Compute (Bridgend) Ltd

SELCOPY C++ Version (SLC) 3.30 New Features

8 Merthyr Mawr Road, Bridgend, Wales UK CF31 3NH

Tel: +44 (1656) 65 2222
Fax: +44 (1656) 65 2227

CBL Web Site - http://www.cbl.com

This document may be downloaded from http://www.cbl.com/documentation.php

CBL Ref: z:\hst\dist\all\snf330c\SELCOPY_CPP_3.30_New_Features.pdf

Contents

SELCOPY C++ Version (SLC) Release 3.30 New Features

Documentation Notes

CBL Ref SNF330c001 (this document) describes changes, new features and fixes introduced in SLC (SELCOPY C++) 3.30 Build Level 001. Note that these changes have not yet been included in the full SELCOPY manual.

Installation packages for the latest GA releases of SLC for each supported platform are available via the CBL web site (http://www.cbl.com) and ftp server (ftp://ftp.cbl.com).

Product packages are removed when no longer in support.

The **SELCOPY C++ version** for z/OS (MVS) and z/VM (CMS) operating systems is provided as an executable load module, SLC, which is included as part of the **SELCOPY Product Suite**, available for download and install from the CBL web site **SELCOPY Downloads** page (URL: http://www.cbl.com/selcdl.php). Corrective service is provided in the form of new build levels via z/OS SMP/E SYSMODs or a VM/CMS VMARC software update bundle.

The **SELCOPY C++ version** for IBM i, UNIX and Windows operating systems is provided as downloadable .TGZ and .ZIP archive files from the CBL web site **SELCOPY Downloads** page.

The following publication documents operation of the SELCOPY program for both the mainframe BAL (Basic Assembler Language) version and the C++ version. It is available in Adobe Acrobat PDF format from the CBL web site **Documentation** page:

- SELCOPY 3.30 User Manual

The following generic terms are used throughout this document:

MVS	-	IBM z/OS, OS/390, MVS/ESA, MVS/XA, MVS/SP, OS.
CMS	-	IBM z/VM, VM/ESA, VM/XA, VM/SP.
AIX	-	IBM AIX
DEC	-	HP Tru64
HPX	-	HP HP-UX
LNX	-	Linux x86 (RHEL or SLES)
LNZ	-	z/Linux (RHEL or SLES)
SCO	-	UnXis (SCO) UnixWare
SUN	-	SUN Sparc Solaris
WNT	-	MicroSoft Windows x86 (NT, Vista, 7, Server 2008)
AS/400	-	IBM i, i5/OS, OS/400
UNIX	-	AIX, DEC, HPX, LNX, LNZ, SCO and SUN.
PC	-	x86 servers or workstations running MicroSoft MS-DOS or Windows.
ALL	-	AIX, DEC, HPX, LNX, LNZ, SCO, SUN, MVS, CMS, AS/400 and WNT.

Overview

Overview on the evolution of the C++ version of the SELCOPY Batch utility product.

Platforms

This document applies to the following platforms:

- UNIX.   (For supported platforms please see Documentation Notes above.
- Microsoft Windows.
- IBM Mainframe z/OS (MVS) and z/VM (CMS).

Background

SELCOPY for the IBM mainframe, first released in 1971, is written in BAL (Basic Assembler Language) and is ongoing.

SELCOPY for the various UNIX and PC platforms and the, first released in 1996, is written in C++ for the benefit of portability between platforms. It has essentially the same syntax as the BAL version.

Certain features in the C++ version now more than justify its release to mainframe users, giving them many benefits, in particular, the ability to read lists from the SELCOPY Interactive environment.

The Future

The BAL version for the mainframe will ultimately be phased out, but not until the C++ version has been adapted to call BAL subroutines for all the critical, cpu-intensive, parts of the product. Thus the raw power of the BAL version will be retained, with the C++ overhead apparant only in the control card analysis at start up.

Program Naming

The IBM Mainframe C++ version will be known as **SLC**.
The IBM Mainframe BAL version will remain as **SELCOPY**.
The IBM i, UNIX and Windows C++ versions may remain as SELCOPY, or be renamed to SLC, depending on the installation's preference.

SELCOPY C++ Version Advantages

- Reads lists as provided by SELCOPY/i. e.g. Vols, DataSets, Members, Queues, Allocs.
- Command line invocation can provide all control statements on the PARM field.
- Case insensitive compare.
- Reverse scan.
- CSV (Comma Separated Variables) support.
- TYPE=C and mixed TYPE arithmetic.
- INCLUDE statement.
- CVDATE statement for converting date formats.
- HEX offsets supported.
- DECLARE variables
- Multiple fields on a PRINT statement.
- CHANGE statement.
- RGX giving the power of unix-type Regular Expressions for IF and CHANGE statements.

SELCOPY BAL Version Advantages

- Raw speed.
- DB2, IMS, ADABAS support.

Recent History

Platform	SELCOPY (SLC) Release	Build Level	Latest Change	Publish Date	Comments
MVS Mainframe	3.00	001	2010/12/01 23:16	2010/12/16	GA. s300 Product Suite dist
WNT Windows	3.00	004	2011/03/01 22:32	2011/03/01	GA.
WNT Windows	3.00	006	2011/05/25 22:28	2011/06/08	GA.
LNX Linux	3.00	007	2011/06/17 11:38	2011/06/22	GA.
ALL SNF300c	3.00	007		2011/06/24	GA. Documentation.
MVS Mainframe	3.00	009	2012/03/04 21:28	2012/03/15	GA. s310 Product Suite dist
MVS Mainframe	3.10	001	2012/04/10 17:51	2012/04/12	GA. (sysmod)
MVS Mainframe	3.10	004	2012/09/21 17:10	2012/09/25	GA. (sysmod)
CMS Mainframe	3.10	004	2012/09/12 17:48	2012/09/20	GA.
WNT Windows	3.10	004	2012/09/12 17:48	2012/09/20	GA.
ALL SNF310c	3.10	004		2012/09/24	GA. Documentation.
For ease of reference, all new features in the SNF300c are repeated in SNF310c. Thus, SNF310c covers all new features, up to SLC 3.10 Build 004, that are not yet officially published in the SELCOPY manual.					
CMS Mainframe	3.10	005	2012/12/13 15.12	2012/12/19	GA.
HPX HP-UX	3.10	006	2013/01/27 21.58	2013/02/08	GA.
LNX Linux	3.10	007	2013/05/10 12:06	2013/05/10	GA.
MVS Mainframe	3.10	008	2013/05/27 17:12	2013/06/21	GA.
CMS Mainframe	3.10	008	2013/05/27 17:12	2013/06/21	GA.
WNT Windows	3.10	008	2013/05/27 17:12	2013/06/21	GA.
LNX Linux	3.10	008	2013/05/27 17:50	2013/06/21	GA.
MVS Mainframe	3.20	001	2013/10/19 22:09	2013/10/22	GA. s320 Product Suite dist
CMS Mainframe	3.20	001	2013/10/19 22:09	2013/10/22	GA.
WNT Windows	3.20	001	2013/10/19 22.09	2013/10/22	GA.
LNX Linux	3.20	001	2013/10/19 22:09	2013/10/22	GA.
ALL SNF320c	3.30	001		2013/10/23	GA. Documentation.
MVS Mainframe	3.30	001	2015/02/19 14:40	2015/04/22	GA. s330 Product Suite dist
CMS Mainframe	3.30	001	2015/02/19 14:40	2015/04/22	GA.
WNT Windows	3.30	001	2015/02/19 14:40	2015/04/22	GA.
LNX Linux	3.30	001	2015/02/19 14:40	2013/10/22	GA.
ALL SNF330c001	3.30	001		2015/04/22	GA. Documentation.

Important Changes

There are 2 important changes which can give different results from earlier releases of SLC, the C++ version of selcopy:

- Repeating keystrokes with [TIMES nnn]   (Windows platform ONLY.)
- CMS Keyed read of an ESDS or RRDS on VSE (VM/CMS platform ONLY.)

These IMPORTANT CHANGES are discussed below.

Repeating keystrokes with [TIMES nnn]

2014/05/21   s320_003

ONLY for MS Windows platform.

SPECIFICATION CHANGE: Use of `[TIMES=n]` for Key Repetition within a string to be written to a specified window as a set of keystrokes HAS BEEN CHANGED.

The *nnn* argument of the TIMES keyword no longer represents the number of times to repeat the previous keystroke.

Instead, the *nnn* argument of the TIMES keyword represents the number of times the previous keystroke is to be ACTIONED IN TOTAL.
e.g.

```
  OPT   KEYENC = "[]"          * Unnecessary as '[]' is the default.
  WRITE  WIN='Title'  'abc [times 11]def [x 9][CR]'

                               * Will write "abc" followed by 11 blanks,
                                                followed by "def",
                                                followed by 9 blanks and
                                                followed by ENTER key.
```

Note that `[X=n]` is a synonym for `[TIMES=n]`.
e.g.

```
  WRITE  WIN='Title'  '[right][times 22]'   * Move cursor 22 places right.
  WRITE  WIN='Title'     '[ri][x 22]'       * Same as above.
```

CMS Keyed read of an ESDS or RRDS on VSE

2015/04/09   s330_001

ONLY for VM/CMS platform.

For the VM/CMS platform only, using the IBM-supplied CMS/VSAM interface for accessing VSE/VSAM files, which is now "Out of Support" by IBM, the keyed read of an ESDS or RRDS, by RBA or REC respectively, no longer points to the requested VSAM record, resulting in the selcopy message ERROR 544.

Keyed read of a KSDS still operates correctly.

The problem is under review by IBM, ref no: PMR 38722,999,866, which was opened on 2015/02/04.

Fixes

Fix 01 - Repeating Combined Keys (Windows)

2013/11/10 s320_002

ONLY for MS Windows platform.

When writing keystrokes to a different window, use of "`[TIMES nnn]`" to define the number of times in total the preceding keystroke is to be actioned, was only successful if the previous keystroke were a single key depression.

If the previous keystroke involved a combination of keys, then only the first key of the combination was used for the nnn repeated keystrokes.
e.g.

```
  wr WIN=abc   '[SHIFT [TAB]]'  '[TIMES 3]'  * Write to the window which
                                             *       has abc in its title.
  wr WIN=abc   '[SHIFT [TAB]][TIMES 3]'      * Same as above,  but less
                                                                readable.
```

In the above example, on earlier releases, the `'[SHIFT [TAB]]'` worked fine for the 1st time, but subsequent repeats only operated the SHIFT key, ignoring the TAB key.

This has now been corrected so that both the SHIFT key and the TAB key are depressed and released properly every time.

Fix 02 - READ LIST=*list-cmd* - File geom (MVS)

2014/09/11 s320_004

Records returned for the pseudo file, list-cmd, are always of fixed length.

However, on earlier releases, LIST input records were reported and treated as RECFM=U, so if the list-cmd happens to be the prime input file, and an existing output file with different geometry is used, the resultant output file could have records truncated.

To overcome this problem on earlier releases, it was necessary to code both RECFM and LRECL on the output file.

Starting with SLC 3.20 Build 004, LIST files are reported accurately in the summary.
e.g.

```
"   77 F"
```

under the LRECL column, indicating RECFM=F and LRECL=77.

New Features and Other Changes

Additional DCL data types: NTS,VCH,CHV

2013/11/22 s320_002

The DCL statement has been extended to support 3 additional character data types, NTS, VCH and CHV, where the length of the data is variable.

```
              NTS | Z|ZC|ZCH|ZCHA|ZCHAR | CHZ|CHAZ|CHARZ | CSTR|CSTRING

DCL varname VCH | V|VC|VCH|VCHA|VCHAR|VARCHAR                    [ (n1) ]

              CHV | CHARV|CHARVARYING
```

For each of the 3 data types and their synonyms listed above:

- The number, (n1), defines the maximum length of data that may be held in the variable. It does not include any prefix or terminator defining current length. Default if (n1) omitted is 127 for NTS and 126 for VCH or CHV.
- Reference to such a variable returns ONLY the data of that variable and ONLY the data for its current length.
- INIT and POS parameters are supported as for the CHA data type. However, the FORMAT parameter is not appropriate, so is disallowed.

DCL varname NTS (*n1*) [INIT '*initial string*']

A DCL variable of type NTS holds a "Null-Terminated-String" which is a data string whose length is defined by a x'00' terminator.
e.g.

```
dcl  zvar   NTS  (100)    ini "XYZ"

print 'zvar is -->'  zvzar  '<--'     * Gives:  "zvar is -->XYZ<--"

zvar = 'ABCDEF'
print 'zvar is -->'  zvzar  '<--'     * Gives:  "zvar is -->ABCDEF<--"
```

DCL varname VCH (*n1*) [INIT '*initial string*']

A DCL variable of type VCH holds a string whose length is defined by a 2-byte binary prefix.

Unused data, up to the maximum data length, is left unchanged.
e.g.

```
dcl  vvar   VCH  (100)    ini 'XYZ'

print 'vvar is:-->'  vvzar  '<--'     * Gives:  "vvar is:-->XYZ<--"

vvar = 'ABCDEF'
print 'vvar is:-->'  vvzar  '<--'     * Gives:  "vvar is:-->ABCDEF<--"
```

DCL varname CHV (*n1*) [INIT '*initial string*']

A DCL variable of type CHV is identical to a VCH variable, with 2 exceptions:

1. Unused data is padded with the FILL char.
2. When written as part of a structure, the 2-byte binary prefix and the full maximum data length of the variable is written. However, DCL structures are not supported by SLC C++, so the CHV data type is best avoided for the time being.

MATCHLEN support for IF stmt

2014/04/08 s320_002

All IF-type stmts, (IF, AND, OR, THENIF, ELSEIF), have been enhanced to support the MATCHLEN parameter, enabling the user to get the length of the string matched.

For normal IF-type stmts, the length is constant and already known, but for an RGX (Regular Expression), described below, the length can vary.

The MATCHLEN parameter must have an argument indicating the @pointer or arithmetic DCL variable to receive the resultant length of the string matched.

MATCHLEN and its argument may be coded anywhere following the IF-type keyword provided it does not separate any other parameter from its argument.

For an example of use of MATCHLEN, please refer to the TILDE operator, used for negation within an RGX, which is described below.

CHANGE stmt CASEI support

2014/08/07 s320_003

The CHANGE stmt, which was introduced for SLC C++ 3.20, has been enhanced to support the CASEI parameter which indicates that a CASE Insensitive search for the CHANGE target string is required.

The CASEI parameter may be coded anywhere following the CHANGE keyword or synonym, provided it does not separate any other parameter from its argument.
e.g.

```
dcl          str1 char              init="ABCDEFG ABCDEFG ABCDEFG ABCDEFG"
change casei str1  'abc' 'zz '  * To get "zz DEFG zz DEFG zz DEFG zz DEFG"
change casei str1  'efg' '.'    * To get "zz D. zz D. zz D. zz D.        "
```

In the above example, the declared variable, *str1*, is a fixed length of 31, so after the 2nd CHANGE command, which reduces the target, *str1* is padded out with the default FILL char, blank.

To avoid padding, and to keep the length of str1 true for the data only, the DCL variable str1 can be declared with data type NTS (Null Terminated String) instead of data type CHAR.

Synonyms for NTS are: Z, ZC, ZCH, ZCHA, ZCHAR, CHZ, CHAZ, CHARZ,

CHANGE stmt Synonyms, C, CH and CHG

2014/08/07 s320_003

New synonyms, C, CH and CHG, have been introduced for the CHANGE command.
e.g.

```
dcl          str1 cha               ini "ABCDEFG ABCDEFG ABCDEFG ABCDEFG"
c   str1  casei    'abc' 'zz '  * To get "zz DEFG zz DEFG zz DEFG zz DEFG"
ch  str1   'efg' casei   '.'    * To get "zz D. zz D. zz D. zz D.        "

p 1 =                                   "ABCDEFG ABCDEFG ABCDEFG"
chg     "ABC" times 2    "abc"  * To get "abcDEFG abcDEFG ABCDEFG"
                                * Only 1st 2 occurrences on curr rec.
c 80 at 1  "EFG" "efg" times 1  * To get "abcDefg abcDEFG ABCDEFG"
```

In the above example, no workarea is mentioned, but an 80 byte blank filled workarea is still made available by default and is treated as the current record until a READ statement is actioned.

WIN RESET support (Windows)

2014/08/07 s320_003

ONLY for MS Windows platform.

When SLC is used to send keystrokes to a different window, the window in focus is changed to that window and when SLC terminates the focus is left on that window.

This is normally what is required by the user, for example using a SLC run to enter the keystrokes for logging in to some service, such as z/OS TSO.

However, as more uses of the keystrokes technique evolve, there are times when it is inconvenient to lose the focus from the initiating window.

WIN RESET has therefore been supported to allow the user to re-focus on the window that originally had focus when SLC was invoked.

WIN RESET will write no data to the window, and if any data field is provided it is silently ignored.
e.g.

```
wr win='My Editor"  '[cr]some command or other[cr]'   \
                    'some different command[cr]'      \
                    '[tab][times 4][right][times 22]' \
                    'abc-xyz[cr]'
   *    The following 3 commands all do the same thing.
wr WIN RESET   * Comment: Refocus on original window.
WIN    RESET   * Comment: Refocus on original window.
wr WIN RESET    "Data:    Gets silently ignored."
end
```

In the above example, a series of literals are sent to the window which has the string "My Editor" (case insensitive) somewhere within its title bar.

Note the use of '\' (backslash) to indicate a continuation line follows. Thus the first 4 lines are a single statement with multiple fields. The fields are concatenated with no intervening blanks. If a blank is required, it must be included within a field.

RGX parameter for Regular Expressions

2014/02/01 s320_002

The long awaited power of unix-type Regular Expressions (RGX) for SELCOPY has finally been provided in the SLC C++ version.

Non-unix programmers may well be unfamiliar with RGX usage and its complexity can appear very daunting.

However, the brevity and simplicity of an RGX, defining a very complicated set of conditions to be matched on a scan, will make it well worth the initial time spent reading on.

Various flavours of RGX syntax have existed for years in the unix arena where RGX was spawned, so it is not possible to match all of them.

KEDIT, by Mansfield Software Inc, provides a powerful set of RGX operators which follow as much as possible the syntax of existing unix implementations and CBL has chosen to follow the KEDIT syntax.

For information on KEDIT, please see http://www.kedit.com.

RGX for SLC C++

An RGX may be used on either an IF-type command (IF AND OR THENIF ELSEIF) or on a CHANGE command.

When an RGX is used on an IF-type command, it automatically implies a range test which will set a pointer to the start of the string matched in the data field.

The keyword RGX, or a synonym (REGEXP REGX), must be coded anywhere following the command, provided it does not separate any other parameter from its argument.

RGX coded on the IF or CHANGE means that the search string for the operation is not just a normal string. It will contain certain special characters, used as operators, to define more complicated conditions to be matched.

To whet the appetite for reading on, the following reduces the need for 5 change commands to 1 by using the [and] operators of an RGX to indicate that any of the enclosed characters will satisfy the search.
e.g.

```
dcl    alf  cha                           ini "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
change alf      rgx '[AEIOU]'  '-'   * Gives: "-BCD-FGH-JKLMN-PQRST-VWXYZ"
```

But there is .... much .... much .... more ....

RGX Operators

The full list of the special chars used as operators for an RGX is:

```
\  ^  $  [  ~  -  ]  (  |  )  {  }  &  +  *  @  #  ?  :
```

Any character, other than the above operator characters, is considered normal text as on a non-RGX scan.

Each operator is discussed below.

ESCAPE Character \ (BackSlash)

In order to treat an RGX operator as normal text data in an RGX, a backslash is required immediately preceding the operator that needs to be escaped.
e.g.

```
dcl     data      charz                ini = "ABC[[DEF]]G"
change  data  '[\[\]]'  null         * Gives: "ABCDEFG"
```

In the above example, the 1st '[' is a genuine operator defining the start of a class of characters any of which will give a match.

The 2nd '[' is escaped, so is just the 1st char of data within the class.

The 1st ']' is also escaped, so again is just data within the class.

The 2nd ']' is not escaped, so is the operator defining the end of the class.

In fact, with a few exceptions, any character may be escaped, even if it is not a genuine RGX operator. So if in doubt, and it's a special char, escape it.

The exceptions, which each have a specific meaning, are all alpha:

```
\a        Apostrophe (Single Quote)                   (Not yet implemented.)
\q        Double Quote                                (Not yet implemented.)

\b    BS  BackSpace         ASCII X'08'     EBCDIC X'16'
\f    FF  FormFeed          ASCII X'0C'     EBCDIC X'0C'
\r    CR  Carriage Return   ASCII X'0D'     EBCDIC X'0D'

\l    NL  New Line                          EBCDIC X'15'  (Not implemented.)

\n    LF  LineFeed          ASCII X'0A'     EBCDIC X'25'
\t    HT  Horizontal Tab    ASCII X'09'     EBCDIC X'05'

\xnn      Hexadecimal character code X'xx'

\nnn      Octal character code nnn     e.g. \072 or \72 (equivalent to \x3A)
```

CARET Character ^ Circumflex - Power Factor

The caret character is used as an operator in an RGX for 2 distinct functions:

1. If caret is the 1st char of the RGX, the caret operator means that whatever follows in the RGX must match the field being checked, but it must match starting at POS 1 of the field.
2. If caret is NOT the 1st char of the RGX, the caret operator means that a POWER FACTOR is to be applied to the previous character or group. (See below for discussion of group.) The number of times to repeat matching the previous char or group must be supplied in the RGX immediately after the caret as a decimal integer, terminated by the first non-numeric char. Leading zeros are permitted, but max power factor allowed is 999.

e.g.

```
dcl  irec         chz(80)         * Another NTS. (Null Terminated String.)
read 'xyz.test.file'   into irec
if   irec = rgx '^[BKWX]^3'
  then print   'irec = "'   irec    '"'   s=20   * Stop after 20 hits.
```

In the above example, the RGX '^[BKWX]^3' used on the IF statement makes use of the caret for 2 different purposes:

The 1st caret means a match will only occur if 1 of the chars in the class [BKWX] is found at POS 1.

The 2nd caret is a power factor, meaning matching of the previous character, which in this case is the class [BKWX], must be done a total of 3 times.

Thus, we will select only those records where the 1st 3 chars are members of the class [BKWX].

DOLLAR Character $ (Dollar Sign)

The dollar sign, not the currency symbol, means that the remainder of the RGX must not only match the data in the field being scanned, but must also have reached the end of the field.
e.g.

```
dcl  irec         chz(80)         * Another NTS. (Null Terminated String.)
read 'xyz.test.file'   into irec
if   irec = rgx '$[BKWX]^003'
  then print   'irec = "'   irec    '"'   s=20   * Stop after 20 hits.
```

In the above example, the only records selected will be those that end with a string of 3 chars, all of which are members of the class [BKWX].

SQUARE BRACKETS [] Class definition

Having already mentioned [ ] above in order to allow simple examples, the full spec for [ ] is:

The [operator in an RGX defines the start of a character class which must be terminated with a] operator.

All characters enclosed within the [and] operators, with 2 execptions, are considered to be members of that class. The exceptions are:

1. ~ (Tilde) is used to negate the class. (See below.)
2. - (Minus) is used to specify a range of chars. (See below.)

A character class is matched against a single char in the field being scanned and is considered to be successful if the char in the field matches any of the chars in the class.
e.g.

```
dcl    wk2  cha                          ini "ABCzzzGHIzzzMNOzzzSTUzzzYZ"
change wk2 rgx '[AEIOU]z^3'  '-yyy'  * Gives: "ABCzzzGH-yyyMN-yyyST-yyyYZ"
```

In the above example, Czzz is not changed, as C is not in the class [AEIOU].

TILDE ~ Negation

The tilde character represents the logical NOT operator and it negates the result of the matching of the single character or sub-RGX which follows it.

There are 2 types of negation:

1. If the tilde is the 1st character within an RGX class specification such as [~AEIOU] then the character at the current position in the source data field is considered to be a match if that character is NOT a member of the specified class. In other words, the class represents all possible 256 characters, with the exception of those mentioned in the negated class. The source field focus is then ADVANCED by 1 char and further matching continues with the next operator or text in the RGX.

   Use of tilde within an RGX class is restricted to accepting the tilde only in the 1st position of the class, otherwise an error is given.

2. If the tilde is NOT within an RGX class, then it means that the matching will be successful only if the current source data char is NOT the same as the char following the tilde in the RGX.

   However, the source field focus is NOT ADVANCED by 1 char, because in reality nothing was matched. It has only been established that the char does not match the RGX char, resulting in TRUE, so further matching continues with the next operator or text in the RGX, without advancing the source field focus.

   If the tilde is followed by a sub-RGX such as a class specification or a group, (see below for a group), then the same thing applies. A false result will be negated, giving a true result, but the source field focus is NOT ADVANCED.

The following example uses the ? operator (Question Mark) which means match any character. (See below for full info on the ? operator.)
e.g.

```
opt dw=66
dcl data3 c    ini "Source data Aa Ad Ad Ae Afff Ag Abxxx"
dcl off3  bin  format '+99'            * Offset from data3 to target found.

dcl rgx3  c    ini "A[~adefg]?"        * Result="Abx"  at offset +32
dcl rgx4  cha  ini "A~a~d~e~f~g?"      * Result="Ab"   at offset +32
dcl rgx5  char ini "A~[adefg]?"        * Result="Ab"   at offset +32

if data3 = rgx3 rgx  ptr=@found  matchlen=@mlen   !t do showit  rgx3
if data3 = rgx4 rgx  ptr=@found  matchlen=@mlen   !t do showit  rgx4
if data3 = rgx5 rgx  ptr=@found  matchlen=@mlen   !t do showit  rgx5
eoj

=showit:=  rgxcur
plog '  Data="' data3 '"'
plog 'RegExp="' rgxcur '"'
@beg3 = data3                          * Start of data3 as an @ ptr for arith.
off3 = @found - @beg3
plog 'Result="' fr @mlen at @found '"   at offset '   off3
plog ' '
=ret=
end
```

In the above example, the RGXs are provided in the declared variables, rgx3, 4 and 5.

When run, all 3 RGXs find the target at offset 32, but rgx4 and rgx5 both have a target length of 2, whereas rgx3 has a target length 3.

This is because rgx3 treats 'b' as a positive match of data, due to 'b' not being a member of the negated class A[~adefg] so the match advances the source field. The other 2 RGXs just ensure that the match is false.

MINUS SIGN - Character Range

Character ranges are defined as a range start character, followed by a hyphen (minus sign), followed by a range end character.

Character ranges are for use within a character class only, as a way of reducing the size of an RGX when specifying consecutive characters.

Ranges can be specified in either order, so 0-9 and 9-0 both define the set 0123456789 of numeric digits.

For EBCDIC platforms only:

> Special action is taken when both ends (range limits) are alpha characters of the same case. This is required because non-alpha code points exist within the alpha range of EBCDIC code points.
>
> These EBCDIC non-alpha code points within the alpha range are therefore disregarded when both ends are lower case letters, or both ends are upper case letters.

If a minus sign is used that is not within a character class, the - sign should be escaped to avoid ERROR 224.
e.g.

```
dcl    wk2  cha                          ini "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
c rgx  wk2   '[G-K]'           'o'   * Gives: "ABCDEFoooooLMNOPQRSTUVWXYZ"
c rgx  wk2   '[G-KR-TW-YCEA]'  'o'   * Gives: "oBoDoFoooooLMNOPQoooUVoooZ"
```

In the above example, the 2nd CHANGE command has an RGX specifying 3 ranges, G-K, R-T and W-Y, as well as the idividual chars, A, C and E.

ASTERISK * 0 Minimal Closure - shortest of 0 or more

* (Asterisk) is the minimal closure repetition operator. It applies to the term preceding it and specifies that 0 or more occurrences be included in the match, but only the minimum number necessary to continue matching with the next term in the RGX.
e.g.

```
dcl  irec           z(256)        * An NTS. (Null Terminated String.)
read 'xyz.test.file'   into irec
if   irec = '^ *DCL '      rgx  casei
  then print   'irec = "'   irec    '"'   s=20   * Stop after 20 hits.
```

In the above example, the '^' ensures we start at the beginning of the record, the `' *'` (blank asterisk) ensures we accept a minimum of 0 blanks, or as many as we need, and the final 4 characters must then be matched case independently. So we get only those records that start with 'DCL ' or 'dcl ' or 'Dcl ', etc, regardless of leading blanks.

Note that the Minimal Closure operator '*' will give a successful match of 0 ocurrences if the * is the last char of the RGX. In other words, it will always match regardless of what source data remains. So for the data record. "XYZZZZ", the RGX "XYZ*" will give a match length of 2, for the "XY" only.

PLUS SIGN + 1 Minimal Plus - shortest of 1 or more

+ (Plus sign) is the minimal plus repetition operator. It applies to the term preceding it and specifies that 1 or more occurrences be included in the match, but only the minimum number necessary to continue matching with the next term in the RGX.

Effectively the same as Minimal Closure, except at least 1 occurrence must be matched.

AT SIGN @ 0 Maximal Closure - longest of 0 or more

@ (At sign) is the maximal closure repetition operator. It applies to the term preceding it and specifies that the maximum of 0 or more occurrences be included in the match.

HASH SIGN # 1 Maximal Plus - longest of 1 or more

# (Hash sign) is the maximal plus repetition operator. It applies to the term preceding it and specifies that the maximum of at least 1 or more occurrences be included in the match.
e.g.

```
dcl    wk2  cha                           ini "ABC    HIJ  MN     TUV XYZ"
c rgx  wk2   " #"   " "              * Gives: "ABC HIJ MN TUV XYZ        "
```

In the above example, the CHANGE command reduces all blank strings to 1 blank.

PARENTHESES () Group definition

( ) (Parentheses) operators delimit a sub-RGX which is used to group terms together, which are then considered to be a single term.

The resultant single term may then have any of the repetition operators, '*', '+', '@', '#', '^' applied to it, or the '~' operator to negate it.
e.g.

```
dcl    wk2  cha                           ini "ABABABABABABABABAB, ABABAB"
c rgx  wk2   "(AB)^2" "xxxx" times 1 * Gives: "xxxxABABABABABABAB, ABABAB"
c rgx  wk2   "(AB)#"  "####"         * Gives: "xxxx####, ####            "
```

In the above example, the 1st CHANGE command has a power factor of 2, so scans for "AB" followed by a 2nd "AB", but only operates on the 1st occurrence of "ABAB" due to use of the TIMES=1 parameter restricting it to just 1 hit.

The 2nd CHANGE command operates on as many "AB" strings as possible. No TIMES parameter is given, so all occurrences will be changed.

But, ... groups have more function available than just the above. Please read on about Alternation using the '|' operator.

OR SIGN | Alternation within a Group

| (the OR sign, vertical bar) operator is used only within a group. It separates alternations within the group, where a match on any one of the alternations is sufficient to give a match for the whole group.
e.g.

```
dcl    wk2  cha                           ini "A fox, an owl and the cow."
c rgx  wk2 "( the | a | an )"  " "   * Gives: "A fox, owl and cow.       "
```

In the above CHANGE command, the blanks in the RGX are essential. The RGX could be coded as "((the)|(a)|(an))" to clarify its meaning. However, each alternation in a group is treated as a group of terms by default.

Note that groups may be nested within groups if so required.

BRACES {} Tagged Data definition

{ } (Braces) delimit a tagged sub-expression, known as a tag, which may then be referenced later within the RGX to use exactly the same data as was matched in the source field for that tag.

The RGX is processed left to right and each tag is given a sequence number starting with 1, which can be referenced with &n, where n is the sequence number of the tag required. (See Ampersand & description below.)

Up to 9 sub-expressions may be tagged.

Nesting is allowed, so it is ok to have a tag within a tag, but there are certain limitations when interacting with groups.

Ok to tag any single element of an alternation within a group. You could tag more than 1 individual alternation, or even all of them, but only the alternation that matches will have a real value. All others will still be valid, but will have a null value.

Ok to tag a complete group, or several groups, but you cannot start a tag outside a group and end it within a group that follows. In other words, a tag cannot be terminated in a group unless it started within the same group and a tag within a group cannot cross an alternation boundary, the '|' sign..

AMPERSAND & Reference a tagged data string

&n (Ampersand followed by a digit) refers to a previously tagged string of matched source field data, defined by the { } braces operators as described above.

&n is particularly useful for the CHANGE command, where part of the string matched can vary, but is needed in the replacement string. However, &n may also be used within the RGX to match a string that is identical to the data already matched for a tag.

&1 refers to the 1st tag defined, &2 to the 2nd, and so on up to &9.

&0 refers to the whole of the matched data for the RGX and is made available by default, without the need to define &0 with { } braces in the RGX.
e.g.

```
opt dw=60               * Data width for printing.
dcl   wk3 chz (60)      * A Null Terminated String.
dcl   wk2 cha (26)                      ini "AB1BEFG2GJKLMNOPQR3RUV4VYZ"
wk3 = wk2   * Curr len will be 26.
plog  wk2 ' = Original for both wk2 and wk3.'
c rgx wk2 '{[A-Z]}[0-9]&1'  '---'  * Gives: "A---EF---JKLMNOPQ---U---YZ"
c rgx wk3 '{[A-Z]}[0-9]&1'  '-&0-' * Gives: "A-B1B-EF-G2G-JKLMNOPQ-R3R-U-V4V-YZ"
plog  wk2  ' = Modified wk2.'      * Print and Log to terminal.
plog  wk3  ' = Modified wk3.'
```

In the above example, the 1st CHANGE command operates on wk2 and the RGX specifies 3 chars. The 1st must be uppercase alpha and is tagged, the 2nd must be numeric, and the third char must be the same as &1, the 1st tag defined.

The 2nd CHANGE command operates on wk3 and uses the same RGX, so the hits will be the same as on the 1st change command, but the replacement string uses &0, representing the whole of the string matched, and a minus sign is added as a prefix and suffix, so the wk3 field is enlarged by 2 bytes for each hit.

Print output is:

```
INPUT   SEL SEL                                                              RECORD
RECNO   TOT ID.          1         2         3         4         5         6  LENGTH
-----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0  ------
    0     1   2 AB1BEFG2GJKLMNOPQR3RUV4VYZ = Original for both wk2 and wk3.      80
    0     1   5 A---EF---JKLMNOPQ---U---YZ = Modified wk2.                       80
    0     1   6 A-B1B-EF-G2G-JKLMNOPQ-R3R-U-V4V-YZ = Modified wk3.               80
                ....,....1....,....2....,....3....,....4....,....5....,....6
```

The record length reported is 80 because nothing has been read and 80 is the size of the default workarea.

QUESTION MARK ? Wild char - Accept any char as a match

? (Question Mark) is the WILD CARD character operator which matches any character. So ? is equivalent to the character class [\x00-\xff].

Any of the repetition operators (* + @ # ^) may be used following the wild char in order to match multiple wild chars. Thus, in the same RGX it is possible to match a string which is followed by some other string with anything in between the 2 strings.
e.g.

```
dcl    wk2  cha                          ini "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
plog  wk2 ' = Original wk2.'
c rgx wk2    '[CNW]?#[GRY]'  '-o-'  * Gives: "AB-o-HIJKLM-o-STUV-o-Z    "
plog  wk2 ' = Modified wk2.'        * Print and Log to terminal.
```

In the above example, the CHANGE command has an RGX that looks for the class of letters, C, N or W. If any member of the class is found, the ? operator, with the # repetition operator, tolerates any number of wild chars until it finds the next text specifier, which is the class of letters, G, R or Y.

The RGX will find 3 hits:

```
INPUT   SEL SEL                                                              RECORD
RECNO   TOT ID.          1         2         3         4         5         6  LENGTH
-----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0  ------
    0     1   1 ABCDEFGHIJKLMNOPQRSTUVWXYZ = Original wk2.                       80
    0     1   3 AB-o-HIJKLM-o-STUV-o-Z     = Modified wk2.                       80
                ....,....1....,....2....,....3....,....4....,....5....,....6
```

COLON : Predefined RGX strings provided

Several commonly used RGXs have been supported as predefined RGXs and can be referenced with a shorthand of the predefined RGX operator, colon ":", followed by a single lower case letter.

Predefined RGXs can be used as the RGX itself or as a sub-RGX in a more complex RGX.

Predefined RGXs supported are:

Name	Definition	Description
`:a`	`[a-zA-Z0-9]`	Alphanumeric character
`:b`	`([\t ]#)`	White space (a string of blanks and tabs).
`:c`	`[a-zA-Z]`	Alphabetic character
`:d`	`[0-9]`	Numeric digit
`:q`	`(("[~"]@")|('[~']@'))`	Quoted string (in single or double quotes).
`:w`	`([a-zA-Z]#)`	Word (a string of alphabetic characters).
`:z`	`([0-9]#)`	Integer (a string of numeric digits).

RGX Quick Ref Summary

Operator	Description

\	Escape the next char, treat as text, except for certain standard ESC sequences:
	`\a        Apostrophe (Single Quote)                (Not yet implemented.)`
	`\q        Double Quote                             (Not yet implemented.)`
	`\b    BS  BackSpace          ASCII X'08'  EBCDIC X'16'`
	`\f    FF  FormFeed           ASCII X'0C'  EBCDIC X'0C'`
	`\r    CR  Carriage Return    ASCII X'0D'  EBCDIC X'0D'`
	`\l    NL  New Line                        EBCDIC X'15' (Not yet implemented.)`
	`\n    LF  LineFeed           ASCII X'0A'  EBCDIC X'25'`
	`\t    HT  Horizontal Tab     ASCII X'09'  EBCDIC X'05'`
	`\xnn      Hexadecimal character code X'xx'`
	`\nnn      Octal character code nnn     e.g. \072 or \72 (equivalent to \x3A)`

^	Must be at POS 1 of source data field if '^' is 1st char of RGX.
$	Must have exhaused the source data field.
[class]	Matches any char in enclosed class.
[~class]	Matches any char NOT in enclosed class. The '~' must be 1st in class.
[a-z]	Matches any char within the range 'a' to 'z', or any other range.
( )	Groups any enclosed RGX to be treated as a unit.
(x1|x2|...)	Alternation where any one of the | separated terms gives a match.
	Repetition Operators
*	0 Min Closure - match prev term 0 or as many times as reqd.
+	1 Min Plus - match prev term 1 or as many times as reqd.
@	0 Max Closure - match prev term 0 or as many times as Possible.
#	1 Max Plus - match prev term 1 or as many times as Possible.
^n	Power factor - match prev term n times precisely, if '^' not 1st char of RGX.
~	Not function - succeeds only if next RGX term is NOT matched. Advances the RGX, but not the source data field.
{ }	Tagged expression enables reference, later in the RGX or in the substitution field of a CHANGE statement, to the data matched within the enclosing braces.
&n	References the data matched in the nth tagged expression
?	Wildcard character - matches any single character
:	Treats the colon and the next char as shorthand for a Predefined Expression. The next char must be one of the lower case letters: a b c d q w z

Messages

ERROR Messages - Control Statement Analysis

`E224 REGULAR EXPRESSION SYNTAX`

1. Reserved.
2. Unbalanced parentheses. Group ')' missing.
3. Minus '-' for range is not in a char class.
4. Missing '[' for starting a char class.
5. Missing ']' for char class.
6. Invalid ESCAPE sequence in char class.
7. The tilde must be 1st char in class list.
8. 1st char of range is missing.
9. Invalid opcode char for Char Class.
10. Unbalanced parentheses. Group '(' missing.
11. Power factor > 255 or invalid.
12. Repeat operator not preceded by char data.
13. Operator conflict. e.g. # and ^nnn coded.
14. Invalid ESCAPE sequence in main expression.
15. Tagged data spans an Alternation boundary.
16. Tag close brace '}' without open brace '{'.
17. Tag braces '{' and '}' are in diff groups.
18. &n reference number is invalid.
19. Tilde '~' misplaced. Can't be last.
20. Unknown Predefined Expression letter following ':'.

`E225 ... E269 RESRVD`

All message nembers between E225 and E269 (inclusive) are reserved.

ERROR Messages - Selection Time

`E619 CLIPBOARD TEXT FMT NOT AVAIL`

The Windows function "IsClipboardFormatAvailable" has returned false for "CF_TEXT" format (plain text) indicating it is not avail. Check the system settings.

`E620 CLIPBOARD OPEN FAILED`

The Windows function "OpenClipboard" has returned false, indicating the open clipboard has failed.

`E621 CLIPBOARD READ FAILED`

The Windows function "GetClipboardData" for CF_TEXT has failed to return a clipboard object.

`E622 CLIPBOARD READ LOCK FAILED`

The Windows function "GlobalLock" for the clipboard object has failed to return a pointer to the clipboard data.

WARNING and Information Messages in Summary

Changes to the text of the information messages that may occurr on the summary are:

None.