# A detailed overview of *OpenSCL* from EBE Computing

## Why ***OpenSCL***?

- Network servers have become increasingly powerful over the years to the point where they now rival the performance of mainframe systems
- Network servers provide freedom of choice with regard to vendors *and* the technology they supply
- Network servers providing the same throughput as mainframes *tend* to be more cost-effective
- The temptation to move from mainframes is therefore very great
- However, there is a reluctance to do so because the sophistication, resilience and integrity of mainframe systems appear to be lacking on network servers
- Conversion costs appear prohibitive
- There is uncertainty with regard to the choice of Operating System to migrate to
- The experience of users who have already moved to network servers appear to corroborate these fears

## What is ***OpenSCL***?

- ***OpenSCL*** is a brand-new *operating environment* for OPEN Systems, including Windows-, Linux- and Unix-based servers
- ***OpenSCL*** complements the host operating system and works seamlessly with it
- ***OpenSCL*** is also a tool that will greatly ease migration to an OPEN Systems environment and provide added value on the target platform
- ***OpenSCL*** consists of -
  - **SCL,** a brand-new (scripting) language for Open Systems that builds on the well-established, tried and tested standards with which enterprise users are so familiar
  - **Developer,** an interactive facility for developing and testing programs written in any language and provides a common look across the enterprise

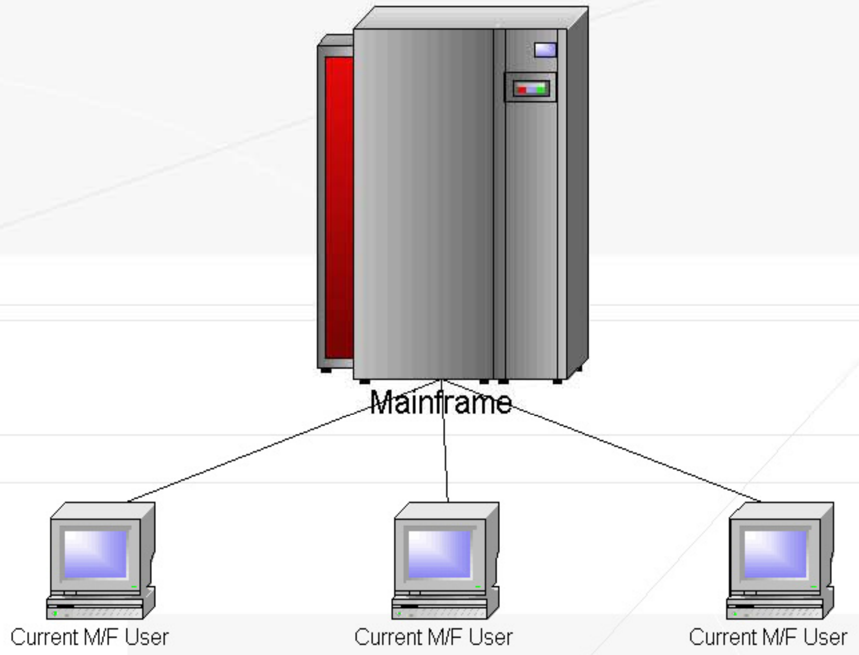
- **Enterprise,** a sophisticated Batch Scheduler for running work in unattended mode across the Enterprise
- **Editor**, a comprehensive editor geared for rapid development of programs and files

## What does ***OpenSCL*** do?

- ***OpenSCL*** provides a common English-like computer language called System Control Language or SCL
- ***OpenSCL*** takes any SCL and translates it to an ANSI-standard C program
- The compiled C program interacts with the ***OpenSCL*** run time system on the target computer, giving the full SCL functionality needed by the C program
- The run time system is bundled with ***OpenSCL*** Developer and Enterprise

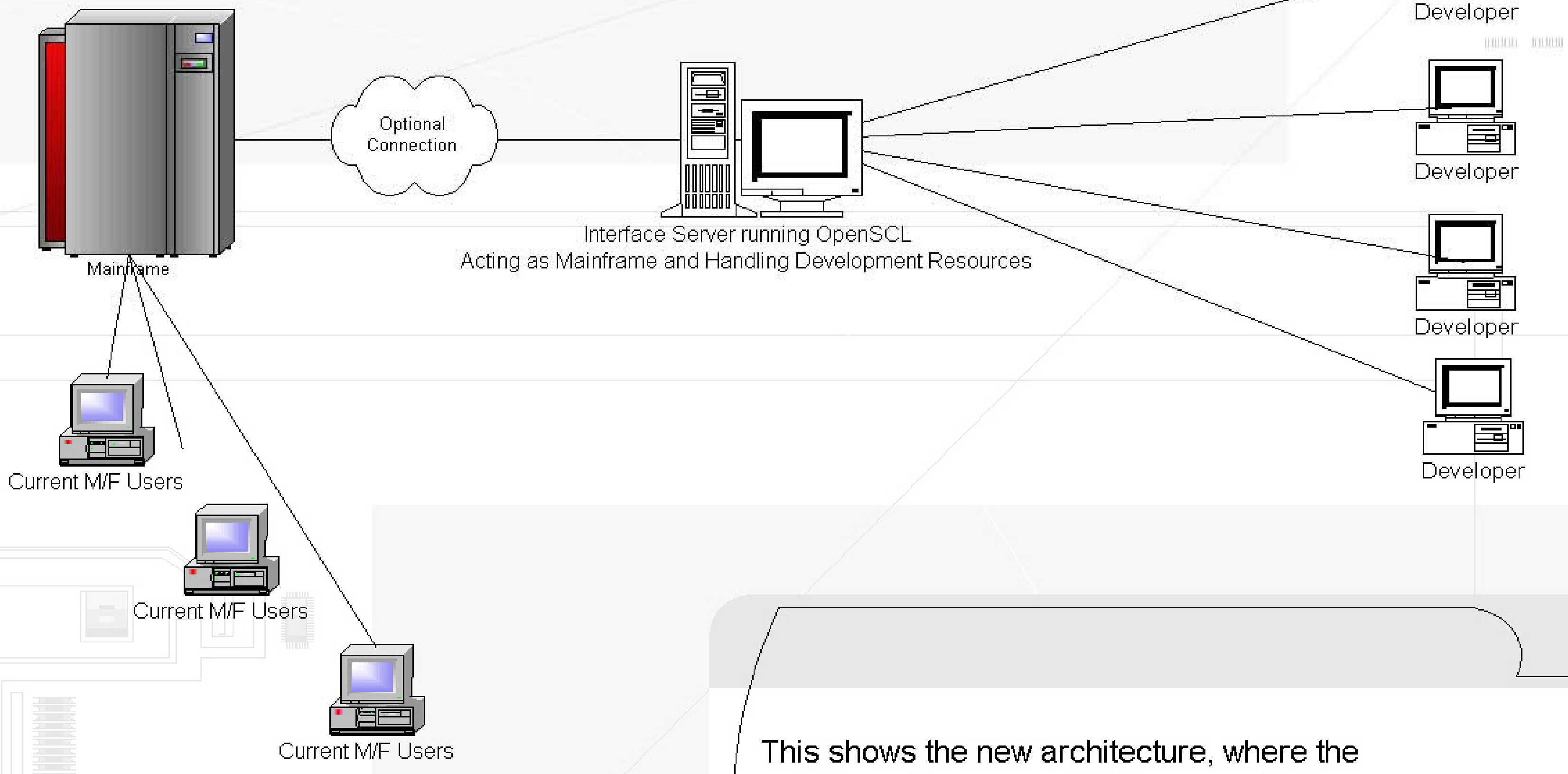
The diagrams on the following pages show ***OpenSCL*** configurations.

## Current Development Architecture

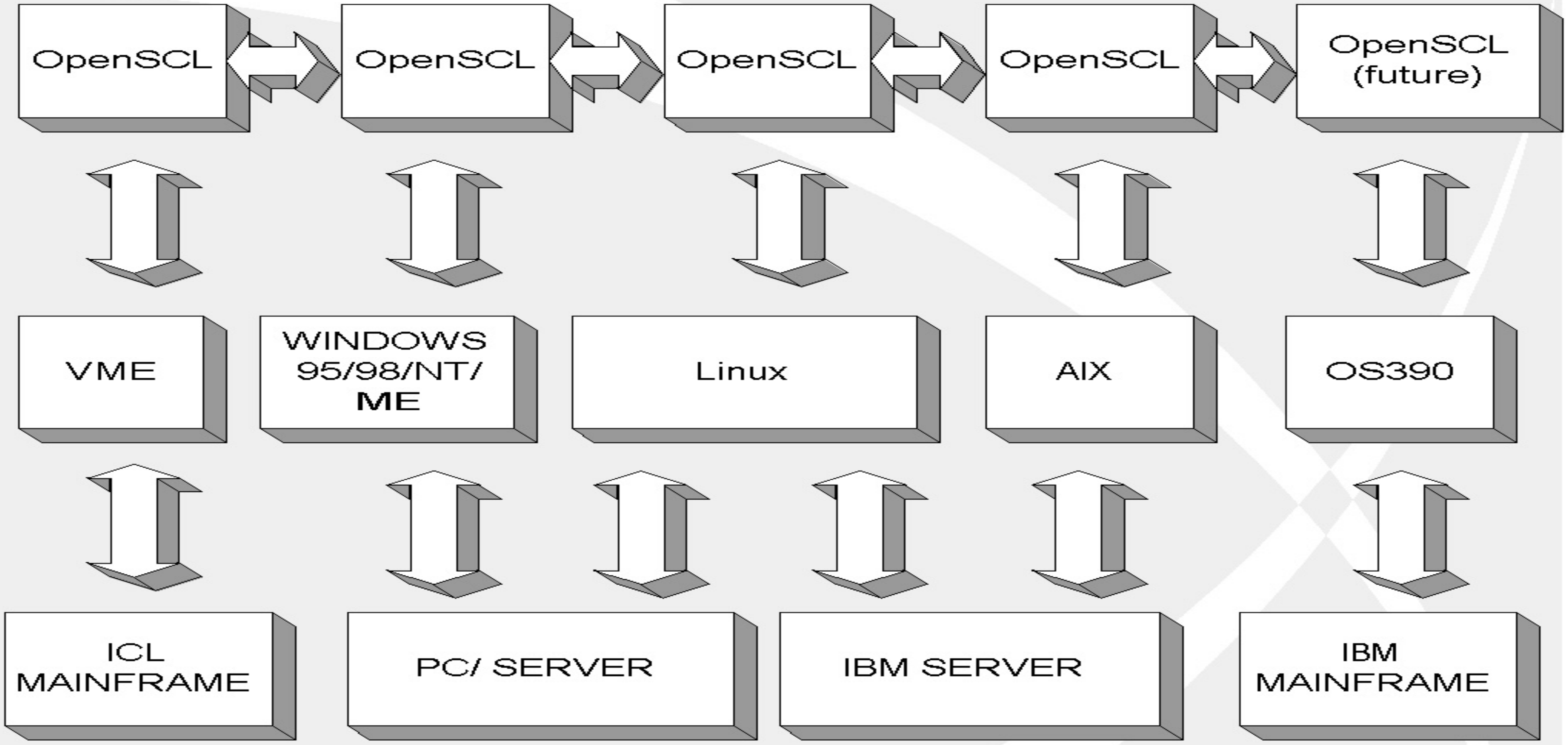


With the current system, development takes place on the mainframe, with current users still accessing it. Development places a heavy toll on resources and will affect performance.

# OpenSCL Architecture



This shows the new architecture, where the mainframe is run on a PC Server and all development and the testing "burden" is passed to the server. Developed work can be tested at any time on the mainframe as well.


OpenSCL
OpenSCL
OpenSCL
OpenSCL
OpenSCL (future)
VME
WINDOWS 95/98/NT/ ME
Linux
AIX
OS390
ICL MAINFRAME
PC/ SERVER
IBM SERVER
IBM MAINFRAME

## What are the benefits?

- For developers, ***OpenSCL*** provides:

  - A low risk development capability
  - Major resource savings in terms of hardware
  - High functionality and performance
  - A highly user-friendly interface with a fast, intuitive development environment
  - Rapid application completion times – complete system testing is facilitated
  - A reduced need to know complex technology
  - A common way to run the same code across the enterprise

- For Users:

  - ***OpenSCL*** provides a single interface in heterogeneous environments
  - An enhanced user interface using run time templates eliminates most errors
  - Multiple Operating Systems can be accessed from a single program
  - Machine resources on the client are therefore minimised
  - No changes are necessary in the user environment - users retain the same interface, eliminating the need for retraining

- For Management:

  - Your investment in SCL is protected - the aim of ***OpenSCL*** is to allow the use of existing SCL across platforms with no manual conversion effort whatsoever
  - Production and development workloads can be split across platforms

    - Production can run, e.g. on Windows Server or AIX
    - Development can run, e.g. Windows 98
    - The processing power of ALL computers on the network can be utilized

- The freedom to choose the best platform
- "Specialist" staff per operating system are no longer an issue - management cannot therefore be "held to ransom"

## Who will benefit?

- Enterprise users who are already using
  - Windows 95/98/NT/2000/XP
  - Linux
  - AIX
- All mainframe users who are moving to the world of OPEN Systems

## What is needed to use ***OpenSCL***?

- The ***OpenSCL*** product will provide you with all you require
- If you are moving SCL from one Operating System to another, a means of transferring the SCL source code, such as compatible tapes or discs, or a file transfer facility, will be necessary
- A "C" compiler, freely available on the Internet, is required
- Under Windows, the C compiler in DevStudio is preferred

## What enhancements are planned?

- Depending on demand, additional target computer platforms will be included, for example, MVS
- Conversion software enabling conversion from other platforms will be supplied, for example, from MVS

## Visual ***OpenSCL***

This is the flagship application that is a complete development environment.

It consists of the following components:

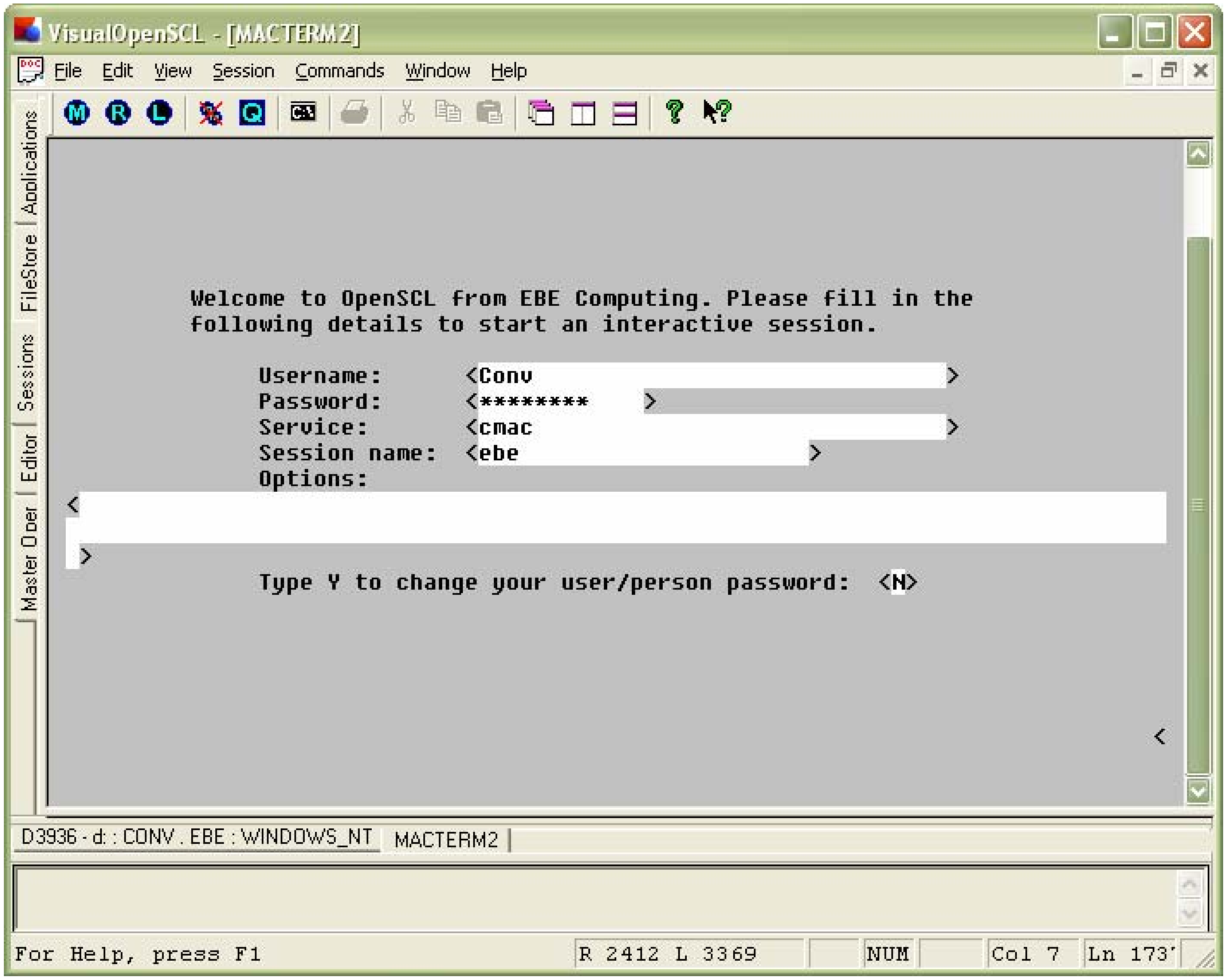
- MAC Sessions
- Filestore view
- SCL Editor
- Applications view
- Master Oper

### MAC Session

Allows a user to login to a secure user. This is the interface that replaces the old green screen that users were used to. The interface:

- Allows remote sessions
- Allows concurrent session access
- Manages remote connections
- Provides secure access
- Has powerful features but yet is easy to use
- Shell commands from the MAC are possible

See the figure below.


VisualOpenSCL - [MACTERM2]
File Edit View Session Commands Window Help
Applications
FileStore
Sessions
Editor
Master Oper

```
Welcome to OpenSCL from EBE Computing. Please fill in the
following details to start an interactive session.

    Username:      <Conv                          >
    Password:      <*********    >
    Service:       <cmac                          >
    Session name:  <ebe                     >
    Options:
<
 >
    Type Y to change your user/person password:  <N>
```

D3936 - d : CONV . EBE : WINDOWS_NT   MACTERM2

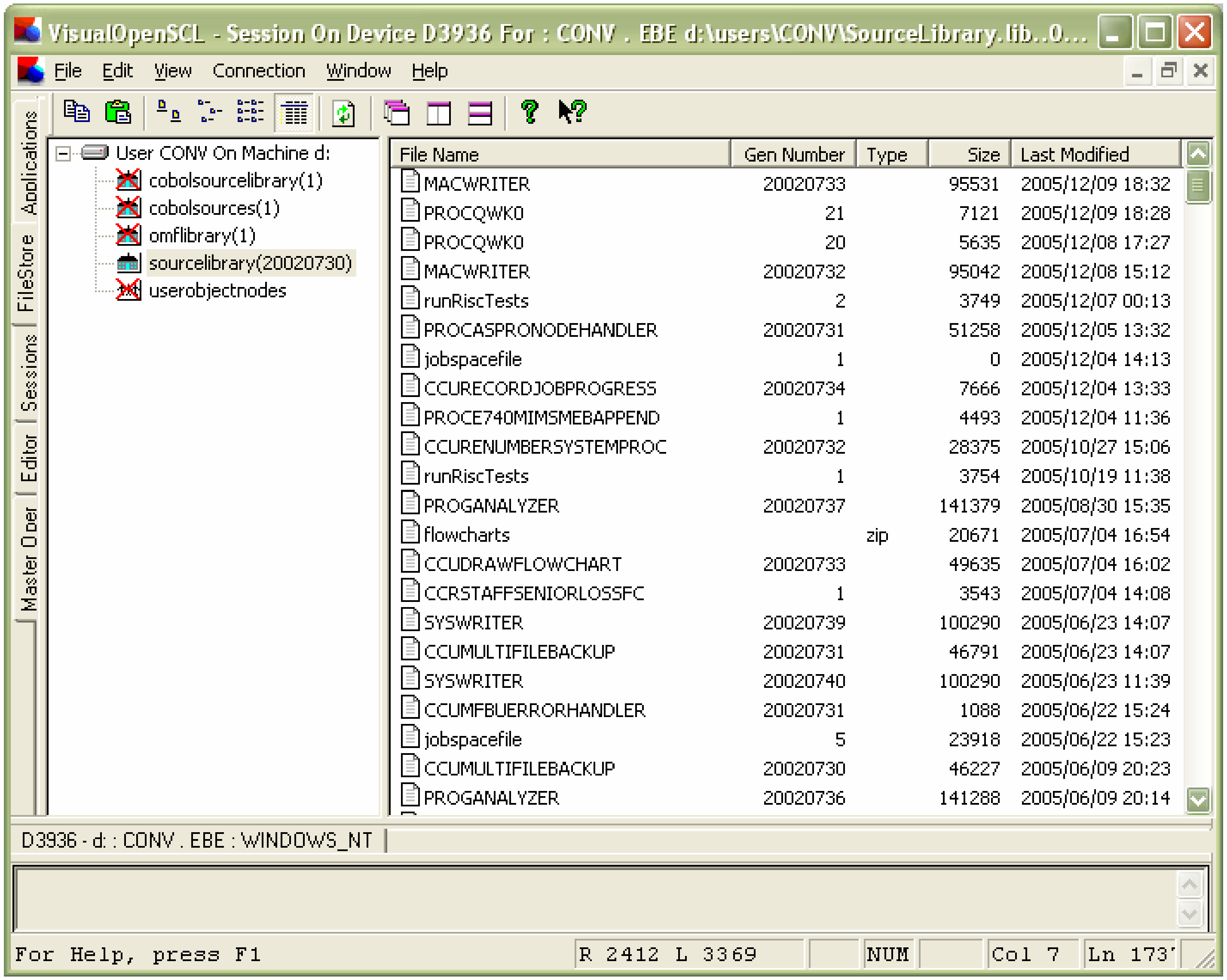
For Help, press F1   R 2412 L 3369   NUM   Col 7   Ln 173'

## Filestore View

A visual mechanism for interacting with files. It has the following features:

- Is modelled after Windows explorer
- Makes accessing and manipulating files easier
- Has context sensitive menus

See the figure below.



VisualOpenSCL - Session On Device D3936 For : CONV . EBE d:\users\CONV\SourceLibrary.lib..0...

File Edit View Connection Window Help

User CONV On Machine d:
- cobolsourcelibrary(1)
- cobolsources(1)
- omflibrary(1)
- sourcelibrary(20020730)
- userobjectnodes

| File Name | Gen Number | Type | Size | Last Modified |
|---|---|---|---|---|
| MACWRITER | 20020733 | | 95531 | 2005/12/09 18:32 |
| PROCQWK0 | 21 | | 7121 | 2005/12/09 18:28 |
| PROCQWK0 | 20 | | 5635 | 2005/12/08 17:27 |
| MACWRITER | 20020732 | | 95042 | 2005/12/08 15:12 |
| runRiscTests | 2 | | 3749 | 2005/12/07 00:13 |
| PROCASPRONODEHANDLER | 20020731 | | 51258 | 2005/12/05 13:32 |
| jobspacefile | 1 | | 0 | 2005/12/04 14:13 |
| CCURECORDJOBPROGRESS | 20020734 | | 7666 | 2005/12/04 13:33 |
| PROCE740MIMSMEBAPPEND | 1 | | 4493 | 2005/12/04 11:36 |
| CCURENUMBERSYSTEMPROC | 20020732 | | 28375 | 2005/10/27 15:06 |
| runRiscTests | 1 | | 3754 | 2005/10/19 11:38 |
| PROGANALYZER | 20020737 | | 141379 | 2005/08/30 15:35 |
| flowcharts | | zip | 20671 | 2005/07/04 16:54 |
| CCUDRAWFLOWCHART | 20020733 | | 49635 | 2005/07/04 16:02 |
| CCRSTAFFSENIORLOSSFC | 1 | | 3543 | 2005/07/04 14:08 |
| SYSWRITER | 20020739 | | 100290 | 2005/06/23 14:07 |
| CCUMULTIFILEBACKUP | 20020731 | | 46791 | 2005/06/23 14:07 |
| SYSWRITER | 20020740 | | 100290 | 2005/06/23 11:39 |
| CCUMFBUERRORHANDLER | 20020731 | | 1088 | 2005/06/22 15:24 |
| jobspacefile | 5 | | 23918 | 2005/06/22 15:23 |
| CCUMULTIFILEBACKUP | 20020730 | | 46227 | 2005/06/09 20:23 |
| PROGANALYZER | 20020736 | | 141288 | 2005/06/09 20:14 |

D3936 - d: : CONV . EBE : WINDOWS_NT

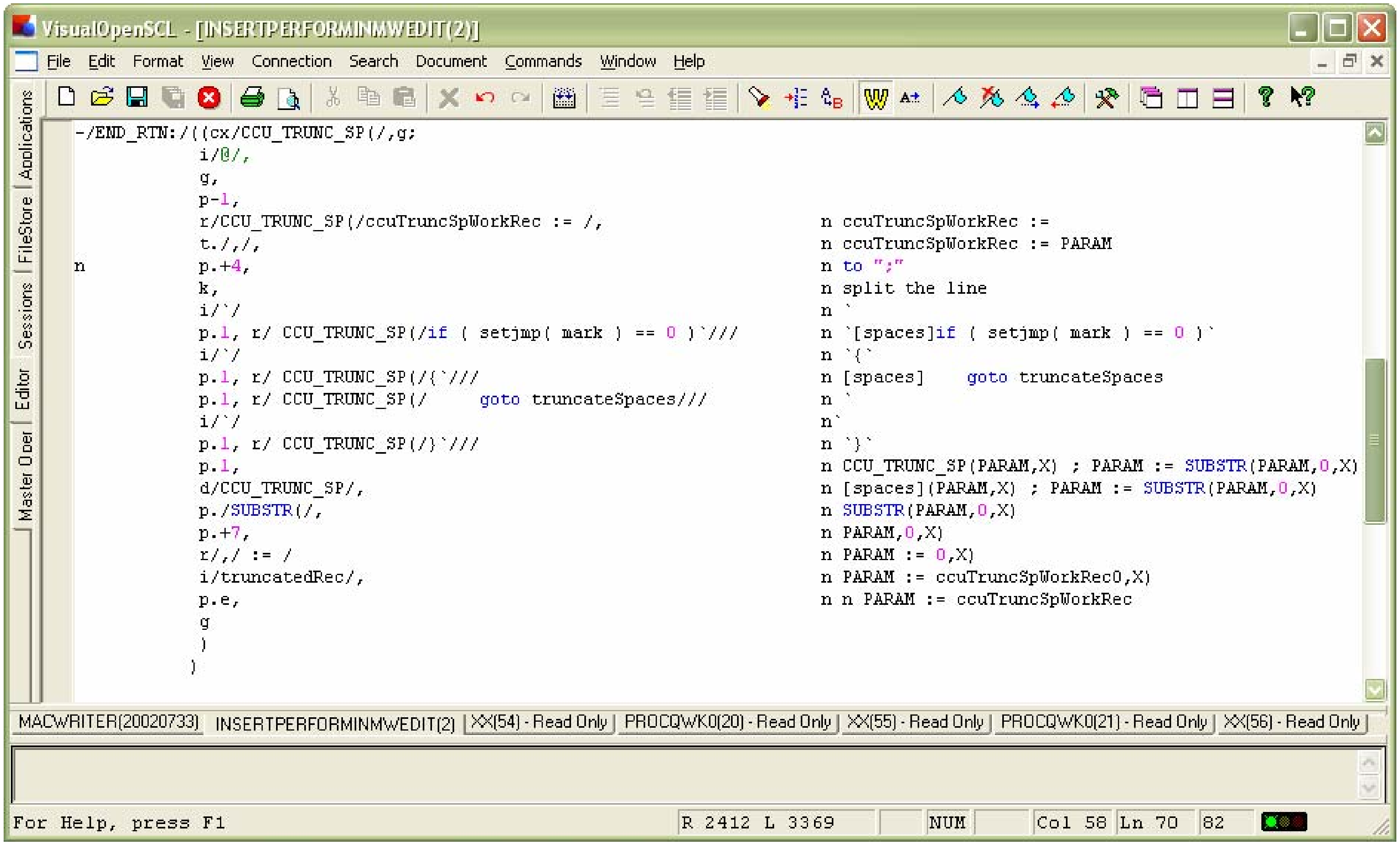
```
For Help, press F1          R 2412  L 3369    NUM   Col 7   Ln 173
```

**SCL Editor and Compiler**

Allows a user to develop new SCL procedures using the following features:

- Auto-complete facility for syntax and commands
- Syntax colouring specific to SCL
- Modelled after mainstream Windows editors such as Editplus
- Compile button and error messages
- Bookmarking

See the figure below.



VisualOpenSCL - [INSERTPERFORMINMWEDIT(2)]
File Edit Format View Connection Search Document Commands Window Help
Applications
FileStore
Sessions
Editor
Master Oper

```
-/END_RTN:/((cx/CCU_TRUNC_SP(/,g;
               i/@/,
               g,
               p-1,
               r/CCU_TRUNC_SP(/ccuTruncSpWorkRec := /,                n ccuTruncSpWorkRec :=
               t./,/,                                                n ccuTruncSpWorkRec := PARAM
n              p.+4,                                                 n to ";"
               k,                                                    n split the line
               i/`/                                                  n `
               p.1, r/ CCU_TRUNC_SP(/if ( setjmp( mark ) == 0 )`///  n `[spaces]if ( setjmp( mark ) == 0 )`
               i/`/                                                  n `{`
               p.1, r/ CCU_TRUNC_SP(/{`///                           n [spaces]    goto truncateSpaces
               p.1, r/ CCU_TRUNC_SP(/     goto truncateSpaces///     n `
               i/`/                                                  n`
               p.1, r/ CCU_TRUNC_SP(/)`///                           n `}`
               p.1,                                                  n CCU_TRUNC_SP(PARAM,X) ; PARAM := SUBSTR(PARAM,0,X)
               d/CCU_TRUNC_SP/,                                      n [spaces](PARAM,X) ; PARAM := SUBSTR(PARAM,0,X)
               p./SUBSTR(/,                                          n SUBSTR(PARAM,0,X)
               p.+7,                                                 n PARAM,0,X)
               r/,/ := /                                             n PARAM := 0,X)
               i/truncatedRec/,                                      n PARAM := ccuTruncSpWorkRec0,X)
               p.e,                                                  n n PARAM := ccuTruncSpWorkRec
               g
               )
              )
```

# *OpenSCL* Technical Overview

## Files

***OpenSCL*** supports the following file types:

- Line Sequential
- Record Sequential
- Indexed Sequential
- Relative
- All files can be fixed or variable length
- These are Microfocus file types and map to their VME equivalents
- Files are assigned dynamically at run time
- No file placement details are recorded in programs
- This provides device-independence to the program
- Files can be on disc or “tape”

The user is oblivious to the format of these files even though Microfocus files have proprietary headers that their VME counterparts do not have. ***OpenSCL*** handles these by interfacing with the Microfocus COBOL external file handler.

## File Descriptions

Each file has a unique description. This description records its:

- Type (sequential, indexed, etc.)
- Minimum record size
- Maximum record size
- Key position
- Key length
- ASCII or EBCDIC attribute

- Block size (future)

**Libraries**

Libraries provide a container for files with similar file descriptions. Libraries, as a whole, can be:

- Copied to other libraries
- Copied to tape
- Deleted
- Tidied

Selected files can be copied and or tidied
Libraries can be on disc or “tape”

**Generation numbers**

- Generation numbers are 9-digit file qualifiers
- All files and libraries can have up to 999-999-999 unique instances
- Generations numbers are automatically appended to file names
- Generation numbers are numbered sequentially but can be specified by the user
- New files start at generation 1 unless changed by the user
- Subsequent files are incremented by 1 unless changed by the user
- The latest generation of a file is alway assigned unless changed by the user.
- Generation numbers can be quote absolutely (by number) or relatively (using + and -)

**Groups and Sub-groups**

- Groups and Sub-groups provide a convenient way of physically grouping files for the same user
- E.g., for a payroll user groups can be created for salaries and wages
- This allows for the same named files to exist for the user but are qualified by the group

- Tax sub-groups can be created for the salaries and wages groups
- Commands exist for bulk manipulation of groups
- E.g., groups and sub-groups can contain files and libraries
- These can be tidied by executing one command

**Compiled SCL, including functions**

The ***OpenSCL*** system provides an SCL compiler to:

- Compile frequently used SCL into procedures
- Provide parameters for different invocations of the same procedure, e.g., daily, weekly or monthly
- Provide speedy execution of SCL
- Compile common routines into functions

**Interpreted SCL**

Interpreted SCL is allowed for:

- Interactive use
- “once-off” SCL requiring no parameters
- SCL can be executed from strings or from files
- This is more typically used to start production SCL procedures

**User object nodes**

- User object nodes are (typically):

    - “Control” files used by SCL procedures
    - Contain small amounts of data
    - Are normal files

  - Do not have generations
  - Can be permanent or temporary
  - Pass information across procedures in different jobs

**Full journal (logging) functionality**

- Every SCL statement issued in a MAC session is logged to a file called a journal
- *All mouse clicks which affect the users environment (e.g., file deletions) are logged*
- All SCL statements executed from files arelogged
- All SCL calls in Batch SCL are logged
- Displays from COBOL programs are logged

**Private journal and spool libraries**

- All journals are recorded in libraries
- The default location is a system user location
- User libraries may be created for sensitive users, e.g., payroll
- User libraries are automatically assigned
- Spool libraries are used for temporary (print) files which are created, listed and then deleted
- These are treated as per journal libraries

**The job space**

- The job space is the “working-storage” for the ***OpenSCL*** system
- It provides for the storage of strings, booleans, integers, etc. both inside procedures
- The job space *outside* procedures is available to any executing SCL
- Procedures can therefore adapt themselves to their environment
- ***OpenSCL*** has no practical limit to the size of the jobspace

## The line Editor (ED)

- A full implementation of the VME line Editor is provided. Some of the facilities include:

  - Transcribing text
  - Deleting text
  - Inserting new text
  - Replacing text
  - Inserting new lines and spaces
  - Controlling an omnibus character
  - Controlling a visible space character
  - Holding a record for re-editing
  - Checking
  - Changing the source of editing instructions
  - Quitting an edit instruction
  - Merging a different old file
  - Commenting of edit instructions

## Common display utilities

All the common display utilities are provided, e.g.

- DisplayUserDetails
- DisplayLibraryDetails
- DisplayGroupDetails
- DisplayFileDetails
- DisplayTapeLibraryDetails
- DisplayUserObjectDetails, etc.

**SCL file I/O**

***OpenSCL*** provides for:

- Opening files
- Reading serial files
- Amending files
- Updating individual records
- Identifying records by string matching
- Reading and amending random access files
- Pseudo-random access to serial files
- Deleting records

in both interpreted and compiled SCL.

- ALL Microfocus file types are supported
- Certain restrictions *within the Microfocus run-time* apply

**Tape handling**

- Production jobs typically back-up intermediate files to tape
- These are not “archives” of multiple gigabytes of data
- These backups can be restored without any operator control
- ***OpenSCL*** provides this functionality by writing these files to a “tape” which is a disc partition
- Production SCL therefore remains unchanged yet provides the same functionality
- ALL intermediate files can be backed to physical tapes in one go since all these reside on one disc
- All tape functionality is provided, e.g.

    - Introducing tapes
    - Tape categories

    - Creating tape files, etc.

**SCL Database I/O**

- ***OpenSCL*** supports Dbase-type and Foxpro databases in SCL
- It uses the royalty-free CodeBase IV database engine and provides facilities to:

    - Create, open and close database files
    - Read, write, update and delete, etc. database records

- Provides very fast access to these database types

**ASCII and EBCDIC**

- ***OpenSCL*** works natively in EBCDIC
- Support for ASCII is provided
- This allows for the reading and writing of both ASCII & EBCDIC files
- The file is simply "described" as ASCII or EBCDIC and ***OpenSCL*** processes the file accordingly
- Support exists for passing parameters to COBOL programs in either EBCDIC or ASCII mode

**Bus Utilities**

- The following, fully-functional, basic utilities are provided:

    - AppendRecords
    - CopyRecords
    - ListRecords
    - MatchRecords
    - DuplicateBlocks

- These support the fselect and rselect parameters

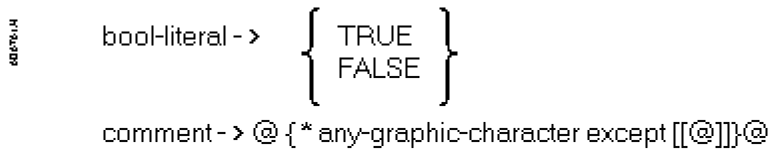
## Macros with textual substitution

- This is an old VME/B type of “procedure”, much like .bat files
- Limited support is provided for these in ***OpenSCL***

## System Control Language (SCL)

SCL is the basis of ***OpenSCL***. It stands for System Control Language and is used to write SCL procedures. System commands are written in SCL and users are able to write their own as well.

- Lexical elements are the building blocks of the SCL language and are equivalent to the words and punctuation in English. Some definitions:
  - alien-data -> *Records delimited by lines starting with ---- and ending with ++++*
  - alien-mode-definer -> *Identifier starting with SPD*
  - any-graphic-character -> *The EBCDIC Codeset*
  - bip -> *built-in procedures – see later*



bool-literal -> $\left\{\begin{array}{l}\text{TRUE}\\ \text{FALSE}\end{array}\right\}$

comment -> @ {* any-graphic-character except [[@]]}@

$$\text{delimiter} \rightarrow \left\{ \begin{array}{l} , \\ ( \\ ) \\ - \\ + \\ / \\ * \\ = \\ \text{space} \\ . \\ \& \\ < \\ > \\ <= \\ >= \\ : \\ := \\ ; \\ +\_ \\ \text{end-of-line} \\ \text{comment} \\ \text{alien-data} \end{array} \right\}$$

$$\text{digit} \rightarrow \left\{ \begin{array}{l} 0 \\ 1 \\ 2 \\ . \\ . \\ 9 \end{array} \right\}$$

end-of-file -> See section D.2

end-of-line -> See section D.2

$$\text{identifier} \rightarrow \text{letter} \left\{ {}^{*} \left\{ \begin{array}{l} \text{letter} \\ \{\text{digit}\} \end{array} \right\} \right\}$$

int-literal -> digit*

```
keyword -> identifier
letter ->  ⎧ A ⎫
           ⎪ B ⎪
           ⎨ C ⎬
           ⎪ . ⎪
           ⎪ . ⎪
           ⎩ Z ⎭

non-delimiter ->  ⎧ identifier                       ⎫
                  ⎪ keyword                          ⎪
                  ⎪ int-literal                      ⎪
                  ⎨ bool-literal                     ⎬
                  ⎪ string-literal                   ⎪
                  ⎪ parameter-literal                ⎪
                  ⎩ parameter-superliteral-element   ⎭

parameter-literal ->

{ { parenthetic-item                                                            } * }
{ { any-graphic-characterexcept [ [ + _ + / , ) space @ / * = " ' end-of-line ] ]  }   }

parameter-superliteral-element ->

{ { parenthetic-item                                                              } * }
{ { any-graphic-characterexcept [ [ + _ + / , ) space @ / * = " ' end-of-line & ] ]  }   }

parenthetic-item ->

{ { parenthetic-item                                                   } * }
{ { any-graphic-characterexcept [ [ + _ + / @ / " space / end-of-line ] ]  }   }

pragma ->  { RESCHEDULE   }
           { NORESCHEDULE }

sci-text -> { * delimiter } { non-delimiter { delimiter *} *}

string-literal ->  ⎧ Ò { * { any-graphic-character except [[Ò]]  } } Ó ⎫
                   ⎪   {   { ÒÓ                                } }   ⎪
                   ⎨                                                 ⎬
                   ⎪ Ô { * { any-graphic-character except [[Ô]]  } } Õ ⎪
                   ⎩   {   { ÔÕ                                } }   ⎭
```

$$\textit{actual-parameter-value} \; \text{->} \; \left\{ \begin{array}{l} \textit{expression} \\ \textit{parameter-literal} \\ \text{VAL } \textit{string-operand} \\ \text{SVAL } \textit{superstring-expression} \\ \left\{ \begin{array}{ll} \left[ \begin{array}{l} \textit{parameter-superliteral-element} \\ \text{VAL } \textit{string-operand} \end{array} \right] & \text{*\&} \end{array} \right\} \end{array} \right\}$$

$$\textit{assignment} \; \text{->} \; \left\{ \begin{array}{l} \text{identifier} \\ \textit{identifier-subscript} \\ \text{SUBSTR(}\textit{identifier,int-expression,int-expression}\text{)} \end{array} \right\} \text{:=expression}$$

$$\textit{block} \; \text{-> BEGIN } \textit{statements} \left\{ \begin{array}{l} \textit{;} \\ \textit{end-of-line} \end{array} \right\} \text{ END}$$

```
bool-expression - >   { bool-operand                                                        }
                      { bool-operand { AND } bool-operand                                   }
                      {              { OR  }                                                }
                      {              { NEQ }                                                }
                      { int-operand relational-operator int-operand                         }
                      { string-operand { relational-operator } string-operand               }
                      {                { INCLUDES            }                              }
                      {                { STARTSWITH          }                              }
                      {                { ENDSWITH            }                              }

bool-operand - >      { identifier            }
                      { bool-literal          }
                      { function-call         }
                      { identifier subscript  }
                      { NOT bool-operand      }
                      { (bool-expression)     }

built-in-procedure-call - > bip [ ( { [ actual-parameter-value ] *, } ) ]

call - > [ ENTER   ] { full-template-call   }
         [ SYSCALL ] { simple-template-call }
                     { no-template-call     }

conditional - > { IF     } conditional-body FI
                { UNLESS }

conditional-body - > bool-expression THEN statements [ ELSE statements         ]
                                                     [ ELSF conditional-body   ]

cycle - > [FOR identifier] [ * { FROM } int-expression ] [ { WHILE } bool-expression ]
                           [   { TO   }                ] [ { UNTIL }                 ]
                           [   { BY   }                ] [                           ]

          DO statements { ;           } REPEAT
                        { end-of-line }

expression - > { int-expression         }
               { bool-expression        }
               { string-expression      }
               { superstring-expression }

ext-proc-declaration - > EXT PROC identifier [(pragma)]
                         IS [ ext-proc-parameters ] [ INT                ]
                                                    [ BOOL               ]
                                                    [ STRING             ]
                                                    [ SUPERSTRING        ]
                                                    [ alien-mode-definer ]

ext-proc-parameter - > { parameter-mode     } [ keyword [ :=expression ] ]
                       { alien-mode-definer } [         [ :=NIL        ] ]

formal-parameter - >   [parameter-mode] [(keyword)] identifier [:=expression]

formal-parameters - >  ([formal-parameter *,])

full-template-call - > identifier [(  [[keyword=][actual-parameter-value]*,])]

function call - > { full-template-call      }
                  { simple-template-call    }
                  { no-template-call        }
                  { built-in-procedure-call }
```

```
int-expression ->      { int-operand* { +       } }
                       {              { -       } }
                       {              { *       } }
                       {              { /       } }
                       {              { AND     } }
                       {              { OR      } }
                       {              { NEQ     } }

int-operand ->         { identifier                }
                       { int-literal               }
                       { function-call             }
                       { identifier subscript      }
                       { (int-expression)          }
                       { { + } int-operand         }
                       { { - }                     }
                       { COUNT identifier          }
                       { LENGTH string-operand     }
                       { BOUND identifier          }

job-control-program -> { { statements            } * } end-of-file
                       { { alien-data            }   }
                       { { procedure-declaration }   }

jump ->  GOTO label

label ->  identifier

macro-body -> MACBEGIN statements { ;           } MACEND
                                  { end-of-line }

macro-declaration -> MACRO identifier [synonyms-and-version]

   IS [macro-formal-parameters] [ INT         ] end-of-line macro-body
                                [ BOOL        ]
                                [ STRING      ]
                                [ SUPERSTRING ]

macro-formal-parameter ->

   [parameter-model][keyword)] macro-identifier [:=expression]

macro-identifier ->  %identifier

no-template-call -> identifier[([expression*,])]

panoramic-conditional -> WHENEVER whenever-condition [THEN statements FI]

parameter-mode ->  { INT              }
                   { BOOL             }
                   { STRING           }
                   { SUPERSTRING      }
                   { LITERAL          }
                   { SUPERLITERAL     }
                   { REF( )INT        }
                   { REF( )BOOL       }
                   { REF INT          }
                   { REF BOOL         }
                   { REF STRING       }
                   { REF SUPERSTRING  }
                   { RESPONSE         }
```

```
proc-body ->  ⎧ PROCBEGIN statements ⎧ ;           ⎫  PROCEND ⎫
              ⎨                      ⎩ end-of-line ⎭          ⎬
              ⎩ BEGIN statements     ⎧ ;           ⎫  END     ⎭
                                     ⎩ end-of-line ⎭

procedure-declaration -> PROC identifier [synonyms-and-version]
    IS [formal-parameters]      ⎡ INT         ⎤ end-of-line proc-body
                                ⎢ BOOL        ⎥
                                ⎢ STRING      ⎥
                                ⎣ SUPERSTRING ⎦

relational-operator ->  ⎧ NE ⎫
                        ⎪ =  ⎪
                        ⎨ >  ⎬
                        ⎪ <  ⎪
                        ⎪ >= ⎪
                        ⎩ <= ⎭

return-statement -> RETURN [expression]

row-declaration -> (int-expression)  ⎧ INT  ⎫   {identifier*,}
                                     ⎩ BOOL ⎭

scalar-or-superstring-declaration ->
⎡ INT                                                  ⎤ ⎧ identifier ⎡⎧ := ⎫ expression⎤ *, ⎫
⎢                                                      ⎥ ⎩            ⎣⎩ IS ⎭           ⎦     ⎭
⎨ BOOL                                                 ⎬
⎢ STRING [(int-expression)]                            ⎥  Note that during repetition := or IS
⎣ SUPERTSRING [([int-expression] [,int-expression])]   ⎦  must not be interchanged

simple-template-call -> identifier [([actual-parameter-value]*,)]

statement -> [label:]  ⎡ assignment                        ⎤
                       ⎢ block                             ⎥
                       ⎢ call                              ⎥
                       ⎢ conditional                       ⎥
                       ⎢ cycle                             ⎥
                       ⎢ ext-proc-declaration              ⎥
                       ⎢ ext-variable-declaration          ⎥
                       ⎢ jump                              ⎥
                       ⎢ panoramic-conditional             ⎥
                       ⎢ return-statement                  ⎥
                       ⎢ row-declaration                   ⎥
                       ⎣ scalar-or-superstring-declaration ⎦

statements -> ⎧ statement* ⎧ ;           ⎫ ⎫
              ⎩            ⎩ end-of-line ⎭ ⎭

string-expression ->  ⎧ string-operand* ⎧ +      ⎫ ⎫
                      ⎪                 ⎪ -      ⎪ ⎪
                      ⎪                 ⎪ AFTER  ⎪ ⎪
                      ⎨                 ⎨ BEFORE ⎬ ⎬
                      ⎪                 ⎪ AND    ⎪ ⎪
                      ⎪                 ⎪ OR     ⎪ ⎪
                      ⎩                 ⎩ NEQ    ⎭ ⎭
```

- ***OpenSCL*** supports ALL the VME SCL language lexical elements and language constructs

  - SCL is a fully functional programming language
  - SCL uses English type commands
  - ***OpenSCL*** SCL provides:

    - panoramic conditionals (WHENEVER)
    - easy string manipulation
    - result checking
    - strings and Superstrings
    - ints and rows of ints
    - bools and rows of bools
    - if, goto, while, repeat, whenever, etc
    - all built-in procedures
    - *Embedded C code in procedures*

**Built-in procedures**

- BIN
- CHARTOINT
- CLOCK
- DIGITS
- FILL
- FIND
- HEX
- HEXTOCHAR
- INDEX
- NUMERIC
- STATUS

- STINT
- SUBSTR

## ***OpenSCL* Enterprise**

- **Batch Scheduler**

### BATCH

Once an SCL procedure has been developed and tested in an interactive MAC session it can be submitted to the Job Scheduler to be run in unattended mode.

- Jobs are submitted to queues with varying priorities.
- The execution of these jobs is controlled by setting concurrencies, for example the scheduler can be set to run only two jobs at any one time.
- Profiles exist per job so that the execution of jobs can be controlled on a group basis.
- These jobs are not scheduled on a processor level.
- The job journal records the results of the job.

Multiple queues

- Priorities amongst queues
- Priorities within queues
- Queues can be held, expressed, released, etc
- Entries in queues can be held, released, expressed
- Executing jobs can be suspended, activated, abandoned, etc
- Concurrencies can be set

Note that the Developer version of ***OpenSCL*** has the ability to run BATCH work. However the overall concurrency can only be set to one.

## Which calls are supported?

***OpenSCL*** can invoke:

- Any DLL
- Any exe
- Any SCL
- C, C++, Visual Basic, etc
- Any WIN API using C embedded in SCL
- Microfocus Advanced Workbench ints and gnts
- Microfocus NetExpress gnts and ints

COBOL programs are called from ***OpenSCL*** via the Microfocus COBOL runtime that is linked with it.

- COBOL programs compiled into INT or GNT executables can be called by name from an SCL procedure.

- Parameter passing. SCL variables can be passed to and from COBOL programs. Byte swapping is handled seamlessly by ***OpenSCL***. Calling conventions allowed on VME are also allowed under ***OpenSCL***.

- COBOL templates. ***OpenSCL*** allows a user to specify templates for calling COBOL programs. Specific default values can be assigned to these calls.

- Error trapping. Run-time exceptions that occur in the COBOL programs are trapped by the ***OpenSCL*** run-time and reported to the user in the MAC session and in the job journal.

- COBOL displays. Output that COBOL programs make to the screen is redirected to the MAC session as well as the job journal.

## Other facilities

- Reading of *any* files on homogeneous networks at source
- Integration with 3rd-party products

## The EBE Computing SCL Generator

- The ***OpenSCL*** Generator with its unique features for generating highly efficient SCL and its associated run-time job restart option ASPRO, is available on all platforms which support ***OpenSCL***