NUMERO esta definido con dos alias y se le da una indicación de valor de integer (entero).

Indicación del Valor:

La indicación del valor indica la clase de valor y si se puede representar como lista y/o rango. La indicación del valor consta de los siguientes elementos:

INDICACION DEL VALOR	DESCRIPCION
parameter value kind	Indica la clase de valor que puede tomar el parámetro.
value list kind OF parameter value kind	Indica si el valor del parámetro puede ser dado como una lista y/o rango de valores.

La indicación del valor es opcional. Si no se indica una clase de valor, el sistema asume que el parámetro es de tipo FILE. Ocurre una excepción si el nombre del parámetro es STATUS; en este caso, se asume un valor VAR OF STATUS.

Clase de Valores del Parámetro:

La clase de valor del parámetro define:

- La clase de valor
- Si es una variable o un arreglo
- Si puede presentarse por uno o más claves (keywords)

o Formatos para Clases de Valores del Parámetro

Los siguientes son formatos válidos para las clases de valores - del parámetro:

- Clase de valor
- KEY clave (keyword)
- clase de valor OR KEY clave

Los siguientes son formatos válidos cuando la clase de valor es un tipo de variable válido:

```
VAR OF  clase de valor
ARRAY OF  clase de valor
```

o Clases de Valores

La siguiente tabla lista las clases de valores

CLASE DE VALOR	DESCRIPCION
FILE	Archivo
NAME	Nombre de SCL
STRING	String
INTEGER	Entero
REAL	Número Real
BOOLEAN	Valor Booleano
KEY	Valor clave (keyword)
STATUS	Variable de estado
ANY	Entrada que indica cualquier -- clase de valor puede ser usada. (Se pu de usar la función $VA-- LUE_KIND para determinar la cla se de valor actual).

o Uso de la Clase de Valor Name

Cuando se usa la clase de valor NAME, se puede indicar su longitud mínima y máxima de caracteres en cualquiera de los siguien-- tes formatos:

```
NAME    longitud
NAME    longitud mínima... longitud máxima
```

Si se omite la longitud, el sistema asume 1..$MAX_NAME (longitud máxima de un nombre).

```
name 10          Nombre con 10 caracteres
name 1..9        Nombre de 1 a 9 caracteres
name 2..2        Nombre con exactamente 2 caracteres
```

- Uso de la Clase de Valor String

  Cuando se usa la clase de valor STRING, se puede indicar la longitud mínima y máxima de caracteres en cualquiera de los siguientes formatos:

  ```
  STRING   longitud
  STRING   longitud mínima..longitud máxima
  ```

  Si se omite la longitud del String, el sistema asume ..$MAX_STRING (máxima longitud de string).

  Los siguientes ejemplos indican strings con longitud específica.

  ```
  string 10      String con 10 caracteres exactamente
  string 1..40   String de 1 a 40 caracteres
  string 1.2     String de 1 a 2 caracteres
  ```

- Uso de la Clase de Valor Integer

  Cuando se usa la clase de valor INTEGER se puede indicar su valor entero mínimo y máximo como sigue:

  ```
  INTEGER  mínimo valor.. máximo valor
  ```

  Si se omiten los valores mínimo y máximo, el sistema asume $MIN_INTEGER..$MAX_INTEGER.

  Los siguientes ejemplos muestran como representar valores enteros:

  ```
  integer 1..100              Entero desde 1 hasta 100
    integer 0..$max_integer   Entero desde    hasta $MAX_INTEGER
  ```

- Uso de la Clase de Valor KEY o una clausula KEY

  Cuando se usa la clase de valor KEY o una clausula KEY, se puede aceptar un nombre de una lista de nombres definidos como valor del parámetro. Por ejemplo, asumamos que el parámetro CONTADOR indica el número de líneas de un archivo a desplegar. Se puede definir el parámetro CONTADOR para aceptar las claves ALL o NONE. Subsecuentemente, al indicar ALL hace que el sistema liste todas las líneas del archivo en cuestión. Puedes definir al parámetro como sigue:

  ```
  contador, c:  key all none
  ```

  Si desea permitir un valor entero y una clave, definir el parámetro CONTADOR como sigue:

  ```
  contador, c:  integer or key all none
  ```

- Uso de las Clausulas VAR OF y ARRAY OF

  Si se desea que el valor pasado a un parámetro sea variable, o si se requiere que el procedimiento regrese un valor, se usa la clausula VAR OF. Si se desea una variable de arreglo con una dimensión mayor que 1 sea pasada a un procedimiento, se usa la clausula ARRAY OF.

  Cuando se usen las clausulas VAR OF o ARRAY OF, se debe además indicar una de las siguientes clases de variable:

STRING	BOOLEAN
INTEGER	STATUS
REAL	ANY

Indicando ANY para la clase de valor se permite entonces una variable con clase de valor STRING, INTEGER, REAL, BOOLEAN o STATUS. Se puede utilizar la función $VALUE_KIND para evaluar la variable y regresar la clase de la variable.

Indicación por Omisión (Default)

La indicación por omisión indica si se quiere que un parámetro sea opcional o requerido. Se incluye la indicación por omisión en el procedimiento con el siguiente formato:

nombres del parámetro: indicación del valor=indicación por omisión

Se puede definir la indicación por omisión de cualquiera de las siguientes formas:

$REQUIRED
$OPTIONAL
lista de valores

La indicación por omisión es opcional. Al omitir la indicación por omisión es èquivalente a indicar $OPTIONAL.

o Indicación $REQUIRED

Si se indica $REQUIRED como indicación por omisión, se debe proporcionar el parámetro cuando se llame al procedimiento. Si se encuentra activo el prompt interactivo, entonces se manda un prompt para el parámetro; de otro modo SCL termina el llamado con un mensaje de diagnóstico.

La siguiente definición del parámetro indica un parámetro requerido:

```
salida, s:  file=$required
```

o Indicación $OPTIONAL

Si se indica $OPTIONAL como indicación por omisión, se puede omitir el parámetro cuando se llame al procedimiento. El sistema no asume un valor por omisión (default) para el parámetro. La siguiente definición del parámetro indica un parámetro opcional:

```
salida, s:  file=$optional
```

o Indicación de una Lista de Valores

Si se indica una lista de valores como indicación por omisión, el parámetro es opcional. Si se omite el parámetro en el llamado del procedimiento, el sistema asume el valor por omisión que fue indicado. El valor por omisión (default) es tratado exactamente como si hubiese sido provisto en la lista de parámetros, con una excepción: al usar la función $SPECIFIED en el cuerpo del procedimiento produce un resultado FALSE.

Por ejemplo, la siguiente definición indica el nombre de archivo por omisión $OUTPUT.

```
salida, s:  file=$OUTPUT
```

Asumamos que tenemos el siguiente encabezado del procedimiento:

```
PROC  desplega_fecha, desf
```

Se puede expander el encabezado del procedimiento DESPLEGA_FE--CHA como:

```
PROC  desplega_fecha, desf (salida, s: file=$OUTPUT)
```

La forma anterior de la definición del parámetro utiliza el formato más simple de la indicación del valor. El parámetro es definido con la clase de valor de archivo.

En el siguiente ejemplo, el procedimiento DESPLEGA_NUMERO esta definido para aceptar el parámetro SALIDA:

```
PROC desplega_numero, desplega_numeros, desn (
    número, numeros, n  :  list 1..10,1..2,..
    range of integer-100..100=$required
    salida, s           : file=$OUTPUT
    status              : var of status=$optional
   )
```

ASIGNACION DE VALORES A PARAMETROS

Establecer parametros con sus clases de valores es una parte muy - importante al escribir procedimientos. Se puede hacer esta actividad utilizando la función $value, como se demuestra en el siguiente procedimiento:

```
PROC  pregunta (
     respuesta, r     :  var of string = $required
     status           :  var of status = $optional
    )

     respuesta_inicial = "
     REPEAT
       accept_line v=respuesta_inicial i=input p='Dar Si o No-'
       respuesta_inicial=$substr($translate (ltu respuesta_inicial)1 1)
     UNTIL (respuesta inicial= 'S' OR respuesta_inicial='N')
     $value (respuesta)= respuesta_inicial

PROCEND  pregunta
```

El procedimiento refleja las siguientes caracteristicas:

- El parámetro simple, RESPUESTA, es una variable string
- El procedimiento utiliza el prompt hasta que se proporcione un valor que inicie con S, s, N, o n.
- El procedimiento luego asigna el valor del parámetro respuesta a S ó N.

El siguiente ejemplo usa el procedimiento:

```
/   contestación="
/ pregunta contestación
Dar Si ó No - Siempre
/ display_value  contestación
  S
```

COMANDOS UTILIZADOS DENTRO DE PROCEDIMIENTOS

Los comandos en la tabla 6-1 son comunmente utilizados dentro de -- procedimientos o para ayudar en la escritura de procedimientos.

Tabla 6-1. Comandos Utilizados dentro de Procedimientos

COMANDO	DESCRIPCION
ACCEPT_LINE	Lee líneas de un archivo dejándolas en una variable string.
COLLECT_TEXT	Lee líneas de texto de la lista de comandos y las escribe en un archivo indicado.
CREATE_VARIABLE	Crea una variable de SCL
DELETE_VARIABLE	Borra declaraciones de variables del - block actual.
DISPLAY_VALUE	Despliega el valor de una expresión.
DISPLAY_VARIABLE_LIST	Despliega las variables disponibles al job actual.
FORMAT_SCL_PROC	Da formato a un procedimiento y detecta errores en el archivo de entrada.
INCLUDE_COMMAND	Hace que el texto de un string específico sea logicamente reemplazado en el comando INCLUDE_COMMAND. Esto permite construir un comando a través de la manipulación de string para luego tener el comando procesado.
INCLUDE_FILE	Inserta en el grupo de comandos un archivo texto, el cual contiene estatutos de SCL.
INCLUDE_LINE	Inserta en el grupo de comandos un - - string el cual contiene estatutos de - SCL.
PUT_LINE	Escribe líneas desde una variable - - string en un archivo.
REQUEST_OPERATOR	Envia un mensaje al operador del sistema y solicita un mensaje de respuesta del operador.
REQUEST_TERMINAL	Asocia un archivo con una terminal en un job interactivo.
WAIT	Suspende el procesamiento de comandos hasta que un número específico de milisegundos hubiese transcurrido, o hasta que otro evento específico o conjunto de eventos tome lugar.

COMO DAR FORMATO A LOS PROCEDIMIENTOS DE SCL.

SCL proporciona una rutina que da formato a los procedimientos de -- SCL, el cual realiza dos tareas principales:

- Dar formato a un archivo que contiene comandos de SCL, haciendo que el procedimiento sea más fácil de leer.
- Detecta ciertos errores en el archivo de entrada y utiliza mensajes de diagnóstico apropiados.

La rutina que da formato lee un archivo de entrada con comandos de - SCL y genera un archivo de salida con su formato correspondiente. - El archivo de entrada puede consistir de uno o más procedimientos de SCL o porciones de procedimientos.

Ejemplo de Como dar Formato:

Las figuras que se muestran más adelante muestran el efecto que tiene la rutina que da formato a procedimientos de SCL sin formato. La figura 6-2 muestra un procedimiento sin formato contenido en el archivo INFILE. La figura 6-3 muestra el mismo procedimiento después de que la fase de dar formato ha concluido.

Para dar formato el procedimiento de ejemplo, ejecutar los siguientes pasos:

1. Llamar a la rutina que da formato de la siguiente manera:

   /format_scl_proc input=infile output=$user.proc_form

   La rutina procesa los estatutos del archivo de entrada INFILE y los escribe en el archivo de salida PROC_FORM en el catalogo - - $USER.

2. Dar el siguiente comando para ver el procedimiento ya con formato (desplegado en la figura 6-3):

   /copy_file file=$user.proc_form

Figura 6-2 Procedimiento de SCL sin Formato

```
proc aaa(
entrada, e    :file=$REQUIRED
salida, s     :file=$OUTPUT
status)
poner_este_mensaje_de _salida 'Este es un mensaje de ..
de salida' 0=$OUTPUT
edit_file XYZ
1,,a
include_file abcc
"$quit
etiqueta_while: while abc >def do
if hij='???' then
create_object_library
add_module  library=:nve.abcdefghijklmopqrstuvwxyz1.abcdefghi..
jklmn2
generate_library a
"$ quit
else; exit
ifend; whilend; procend
```

```
proc  aaa(
    entrada, e    :  file=$required
    salida, o     :  file=$OUTPUT
    status        :  var of status=$optional
   )

    poner_este_mensaje_de_salida 'Este es un mensaje de salida'..
        .   o=$output
    EDIT_FILE XYZ
      1,,a
      include_file abcc
"$quit
etiqueta_while:
    WHILE abc > def   DO
      IF hij='???' THEN
        CREATE_OBJECT_LIBRARY
          add_module library=:nve.abcdefghijklmnopqrstuvwxyz1..
          .abcdefghijklmn2
          generate_library a
$$quit
      ELSE
        EXIT
      IFEND
    WHILEND etiqueta_while

PROCEND aaa
```

Figura 6-3. Procedimiento SCL con Formato

Comando FORMAT_SCL_PROC:

Formato

```
FORMAT_SCL_PROC o
FORMAT_SCL_PROCS o
FORSP
  INPUT=file
  OUTPUT=file
  PAGE_WIDTH=integer
  INITIAL_INDENT_COLUMN=integer
  KEY_CHARACTER=character
  UTILITY_DEFINITION_FILE=file
  PROCESS_COLLECT_TEXT=boolean
  STATUS=status variable
```

Parametros

INPUT (I)
Nombre del archivo del cual los comandos serán leídos. Este parámetro es requerido. Este archivo debe tener los siguientes atributos de archivos.

- El atributo FILE_CONTENT debe ser LEGIBLE o UNKNOWN.
- El atributo FILE_PROCESOR debe ser SCL o -- UNKNOWN.
- El atributo FILE_STRUCTURE debe ser DATA o UNKNOWN.

OUTPUT (O)
Nombre del archivo en el cual se escribirán los comandos con formato. El archivo debe tener -- los mismos atributos que el archivo de entrada (INPUT). El parámetro es requerido.

PAGE_WIDTH (PW)
Ancho de página del archivo OUTPUT. El valor - que se indique es el margen derecho del archivo OUTPUT. El valor debe ser al menos mayor en 64 que el valor indicado en el parámetro INITIAL_ INDENT_COLUMN. El valor por omisión es de 110.

INITIAL_INDENT_COLUMN (IIC)
Columna inicial de la primera línea escrita en el archivo de salida. Este parámetro garantiza el formato de una sección de comandos no con -- formato normal (como aquellos que aparecen como entrada para COLLECT_TEXT). Estos comandos son extraídos de un procedimiento, y reemplazados - en el procedimiento con formato. La columna -- inicial por omisión es 1.

KEY_CHARACTER (KC)
Caracter que provoca que la línea completa de - entrada sea escrita en la línea de salida sin - ningún proceso de formato. Este caracter debe aparecer en la columna 1 de la línea de entrada. El caracter clave por omisión es el asterisco - (*).

UTILITY_DEFINITION_FILE (UDF)
Nombre del archivo que lista los comandos con - que se inician y terminan los comandos de utile ría. Si se crean utilerías propias, se necesi- ta crear un archivo que contenga todos los - - comandos de la utilería usados en el procedi- - miento. Si se omite este parámetro, unicamente los nombres de utilerías definidas en el siste- ma y sus terminaciones son incluidas en la sali da con formato.

PROCESS_COLLECT_TEXT (PCT)
Valor booleano que indica si las líneas dentro del comando COLLECT_TEXT y otros comandos simi- lares serán procesados para darles formato. El valor por omisión es FALSE (las líneas no debe- rán darseles formato).

STATUS
Condición de terminación para el Formato.

Comentarios

Si el archivo de salida en el comando FORMAT_ SCL_PROC no ha sido previamente abierto y cual- quiera de los atributos FILE_CONTENTS, FILE_ PROCESSOR, y FILE_STRUCTURE son establecidos --

Entrada que Recibe la Rutina que da Formatos a Procedimientos:

Se crea un archivo que contenga comandos de SCL el cual será el archivo de entrada para la rutina que da Formatos. El archivo puede consistir de uno o mas procedimientos de SCL, cada uno iniciando con un estatuto PROC y finalizando con un estatuto PROCEND.

No es necesario que la entrada consista de procedimientos completos. Cualquier colección de comandos de SCL es procesada por la rutina de formatos. Sin embargo, los blocks estructurados en SCL en el archivo de entrada deben estar completos, o el comando FORMAT_SCL_PROC generará mensajes del diagnóstico.

Control de Proceso de Formateo:

Para controlar el proceso de dar formato, se pueden proporcionar comentarios especiales de SCL en el archivo de entrada. Se inicia un comentario especial de SCL con los siguientes caracteres.

Formato

El comentario especial tiene el siguiente formateo:

```
"$
     COMMAND=nombre(name)
     FORMAT=booleano(boolean)
```

Parametros

COMMAND (C)
Comando de utilería o terminación de utilería. Este parámetro informa a la rutina de dar formato que el comando indicado será leído por la rutina pero ignorado por el sistema operativo cuando el procedimiento sea ejecutado.

FORMAT (FMT o F)
Valor booleano que notifica a la rutina que da formato si lo habilita o deshabilita. Se usa este parámetro para intercalar el formato on y off.

Comentarios

- El comentario puede empezar en cualquier posición de la línea, pero solo el espacio puede preceder al comentario en la línea de entrada.
- Si un comando aparece en la línea después de un comentario especial y estan separados por un punto y coma, la acción indicada se aplica al comando, también como las líneas consecutivas.

7.- FUNCIONES

OBTENIENDO INFORMACION DE FUNCIONES DEL SISTEMA 7-1

LLAMANDO A UNA FUNCION 7-2

LISTA Y DESCRIPCION DE FUNCIONES DE SCL 7-3
- $ACCESS_MODE 7-6
- $CATALOG 7-7
- $CHAR 7-7
- $DATE 7-7
- $FILE 7-8
- $FNAME 7-11
- $INTEGER 7-12
- $INTERACTION_STYLE 7-12
- $JOB 7-13
- $JOB_LIMIT 7-18
- $JOB_STATUS 7-19
- $MAX_INTEGER 7-19
- $MAX_NAME 7-20
- $MAX_STRING 7-20
- $MAX_VALUE_SETS 7-20
- $MAX_VALUES 7-21
- $MIN_INTEGER 7-21
- $MOD 7-21
- $NAME 7-22
- $ORD 7-22
- $PATH 7-23
- $QUOTE 7-24
- $REMOTE_VALIDATION 7-24
- $RING 7-25
- $SCAN_ANY 7-25
- $SCAN_NOT_ANY 7-25
- $SCAN_STRING 7-26
- $STATUS 7-27
- $STRING 7-28
- $STRLEN 7-28
- $STRREP 7-29
- $SUBSTR 7-29
- $TERMINAL_MODEL 7-31
- $TIME 7-31
- $VNAME 7-32

7.- FUNCIONES

SCL proporciona dos conjuntos de Funciones, que son usados para regresar valores de varios atributos de sistema y variables.

- Funciones usadas en cualquier parte dentro de un estatuto SCL.
- Funciones usadas dentro de procedimientos para determinar varios atributos de parametros de procedimiento.

Ambos tipos de funciones se describen en este capítulo. Una lista tópica de funciones SCL es seguida por descripciones completas en forma alfabetica.

Algunas utilerías de NOS/VE también proporcionan funciones. Estas funciones estan disponibles mientras la utilería esta siendo ejecutada.

Para desplegar funciones disponibles use el parámetro DD=F del comando DISPLAY_COMAND_LIST_ENTRY.

Nota:

Ya que una función es definida en una utilería esta es solo disponible mientras que la utilería se esta ejecutando, la función no puede ser evaluada en otra tarea. Por ejemplo, la función de la utilería EDIT_CATALOG, $CURRENT_FILE no esta evaluada por EDIT_CATALOG cuando ejecutas el siguiente comando de EDIT_CATALOG:

```
fortran i = $current_file
```

La evaluación no ocurre porque el comando anterior causa otra tarea a ser ejecutada (el compilador FORTRAN); la función $CURRENT_FILE no es conocida para esta tarea.

Ver el manual de ejemplos en línea para un procedimiento que es usado para evaluar un comando usando la función $CURRENT_FILE y después ejecutarla de utilería EDIT_CATALOG. El nombre del procedimiento es SUBSTITUTE_EDIT_CATALOG_$CF.

OBTENIENDO INFORMACION DE FUNCIONES DEL SISTEMA

Para ver información acerca de una función específica, use el comando DISPLAY_FUNCTION_INFORMATION. Este comando se describe en los siguientes parrafos.

Propósito:	Despliega información acerca de una función.
Formato:	DISPLAY_FUNCTION_INFORMATION ó DISFI Función=nombre Output=archivo Status=variable de estado

Parametros:

FUNCION (F)
Función de la cual tu quieres información. Este parámetro se requiere.

OUTPUT
Nombre del archivo que contendrá la información escrita la función. El archivo por omisión es OUTPUT.

STATUS
Condición de terminación del comando.

Ejemplos:

El siguiente ejemplo requiere información acerca de la función $DATE.

```
/display_function_information function=$date

parámetro 1= key month, moly, dmy, iso, isod,..
             ordinal, default=default.
```

Esta descripción nos dice lo siguiente:

- La función $DATE tiene un parámetro.
- Posibles entradas para este parámetro son las siguientes palabras claves: MONTH, MDY, DMY, ISO, ISOD, ORDINAL Y DEFAULT.
- El parámetro es opcional. Si tu omites este parámetro, el sistema asume que DEFAULT esta efectivo. Actualmente el valor para DEFAULT es ISOD.

LLAMANDO A UNA FUNCION

Para llamar una función indique el nombre de la función, seguido opcionalmente por una lista de parametros adjuntos dentro del parentesis como siguie:

$function name (lista de parametros)

$function name — Todos los nombres de las Funciones SCL empiezan con el caracter del signo de dolares; un un ejemplo del nombre de una función es - - $DATE. El nombre de la función se requiere.

(lista de parametros) — Cada parámetros en la lista representa un valor y es separado de otros parametros por un espacio o coma como sigue:

```
(value)
   ó
(valor, valor,...,valor)
```

Espacios seguidos al abrir paréntesis o antes de cerrar el paréntesis es opcional.

Si no incluyes una lista de parametros, no necesitas indicar el abrir y cerrar de los paréntesis.

No puede haber espacio entre el nombre de la función y el abrir paréntesis de una lista de parametros.

El siguiente ejemplo llama a la función SCL que regresa la fecha actual:

```
$date
```

También puedes llamar la función como sigue:

```
$date ( )
```

La llamada de la función $DATE es reemplazada por un string representando la fecha actual. Asume por ejemplo, que la siguiente expresión es ejecutada el 28 de marzo, 1987:

```
'La Fecha actual esta' // $date
```

El resultado es un string con el siguiente valor:

```
The current date is 1987-03-28
La fecha actual es 1987-03-28
```

La función $DATE acepta un parámetro opcional que especifica el formato en el cual la fecha debe ser regresada. Si omites este parámetro, el formato ISOD es usado. Si especificas el parámetro MONTH, la fecha se regresa como se muestra en el siguiente ejemplo:

```
March 28, 1987
```

La función de llamada que regresa la fecha mostrando el mes primero tiene el siguiente formato:

```
$date(month)
```

Para información más completa acerca de valores de parametros, ver capítulo 4.

LISTA Y DESCRIPCIONES DE FUNCIONES DE SCL

La tabla 7.1 presenta una lista de funciones de SCL organizada por las funciones que ejecuta. Ensequida de esta lista hay descripciones completas de las funciones en forma alfabética por nombre de función.

Tabla 7.1 Lista Topica de funciones de SCL

TOPICO	NOMBRE DE FUNCION
Accesar fecha y tiempo:	
Accesar la fecha actual	$DATE
Accesar el tiempo actual	$TIME
Convertir datos:	
Conversión de string a nombre de un archivo	$FNAME
Conversión de un string ó booleano a entero	$INTEGER
Conversión de un string a un nombre	$NAME
Conversión de un nombre, archivo, o variable a un string	$STRING
Conversión de cualquier valor a un string	$STRREP

Conversión de un string a un nombre variable	$VNAME
Regresando Valores Cons-- tantes:	
Regresando el valor máximo de un entero	$MAX_INTEGER
Regresando el valor mínimo de un entero	$MIN_INTEGER
Regresando la longitud -- máxima de un nombre	$MAX_NAME
Regresando la longitud -- máxima de un string	$MAX_STRING
Regresando el número máximo de valores	$MAX_VALUES
Manipulando Strings:	
Encontrar el caracter dado el original	$CHAR
Encontrar el ordinal dado el caracter	$ORD
Determinado largo de un - string	$STRLEN
Cambiando caracteres en - un string	$TRANSLATE
Copiando un string	$QUOTE
Borrar rastreo de espacios de un string	$TRIM
Buscar un caracter en un par de caracteres	$SCAN_ANY
Buscar un caracter no en un conjunto de caracteres	$SCAN_NOT_ANY
Buscar un string por otro string	$SCAN_STRING
Procesando substrings	$SUBSTR
Obtención de Información de Archivo y Catalogo:	
Determinando el modo de - acceso de un archivo	$ACCESS_MODE
Interrogando atributos de un archivo	$FILE
Accesando el nombre del - catalogo en uso	$CATALOG
Interrogando caminos o rutas de un archivo	$PATH
Obtención de Información del equipo (HARDWARE):	
Regresando atributos del mainframe	$MAINFRAME
Regresando atributos del procesador	$PROCESSOR

Obtención de Información Acerca de Jobs:	
Interrogando límite de recursos de un job	$JOB_LIMITS
Interrogando atributos - un job	$JOB
Determinando el estado - de un job	$JOB_STATUS
Determinando el modelo - de la terminal	$TERMINAL_MODEL
Interrogando validación remota	$REMOTE_VALIDATION
Determinando tu estilo - interactivo	$INTERACTION_STYLE
Determinando el nombre - de un job	$TASK_NAME
Determinando el estado - de una tarea	$TASK_STATUS
Determinando el nivel -- del mensaje	$MESSAGE_LEVEL
Determinando el lenguaje de mensajes	$NATURAL_LANGUAGE
Determinando el anillo - de ejecución	$RING
Interrogando atributos - de variables	$VARIABLE
Determinando la familia por omisión	$DEFAULT_FAMILY
Obtención de Información acerca de Salida de Ar-- chivos:	
Interrogando atributos - de un archivo de salida	$JOB_OUTPUT
Determinando el estado - de un archivo de salida	$OUTPUT_STATUS
Funciones que tratan con Valores de Estado:	
Determinando (el código de condición)	$CONDITION_CODE
Determinando (el nombre de condición)	$CONDITION_NAME
Determinando la severi-- dad de una condición	$SEVERITY
Construyendo un valor -- de estado	$STATUS
Obtención el estado completo de un comando	$PREVIOUS_STATUS

$ACCES_MODE

Propósito:

Regresa un valor booleano indicando si los modos de Acceso especificados son asigna-- dos a un archivo.

Formato:

```
$Acces=Mode
(file,
    keyword)
```

Parametros:

FILE
Nombre del archivo del cual desea conocer sus modos de acceso. Este parámetro es requerido.

Keyword

Modos de Acceso que tu quieres checar. Este parámetro es requerido. Usar uno ó mas de las siguientes palabras claves.

READ
Puedes leer el archivo.

APPEND
Puedes anañdir información al final del archivo.

MODIFY
Puedes alterar datos existentes en el archivo.

SHORTEN
Puedes suprimir datos del final del archivo.

WRITE
Puedes dar append, modify y shorten al archivo.

EXECUTE
Puedes ejecutar el código objeto en el archivo.

ALL
Una combinación de las formas de acceso. - READ, APPEND, MODIFY, SHORTEN Y EXECUTE.

Nota: Cuando el archivo es asignado con - modos de acceso que especificas, TRUE es - regresado. Cuando el archivo no es asignado con modos de acceso que especificas, -- FALSE es regresado.

Ejemplos:

El siguiente ejemplo usa la función $ACCESS_ MODE para determinar si una lista de estatutos debe ser ejecutada.

```
Si $access_mode (data_file_1,
```

```
If $access_mode (archivo_datos, append, --
   shorten) THEN
   :
   :
   "Estatutos que necesitan permisos para
   "acceso de APPEND y SHORTEN
IFEND.
```

$CATALOG

Propósito: Regresa un valor que representa el catalogo actual en uso.

Formato: $CATALOG

Parametros: Ninguno

Ejemplos: El siguiente ejemplo compara el catalogo - actual en uso con el Catalogo $USER. Si - el catalogo en uso es $USER se cambia a -- $LOCAL.

```
If $string($catalog)=$string($fname('$user'))THEN
   set_working_catalog $local
IFEND
```

$CHAR

Propósito: Regresa el caracter ASCII que corresponde al número entero que especificas.

Formato: $CHAR
(entero)

Parametros: Entero por el cual tu deseas el caracter - ASCII regresado. El entero es un ordinal; representa la posición del caracter en - - cuestión dentro de la secuencia de ASCII. Este parámetro se requiere.

Ejemplos:

- El siguiente ejemplo regresa el carac-- ter que es representado como código - - ASCII 25 (hexadecimal):

```
/display_value $char(25(16))
%
```

- El siguiente ejemplo despliega los ca- - racteres ASCII con los ordinales decima les 65, 66 y 67:

```
/display_value ($char(65),$char(66),$char(67))
 A
 B
 C
```

$DATE

Propósito: Regresa la fecha actual como un string.

Formato:

$DATE
(parámetro)

Parametros:

Parámetro
Forma en la cual la fecha se regresa. Las siguientes palabras clave son válidas:

MONTH
Regresa la fecha según se muestra en el siguiente ejemplo:
March 28, 1987
La longitud del string es variable.

MDY
Regresa la fecha en 8 caracteres según se muestra en el siguiente ejemplo.
03/28/89

DMY
Regresa la fecha en 8 caracteres según se muestra en el siguiente ejemplo:
28.03.87

ISOD ó ISO
Regresa la fecha en 10 caracteres en el siguiente ejemplo
1987-03-28
ISO es una sigla para la Organización Internacional de Standares.

ORDINAL
Regresa una fecha Juliana de 7 caracteres en la forma del año y número de día en el año. Por ejemplo:
1987087

DEFAULT
Regresa el fomrato de omisión para la fecha. Actualmente el valor por omisión es ISOD.

Si omites este parámetro, DEFAULT es usado.

$FILE

Propósito:

Regresa ciertos atributos del archivo.

Formato:

$FILE
(file,
keyword)

Parametros:

file
Nombre del archivo cuyos atributos estan cuestionando. Este parámetro es requerido.

Keyword

Nombre del atributo que estas cuestionando. La Tabla 7-2 lista y describe los nombres de atributo. Este parámetro es requerido.

Tabla 7-2 Atributos del archivo

Nombre del Atributo	Descripción	Resultado de la función
ASSIGNED (A)	Archivo dentro del job requisitado es asignado a un dispositivo.	Booleano
ATTACHED	El archivo esta asociado.	Booleano
CYCLE_NUMBER (CN)	Número cíclico de un archivo	Entero
DEVICE_CLASS (DC)	El archivo es asignado a un dispositivo NULL.	"NULL"
	El archivo es asignado (almacenaje masivo)	'MAS_STORAGE'
	El archivo es asignado a una unidad cinta magnética.	'MAGNETIC_TAPE'
	El archivo es asignado a una terminal.	'TERMINAL'
FILE_CONTENT (FC)	El contenido es desconocido.	'UNKNOWN'
	Caracter de Datos	'LEGIBLE'
	Módulo de Objeto	'OBJECT'
	Formato de lista o reporte.	'LIST'
FILE_ORGANIZATION (FO)	La organización del archivo es secuencial.	'SEQUENTIAL'
	La organización del archivo es acceso directo	'DIRECT_ACCESS'
	La organización es secuencial indexada	'INDEXED_SEQUENTIAL'
FILE_PROCESOR (FP)	Procesador Desconocido	'UNKNOWN'
	Texto ADA	'ADA'
	Texto APL	'APL'
	Texto ensamblador de NOS/VE	'ASSEMBLER'
	Texto Basic	'BASIC'
	Texto C	'C'

	Texto Cobol	'COBOL'
	Texto CYBIL	'CYBIL'
	Archivo con objeto en debyg	'DEBUGGER'
	Texto FORTRAN	'FORTRAN'
	Texto LISP	'LISP'
	Texto SCL	'SCL'
	Libreria SCV	'SCV'
	Texto PASCAL	'PASCAL'
	Texto PLI	'PLI'
	Texto ensamblador -- NOS PP	'PPU=ASSEMBLER'
	Texto PROLOG	'PROLOG'
	Asociado con VX/VE	'VX'
FILE_STRUCTURE (FS)	La estructura es desconocida	'UNKNOWN'
	Archivo de datos	'DATA'
	Archivo de libreria	'LIBRARY'
GLOBAL_FILE_POSITION (GFP)	El archivo es colocado al inicio de la información después del último acceso al archivo	'BOI'
	El archivo es colocado al inicio de la partitución después del último acceso de archivo	'BOP'
	El archivo es colocado al final de la información después del último acceso de archivo	'EOI'
	El archivo es colocado al final de la partición después del último acceso de archivo.	'EOP'
	El archivo es colocado al final del registro después del último - - acceso al archivo	'EOR'
	El archivo es colocado entre el inicio y el - final de un registro - después del último - - acceso	'MID_RECORD'

OPENED (O)	El archivo esta abierto	Booleano
OPEN_POSITION (OP)	Es colocado al princi-- pio de la información - después de una opera- - ción de open	'$BOI'
	El archivo es colocado al final de la informa- ción después de una ope_ ración	'$EOI'
	El archivo no esta colo_ cado despupes de una -- operación de open	'$ASIS'
PERMANENT (P)	El archivo es permanen- te	Booleano
SIZE (S)	Largo del archivo en -- bytes	ENTERO

$FNAME

Propósito: Convierte un string a un nombre de archivo.

Formato:
```
$FNAME
   (string)
```

Parametros:
string
Variable tipo String que quieres convertir - al nombre de un archivo. Este parámetro se requiere.

Ejemplos: El siguiente ejemplo usa el resultado de la función $FNAME como el parámetro del nombre de un archivo.

```
/catalog name = '$user'
/file_name = 'data_file_1'
/attach_file $fname (catalog_name//'.'
//file_name)
```

En el ejemplo anterior, la variable CATALOG_ NAME es creada con el valor string '$user'. Similarmente, la variable FILE_NAME es crea- da con el valor string 'data_file_1'. La -- expresión de string es pasada a la función - $FNAME, que es evaluada de la siguiente for- ma:

```
$user.data_file_1
```

Subsequentemente, la función $FNAME convier- te el string evaluado al nombre de archivo - $USER.DATA_FILE_1, y el comando ATTACH_FILE es procesado con el parámetro convertido co- mo sigue:

Attach_file $USER.DATA_FILE_1

ATTACH_FILE $USER.DATA_FILE_1

$INTEGER

Propósito: Convierte un string ó booleano a un entero.

Formato:
$INTEGER
(parámetro)

Parametros:
parametros
Un valor String ó booleano que quieres convertir a un entero. Este parámetro es requerido.

Notas:
- Si usas esta función para convertir un string, debes estar seguro que el contenido en el string conforme a las reglas para formar un entero constante.
- Valores booleanos son convertidos a cero 0 cuando el valor es FALSE, y a 1 cuando el valor es TRUE.

Ejemplos:
- Los siguientes ejemplos muestran la creación de variables string y el uso de la función $INTEGER para convertir estos strings a enteros.

```
/create_variable sample_1 kind=string...
.../value='off (16)'
/display_value $integer (sample_1)
off (16)

/create_variable sample_2 kind=string...
.../value=' 123(10)'
/display_value $integer (sample_2)
123 (10)
```

El SCL mantiene la base de cada variable entera que tu creas. Si quieres saber el valor entero actual de una variable, haga referencia de la variable por su nombre; SCL regresa el entero con la misma base que usaste cuando create la variable.

- El siguiente ejemplo convierte un valor booleano a un entero.

```
/display_value $integer (false)
0
```

$INTERACTION_STYLE

Propósito: Regresa el estilo de interacción actual de tu terminal.

Formato:
$INTERACTION_STYLE
(style)

Parametros:

style
Estilo de interacción que tu quieres verificar. Este parámetro es requerido. Usa una de las siguientes entradas.

LINE (L)
SCREEN (S)

Si el sistema regresa un valor TRUE, el estilo interacción que especifica esta vigente. Si el sistema regresa a un valor FALSE, el otro estilo de interacción esta vigente.

Ejemplos:

El siguiente ejemplo indica que el estilo de interacción actual es el modo en pantalla:

```
/display_value $interaction_style
(screen)
TRUE
```

$JOB

Propósito:

Regresa atributos específicos de un job actual.

Formato:

```
$JOB
(keyword)
```

Parametros:

keyword
Nombres de atributos de un job que quieres regresar. La tabla 7-3 lista y describe los nombres de atributos válidos para un job, junto con los resultados de función. Este parámetro es requerido.

Nota: El tipo de resultado regresado depende del atributo que esta siendo probado (tabla 73). Cuando el valor string es regresado, todas las letras son convertidas en mayúsculas.

Ejemplos:

El siguiente ejemplo de un procedimiento SCL regresa el modo del job actual. Si el modo es interactivo, un mensaje es desplegado.

```
IF $job(mode)= 'INTERACTIVE' THEN
   display_value 'Welcome to NOS/VE'
IFEND
```

TABLA 7-3 ATRIBUTOS JOB

Nombre del atributo	Descripción	Resultado de la Función
C170_OS_TYPE	El compañero del dual-state para el job actual es NOS.	'NOS'
	El compañero del dual-state para el job actual es NOS/BE.	'NOS/BE'
	No hay compañero del dual-state; el job esta ejecutando en un solo sistema.	'NONE'
COMMENT_BANNER (CB)	String de caracter asumido desplegado con archivos de salida generados por el job. Usado en un comentario acerca de la salida impresa.	String
CONTROL_FAMILY (CF)	Nombre de familia del usuario del control. Para muchos jobs, este es el nombre de familia para el log in user.	Name
CONTROL_USER (CU)	Nombre del control del usuario. Para casi todos los jobs, esto es el nombre del login user.	Name
COPIES (C)	Número asumido de copias que deben hacerse para archivos de salida generados por el job.	Entero
CYCLIC_AGING_ INTERVAL (CAI)	Número en milisegundos al cual el conjunto de trabajo del job es avanzado.	Entero
DETACHED_JOB_ WAIT_TIME (DJWT)	Número de segundos en que el job permanece suspendido si es separado o desconectado de la sesión de la terminal antes de que termine el job.	Entero
DEVICE (D)	Nombre del dispositivo que cuando combinado con el nombre de la estación, identifica un dispositivo de salida en particular	Name
DISPATCHING_ PRIORITY (DP)	Remitir prioridad asignado como default a todas las tareas del usuario	Name

EXTERNAL_CHARACTERISTICS (EC)	String de caracteristicas externas por default para salida de archivos generados por el job.	String
FORMS_CODE (FC)	String de formas de codigo por default para archivos de salida generados por el job.	String
JOB_ABORT_DISPOSITION (JAD)	Acción tomada cuando un job es abortado debido a una falla del sistema.	
	Job vuelve a empezar	RESTART
	Job es terminado	TERMINATE
JOB_CLASS (JC)	Clase del Job actual	Name
JOB_MODE (JM)	Modo del job actual es -- batch	'BATCH'
	Modo del job actual es interactivo.	'INTERACTIVE'
	Job interactivo fue desconectado de una terminal como resultado de un comando DETACH_JOB.	'INTERACTIVE_COMMAND_DISCONNECT'
	Job interactivo fue desconectado de una terminal debido a un problema con la línea de comunicación (tal como colgar un teléfono)	'INTERACTIVE_LINE_DISCONNECT'
	Job interactivo fue desconectado de la terminal debido a una falla en el sistema.	'INTERACTIVE" SYSTEM_DISCONNECT'
JOB_RECOVERY_DISPOSITION (JRD)	Acción tomada siguiendo una interrupción del sistema.	
	El job continua después del punto de interrupción	CONTINUE
	El job se vuelve a empezar	RESTART
	El job es terminado	TERMINATE
JOB_SIZE (JS)	Tamaño en los bytes del job de la entrada de un archivo. Para jobs interactivos, este valor es siempre cero.	Entero

LOGIN_ACCOUNT (LA)	Nombre de la cuenta bajo el cual el job es registrado y corrido	Name
LOGIN_FAMILY (LF)	Nombre de la familia bajo el cual el job es registrado y corrido.	Name
LOGIN_PROJECT (LP)	Nombre del proyecto bajo el cual el job es registrado y corrido	Name
LOGIN_USER (LU)	Nombre del usuario bajo el cual el job es registrado y corrido	Name
MAXIMUM_ WORKING_SET (MAXWS)	Número máximo de páginas de memoria permitidas en el conjunto de trabajo del job	Entero
MINIMUM_ WORKING_SET (MINWS)	Número mínimo de páginas de memoria permitidas en el conjunto de trabajo del job	Entero
OPERATOR (O)	Indica si el job actual tiene privilegios del operador	Booleano
OPERATOR_FAMILY (OF)	Nombre de familia asumido de un operador de una estación privada que recibe archivos de salida generados por el job.	Name
OPERATOR_USER (OU)	Nombre asumido del operador de una estación privada que recibe archivos de salida generados por el job	Name
ORIGINATING_ APPLICATION_NAME (OAN)	Nombre de aplicación que causó el job para ser entrado dentro del sistema	Name
OS_VERSION	Nombre y versión de un sistema operativo	String
OUTPUT_CLASS (OC)	Clase de salida asumida para archivos de salida generados por el job	Name
OUTPUT_ DESTINTATION (ODE)	Identificador asumido del sistema o estación dando archivos de salida estan siendo impresos	String
OUTPUT_ DESTINATION_USAGE (ODU)	Uso asumido del destino al cual archivos de salida generado por el job son enviados. Una de las siguientes actividades toma lugar:	

	La salida es enviada al sistema NOS ó NOS/B que comparte el mainframe. Si no hay ningún compañero del dual-state, la salida es indefinidamente reciclada en la cola.	DUAL_STATE
	La salida es enviada vía la facilidad de red de transferencia a un sistema remoto para impresión	NTF
	La salida es enviada a una estación pública CDNET 1/0	PUBLIC
	La salida es enviada vía transferencia de la cola de archivo a un sistema remoto para impresión	QTF
OUTPUT_DISPOSITION (ODI)	Especifica como la salida del job es colocada:	
	La salida es dirigida a un archivo específico	FILE
	Toda la salida es desechada	DISCARD_ALL_OUTPUT
	La Salida Standar es descartada	DISCARD_STANDARD_OUTPUT
	Salida generada por la corrida de un job en un sistema remoto dirido a la impresora en el sistema remoto	LOCAL
	Salida es dirigida al propietario del job subcatalogo $WAIT-QUEUE	WAIT_QUEUE
OUTPUT_PRIORITY (OP)	Incremento prioritario asumido que es agregado a la salida inicial prioritaria de archivos de salida generados por el job	Name
REMOTE_HOST_DIRECTIVE (RHD)	Directivos asumidos para archivos de salida generados por el job si el parámetro OUTPUT_DESTINATION_USAGE ya sea DUAL_STATE ó nombres a facilidad de transferencia	String
ROUTING_BANNER (RB)	String caracteres asumidos que es desplegado con archivos de salida generados por el job	String

SERVICE_CLASS (SC)	Clase de servicio de un job actual	Name
SITE_INFORMATION (SI)	String de caracteres que es establecido por el SITE cuando el job es reciclado	String
STATION (S)	Nombre de estación asumido al cual los archivos de salida generado por el job son enviados	Name
SWITCHn	Especifica si el sense switch de job actual esta encendido ó apagado (n es un entero del 1 al 8)	Booleano
SYSTEM	Especifica si el job actual es el job del sistema	Booleano
SYSTEM_JOB_NAME (SJN)	Nombre asignado al job por el sistema	Name
USER_INFORMATION (UI)	String de caracteres de 1 a 256 que es pasado a todos los archivos de salida generado por el job	String
USER_JOB_NAME (UJN)	Nombre que fue proporcionado por el usuario para el job	Name

---

$JOB_LIMIT

Propósito: Regresa información acerca de los recursos límites para un job.

Formato:

```
$JOB_LIMIT
   (limit,
      keyword)
```

Parametros:

limit
Nombre del recurso límite para un job del cual deseas información. Este parámetro es requerido. Utilice uno de los siguientes nombres:

CPU-TIME: Indica el tiempo de CP (control de proceso), en microsegundos

SRU: Indica las unidades de recursos del sistema que acumulo el job

TASK: Indica la tarea que actualmente esta siendo ejecutada dentro del job

keyword
Forma en la cual deseas la información. Este parámetro es requerido. Utilice una de las siguientes claves:

ACTIVE
Indica si el límite indicado esta activo para el job. Si se indica esta clave, la función $JOB_LIMIT regresa un valor booleano.

ACCUMULATOR
Indica el valor actual del acumulador. Si se indica esta clave, la función $JOB_LIMIT regresa un valor entero.

RESOURCE_LIMIT
Indica el valor actual del límite de recurso. Si se indica esta clave la función $JOB_LIMIT regresa un valor entero.

ABORT_LIMIT
Indica el valor máximo permitido para el límite de recurso, esto es, el punto en el cual su job abortará. Si se indica esta clave, la función $JOB_LIMIT regresa un valor entero.

$JOB_STATUS

Propósito: Regresa el estado de un job.

Formato:
```
$JOB_STATUS
  (name,
     keyword)
```

Parametros:

name
Nombre del job del cual deseas conocer su estado o situación actual. Es válido si el nombre es indicado por ya sea el sistema o el usuario. Si el usuario proporciona el nombre del job, este deberá ser único, de lo contrario, la función fallará. Este parámetro es requerido.

keyword
Tipo de información que usted desea conocer del job mencionado.

$MAX_INTEGER

Propósito: Regresar el máximo entero positivo permitido para un parámetro.

Formato: $MAX_INTEGER

Parametros: Ninguno

Comentarios: El valor regresado es 281, 474, 976, 710, 655.

Ejemplo: En el siguiente ejemplo, el sistema es - - cuestionado para que proporcione el valor de $MAX_INTEGER:

```
/display_value  $max_integer
281474976710655
```

$MAX_NAME

Propósito: Regresar la longitud máxima permitida para un nombre utilizado como parámetro.

Formato: $MAX_NAME

Parametros: Ninguno

Comentarios: El valor regresado es 31.

Ejemplo: En el siguiente ejemplo, el sistema es - - cuestionado para que indique el valor de - $MAX_NAME:

```
/display_value  $max_name
31
```

$MAX_STRING

Propósito: Regresar la longitud máxima permitida para un string utilizado como parámetro.

Formato: $MAX_STRING

Parametro: Ninguno

Comentarios: El valor regresado es 256.

Ejemplo: En el siguiente ejemplo, el sistema es - - cuestionado para que indique el valor de - $MAX_STRING:

```
/display_value  $max_string
256
```

$MAX_VALUE_SETS

Propósito: Regresar el número máximo permitido de - - sets para un parámetro.

Formato: $MAX_VALUE_SETS

Parametros: Ninguno

Comentarios: El valor regresado es 2, 147, 483, 647.

Ejemplos: En el siguiente ejemplo, el sistema es - - cuestionado para que indique el valor de - $MAX_VALUE_SETS:

```
/display_value $max_value_sets
2147483647
```

$MAX_VALUES

Propósito: Regresar el número máximo de valores permitido para cada set de valores.

Formato: $MAX_VALUE

Parametro: Ninguno

Comentarios:. El valor regresado es 2, 147, 483, 647.

Ejemplos: En el siguiente ejemplo, el sistema es cuestionado para que indique el valor de $MAX VALUES

```
/display_values  $max_value
2147483647
```

$MIN_INTEGER

Propósito: Regresar valor del menor entero permitido para un parámetro.

Formato: $MIN_INTEGER

Parametros: Ninguno

Comentarios: El valor regresado es de -281, 474, 976, -- 710, 655.

Ejemplos: En el siguiente ejemplo, el sistema es cuestionado para que indique el valor de $MIN_INTEGER:

```
/display_value  $min_integer
-2814749676710655.
```

$MOD

Propósito: Regresar el modulo de un entero con respecto a otro entero.

Formato:

```
$MOD
  (integer 1,
   integer 2)
```

Parametros:

integer1
Número en el cual el modulo será regresado. Este parámetro es requerido.

integer2
Número para el cual el modulo es relativo. Este parámetro es requerido.

Comentarios: El resultado es calculado usando la siguiente formula (a y b son enteros, y la división es división entera):

```
modulo (a,b) = a-((a/b)*b)
```

Ejemplos: El siguiente ejemplo regresa el modulo de -134 con respecto a 10:

```
/display va ue  $mod (134,10)
4
```

$NAME

Propósito: Convertir un string a un nombre de SCL.

Formato:
```
$NAME
  (string)
```

Parametros:
string
String que desea convertir a un nombre de SCL. Este parámetro es requerido.

Comentarios:
- Con esta función, se puede construir un nombre utilizando un string y posteriormente utilizarlo como un valor de un parámetro tipo name.
- El resultado de esta función es convertido a caracteres mayúsculas.

Ejemplos:
- Los siguientes ejemplos crean un nombre de variable tipo string DATE_FORMAT. Este valor es convertido a un nombre y pasado a la función $DATE como opción de la fecha.

```
/date_format = 'month'
/display_value  date_format
month
/display_value $name(date_format)
MONTH
/display_value $date($name(date_format))
 March 28, 1987.
```

- Los siguientes ejemplos ilustran el resultado cuando se proporciona un nombre inválido en la funcion $NAME.

```
/date_format = 'mon th'
/display_value $date($name(date_format))
--ERROR-- Improper name: mon th
```

$ORD

Propósito: Regresar el entero que corresponde al caracter ASCII que fue indicado.

Formato:
```
$ORD
  (string)
```

Parametros:
string
Caracter ASCII para el cual se desea obtener el número entero correspondiente. Este parámetro es requerido.

Comentarios: El entero regresado es un ordinal; representa la posición del caracter en cuestión dentro de la secuencia ASCII.

Ejemplos:
- El siguiente ejemplo regresa el entero decimal que corresponde al caracter # de ASCII:

```
/display_value $ord('#')
 35
```

. El siguiente ejemplo realiza la misma operación, pero despliega el valor hexadecimal del caracter ordinal:

```
/post_radix = '(16)'
/display_value $strrep ($ord('#',16)//post_radix
23(16)
```

$PATH

Propósito:

Regresa ya sea una porción de la ruta (un string) o el número de elementos en la ruta (un entero)

Formato:

```
$PATH
  (path,
    keyword)
```

Parametros:

path
Nombre de la ruta o camino que se esta consultando. Este parámetro es requerido.

keyword
Entrada que indica si la porción de la ruta o el número de elementos será lo que regrese. Este parámetro es requerido. Use uno de los siguientes:

CATALOG
Hace que la porción de catalogo de una ruta sea regresada en un string.

LAST
Hace que el último elemento de la ruta sea regresado en un string.

COUNT
Regresa un contador entero de los elementos contenidos en la ruta.

Comentarios:

Cuando la función regresa un valor string todas las letras dentro del string son mayúsculas.

Ejemplos:

. El siguiente ejemplo despliega la porción de catalogo de una ruta de archivo $USER.DATA_FILE_1. El nombre del usuario es USER_123 y el nombre de familia es FAMILY_Z.

```
/display_value $path($user.data_file_1, catalog)
:FAMILY_Z.USER_123
```

. El siguiente ejemplo despliega el último elemento en la ruta de archivo anterior.

```
/display_value $path($user.data_file_1, last)
DATA_FILE_1
```

. El siguiente ejemplo despliega el # de elementos de la ruta de archivos anterior. - El # 3 se obtiene del nombre de familia - FAMILY_Z, el nombre de usuario USER_123, y el nombre de archivo data_file_1.

```
/display_value  $path($user.data_file_1, count)
```

$QUOTE

Propósito:

Copiar un string a otro string y agregarles delimitadores de strings.

Formato:

```
$QUOTE
  (string)
```

Parametros:

string
String que desea copiar.

Ejemplos:

El siguiente ejemplo copia el string S al - string Q:

```
/s = 'ABC''DEF'
/display_value s
ABC'DEF
/q = $quote(s)
'ABC''DEF'
```

$REMOTE_VALIDATION

Propósito:

Regresar un valor booleano indicando si hay validación para el acceso de archivos en la localidad remota indicada.

Formato:

```
$REMOTE_VALIDATION
  (location)
```

Parametros:

Location
Nombre de la localidad remota en donde residen los archivos a accesar. Si la validación es establecida para esa localidad, la función regresa un valor TRUE. Este parámetro es requerido.

Ejemplos:

El siguiente ejemplo consulta si esta establecida una validación remota con una localidad dada (MACHINE_A) y, si no esta establecida, despliega un mensaje:

```
IF $remote_validation(machine_a)=FALSE THEN
   display_value 'remote validation not defined..
   for MACHINE_A'
IFEND
```

$RING

Propósito: Regresar un entero indicando el anillo de ejecución actual para un tarea (task).

Foramato: $RING

Parametros: Ninguno

Ejemplos: El siguiente ejemplo indica que la tarea actual esta asociada con un anillo de ejecución 11.

```
/display_value $ring
11
```

$SCAN_ANY

Propósito: Buscar un string para cualquiera de un conjunto especifico de caracteres.

Formato:
```
$SCAN_ANY
  (string1,
   string2)
```

Parametros:

string1
Conjunto de caracteres a ser localizados. Este parámetro es requerido.

string2
String donde se hará la busqueda. Este parámetro es requerido.

Comentarios: Esta función regresa un número indicando la posición del primer caracter en el string2 que también fue localizado en el string1. Si ningún caracter en el string1 aparece en el string2, el entero Ø (cero) es regresado.

Ejemplo: El siguiente ejemplo busca por la posición del primer caracter en el string S que también ocurra en el string D:

```
/d = '0123456789'
/s = 'temp 32'
/display value $scan-any (d,s)
6
```

Se regresa un 6 porque el primer caracter en el string s que también estuviera en el string D es el número 3; este caracter esta en la sexta posición dentro del string s.

$SCAN_NOT_ANY

Propósito: Buscar un string para cualquier caracter que no se encuentre en un conjunto de caracteres dado.

Formato:
```
$SCAN_NOT_ANY
  (string1,
   string2)
```

Parametros:

string1
Conjunto de caracteres. El sistema busca - por cualquier caracter que no se encuentre en este conjunto. Este parámetro es requerido.

string2
String donde se hará la busqueda. Este parámetro es requerido.

Comentarios:

Esta función regresa un número indicando la posición del primer caracter en string2 que no se encuentre en string1. Si solo caracteres del string1 aparecen string2, el entero Ø (cero) es regresado.

Ejemplos:

El siguiente ejemplo regresa un número mostrando la posición del primer caracter en - string S (llamado, t) que no se encontró en el string D.

```
/d = '0123456789'
/s = 'temp_32'
/display_value $scan_not_any (d,s)
1
```

$SCAN_STRING

Propósito:

Buscar un string para localizar las ocurrencias de otro string (llamado patrón)

Formato:

```
$SCAN_STRING
  (string1,
   string2)
```

Parametros:

string1
String patrón (string donde se hará la busqueda).
Este parámetro es requerido.

string2
String donde se hará la busqueda. Este parámetro es requerido.

Comentarios:

Esta función regresa un número indicando la posición del primer caracter de la primera ocurrencia del string patrón donde se encontró el string donde se hizo la busqueda. Si el string patrón es un string nulo, el entero 1 se regresa como resultado. Si el patrón no se encontró en el string, el entero 0/ es regresado.

Ejemplos:

El siguiente ejemplo regresa un número mostrando la posición en el string S de la primera ocurrencia del string patrón P (la sexta posición):

```
/s = '0123_abc_9'
/p = 'abc'
/display_value $scan_string(p,s)
6
```

$STATUS

Propósito: Regresar un valor de estado con los campos - del registro de estado indicado.

Formato:

```
$STATUS
  (normal,
   identifier,
   condition,
   message)
```

Parametros:

normal
Valor booleano del campo NORAML de un registro de estado. Si el valor de este parámetro es TRUE, el resto de los parametros son ignorados. Si el valor de este parámetro es FALSE, se debe indicar el parámetro de condición o los parametros de condition e identifier. Este parámetro es requerido.

identifier
El campo IDENTIFIER de un registro de estado. Es un string de 2 caracteres.

condition
Campo CONDITION de un registro de estado. -- Puede ser ya sea un entero o un nombre de la condición de estado.

message
El campo TEXT de un registro de estado. Es - un string de hasta 256 caracteres conteniendo hasta 10 parametros de mensajes separados El valor por omisión es un string nulo.

Ejemplos:

- El siguiente ejemplo despliega un valor de estado construido con 2 parametros de mensaje:

```
/display_value $status(false,'CL',..
../cle$wrong kind of value, 'INTEGER','STRING')
--ERROR-- Expecting INTEGER value, found STRING.
```

- El siguiente ejemplo ilustra que se despliega si el sistema no puede encontra un mensaje de status asociado con la condición - de estado.

```
/display_value $status..
../(false,'MY',Ø,'param1','param2')
--ERROR-- CC=MY Ø TEXT=?param1?param2
```

$STRING

Propósito:

Convertir una referencia de archivo, nombre o referencia de variable a un string de mayúsculas.

Formato:

```
$STRING
  (parameter)
```

Parametros:

parameter
Referencia de nombre, archivo o variable a ser convertido. El parámetro es requerido.

Comentarios:

- Si se va a convertir un archivo a string, se puede indicar como el archivo será posicionado antes de su uso.
- Esta función es útil cuando se desea comparar 2 nombres. Debido a que los nombres no pueden compararse directamente con operadores relacionales, deben convertirse primero a strings y los strings resultantes ser comparados.

Ejemplos:

El siguiente ejemplo utiliza la función $VALUE para comparar 2 nombres de archivo que fueron pasados a un procedimiento de SCL:

```
IF $string($value(from))=$string($value(to))THEN
   .
   .
   .
IFEND
```

La función $value regresa el valor actual que fue indicado por un parámetro de procedimiento. El ejemplo asume que el procedimiento fue llamado con los parametros de archivo FROM y TO. La función $VALUE(FROM) proporciona el valor actual indicado en el parámetro FROM. La función $STRING luego convierte ese valor (un nombre) a un string el cual puede ser comparado con el string resultante de $STRING(VALUE(TO)). Este string contiene el nombre indicado en el parámetro TO. Si los nombres pasados al procedimiento son idénticos, el procedimiento establece un estado apropiado y termina.

$STRLEN

Propósito:

Regresar un entero que represente la longitud actual de un string.

Formato:

```
$STRLEN
  (string)
```

Parametros:

string
string del cual desea saber su longitud. Este parámetro es requerido.

Comentarios: La longitud regresada es la longitud actual, no la longitud máxima definida. La longitud actual es igual a la posición del caracter último en un string.

Ejemplos: En el siguiente ejemplo, el sistema pregunta por la longitud actual del string 'abc'//'123':

```
/display_value #strlen ('abc'//'123')
```

$STRREP

Propósito: Convertir cualquier valor (entero, booleano, archivo, nombre, valor real, estado, string, o variable) a string.

Formato:

```
$STRREP
  (parameter
    radix)
```

Parametros:

parameter
Entero, valor booleano, archivo, nombre, valor real, estado, string o variable que desea convertir a string. Este parámetro es requerido.

radix
Raíz o base asociada con un valor entero. La raíz no es parte de un valor string creado; el valor de la raíz por omisión es el decimal.

Ejemplos:

- El siguiente ejemplo convierte un entero 256 a hexadecimal y regresa el string resultante:

```
/display_value $strrep(256,16)
1ØØ
```

- El siguiente ejemplo convierte el valor booleano TRUE a su representación en string en caracteres mayúsculos:

```
/display value 'El resultado es'..
..///$strrep(true)//'.'
El resultado es TRUE.
```

- El siguiente ejemplo convierte el número hexadecimal ØFF a decimal:

```
/display value $strrep(ØFF(16))
255
```

$SUBSTR

Propósito: Regresar una porción específica (substring) de un string.

Formato:

```
$SUBSTR
  (string,
   position,
   length,
   character)
```

Parametros:

string
String del cual se extraerá un substring - - deseado. Este parámetro es requerido.

position
Posición en el string donde se desea iniciar el substring. Este parámetro es requerido.

length
Longitud (en caracteres) del substring que - se desea obtener. Por omisión la longitud - es 1.

character
Caracter utilizado para rellenar el substring cuando el string por completo es menor que - la longitud requerida para el substring. El substring es rellenado a la derecha con el - caracter que se indique. El valor por omi-- sión es el caracter espacio de la tabla - - ASCII..

Comentarios:

Para utilizar $SUBSTR en su forma más simple proporcione el string y la posición inicial del substring. $SUBSTR regresa el caracter que se encuentre en la posición indicada.

Ejemplos:

. El siguiente ejemplo acepta la longitud -- por omisión de un caracter para el subs- tring. Regresa un string de un caracter - empezando en el decimo caracter del string. 'abcdefghijklm'.

```
/display_value $substr('abcdefghijklm',10)
 j
```

. El siguiente ejemplo indica un substring - de tres caracteres de longitud. Regresa - un string de 3 posiciones a partir del dé- cimo caracter del string 'abcdefghijklm'.

```
/display_value $substr('abcdefghijklm',10,3)
 jkl
```

. El siguiente ejemplo indica el caracter -- con el cual rellenar el substring cuando - este sea mayor en longitud que el string. Regresará un substring de ocho posiciones iniciando en el décimo caracter del string 'abcdefghijklm' e indicando que el carac-- ter de relleno es el '-'.

```
/display_value $substr('abcdefghijklm'10,8,'-')
 jklm-----
```

. En el siguiente ejemplo, la función $DATE se utiliza para obtener un substring con - la función $SUBSTR.

La fecha juliana actual es el valor que - se imprime.

```
/display_value  $date(ordinal)
1987087
/display_value  $substr($date(ordinal),5,3,)
Ø87.
```

$TERMINAL_MODEL

Propósito: Regresa el número del modelo de la terminal.

Formato:
```
$TERMINAL_MODEL
  (file)
```

Parametros:

file
Nombre de un archivo de terminal asociado - con la terminal.

Comentarios:

- Para que esta función regrese el modelo - de la terminal, el modelo primero debe es tablecerse con el comando CHANGE_TERMINAL_ _ATTRIBUTES.
- El número de modelo que regresa es un - - string. Si no se ha definido un modelo - de terminal, un string nulo se regresará.

Ejemplos: El siguiente ejemplo consulta en el sistema para obtener el nombre del modelo de terminal:

```
/display_value  $terminal_model
DEC_VT220
```

$TIME

Propósito: Regresa a la hora actual en string.

Formato:
```
$TIME
  (parameter)
```

Parametros:

parameter
Forma en la cual la hora será regresada. -- Utilice alguna de las siguientes claves:

AMPM
Regresa la hora en terminos de 12 horas - del reloj con la designación AM o PM, como en el ejemplo:

```
6:38 PM
```

HMS
Regresa la hora en terminos de 24 horas, como en el ejemplo:

```
18:38:14
```

La designación de 24 horas para el formato de la hora se obtiene sumando 12 a cada hora después del mediodía y conside--

rando la medianoche como 00:00:00.

ISOT
Regresa la hora en formato horas, minutos, segundos y centésimas de segundo, como en el ejemplo:

18.38.14,79

MILLISECOND (MS)
Regresa la hora en formato horas, minutos, segundos y milesimas de segundos como en - el ejemplo:

18.38.14,796

DEFAULT
Regresa el formato por omisión en la hora. Actualmente es HMS.

Ejemplos: El siguiente ejemplo despliega la fecha y - hora actual:

```
/display_value $time(ampm)//'on'//$date(month)
6:38 PM on March 28, 1987.
```

$VNAME

Propósito: Convertir un string a nombre de variable.

Formato:
```
$VNAME
  (string)
```

Parametros:
string
String que se desea convertir a nombre de variable. Este parámetro es requerido.

Comentarios: Esta función hace posible referenciar a -- una variable vía string.

Ejemplos: Los siguientes ejemplos crean una variable entera con nombre COUNT y una variable - - string llamada COUNT_POINTER. Para acce-- sar el valor de COUNT, la función $VNAME - evalua el valor en COUNT_POINTER.

```
/COUNT = 10
/COUNT_POINTER='count'
/display_value $vname (count_pointer)
10
```

CONCLUSIONES

SCL es un lenguaje demasiado amplio que inclusive no tiene límites, los límites se encontrarían en última instancia en quien utilice - el equipo de CONTROL DATA (CYBER), ya que SCL ofrece la posibili-- dad de un crecimiento según a las necesidades de cada usuario.

SCL da un giro en los lenguajes de programación, ya que es lo su-- ficientemente inteligente como para tomar una decisión adecuada o programada en caso de que se presente alguna falla o irregulari- - dad en la ejecución de procesos de un sistema en general, cosa que la mayoría de los lenguajes no estan capacitados para hacer.

La posibilidad de que existan lenguajes similares o parecidos a -- SCL no es muy remota, de hecho ya existen. Lo interesante aquí -- sería un analisis de lenguajes para ver los pros y contras de cada uno de ellos, con una sola finalidad: obtener un lenguaje ideal o más perfecto.