Macro-PLUS™ Programming

OPTICAL RESEARCH ASSOCIATES

3280 East Foothill Boulevard
Pasadena, California 91107 USA
(626) 795-9101 Fax (626) 795-0184
e-mail: service@opticalres.com
World Wide Web: http://www.opticalres.com

What is Macro-PLUS?

- Macro-PLUS is a programming language built into CODE V
- It allows you to customize CODE V to your needs
- It adds flexibility and power to CODE V
- Historically, macros started as simple sequences of commands
  - That is why macros are saved as .SEQ files
  - In version 7.20 (about 1988), Macro-PLUS was added to CODE V
  - The terms “sequence” and “macro” are roughly synonymous
    - Some use the name "sequence" for simple sequences of commands (such as the commands for an AUTO run) and the name "macro" for more complex calculations
    - However, they are both stored in .SEQ files and are executed the same way (with the IN command)

A Typical Simple Macro

```
!  Macro to list individual element focal lengths, clear aperture
!  diameters, and ET's at max of the two semi-diameters.
^format == "'3d' '3d' '9d.4d'  '8c' '4d.4d' '7d.4d'"
wri "Surfaces   Focal Length Glass     C.A. Diam.  Edge Thickness"
for ^s 1 (num s)-1
   if (gla s^s) <> ""
       ^s2 == ^s+1
       ^c1 == (cuy s^s) ; ^c2 == (cuy s^s2)
       if ^c1=0 and ^c2=0
              ^FL == 0
       else
              ^n == (ind s^s) ; ^t == (thi s^s)
              ^FL == 1/(^n-1)/(^c1-^c2+^t*(^n-1)*^c1*^c2/^n)
       end if
       ^sd == maxf((sd s^s),(sd s^s2))
       ^et == (thi s^s) - sagf(^s,1,0,^sd) + sagf(^s2,1,0,^sd)
       wri q^format ^s ^s2 ^FL (gla s^s) 2*^sd ^et
   end if
end for
```

Formatted output

Loop

Variables

Access to lens data

IF statement

Functions

OPTICAL RESEARCH ASSOCIATES

What Can I Do With Macros?

- **Level 1:** Nothing
  - You can ignore macros if you want to - they are not required to run CODE V.
- **Level 2:** Use pre-written macros.
  - ORA supplies over 200 macros with CODE V - no programming needed.
  - Many of these are integrated into the CODE V GUI without you realizing it!
- **Level 3:** Use macro expressions in place of numbers
  - Use CODE V as a calculator, using expressions in place of numbers.
- **Level 4:** Command sequences
  - These are text files of CODE V commands, such as optimization or tolerancing input.
- **Level 5:** Complex macros
  - These are macros which may include loops and branching and compute user-desired quantities or output.

Sample Macros Supplied With CODE V

- Over 200 macros are supplied with CODE V.
- These sample macros have three purposes:
  - Demonstrate the type of things that can be done with macros
  - Extend the capabilities of CODE V in many areas
  - Provide examples to users who wish to modify the samples or write their own macros
- You do not have to program to use these sample macros.

Integrated Macros

- Some macros have been integrated into the user interface as standard features.
- These include macros for user-defined tolerancing, inserting various prisms, and distortion grid plotting.
- **Removing or changing certain macros in the CV_MACRO: directory may disable some program features, so please don't change them!**
  - Copy any macros that you wish to change to another folder before editing.

Using Supplied Macros

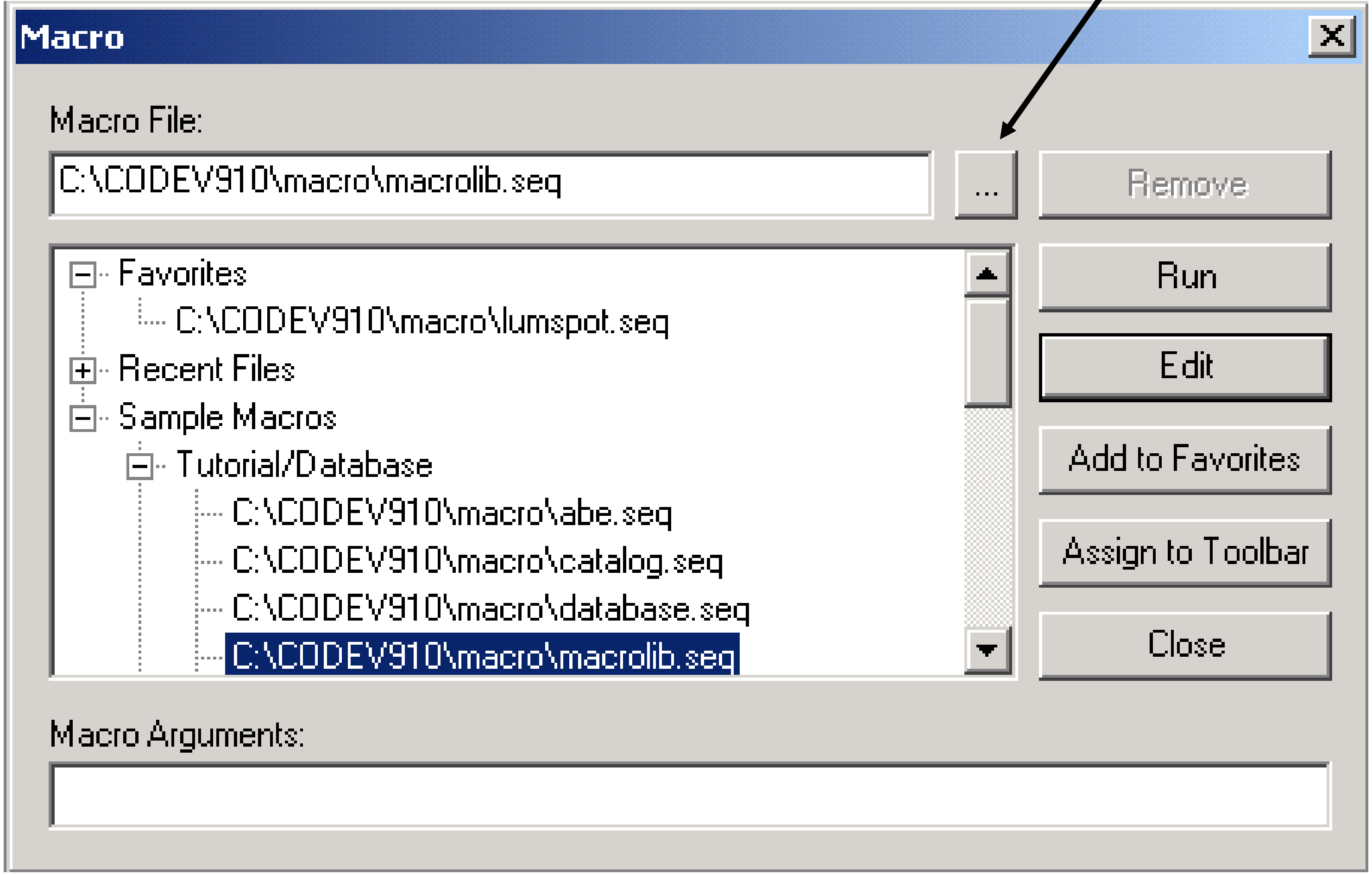
- Macros are executed via a dialog box launched from the **Tools > Macro Manager** menu.
  - Also available from the command line (**IN** command)
- From here you can browse through available macros, including categorized ORA samples as well as your favorites and recently used macros.
- You can also edit the selected macro, or just view it in the editor.
- You can assign a frequently used macro to a toolbar icon.
- A special library macro exists for viewing sample macros from the command prompt:

```
CODE V> IN CV_MACRO:MACROLIB
```

The next slide shows this macro via the **Tools > Macro Manager** dialog

The Tools > Macro Manager Dialog

Browse Button



Macro File:

C:\CODEV910\macro\macrolib.seq

Favorites
- C:\CODEV910\macro\lumspot.seq

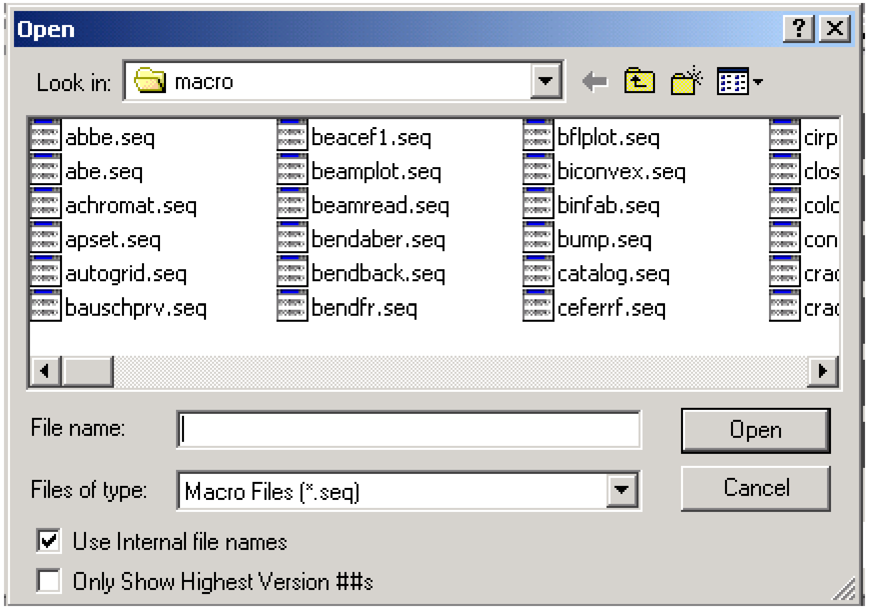
Recent Files

Sample Macros
- Tutorial/Database
  - C:\CODEV910\macro\abe.seq
  - C:\CODEV910\macro\catalog.seq
  - C:\CODEV910\macro\database.seq
  - C:\CODEV910\macro\macrolib.seq

Macro Arguments:

Running Macros from Tools > Macro Manager

- Choose **Tools > Macro Manager** and select a macro from the navigation tree, then click the Run button
  - The last macro you ran will be shown in the file name field, and you can click **Run** to re-run it
- If the macro is not in the navigation tree, click the browse button to locate it:



Open

Look in: macro

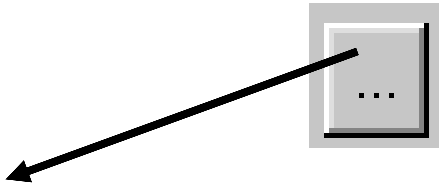
abbe.seq
abe.seq
achromat.seq
apset.seq
autogrid.seq
bauschprv.seq
beacef1.seq
beamplot.seq
beamread.seq
bendaber.seq
bendback.seq
bendfr.seq
bflplot.seq
biconvex.seq
binfab.seq
bump.seq
catalog.seq
ceferrf.seq
cirp
clos
colc
con
crac
crac

File name:

Files of type: Macro Files (*.seq)

☑ Use Internal file names

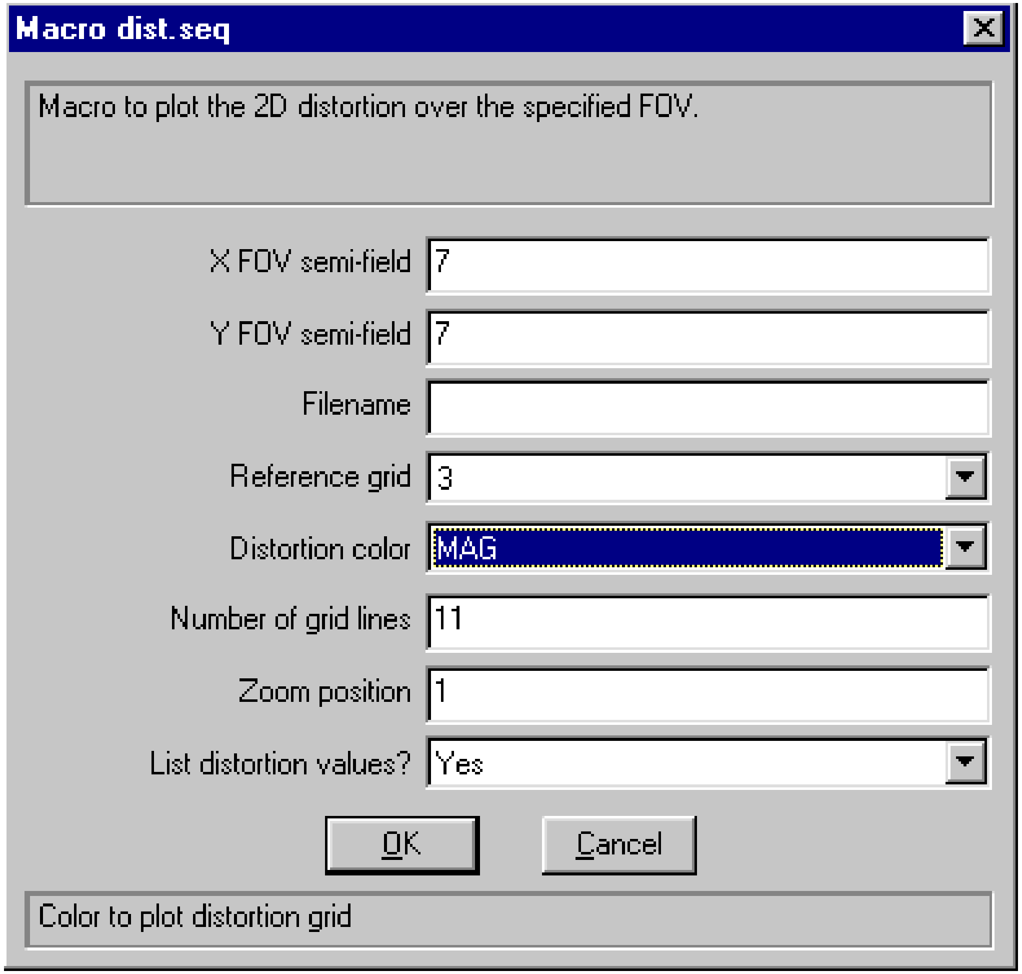
□ Only Show Highest Version ##s



Macro Input Dialog Boxes

- When you launch a macro from the Macro dialog box, it will display its own input dialog box if one is defined for it
  - The dialog box is created via specially formatted “! ARG” comments in the macro source code
  - These comments are interpreted at run time to build the dialog box
- In the dialog box, you can enter values for any input arguments that the macro requires. (More about this subject later.)
- The input dialog box does *not* appear if you run a macro from the command line with the **IN** command

Macro Dialog Example



Macro dist.seq

Macro to plot the 2D distortion over the specified FOV.

X FOV semi-field: 7

Y FOV semi-field: 7

Filename:

Reference grid: 3

Distortion color: MAG

Number of grid lines: 11

Zoom position: 1

List distortion values?: Yes

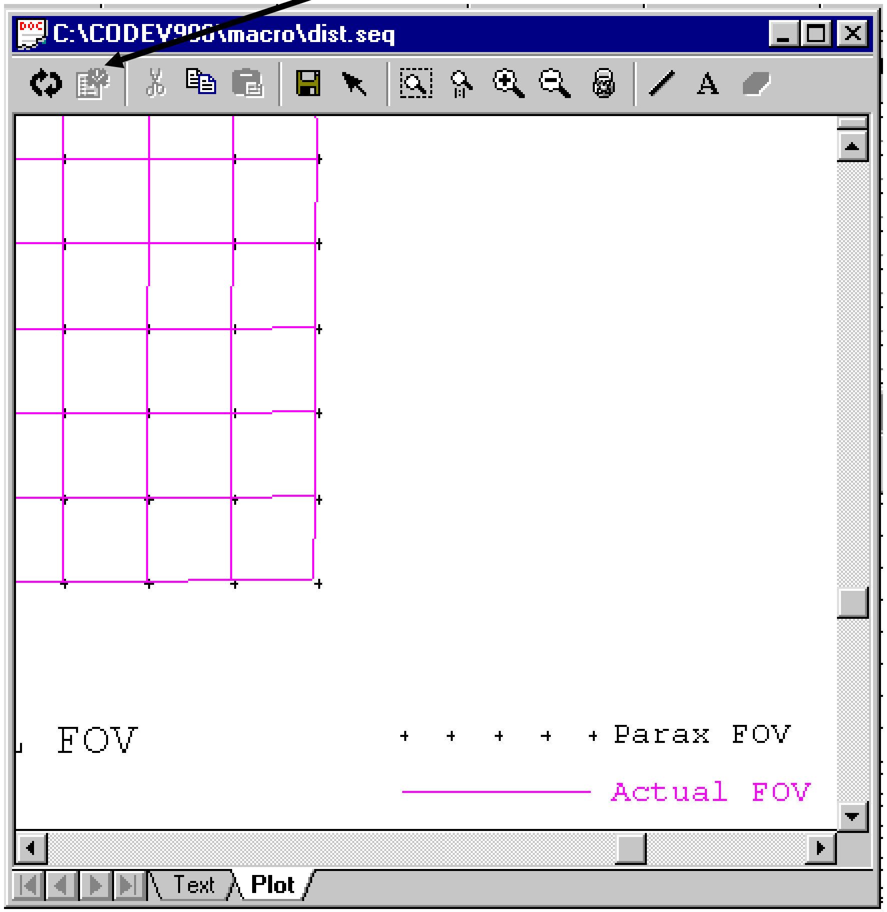
Color to plot distortion grid

Tabbed Output Windows for Macros

- Output from a macro run from the macro dialog appears in a tabbed output window (TOW) similar to that of a CODE V options.
  - A text tab always appears; numbered graphics tab(s) and an info tab may appear depending on the macro.
  - Any text output also appears in the command window.
- You can re-run the macro with the re-execute button (for example, if you change the lens)
- As of version 10.2, you can re-run the macro with new inputs by using the settings button.
- You can also create a TOW for a macro run from the command line by using the **TOW** command:
  - `CODE V > TOW; IN CV_MACRO:SPOT2D`
  - All output from all commands on the same line as the **TOW** command will be written/displayed in a TOW
    - Use ";" to separate commands on the same line

Example of Tabbed Output Window

- Note that the Settings button is dimmed out, because this picture is older than version 10.2!
- In versions older than 10.2, you must re-run the macro from the **Tools > Macro** menu (or from the command line) to modify the input arguments.



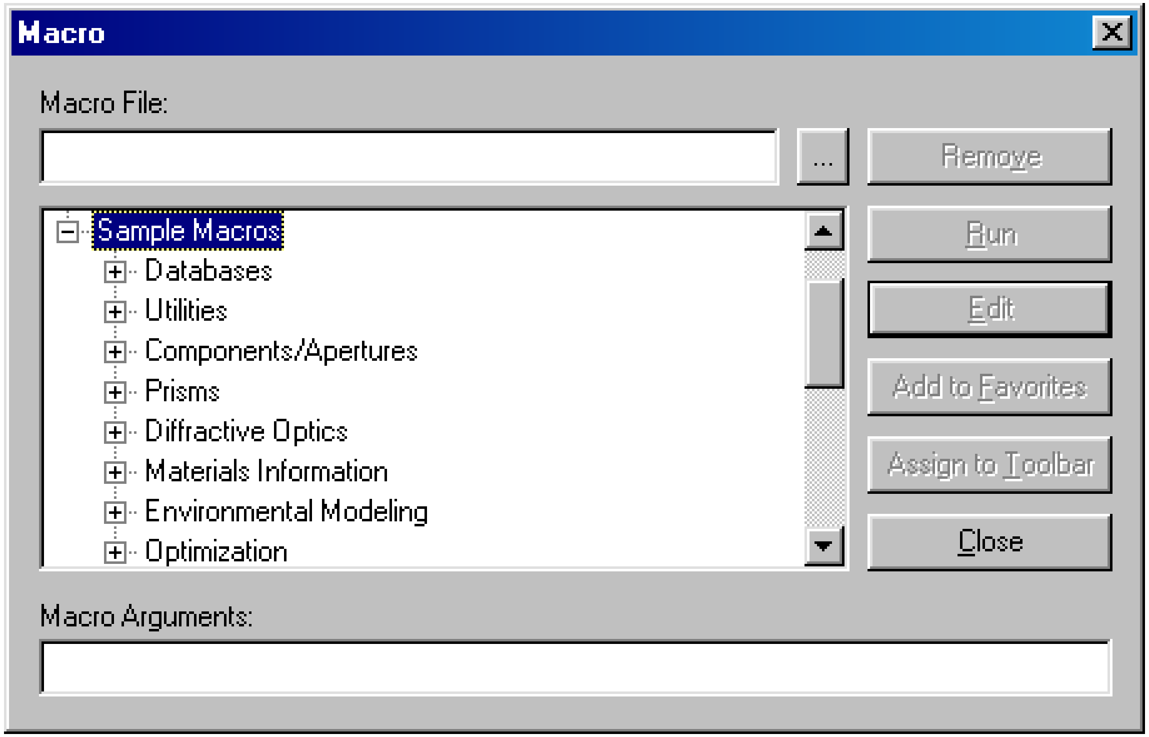
Settings Button

Running Macros as Commands

- You can run macros in the Command Window using the **IN** command.
  - For example, `IN CV_MACRO:FL`
  - The **PTH** command is very useful:
    `CODE V > PTH SEQ APP CV_MACRO:`
    - This allows you to simply type `IN FL`
- The default behavior for output is that text appears only in the Command Window, and graphics appear in separate, non-updating plot windows.
  - If you precede the **IN** command with the **TOW** command, the output will appear in a tabbed output window, such as `TOW IN FL`
- A few of the older ORA-supplied macros only work well from the Command Window (e.g., the LOOP* animation macros)

Supplied Macro Categories

- The ORA Sample Macros are organized in a collapsing tree structure with 17 categories.
- These categories make it easier to find what you want.



Macro

Macro File:

Sample Macros
- Databases
- Utilities
- Components/Apertures
- Prisms
- Diffractive Optics
- Materials Information
- Environmental Modeling
- Optimization

Macro Arguments:

Use the Edit Button to View Comments

- Even if you don’t plan to write or change macros, the Edit button can still be very useful.
- Most ORA Sample macros have documentation comments at the top of the macro source code.
  - These may provide a more thorough description of what the macro does, its history, algorithm used, caveats, etc.
- Select the macro and click the Edit button to view the macro in a separate editor window.
- **Don’t change an ORA-supplied macro!**
  - If you want to make a customized version of a supplied macro, use **File > Save As** to save a copy elsewhere, and modify the copy!

Some Sample Macros Supplied With CODE V

- Utilities – Convert surface(s) to GLB decenter (MAKEGLBS)
- Utilities - Reference Ray Check (REFCHECK)
- Utilities – Interactive Macro-recording Macro (MACRO)
- Materials Information - Private Catalog for plastics (PLASTICPRV)
  - The old name PRVPL still works.
- Optimization - AUTO Ray Grid Plot (AUTOGRID)
- 1st Order Analysis - BFL vs. Wavelength (BFLPLOT)
- Geometrical Analysis- False color Spot Diagram (LUMSPOT)
- Diffraction Analysis - PSF Slice Plot (PSFPLOT)
- Diffractive Optics - HOE Phase Plot (HCOPLOT)
- Polarization - Polarization Pupil Map (POLDSP)

Tips for Running the Sample Macros

- Set the **PTH** command if running from the Command Window.

- Turn off the verify flag:
  - `CODE V> VER NO`
  - This suppresses the echoing of macro commands during macro execution, so the output is cleaner and less cluttered.
    - Turning verify on (**VER Y**) is useful for debugging.

- The **VER** command can also be included in your DEFAULTS.SEQ sequence (discussed later).

Macro-PLUS Major Programming Features

- Global and local variables
- Numeric and string variables
- One- and two-dimensional arrays
- Access to the lens database
- Many built-in functions (math, string, optical, etc.)
- User-defined macro functions
- Control structures and branching (WHILE, UNTIL, IF, etc.)
- Subprograms (macros can call macros)
- Interaction (read, write) with text files
- Format control of input and output
- Unlimited program size

Programming in Macro-PLUS

- You need to be familiar with CODE V command mode
  - LDM commands
  - Option commands
- You need to know Macro-PLUS commands
  - Defining and assigning variables
  - Expressions
  - Branching and looping
  - Input and output
- You need to be familiar with basic programming concepts
  - If you ever learned Basic, C, Fortran, Pascal, etc., you'll do fine!
- CVEdit is a text editor supplied with CODE V
  - Supports file versioning
  - Open from CODE V command line:

```
CODE V > edi test.seq
```

Macro Variables

- Macro-PLUS has both numeric and string variables.
- The names of variables all start with **^** (caret or "hat"):
  `^x   ^bigval   ^string3`
  - This is to avoid conflict with CODE V commands
- Variable names can contain letters, numbers, and underscores (_)
  - Example: **`^the_original_value`**
  - The name must start with a letter.
- Once a variable is declared as numeric or string type, its type cannot be changed.
  - Trying to assign the wrong data type to a variable will result in an error.
  - Variables can be dropped (**DROP** command) and redeclared.

Declaring Variables

- Macro-PLUS must know what type a variable is (numeric or string).
- This declaration can be done implicitly or explicitly.
- Implicit declaration is done by simply assigning data to a new variable:

```
^x == 3           - implicitly declared as numeric type
^y == "hello"     - implicitly declared as string type
```

- Explicit declaration is done with the **NUM** or **STR** command:

```
num ^a ^b ^num   - explicitly declared as numeric types
str ^m ^n ^text  - explicitly declared as string types
```

- In macros, variable declarations must be made before any other executable statements.
- The command **CHK Y** forces all variables to be explicitly declared before use. This is useful for debugging.
- There is no integer variable type.

Local and Global Variables

- **Local variables** are only known in their current context.
  - If a local variable is defined interactively (at the CODE V> prompt), it is not known in any macro.
  - If a local variable is defined in any macro, it is not known in other macros or at the interactive level.
  - Declare with **LCL** command (e.g., `lcl num ^x`).
- **Global variables** are known throughout CODE V.
  - Global variables defined in any macro or interactively are always known.
  - Declare with **GBL** command (e.g., `gbl str ^text`).
- If **LCL** or **GBL** is not given in a declaration, **LCL** is assumed
- Global and local variable conflicts are possible.
  - An implicitly declared variable will use the global definition, if it exists.
  - Local variables explicitly declared in a macro will override any global variables previously defined with the same name.
    - It is always good practice to explicitly declare local variables in macros to avoid possible conflicts with global variables.

Array Variables

- One- and two-dimensional arrays can be defined
  - Can be numeric or string
- Array variables must be explicitly declared before use
- One-dimensional array (vector)

```
num ^field(5)
^field(2) == (yan f2)
```

- Two-dimensional array (matrix)

```
num ^x(10,10)
^x(5,5) == 25
```

- The array subscript can start at numbers other than 1

```
num ^x(11) ^y(-5..5) ^z(20..31)
str ^a(11,11) ^b(-5..5,10..21)
```

- Array sizes are unlimited (limited only by memory)

Database Access

- Most of the data in the lens database is available to Macro-PLUS.
  - Lens data (radii, thickness, EPD, fields, aspheric coefficients, etc.)
  - Calculated values (EFL, ray trace data, etc.)
- A database item usually has the same name as the corresponding lens data command, or an **AUT** constraint.
- The database names are always enclosed in parentheses.
  - This identifies them as database items.
- Examples:

```
(rdy s3)
(epd)
(y r1 f3 w2 z2 s7)
```

- Database items can return numeric data or string data

```
eva (rdy s3)
   (rdy s3) = 57.1234
eva (gla s1)
   (gla s1) = "BK7"
```

Use of Database Items

- Database items can be used for variable assignment (note that the assignment operator in Macro-PLUS is the double ==)

  ```
  ^rdy_s1 == (rdy s1)
  ^y3 == (y s3)
  ```

- They can be used in expressions

  ```
  ^x == (rdy s3)*2 + (thi s4)/(thi s5)
  ^y == sinf((ade s2)/57.29578)
  ```

- They can be used in CODE V commands

  ```
  rdy s1 -(rdy s1)
  epd (epd)
  auto;efl = (efl);go
  ```

- They can be used in data queries (with **EVA** command)

  ```
  eva (rdy s2)
  eva (efl)
  ```

Macro Expressions

- Expressions are mixtures of variables, database items, functions, and operators
- Expressions have many uses
  - Variable assignment
    `^pupil_area == ^pi/4*(epd)**2`
  - Can be used in place of numbers in CODE V commands
    `thi s3 90-(thi s2)`
    - Note that this does NOT create a permanent relationship (database items are just numbers or strings)
    - To form a relationship, use a pickup
      `pik thi s3 thi s2 -1 90`
- Always remember to enclose database items in parentheses!

Rules for Expressions

- Expressions consist of operators, parentheses, and operands
  - Operators
    - Unary + or -
    - Arithmetic (+, -, *, /, **)
    - Relational (<, <=, =, >=, >, <>)
    - Logical (NOT, AND, OR)
  - Parentheses are used for database items, function arguments, and to change the order of evaluation of operators
  - Operands are variables, functions, constants, database items
- In variable assignments, expressions can include spaces for readability

  ```
  ^x == (^a1 + ^a2) / (^b1 + ^b2) + ^c
  ```

- In CODE V commands, expressions should not include spaces unless enclosed in parentheses

  ```
  yan 0 10 10 + 10          - gives an error
  yan 0 10 (10 + 10)        - OK
  ```

Expression Examples

- Evaluation

```
EVA (EFL)
EVA TANF((ADE S3)/57.2958)
```

- Lens data definition

```
THI S5 -(THI S3)/2
```

- AUTO constraints

```
EFL = (EFL)
```

- Miscellaneous

```
^i == 8
^tau == (thi s^i) / (ind s^i)         - t/n
^nmn == (ind s^i+1) - (ind s^i)       - n' - n
^radial_dist_3 == sqrtf((x s3)**2 + (y s3)**2))
^metric == (dim)='M' or (dim)='C'     - TRUE = 1
^normal == (yan f1) > (yan fL)        - FALSE = 0
```

Loops

- Macro-PLUS has three methods for looping through program steps
- **FOR** loop
  - Loops through steps a fixed number of times, based on starting value, final value, and increment (they do not need to be integers)

    ```
    FOR ^i 1 20 2
    ```
- **WHILE** loop
  - Continues to loop through steps while some condition is true
  - Test is at the top of the loop, so execution occurs 0 or more times

    ```
    WHILE ^x = 3
    ```
- **UNTIL** loop
  - Loops through steps until some condition is true
  - Test is at bottom of loop, so execution occurs at least one time

    ```
    UNTIL
    ...
    END UNTIL ^x = 5
    ```

Loop Examples

- Macro to list radius, thickness, and glass for all surfaces
  - With **FOR** loop:

```
for ^i 0 (num s)
  wri ^i (rdy s^i) (thi s^i) (gla s^i)
end for
```

  - With **WHILE** loop:

```
^i == 0
while ^i <= (num s)
  wri (rdy s^i) (thi s^i) (gla s^i)
  ^i == ^i + 1
end while
```

  - With **UNTIL** loop:

```
^i == 0
until
  wri (rdy s^i) (thi s^i) (gla s^i)
  ^i == ^i + 1
end until ^i > (num s)
```

IF Tests

- **IF** tests allow conditional execution of a section of code
  - The basic idea is "If TRUE, do this, if FALSE, skip it"
- The **IF** test allows multiple additional tests (**ELSE IF**), plus allows a final choice (**ELSE**)
- The basic structure is

```
IF test
...
ELSE IF test
...
ELSE
...
END IF
```

optional

- Note that unlike FORTRAN, **END IF** is two words (otherwise, Macro-PLUS does not know what you are ending, since it only reads the first 3 letters)

IF Example

- Test to check on lens units, and set a scale factor of mm per lens unit

```
if (dim) = "I"
   wri "Dimensions are inches"
   ^scale == 25.4
else if (dim) = "C"
   wri "Dimensions are centimeters"
   ^scale == 10.0
else
   wri "dimensions are millimeters"
   ^scale == 1.0
end if
```

The GOTO Command

- The **GOTO** command transfers execution unconditionally to a corresponding **LBL** statement (label)

  ```
  GOTO NEXT
  ...
  LBL NEXT
  ```

- The **LBL** statement can be after, or before the **GOTO** command
  - There can be only one **LBL** statement with a given label
- Note that there is no space between GO and TO
  - `GOTO NEXT`, *not* `GO TO NEXT`
- Excessive use of **GOTO**s can make a macro hard to understand
  - Programming purists think they're inelegant.
  - However, they are useful in many situations.

Macros Calling Macros

- Complex macros can often be broken into smaller macros where some macros call others (like subroutines in other programming languages).
- Macros can call other macros, which in turn can call other macros, etc.
  - The calling depth is unlimited.
  - When a macro terminates, it returns execution to wherever it was called from.
  - A macro terminates when it reaches its end or when it encounters an **RTN** command (return).
- Note that when any macro encounters an error, even if it's in a subroutine call, execution stops and you are returned to the interactive level.

Replacement Fields (Arguments)

- When you run a macro, you often want to pass parameter values to it.
- Replacement fields can be used to pass up to 9 parameters to a macro.
  - The parameters can be numbers, strings, or literals (strings without the quotes).
- Replacement fields are used to send data *to* macros, they do not return data *from* macros.
  - Use global variables or worksheet buffers to communicate data between macros.
- Replacement fields are not variables.
  - They work by pure text substitution when the macro is called.

Use of Replacement Fields

- Use **#n**, where n is 1, 2,..., 9, wherever you will pass in the appropriate parameter from the calling line
- Macro MYMTF.SEQ

```
MTF
MFR #1
IFR #2
PLO FRE #3
GO
```

- Macro execution

```
CODE V> IN MYMTF 200 10 Yes
                   ↑  ↑   ↑
                  #1 #2  #3
```

Replacement Field Defaults

- The **RFD** command (replacement field default) allows specification of defaults for passed parameters
  - The RFD command must be the first executable command in a macro
- Example MYMTF.SEQ

```
RFD 200 10 Y
MTF
MFR #1
IFR #2
PLO FRE #3
GO
```

- Macro execution

```
CODE V> IN MYMTF 100 5 N   (uses no defaults)
CODE V> IN MYMTF 150       (uses defaults for #2 and #3)
CODE V> IN MYMTF           (uses defaults for all params)
```

Reading Data

- Macro-PLUS can read data during execution
  - From the Command Window input line (interactively)
  - From a file (see Appendix 2 to this section)
- Reading is done with the **REA** command.
- When reading data interactively, the macro stops and prompts for data:

```
REA ^X
READ > 35              (^x is given the value 35)
```

- The read prompt can be set with **RPR** (read prompt)

```
RPR "INPUT A NUMBER, PLEASE:"
REA ^X
INPUT A NUMBER, PLEASE: 35
```

  - Giving the **RPR** command with no string following it resets the prompt to the default.

Reading Data (cont.)

- More than one datum can be read at a time:

  ```
  REA ^X ^Y ^Z
  ```

  - Separate multiple inputs by spaces:

    ```
    INPUT > 10 20 30
    ```

  - If not enough values are entered, the variables are given values of 0.

- When reading text strings, text strings are separated by spaces (unless formatted reads are used).

  ```
  STR ^A ^B
  REA ^A
  INPUT > CODE V             (A = "CODE")
  REA ^A
  INPUT > "CODE V"           (A = ""CODE")
  REA ^A ^B
  INPUT > CODE V             (A = "CODE", B = "V")
  ```

  - If no input is given, the variable is made a null string ("").

Writing Data

- The **WRI** command writes data
  - To the Command Window, or
  - To a file (see Appendix 2)
- Example: `WRI ^X`
- Multiple data items can be written with one **WRI** command

  ```
  WRI "The values of x and y are" ^X "and" ^Y
  ```
- Numeric values are written with a default format which
  - Resembles Fortran G format, and
  - Provides about 6 significant figures.
- Each **WRI** command starts writing on a new line.
  - A standalone **WRI** command writes a blank line.

Formatted READ and WRITE

- By default, reading and writing is done unformatted.
  - Which means it uses a default format
- You can specify your own format with a template string called a **Q** format.
  - It uses the qualifier **Q** followed by a text string defining the format.
- The template contains characters representing different types of data:
  - **D** for numeric digit in fixed format
  - **E** for numeric digit in exponential format
  - **G** for numeric digit in general format (switches between fixed and exponential format depending on magnitude)
  - A period (**.**) for a decimal point location
  - **C** for a text character
  - The format can also contain other text and spaces as desired.

Formatted READ and WRITE (cont.)

- The Q format is a text string which must be enclosed in double quotes.
  - *This is the only place in CODE V where it matters whether you use single quotes or double quotes.*
- For each datum, the individual format string is enclosed in single quotes:

```
'ddd.ddd'  '3d.4d'  'dd'  '3g.5g'  'cccccccc'  '10c'
```

- Examples

```
WRI Q"The value of x is 'ddd.ddd'" ^x
WRI Q"  'dd'  'ddd.dddd' 'cccccccc'" ^x ^y ^text
WRI Q"  '2d'  '3d.4d' '8c'" ^x ^y ^text
```

- The format string can be stored in a string variable

```
^format == "  'ddd.ddd'   'dd.dddd'   'cccccccc'"
wri q^format 1.23 4.56 "text"
```

  - This is useful if you want to re-use the format in several **WRI** commands

Formatted READ and WRITE (cont.)

- If numbers do not fill the format, leading spaces are added

```
WRI Q"'dddd.ddd'" 1
   1.000
```

- If a number is too large to fit in a fixed format, the format is automatically expanded as needed

```
WRI Q"'dddd.ddd'" 1000000
1000000.000
```

- If text does not fill a character format, the text is left justified in the format and trailing spaces are added

```
WRI Q"'cccccccc' = 'ddd.ddd'" "radius" 10
radius   =  10.000
```

- If text is larger than a character format, the text is truncated

```
WRI Q"'ccc' = 'ddd.ddd'" "radius" 10
rad =  10.000
```

Built-in Functions

- **Math functions**
  - SINF, COSF, TANF, ASINF, ACOSF, ATANF, ABSF, EXPF, LOGF, LOG10F, MAXF, MINF, MODF, RANDF, ROUNDF, SQRTF, SIGNF, ZFRFIT, ZRNFIT, GAUSSWTS, functions pertaining to FFTs
- **Array functions**
  - STDEV, SUMF, ARR_TO_BUF, BUF_TO_ARR, + many more
- **Optical functions**
  - RAYRSI, RAYSIN, RAYTRA, SAGF, SASF, TRANSFORM, ZFRCOEF, INDEX, BESTSPH, SURFSAGD, ZERNIKE, EVALZERN, NORMRADIUS, FITERROR, POLGRID, RAYPOL
- **Functions that emulate CODE V options**
  - GAUSSBEAM, TRA_1FLD, MTF_1FLD, RMSWFE, RMS_1FLD, ZERNIKEGQ , RMSSPOT, SPOTDATA
- **String Functions**
  - CONCAT, LENSTR, LOCSTR, LOWCASE, SUBSTR, NUM_TO_STR, RFSTR, STR_TO_NUM, TRUNC, UPCASE
- **Others**
  - EOFILE, CVERROR, Image simulation (IMS) functions

Using Built-in Functions

- Functions can take none, one, or several arguments.
  - Multiple arguments are separated by commas
- All functions requiring arguments have their arguments enclosed in parentheses.
  - Database arguments require their own parentheses in addition
  - **`RANDF`**, **`SQRTF(10)`**, **`MAXF(^X,^Y)`**, **`SINF((ADE S3)/57.2958)`**
- Math functions, array functions, and optical functions all return a single numeric value.
  - Some functions load array variables with values.
- String functions return a single numeric or string value.