CA-CLIPPER

CA-CLIPPER Introduction
Student Guide
CL001

Release 5.0

November 1992

COMPUTER® ASSOCIATES
Software superior by design.

E155NCS0SCP

Release 5.0, November 1992

Call Computer Associates technical services for any information not covered in this manual or the related publications. In North America, see your Computer Associates *Product Support Directory* for the appropriate telephone number to call for direct support, or you may call 1-800-645-3042 or 1-516-342-4683 and your call will be returned as soon as possible.

Outside North America, contact your local Computer Associates technical support center for assistance.

Table of Contents

Introduction to the CA-Clipper Language

Release 5.0

CA-CLIPPER 5.0

About CA-Clipper 5.0

. CA-Clipper is designed for application development needs

. Open architecture

. Third-party product support

The CA-Clipper 5.0 Package

- CA-CLIPPER.EXE
- RTLINK.EXE
- RMAKE.EXE
- CLD.EXE
- DBU/ RL (source code and .EXE)
- NG.EXE

The CA-Clipper 5.0 Package

- Include Files (.ch)
- Library Files (.LIB)
- GetSys.prg
- ErrorSys.prg
- DBU and RL Source Codes
- Samples (.prg)

The CA-Clipper Installation Process

The **Install** program will create the following subdirectories:

```
\CA-CLIPPER5
        \BIN
        \LIB
        \INCLUDE
        \PLL
        \SOURCE
                \SYS
                \DBU
                \PE
                \RL
                \SAMPLE
\NG
```

After the installation process, you need to be sure you have:

- DOS environmental variables properly set
- PATH to \CA-CLIPPER5\BIN and \NG directories:

```
SET PATH=C:\CA-CLIPPER5\BIN;\NG;....
```

The Environmental Variables

- CA-CLIPPER
- LIB
- INCLUDE
- PLL
- CA-CLIPPERCMD
- RTLINKCMD
- TMP
- OBJ
- RMAKE

SET CA-CLIPPER

```
SET CA-CLIPPER=Fxxx SWAPPATH:"drive:\path"
```

- Be sure your CONFIG.SYS contains:

  ```
  FILES=xxx
  ```

  where xxx indicates the same number of the F parameter, plus 5.

- You have two ways to set this variable:

  1. Before your application is executed:

     ```
     C:\TEST> SET CA-CLIPPER=F55 SWAPPATH:"d:\"
     C:\TEST> MyApp
     ```

  2. In the command line that invokes your application:

     ```
     C:\TEST> MyApp //F55 //SWAPPATH:"d:\"
     ```

How to create an .EXE file

- Write your Source Code, using any program editor --> .prg file
- Compile the source file with CA-CLIPPER.EXE --> .OBJ file
- Link the object file with RTLink --> .EXE file
- Run your program (executable file)

The CA-Clipper Compiler

Source File --> **Object File**
(.prg) (.OBJ)

- Assumes **.prg** source file extension
- You can define your own extension; in this case, you must tell it to CA-Clipper:
- You invoke the Compiler, typing at the DOS prompt:

```
C:\TEST> CA-CLIPPER MyApp
```

or,

```
C:\TEST> CA-CLIPPER MyApp.1
```

- Actually, CA-Clipper calls the Pre-processor first
- and then, it sends the pre-processed code to the Compiler
- Checks for syntax and variable declaration
- If no error, creates the object file
- CA-Clipper returns either ERRORLEVEL 0 (no errors) or ERRORLEVEL 1 (errors found)

Compiler Switches

- Switches may be indicated with / or -
- Some of them are:

B	Include Debug information (required to run CLD)
D	Define an identifier
I	Expand the Include file directory search list
M	Compile the current module only
N	Suppress automatic Main procedure
O	Compile to specified OBJ file
P	Generate pre-processed file (.ppo)
S	Syntax check only, don't generate .OBJ file
W	Issue warning message for ambiguous references

- Using compiler switches:

```
C:\TEST> CA-CLIPPER MyApp  /N /W /P /B
```

or, if you prefer, you can pre-define your switches in a command line, batch file or even in your AUTOEXEC.BAT, using:

```
SET CA-CLIPPERCMD=/N /W /P /B

.
.
CA-CLIPPER MyApp1

CA-CLIPPER MyApp2
```

- You can also redirect screen output to a DOS file with >:

```
C:\TEST> CA-CLIPPER MyApp /P >Error.log
```

Linking with RTLink

Object File (.OBJ) ---> Executable File (.EXE)

To create an executable file from an object file (compiled file)

- **Dynamic Overlay Linker**
  RTLink works in pages of 4Kbytes

- RTLink allows:
  - positional and free format (default) modes
  - Incremental linking
  - Pre-linked libraries
  - define RTLINKCMD variable

- Syntax:

```
SET RTLINKCMD=/PLL:BASE50
RTLINK FILE MyApp1, MyApp2, MyApp3  OUTPUT MyApp  /PLL:BASE50

SET RTLINKCMD=/POSI   /PLL:BASE50
RTLINK MyApp1 MyApp2 MyApp3, MyApp;
```

- To use pre-linked libraries, you need:

```
SET PLL=C:\CA-CLIPPER5\PLL
```

RMake

- It works like a batch file to compile and link every source file belonging to your program
- **BUT**, RMake keeps files up to date by comparing date and time
- RMake minimizes the number of steps taken to create an .EXE
- Assumes file extension of **.RMK** but you may use whatever you want.
- Syntax:

```
RMAKE <MakeFileName>
```

Program Execution - VMM

VM

limited amount of REAL memory to emulate a larger amount of memory (VIRTUAL)

VMM

segmented memory manager

- most VM segments are removable
- VMM can reorganize them to get maximum utilization of real memory
- VMM does not run out of available real memory
- VMM makes room for new segments swapping out the ones least recently used

Modular Programming

- A CA-Clipper application is based on Modulatiry,
- One Main module, calling several others
- A Module consists in a block of statements to implement a pre-defined routine
- It should be placed in a Source file that contains a set of related routines
- The Main module will call the others when you run your application
- Use RMAKE to manage the compiling/linking process for you

Advantages of using Modular Programming

- Easy mantainance
- Design reusable tools
- Each module should depend on nothing more than information it receives via parameters or creates internally

Main Function

- A program statement can take many different forms including:
  - Comment
  - Variable Declaration
  - Variable Assignment
  - Expressions
  - Control structure
  - Function call (library, another UDF, pseudo)
  - Command invocation (internal, User-Defined Command)
  - Pre-processor directive

- You can use:

```
FUNCTION Main()
      .
      .
      .
```

- or, you can choose any name:

```
FUNCTION Sys1()
      .
      .
      FuncA()
      FuncB()
      .
      .
```

- Using this procedure you **MUST** use the **/N** compile swtich

Comments

- You can use:
  - a multi-line comment

```
/*

   Test.prg
   Tests something

*/
```

  - or a single line comment

```
//   Variable declaration
//
STATIC   nTotal  :=  0      // Stores the Total of sales
.
.
.
```

Variables

- Store data, temporarily
- Data types supported by the CA-Clipper language:

Character	C
Numeric	N
Date	D
Logical	L
Memo	M
Array	A
Code Block	B
Object	O
NIL	U

- Use VALTYPE() to determine the data type
- Attributes:

  Name
  Scope (Lifetime / Visibility)
  Value

Variable Name

- Give names according to data type
- **Hungarian Notation:**
  - First letter is always lowercase and indicates the data type for a variable
  - Second letter is always uppercase and is the first significant letter of the variable name
  - Other significant letters may be uppercase also
- Examples:

```
cName, cLastName, dDueDate, nCounter, lMarried, nMenuOption
```

Variable Declaration

- **Declaration scope:**
  - FUNCTION/ PROCEDURE
  - FILE
  - SYSTEM
- **Declaration types:**
  - PUBLIC
  - PRIVATE
  - LOCAL
  - STATIC
- Example:

```
LOCAL cMessage
STATIC nPageNo
PUBLIC cMessage
PRIVATE nCode
```

Variable Lifetime and Visibility

- Depends on the Scope and Declaration

	Lifetime	Visibility
PUBLIC	Global	Global
PRIVATE	Creator/Called	Creator/Called
STATIC	Global	Creator
LOCAL	Creator/Called	Creator

Variable Assignment

- Use the assignment operator (:=) to initialize a variable
- Can be declared and initialized on the same line
- Can be declared and then, during subsequent execution, initialized
- When a variable is declared, NIL is its value, until iformal initialization
- Example:

```
FUNCTION Main()
  PUBLIC nPub := 1
  LOCAL nLocal:= 30

  QOUT( nPub )
  QOUT( nLocal )

  Func1()

  QOUT( nPub )
  QOUT( nLocal )
  QOUT( nPrv )
  .
  .
  .
```

```
FUNCTION Func1()
  LOCAL nLocal := 50
  PRIVATE nPrv := 20

  QOUT( nPub )
  QOUT( nPrv )
  QOUT( nLocal )

  nPub := 100

  Func2()

  QOUT( nPub )
  QOUT( nPrv )
  QOUT( nLocal )
  .
  .
```

```
FUNCTION Func2()
  LOCAL nLocal := 5

  QOUT( nPub )
  QOUT( nPrv )
  QOUT( nLocal )
  .
  .
  .
```

LOCAL Variables

```
FUNCTION MyFunc()
      LOCAL nVar := 34

      OtherFunc()

      QOUT( nVar )          // Prints 34

      RETURN ( NIL )


FUNCTION OtherFunc()
      LOCAL nVar      // This is not the same as above

      nVar := 8765

      RETURN ( NIL )
```

STATIC - Internal Variables

```
FUNCTION PrintPageNo()
      STATIC  nPageNo := 0

      @ nRowNo, 70 SAY nPageNo++

      RETURN ( NIL )
```

- Initialization is done just once at start-up time
- Variable hold its value during program execution
- May be initialized only with a constant

STATIC - External Variables

- Also called file wide STATIC
- Same as STATIC internal, but every Function inside the .prg file can manipulate it
- You **MUST** compile your program with /N

```
// Beginning of source file
STATIC  nLineNo := 10

FUNCTION PrintHeader()
      @ nLineNo, 5 SAY "Customer"
      .
      .
      .
      RETURN ( NIL )


FUNCTION PrintRec()
      @ ++nLineNo, 5 SAY Customer->Name
      .
      .
      .
      RETURN ( NIL )
```

Considerations

- Declare ALL memory variables
- /W compile switch will check for undeclared or ambiguous variable references
- Use LOCAL and STATIC declarations

Expressions

- Collection of items to perform some operation
- CA-Clipper allows you to put an expression any place
- To define an expression, you can use:

  Constants
  Variables
  Mathematical operators
  Logical operators
  Function calls

- You can invoke any User Defined Function as part of an expression

Operators

- Assignment :=
- Logical .AND., .OR., .NOT.
- Relational <, <=, >, >=, =, ==, !=, $
- Numeric +, -, *, /, %, **, +=, -=, *=, /=, **=, %=
- Date +, -, +=, -=
- Character +, -, +=, -=

Increment and Decrement Operators

- Pre/ post ++
- Pre/ post --
- Example:

```
@ ++nRowNo, 10 SAY "Something"
nPageNo++

WHILE  nCnt--  > 0
       .
       .  // Do something
       .
END
```

Special Operators

()	Grouping parentheses
[]	Array element
{}	Array definition
{|| ....}	Code block
->	Alias
&	Macro
@	Argument passed by reference

UDFs

User-Defined Function

- Consists of a block of a logical statements sequence to codify a particular routine
- The statements can take the form of:
  - Comment
  - Variable Declaration
  - Variable Assignment
  - Expression
  - Control structure
  - Function call (library, another UDF, pseudo)
  - Command invocation (internal, User-Defined Command)
  - Pre-processor directive
- You can write a UDF as:

```
PROCEDURE  MyProc()
      .
      . // Block of Statements
      .
      RETURN
```

or,

```
FUNCTION  MyFunc()
      .
      . // Block of Statements
      .
      RETURN <ReturnValue>
```

- the return value can be NIL

FUNCTION Parameters

- Declare the parameter list as:

```
FUNCTION MyTest( xParm1, xParm2, xParm3, xParm4 )
```

- The parameters will be LOCAL
- You can omit parameters by just skipping them:

```
MyTest( 10, , , "Abcde" )
.
.
```

- The value of skipped parameters is NIL
- Parameter checking:

```
xParm1 := IF( xParm1 == NIL, "Default value", xParm1 )
.
.
.
```

- Variables passed by reference pass either their value or a self-reference

```
MyTest( cName, @nSalary )
```

STATIC FUNCTION

- A Function (or Procedure) can be declared STATIC
- It will be seen only inside the .prg file
- Prevention against name conflicts
- A.prg

```
FUNCTION A()
        .
        .
        .
        Msg( "Function A" )
        .
        .
        RETURN ( NIL )

STATIC  FUNCTION  Msg( cMessage )
        @ MAXROW(), 0  SAY cMessage
        RETURN  ( NIL )
```

- B.prg

```
FUNCTION B()
        .
        .
        .
        Msg( "Function B", 20 )
        .
        .
        RETURN (NIL )

STATIC FUNCTION  Msg( cMes, nRow )
        @ nRow, 0 SAY PADC( cMes, MAXCOL()+1 )
        RETURN ( NIL )
```

Evaluating Expressions

- Expressions can be nested
- Always evaluated from the innermost expression outward
- Use grouping paretheses to clarify what you want
- You can write Multiple-expressions:
  - Just enclose every expression in parentheses, comma separated
  - Last one evaluated is returned
- Everyplace you use a WHILE clause you can call a UDF, where you can do *anything* you want, since this function returns a logical value

```
FUNCTION PrintLabel()
      .
      .
      SELECT Test
      LABEL FORM Test WHILE PrintCtr()
      .
      .

FUNCTION PrintCtr()
      LOCAL cScreen, nOption

      cScreen := SAVESCREEN( 5, 0, 10, 40 )
      nOption := 1

      @ 6, 10 PROMPT "Continue"
      @ 8, 10 PROMPT "Abort    "
      MENU TO nOption

      RESTSCREEN( 5, 0, 10, 40, cScreen )

      RETURN ( nOption == 1 )
```

Shortcutting

- CA-Clipper 5.0 automatically performs optimizations (shortcutting) on logical expressions containing .AND. or .OR. operators:

x	y	x .AND. y	
.T.	.T.	.T.	
.T.	.F.	.F.	
.F.	.T.	.F.	shortcut
.F.	.F.	.F.	shortcut

x	y	x .OR. y	
.T.	.T.	.T.	shortcut
.T.	.F.	.T.	shortcut
.F.	.T.	.T.	
.F.	.F.	.F.	

- Different behavior from Summer' 87
- /Z compile switch suppresses shortcutting for the compiled file

Extended Expressions

- Many places when you have to use memory variable contents instead of the variable itself, you can write it inside () as an expression
- Use:

```
cFileName := "Customer"
USE ( cFileName )          // in place of  USE Customer
```

Macro Substitution

- Text substitution to code
- Allows run time compilation, by a tiny version of the CA-Clipper Compiler
- The macro will be both compiled and then executed, every time
- Use:

```
// This was typed by an end-user
// through @...GET
cCond = "Customer->LastName == 'Smith'"

WHILE  &cCond .AND. !EOF()
       .
       .
       .
END
```

Control Structures

- Loop

  WHILE...END

  FOR...NEXT

- Decision

  IF...ELSEIF... ELSE... ENDIF

  DO CASE...ENDCASE

  IF()

WHILE...END

- Executes same block of statements as long as the controlling logical expression evaluates to .T.
- May nest WHILE...END blocks to any depth
- Supports EXIT and LOOP
- Syntax:

```
WHILE <condition>
      .
      .   //  Statements
      .
END
```

- Example:

```
.
.
WHILE ! EOF()
      HeaderPage()
      PrintRec()
      DBSKIP()
END
.
.
```

FOR...NEXT

- Executes same block of statements specified number of times
- Expression in the FOR statement is reevaluated each pass
- May nest FOR...NEXT blocks to any depth
- Optional STEP clause to control how to increment
- Supports EXIT and LOOP
- Syntax:

```
FOR <xVar> := <startValue> TO <finishValue> STEP <increment>
      .
      .     // statements
      .

NEXT
```

- Example:

```
.
.
FOR i := 1 TO 12
      nTotal += aSales[i]

NEXT
.
.
```

IF...ELSEIF...ELSE...ENDIF

- A set of mutually exclusive statement blocks, condition related
- The structure forces execution of the first block whose condition evaluates to .T.
- Each IF/ELSEIF evaluates any valid CA-Clipper logical expression
- ELSE is an optional clause to trap when all conditions have failed
- Syntax:

```
IF  <condition>
      .
      . // first block of Statements
      .
ELSE
      .
      . // second block of Statements
      .
ENDIF
```

or,

```
IF  <condition1>
      .
      . // first block of Statements
      .
ELSEIF <condition2>
      .
      . // second block of Statements
      .
ELSEIF  <condition3>
      .
      . // third block of Statements
      .
ELSEIF <condition4>
      .
      .
      .
ELSE
      .
      .  //  last one block of Statements to be executed if
      .  //  ALL conditions have failed
      .
ENDIF
```

- Examples:

```
// About discount...
IF lDiscount
      IF cCodeProd == "1111"
            nPrice *= .2

      ELSE
            nPrice *= .15

      ENDIF

ENDIF


// Processing a Menu selection
IF nOption == 1
      // do something

ELSEIF nOption == 2
      // do another thing

ELSEIF nOption == 3
      // ...

ELSE
      // ...

ENDIF
```

DO CASE...CASE...ENDCASE

- Set of mutually exclusive statement blocks whoseexecution depends on certain conditions
- Each block starts with its own CASE statement
- Each CASE statement can evaluate any CA-Clipper logical expression
- OTHERWISE is an optional clause to trap when all CASEs have failed
- Identical to IF...ELSEIF...ELSE...ENDIF
- Syntax:

```
DO CASE
      CASE <condition1>
            .
            .  //  First block of statements
            .
      CASE <condition2>
            .
            .  //  Second block of statements
            .
      CASE <condition3>
            .
            .  //  Third block of statements
            .
      OTHERWISE
            .
            .  //  block of statements to be executed if
            .  //  ALL conditions have failed
            .
ENDCASE
```

- Example:

```
DO CASE
      CASE nOption == 1
            // do something

      CASE nOption == 2
            // do another thing

      CASE nOption == 3
            // ...

      OTHERWISE
            // ...

ENDCASE
```

IF()

- Provides in-line IF...ELSE...ENDIF structure
- Forces CA-Clipper to evaluate one of two in-line expressions
- The resultant value can be stored directly into a memory variable
- Syntax:

```
xVar := IF( <condition>, <expression 1>, <expression 2> )
```

IF <condition> evalutes to:	CA-Clipper executes:
.T.	<expression 1>
.F.	<expression 2>

and assigns the resultant value into xVar

- Examples:

```
// Rewriting the above discount decision...
IF lDiscount
      nPrice *= IF( cCodeProd == "1111", .2, .15 )

ENDIF


FUNCTION MyFunc( xParm1, xParm2, xParm3 )
      // Parameter checking
      xParm1 := IF( xParm1 == NIL, "Default value", xParm1 )
      .
      .
```

Arrays

- Is a **reference** to a set of values related to the same name
- A subscript points out to a particular array element
- CA-Clipper works with Multi-dimensional Arrays
- Each array element can be another array
- 4096 elements per dimension

Declaring Arrays

- An array is contained in a variable
- the variable can be LOCAL, STATIC, PRIVATE, PUBLIC
- You can size it at its declaration or do it during program execution

```
LOCAL aSales[12]

LOCAL aSales
.
.
.
aSales := ARRAY(12)
```

Assigning values

- You can assign a value to each element of an array:

```
aSales[1] := 20000
aSales[2] := 35000
.
.
.
```

Working with Arrays

```
nTotal := aSales[1] + aSales[2] + ...
```

or, in a more efficient way:

```
FOR i := 1 TO 12
      nTotal += aSales[ i ]
NEXT
```

Multi-Dimensional Array

```
LOCAL aSales[2][12]      // Sales from the 2 subsidiaries
LOCAL aTotal[2]

// Input data for the First subsidiary
FOR nMonth := 1 TO 12
      @ 10+nMonth, 15 GET aSales[1][nMonth]

NEXT
.
.
.

// Calculating the totals
FOR nSub := 1 TO 2

      FOR nMonth := 1 TO 12
            aTotal[ nSub ] += aSales[ nSub ][ nMonth ]

      NEXT

NEXT

.
.
.

// Printing the totals
@ nRowNo, 30 SAY aTotal[ 1 ]
@ nRowNo, 60 SAY aTotal[ 2 ]
.
.
.
```

Dynamic initialization

```
STATIC aMonths := { "JAN", "FEB", "MAR",..., "DEC" }

LOCAL aMenu := {  { "Add",    { || AddRec()  } },;
                  { "Edit",   { || EditRec() } },;
                  { "Print",  { || PrintDB() } }   }
```

Code Blocks

- Small piece of executable code
- A way to export code from one place in a system to another
- Treated as data
- You can think a code block as a function WITHOUT A NAME

Writing Code Blocks

- Code Block works as a Function, it can receive parameters and return a value
- It can be stored in a memory variable
- Can contain any CA-Clipper expression or a comma-separated list of expressions, for example:
  - Variable Assignment
  - Function call (library, UDF, pseudo-functions)
- You can build an array of Code Blocks
- Syntax:

```
bBlock := { | <argument list> | <expressionList> }
```

- Example:

```
bCalc := { |nVal1, nVal2| Func1( nVal1 ) + Func2( nVal2 ) }

aActions := { { || AddRec() },;
              { || Report() },;
              { || Quit() } }
```

Evaluating a Code block

- Code Block remains data until you invoke its execution
- Compilation is completed at compile-time
- It is evaluated at run time by **EVAL()**
- Syntax:

```
EVAL( <CodeBlock >, <argumentList> )
```

- Example:

```
EVAL( bCalc, 100, 345 )

nOption := ACHOICE( 10, 5, 13, 15, aOptions )
EVAL( aActions[ nOption ] )
```

The Pre-Processor

- Search and Replace filter
- Reads your source file and converts the lines of code into internal CA-Clipper functions
- Every command is pre-processed into function calls
- You can see the result using the /P switch when you invoke the Compiler
- CA-Clipper gives you the opportunity to emulate any Language or DBMS or even your own language.

Pre-Processor directives

#define

#include

#ifdef
#else
#endif

Manifest Constants

#define

To create manifest constants

Instead of:

```
IF LASTKEY() == 27          // ESC key
      lContinue := .F.

ENDIF
```

you can write:

```
#define   K_ESC          27
#define   STOP_LOOP      .F.
.
.
.
IF LASTKEY() == K_ESC
      lContinue := STOP_LOOP

ENDIF
```

- Scope is file-wide only
- #define is case sensitive
- The right place to define your Manifest Constants is at the top, but you can #define anyplace
- The use of Manifest Constants improves readability of your code

Pseudo-Functions

#define

To create pseudo-functions

```
#define   Msg(x)     @ MAXROW(),0 SAY PADC( x, MAXCOL()+1 )
FUNCTION Main()
      .
      .
      .
      Msg( "Testing..." )
```

- Looking at the .ppo file, we can see what is going to be sent to the compiler:

```
<blank line>
FUNCTION Main()
      .
      .
      .
      DEVPOS(MAXROW(), 0); DEVOUT(PADC("Testing...", MAXCOL()+1))
```

Include Files

- You can choose to write a set of manifest constants as a file
- Default extension is .ch (CA-Clipper Header)

```
// Screen.ch
#define     HEADER_MSG          "ABC System"
#define     HEADER_COLOR        "B/W"
#define     ROW_HEADER_MSG      1
#define     FOOTER_MSG          "Please, choose an option"
#define     FOOTER_COLOR        "W/R"
#define     ROW_FOOTER_MSG      MAXROW() - 2
.
.
.
```

Using the above Header file at the beginning of your source code:

```
#include "Screen.ch"
```

The CA-Clipper Header Files (.ch)

- CA-Clipper has already defined several Manifest Constants for you in specific Header files:

STD.ch	Set.ch	InKey.ch
Box.ch	Error.ch	SetCurs.ch
Achoice.ch	DBEdit.ch	Directry.ch
FileIO.ch	MemoEdit.ch	DBStruct.ch

- #include looks for these files at:
  - current directory
  - INCLUDE variable

    ```
    SET INCLUDE=C:\CA-CLIPPER5\INCLUDE;C:\TEST
    ```

  - /I compile switch

STD.ch

- Contains the definitions for ALL commands
- Compatibility between CA-Clipper 5.0 and dBase III and previous CA-Clipper versions
- Scope is compile wide
- From STD.ch:

```
***
*  Basic statement synonyms
*
#command DO WHILE <exp>          => while <exp>

#command END <x>                 => end
#command END SEQUENCE            => end
#command ENDSEQUENCE             => end
#command ENDDO     <*x*>         => enddo
#command ENDIF     <*x*>         => endif
.
.

**
  System SETs

command SET EXACT <x:ON,OFF,&>   => Set( _SET_EXACT, <(x)> )
command SET EXACT (<x>)          => Set( _SET_EXACT, <x> )

command SET FIXED <x:ON,OFF,&>   => Set( _SET_FIXED, <(x)> )
command SET FIXED (<x>)          => Set( _SET_FIXED, <x> )

command SET DECIMALS TO <x>      => Set( _SET_DECIMALS, <x> )
command SET DECIMALS TO          => Set( _SET_DECIMALS, 0 )

command SET PATH TO <*path*>     => Set( _SET_PATH, <(path)> )
command SET PATH TO              => Set( _SET_PATH, "" )

command SET DEFAULT TO <(path)>  => Set( _SET_DEFAULT, <(path)> )
command SET DEFAULT TO           => Set( _SET_DEFAULT, "" )
.
.
```

Conditional Compiling

```
#ifdef <constant>
       <statements>
#else
       <statements>
#endif
```

- To build decision for the compiler
- It bases on the existence of a constant created with /D compiler switch

```
/******
 * MyApp.prg
 *
 */

FUNCTION WriteRec()

      #ifdef NETWORK
            RLOCK()
      #endif
      .
      .       // Replace statements
      .

      #ifdef NETWORK
            UNLOCK()
      #endif
      .
      .
      .
```

- Use one Source Code to generate two executable file versions, a single user and a multi-user:

```
CA-CLIPPER MyApp

CA-CLIPPER MyApp  /DNETWORK
```

User-Defined Commands

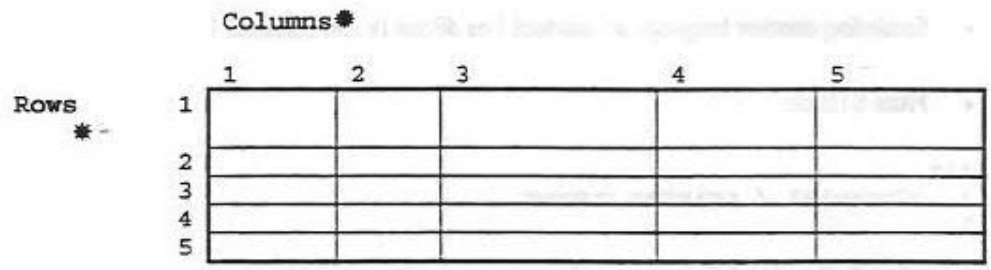
#command

- To create your own Commands
- Programming style
- Emulating another language or product (as dBase IV, for instance)
- From STD.ch:

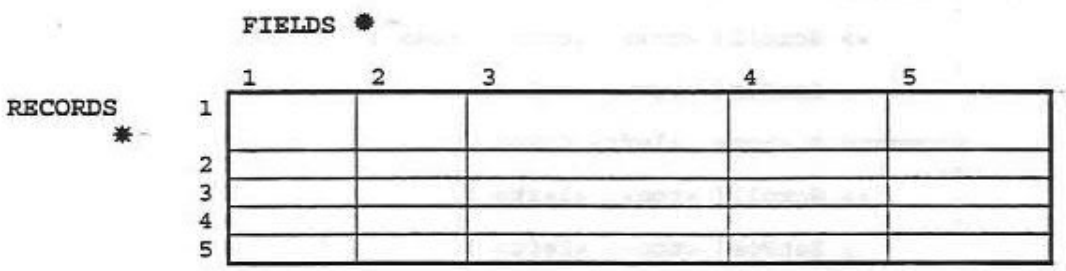
```
***
*  "Console" / printer output
*

#command ?  [ <list,...> ]       => QOut( <list> )
#command ?? [ <list,...> ]       => QQOut( <list> )


***
*  Clear screen
*

#command CLS    =>     Scroll(); SetPos(0,0)

#command CLEAR SCREEN
;
      => CLS

#command @ <row>, <col>
;
      => Scroll( <row>, <col>, <row> )
;
       ; SetPos( <row>, <col> )

#command @ <top>, <left> CLEAR
;
      => Scroll( <top>, <left> )
;
       ; SetPos( <top>, <left> )


#command @ <top>, <left> CLEAR TO <bottom>, <right>
;
      => Scroll( <top>, <left>, <bottom>, <right> )
;
       ; SetPos( <top>, <left> )
```

Storing Data

- Every single piece of information allocated thru memory variables is lost when you exit an application
- Database is a table which holds data for storage:



- A row in our table defines a **RECORD**
- The intersection between a row and a column is named **FIELD**
- Each **field** stores a memory variable content
- Each **record** contains information from several memory variables related



Database Structure

- A database file has a header which contains a description of the information stored in it as: **name, type, length, decimals**
- Give to each piece of information a name, no more than 10 characters long
- Choose what type:

Data Type	Field Type
Numeric	Numeric
Character	Character / Memo
Date	Date
Logic	Logic

- give to this field a fixed length
- and number of decimals, if numeric

Field Name	Type	Lenght	Decimal
Name	C	30	0
Code	C	5	0
Salary	N	9	2
BirthDate	D	8	0
Address	C	30	0
City	C	15	0
State	C	2	0
Zip	C	5	0
Phone	C	10	0

- Remember! Avoid duplicate information.

Working with a Database File

- Open the database file in a work area:

```
USE <FileName> NEW
```

or,

```
DBUSEAREA( .T., "dbfntx", <cFileName> )
```

where:
.T. means NEW
dbfntx is the database driver. This is the default, you do not need to type it

- Close the opened work area:

```
CLOSE
```

or,

```
Alias->DBCLOSEAREA()
```

Indexing Files

- Logically sort a database accordingly to some criteria
- The key expression is any valid CA-Clipper expression
- use DTOS() for date fields
- up to 15 indexes per workarea

Working with Indexes

- Create an Index:

```
INDEX ON <indexExp> TO <IndexName>
```

or,

```
DBCREATEINDEX( <cIndexName>, <cExp>, <bExp> )

Alias->( DBCREATEINDEX( <cIndexName>, <cExp>, <bExp> ) )
```

- Set the Index file list with

```
SET INDEX TO <IndexFileList>

SET INDEX TO ( <cIndexFileName> )
```

or,

```
DBSETINDEX( <cIndexFile> )    // ONE per time

Alias->( DBSETINDEX( <cIndexFile> ) )
```

- Change the index order:

```
DBSETORDER( <nOrderNumber> )

Alias->( DBSETORDER( <nOrderNumber> ) )
```

0 means natural order (by record number)

Seeking a record

- Normal procedure

```
SEEK <exp>

DBSEEK( <exp> )

Alias->( DBSEEK( <exp> ) )
```

  - Points to the End-Of-File after a failed key search on index

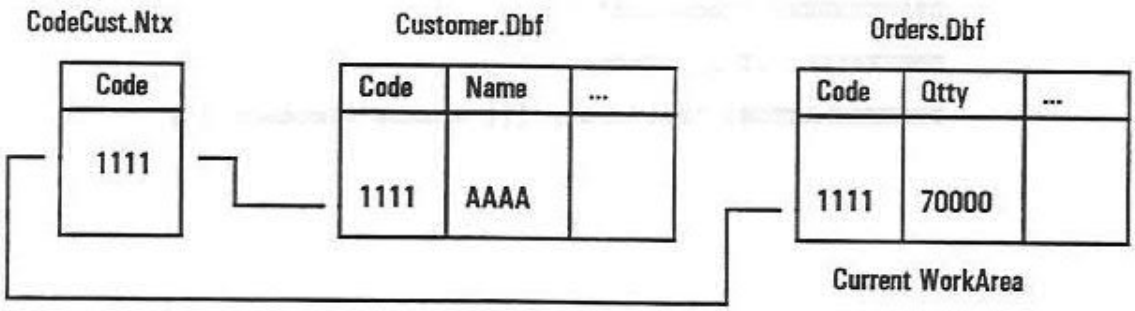
- Using SOFTSEEK

```
SET SOFTSEEK ON
SEEK <exp>
SET SOFTSEEK OFF

DBSEEK( <exp>, .T. )

Alias->( DBSEEK( <exp>, .T. ) )
```

  - Points to the first key with higher value after a failed index search

Relating databases

- Think about data security and redundant information
- You can have the same key as part of the structure of several files
- We are going to use the index key as a "link" to "join" them
- One to many (8)
- Physically:



- Logically:

Code	Name	...	Qtty	...
1111	AAA		700	

Setting a relation

- Open the related database and associated index files
- Open the main database file
- Set the relation between the two:

```
DBSETRELATION( <cAlias>, <bRelation> )
```

- Example:

```
DBUSEAREA( .T.,, "Product" )
DBSETINDEX( "CodeProd" )

DBUSEAREA( .T.,, "Orders" )

DBSETRELATION( "Product", {|| Orders->Product } )
```

Applications

- You are going to see sample code performing:
  - Database file creation
  - a Menuing system, that includes options to add records and print a report
- Before you start programming, make sure you have:

```
SET CA-CLIPPERCMD=/W /N
SET RTLINKCMD=/PLL:BASE50
```

- Use the compile command line:

```
CA-CLIPPER <fileName>
```

- Use the link command line:

```
RTLINK FILE <OBJfile>
```

Creating a Database File

```
/*****
 *
 * Customer.prg
 *
 */

FUNCTION CreateDB()
      LOCAL aStructure :=  {  { "Code",        "C",  5,   0 },;
                              { "Name",        "C",  20,  0 },;
                              { "Address",     "C",  20,  0 },;
                              { "City",        "C",  10,  0 },;
                              { "State",       "C",  2,   0 },;
                              { "Zip",         "C",  5,   0 },;
                              { "Phone",       "C",  7,   0 },;
                              { "Extension",   "C",  3,   0 }    }

      DBCREATE( "Customer", aStructure )


      DBUSEAREA(,, "Customer" )      //  USE Customer

      DBCREATEINDEX( "Code", "Customer->Code", {|| Customer->Code } )

      RETURN ( NIL )
```

Building a Menu

- We are going to see several versions, using
  - @...PROMPT
  - ACHOICE()

Menu with @...PROMPT

```
/*****
 *
 *  Menu.prg - Version 1
 *
 */

#define     CHOICE_QUIT     3
#define     MSG_ROW         MAXROW()


FUNCTION Main()
      LOCAL nChoice

      SCROLL()
      SET MESSAGE TO  MSG_ROW  CENTER
      SET WRAP ON

      WHILE nChoice != CHOICE_QUIT
            @ 06,30 PROMPT "Add    " MESSAGE "Add a new record"
            @ 10,30 PROMPT "Report" MESSAGE "Print Data"
            @ 12,30 PROMPT "Quit   " MESSAGE "Quit"

            MENU TO nChoice

            IF nChoice == 1
                  AddRec()

            ELSEIF nChoice == 2
                  ReportDB()

            ENDIF

      END

      RETURN ( NIL )
```

```
/*****
 *
 *  Menu.prg - Version 2
 *
 */

#define     CHOICE_QUIT     3
#define     MSG_ROW         MAXROW()

FUNCTION Main()
      LOCAL aOptions := {"Add    " ,;
                         "Report" ,;
                         "Quit   "   }

      LOCAL aMessages := {"Add a new record",;
                          "Print Data",;
                          "Quit"    }

      LOCAL aActions := { { || AddRec()   },;
                          { || ReportDB() }   }

      LOCAL nChoice

      SCROLL()
      SET MESSAGE TO  MSG_ROW  CENTER
      SET WRAP ON

      WHILE nChoice != CHOICE_QUIT
            @ 06,30 PROMPT aOptions[1] MESSAGE aMessages[1]
            @ 08,30 PROMPT aOptions[2] MESSAGE aMessages[2]
            @ 10,30 PROMPT aOptions[3] MESSAGE aMessages[3]
            @ 12,30 PROMPT aOptions[4] MESSAGE aMessages[4]

            MENU TO nChoice

            IF nChoice != CHOICE_QUIT
                  EVAL( aFunctions[ nChoice ] )

            ENDIF

      END

      RETURN ( NIL )
```

```
/*****
 *
 *  Menu.prg - Version 3
 *
 */

#define     CHOICE_QUIT      3
#define     MSG_ROW          MAXROW()
#define     MENU_COLOR        "GR+/W"

FUNCTION Main()
      LOCAL aCoord := { { 06, 35 },;
                        { 08, 35 },;
                        { 10, 35 } }

      LOCAL aOptions := {"Add    " ,;
                         "Report" ,;
                         "Quit   "   }

      LOCAL aMessages := {"Add a new record",;
                          "Print Data",;
                          "Quit"    }

      LOCAL aActions := { { || AddRec()   },;
                          { || ReportDB() }   }

      LOCAL nChoice

      BuildScr()
      SET MESSAGE TO  MSG_ROW  CENTER
      SET WRAP ON

      WHILE nChoice != CHOICE_QUIT

            FOR i := 1 TO LEN( aOptions )
                  @ aCoord[i][1], aCoord[i][2];
                        PROMPT aOptions[i] MESSAGE aMessages[i]
            NEXT

            MENU TO nChoice

            IF nChoice != CHOICE_QUIT
                  EVAL( aActions[ nChoice ] )

            ENDIF

      END

      RETURN ( NIL )
```

```
/*****
 *
 *  BuildScr()  -->  NIL
 *
 */

STATIC FUNCTION BuildScr()
     SCROLL()

     @  1, 0 SAY PADC( "Main Menu", MAXCOL()+1 ) COLOR MENU_COLOR

     RETURN  ( NIL )
```

```
/*****
 *
 * Menu.prg - Version 4
 *
 */

#define     CHOICE_QUIT     3
#define     MSG_ROW         MAXROW()
#define     MENU_COLOR      "GR+/W"

FUNCTION Main()
      LOCAL aCoord := { { 06, 35 },;
                        { 08, 35 },;
                        { 10, 35 } }

      LOCAL aOptions := {"Add    " ,;
                         "Report" ,;
                         "Quit   "   }

      LOCAL aMessages := {"Add a new record",;
                          "Print Data",;
                          "Quit"    }

      LOCAL aActions := { { || AddRec()   },;
                          { || ReportDB() }   }

      LOCAL nChoice

      BuildScr()
      SET MESSAGE TO  MSG_ROW  CENTER
      SET WRAP ON

      WHILE nChoice != CHOICE_QUIT

            DrawMenuPrompt( aCoord, aOptions, aMessages )
            MENU TO nChoice

            IF nChoice != CHOICE_QUIT
                  EVAL( aActions[ nChoice ] )

            ENDIF

      END

      RETURN ( NIL )
```

```
/*****
 *
 *  BuildScr()  -->  NIL
 *
 */

STATIC FUNCTION BuildScr()
      SCROLL()

      @  1, 0 SAY PADC( "Main Menu", MAXCOL()+1 ) COLOR MENU_COLOR

      RETURN  ( NIL )


/*****
 *
 * DrawMenuPrompt()
 *
 */

STATIC FUNCTION DrawMenuPrompt( aCoord, aOptions, aMessages )
      LOCAL i

      FOR i := 1 TO LEN( aOptions )

            @ aCoord[i][1], aCoord[i][2];
                  PROMPT aOptions[i] MESSAGE aMessages[i]

      NEXT

      RETURN ( NIL )
```

Menu with ACHOICE()

```
/*****
 *
 *  Menu.prg - Version 1
 *
 */

#define     CHOICE_QUIT       3
#define     MENU_COLOR        "GR+/W"

FUNCTION Main()
      LOCAL aChoices := { "Add    ",;
                          "Report",;
                          "Quit   "    }

      LOCAL aActions := { {||AddRec()},;
                          {||ReportDB()}   }

      LOCAL nChoice

      BuildScr()

      WHILE nChoice != CHOICE_QUIT

            nChoice := ACHOICE( 10, 10, 13, 20, aChoices )

            IF nChoice != CHOICE_QUIT .AND. nChoice != 0
                  EVAL( aActions[nChoice] )

            ENDIF

      END

      RETURN ( NIL )

/*****
 *
 *  BuildScr()  --> NIL
 *
 *  Build the Main Screen
 *
 */

STATIC FUNCTION BuildScr()
   SCROLL()

   @  1, 0 SAY PADC( "Main Menu", MAXCOL()+1 ) COLOR MENU_COLOR

   RETURN ( NIL )
```

```
/*****
 *
 *  Menu.prg - Version 2
 *
 */

#include "InKey.ch"
#include "Achoice.ch"

#define     CHOICE_QUIT       3
#define     MSG_ROW           MAXROW()
#define     MSG_COLOR         "R/W"

STATIC aMessages :=  { "Add a new record",;
                       "Print Data"      ,;
                       "Quit"                }

FUNCTION Main()
      LOCAL aChoices := { "Add    ",;
                          "Report",;
                          "Quit   "    }

      LOCAL aActions := { {||AddRec()},;
                          {||ReportDB()}  }

      LOCAL nChoice

      BuildScr()

      WHILE nChoice != CHOICE_QUIT

            KEYBOARD CHR( K_CTRL_PGUP )

            nChoice := ACHOICE( 10, 10, 13, 20, aChoices,, "CtrAch" )

            IF nChoice != CHOICE_QUIT .AND. nChoice != 0
                  EVAL( aActions[nChoice] )

            ENDIF

      END

      RETURN ( NIL )
```

```
/*****
 *
 *  BuildScr()  -->  NIL
 *
 *  Build the Main Screen
 *
 */

STATIC FUNCTION BuildScr()
   SCROLL()

   @  2, 1 SAY PADC( "Main Menu", MAXCOL()-2 ) COLOR MENU_COLOR

   RETURN ( NIL )


/*****
 *
 * CtrAch()
 * Achoice control
 *
 */

FUNCTION CtrAch( nMode, nElement, nRelPos )
      LOCAL nKey := LASTKEY()
      LOCAL nRet := AC_CONT

      IF nMode == AC_IDLE
            @ ROW_MENS, 0 SAY PADC( aMessages[nElement], MAXCOL()+1 );
                  COLOR MSG_COLOR

      ELSEIF nMode == AC_HITTOP
            KEYBOARD CHR( K_CTRL_PGDN )

      ELSEIF nMode == AC_HITBOTTOM
            KEYBOARD CHR( K_CTRL_PGUP )

      ELSEIF nMode == AC_EXCEPT
            IF nKey == K_ENTER
                  nRet := AC_SELECT

            ELSEIF nKey == K_ESC
                  nRet := AC_ABORT

            ELSE
                  nRet := AC_GOTO

            ENDIF

      ENDIF

      RETURN ( nRet )
```

Entering Data

```
/*****
 *
 *  GetData.prg
 *
 */

#include "InKey.ch"
#include "Set.ch"
#include "Customer.ch"

#define  ON      .T.

STATIC aScr := {  { 04, 10, 22, "Code           : " }, ;
                  { 06, 10, 22, "Name           : " }, ;
                  { 08, 10, 22, "Address        : " }, ;
                  { 10, 10, 22, "City           : " }, ;
                  { 12, 10, 22, "State          : " }, ;
                  { 14, 10, 22, "Zip            : " }, ;
                  { 16, 10, 22, "Phone          : " }, ;
                  { 18, 10, 22, "Extension      : " }  }

FUNCTION AddRec()
      LOCAL cScreen, cColor, lConfirm
      LOCAL aGet

      cScreen     := RESTSCREEN( 0, 0, MAXROW(), MAXCOL() )
      lConfirm    := SET( _SET_CONFIRM, ON )
      cColor      := SETCOLOR( "B+/N,GR+/B,,,RB/N" )

      BuildScr()

      DBUSEAREA( ,, "Customer" )

      WHILE .T.
            Customer->( DBSKIP( LASTREC() + 1 ) )
            aGet := Customer->( InitData() )
            GetData( aGet )

            IF LASTKEY() == K_ESC
                  EXIT
            ENDIF

            Customer->( DBAPPEND() )
            Customer->( ReplData( aGet ) )

      END
```

```
      Customer->( DBCLOSEAREA() )

      SET( _SET_CONFIRM, lConfirm )
      SETCOLOR( cColor )
      RESTSCREEN( 0, 0, MAXROW(), MAXCOL(), cScreen )

      RETURN ( NIL )


/*****
 *
 *  BuildScr()
 *
 */

STATIC FUNCTION BuildScr()
      LOCAL i

      SCROLL()
      @ MSG_ROW, 0 SAY PADC("Press ESC to quit", MAXCOL()+1);
            COLOR MSG_COLOR

      FOR i := 1 TO LEN( aScr )
            @ aScr[i][1], aScr[i][2] SAY aScr[i][4]
      NEXT

      RETURN ( NIL )


/*****
 *
 *  InitData() --> aGet
 *
 */

FUNCTION InitData()
      LOCAL nFCnt, aGet, i

      nFCnt := FCOUNT()
      aGet  := ARRAY( nFCnt )

      FOR i := 1 TO nFCnt
            aGet[i] := FIELDGET( i )

      NEXT

      RETURN ( aGet )
```

```
/*****
 *
 *  ReplData( aGet ) --> NIL
 *
 */

FUNCTION ReplData( aGet )
      LOCAL i

      FOR i := 1 TO LEN( aGet )
            FIELDPUT( i, aGet[i] )

      NEXT

      RETURN ( NIL )
```

```
/*****
 *
 *  GetData( aGet ) --> NIL
 *
 */

FUNCTION GetData( aGet )
     LOCAL GetList := {}

     @ aScr[CODE][1], aScr[CODE][3] GET aGet[CODE];
          PICTURE "99999" ;
          VALID !EMPTY( aGet[CODE] )

     @ aScr[NAME][1], aScr[NAME][3] GET aGet[NAME];
          PICTURE "@!" ;
          VALID !EMPTY( aGet[NAME] )

     @ aScr[ADDRESS][1], aScr[ADDRESS][3] GET aGet[ADDRESS];
          PICTURE "@!"

     @ aScr[CITY][1], aScr[CITY][3] GET aGet[CITY];
          PICTURE "@!" ;
          WHEN  !EMPTY( aGet[ADDRESS] );
          VALID !EMPTY( aGet[CITY] )

     @ aScr[STATE][1], aScr[STATE][3] GET aGet[STATE];
          PICTURE "@!A" ;
          WHEN  !EMPTY( aGet[ADDRESS] );
          VALID !EMPTY( aGet[STATE] )

     @ aScr[ZIP][1], aScr[ZIP][3] GET aGet[ZIP];
          PICTURE "99999" ;
          WHEN  !EMPTY( aGet[ADDRESS] );
          VALID !EMPTY( aGet[ZIP] )

     @ aScr[PHONE][1], aScr[PHONE][3] GET aGet[PHONE];
          PICTURE "@R 999-9999"

     @ aScr[EXTENSION][1], aScr[EXTENSION][3] GET aGet[EXTENSION];
          PICTURE "999" ;
          WHEN !EMPTY( aGet[PHONE] )

     READ

     RETURN ( NIL )
```

Creating a Report

```
/*****
 *
 *  Report.prg
 *
 */

#include "Set.ch"

#define    LINES_PER_PAGE    60
#define    START_LINE        5
#define    ON                .T.
#define    OFF               .F.

STATIC nLineNo := LINES_PER_PAGE + 1
STATIC aCol := {  1, 8, 30, 52, 64, 71 }


FUNCTION ReportDB()
      SCROLL()

      DBUSEAREA( ,, "Customer" )
      DBSETINDEX( "Code" )
      DBGOTOP()

      SET( _SET_PRINTER, ON )
      SET( _SET_CONSOLE, OFF )
      SET( _SET_DEVICE, "PRINTER" )

      WHILE ! EOF()

            HeaderPage()
            PrintRec()
            DBSKIP()

      END

      SET( _SET_PRINTER, OFF )
      SET( _SET_CONSOLE, ON )
      SET( _SET_DEVICE, "SCREEN" )

      DBCLOSEAREA()

      RETURN ( NIL )
```

```
/*****
 *
 *  HeaderPage()  -->  NIL
 *
 */

STATIC FUNCTION HeaderPage()
      IF nLineNo > LINES_PER_PAGE
            PrintHead()

      ENDIF

      RETURN ( NIL )


/*****
 *
 *  PrintHead()  -->  NIL
 *
 */

STATIC FUNCTION PrintHead()
      STATIC nPageNo := 0

      nLineNo := START_LINE

      @ nLineNo++, 70 SAY TRANSFORM( ++nPageNo, "Page No. 99" )

      @ nLineNo++, 0 SAY PADC( "Customer's Address Listing" , 80 )
      @ nLineNo++, 0 SAY PADC( REPLICATE( "-", 26 ), 80 )

      @ ++nLineNo, aCol[1] SAY "Code"
      @   nLineNo, aCol[2] SAY "Name"
      @   nLineNo, aCol[3] SAY "Address"
      @   nLineNo, aCol[4] SAY "City"
      @   nLineNo, aCol[5] SAY "State"
      @   nLineNo, aCol[6] SAY "Zip"

      @ ++nLineNo, aCol[1] SAY REPLICATE( "-",  5 )
      @   nLineNo, aCol[2] SAY REPLICATE( "-", 20 )
      @   nLineNo, aCol[3] SAY REPLICATE( "-", 20 )
      @   nLineNo, aCol[4] SAY REPLICATE( "-", 10 )
      @   nLineNo, aCol[5] SAY REPLICATE( "-",  5 )
      @   nLineNo, aCol[6] SAY REPLICATE( "-",  5 )

      RETURN ( NIL )
```

```
/*****
 *
 *  PrintRec()  -->  NIL
 *
 */

STATIC FUNCTION PrintRec()
     @ ++nLineNo, aCol[1] SAY Customer->Code
     @   nLineNo, aCol[2] SAY Customer->Name
     @   nLineNo, aCol[3] SAY Customer->Address
     @   nLineNo, aCol[4] SAY Customer->City
     @   nLineNo, aCol[5] SAY Customer->State
     @   nLineNo, aCol[6] SAY Customer->Zip

     RETURN ( NIL )
```

Using RMake

- Create the RMake file:

```
/*****
 *
 * CUSTOMER.RMK
 *
 * Make file to create CUSTOMER.EXE
 *
 */

// Compile files
.prg.OBJ :
   SET CA-CLIPPERCMD=/N /W /P /B
   CA-CLIPPER $<

MENU.OBJ     : Menu.prg
GETDATA.OBJ  : GetData.prg
REPORT.OBJ   : Report.prg

// Link files
.OBJ.EXE :
   SET RTLINKCMD=/POSI /PLL:BASE50
   RTLINK $**, $*;

CUSTOMER.EXE : MENU.OBJ GETDATA.OBJ REPORT.OBJ
```

- From the DOS prompt:

```
C:\TEST>RMAKE Customer
```