

# Introduction to the Syntax of EViews

B. Essama-Nssah  
Poverty Reduction Group (PRMPR)  
The World Bank  
May, 2006

# Foreword

---

- "The structure of a thing is the way it is put together.
- Anything that has structure, then, must have parts, properties, or aspects which are somewhat related to each other.
- In every structure we may distinguish the *relation* or *relations*, and the items which are *related*."
  - Langer (1967)

# Outline

---

- Introduction
- Objects and their Containers
  - Workfile
  - Database
  - Structure of an Object
  - Types of Objects
  - Object Declaration and Assignment
- Working with Objects
  - Object Commands
  - Managing Objects
  - The Model Object

# Outline

---

- Elements of Programming
  - Basics
  - Program Variables and Arguments
    - Control variables
    - String variables
    - Replacement variables
    - Program arguments

# Outline

---

- Elements of Programming
  - Control of Execution
    - Mode
    - If Statements
    - Loops
  - Subroutines
    - Global
    - Local

# Introduction

---

- EViews stands for Econometric Views, a Windows application designed by Quantitative Micro Software (QMS) for general econometric analysis and model simulation.
- The language is structured around the notion of **object**, a collection of related information and operations.
- EViews offers both an interactive interface and a batch mode of command execution.

# Introduction

---

- The available simulation tools can also be used to build and solve general equilibrium models (CGEs).
- The same tools make it possible to build and run **macro-micro simulation** models for the study of the poverty and distributional impact of economic shocks and policies.

# Introduction

---

- All objects are held in **workfiles** or in **databases**. These are known as object containers.
- First step in any EViews session involves the creation of a new workfile or loading in the memory an existing one.

# Workfile

---

- The purpose of a workfile is to hold objects during processing.
- Its **structure** is therefore determined by the nature of such objects.
- Objects designed to hold the contents of datasets (hence the type of data to analyze) ultimately determine the structure of a workfile or workfile page.

# Workfile

---

- A **dataset** is a collection of observations on one or more variables.
- An **observation** is a measurement of some manifestation of a phenomenon.
- Observations are made over **time** or across **space**, therefore there is a need to uniquely identify each observation in the dataset.

# Workfile

---

- **Identifiers** provide information about the observed phenomenon, the date or place of observation, or the individual entity to which the observation pertains.
- Variable name identifies the observed phenomenon.
- For annual data about a process, one can use year identifiers such as "2001", "2002" etc...

# Workfile

---

- For **cross-sectional data** on household, for instance, a code for household ID uniquely identifies a record in a dataset.
- For **longitudinal or panel data**, one uses both a cross-section ID and a date ID to identify each observation.

# Workfile

---

- Observation identifiers are used to describe the structure of a workfile.
- Patterns for **dated regular frequency** data:
  - Annual
  - Semi-annual
  - Quarterly
  - Monthly
  - Weekly
  - Daily (5 day week)
  - Daily (7 day week)

# Workfile

---

- **Undated** data
  - Considered as unstructured.
  - Use default integer identifiers (1, 2, 3, . . . , n) that enumerate observations in the dataset.
- A **balanced panel** represents a regular frequency panel data structure.
  - [Panel data implies each observation has both a group or **cross-section** (e.g. country) and a cell or **within group** (e.g. year) identifiers.]

# Workfile

- Examples of workfile creation:
  - `wfcreate(wf=kenya, page=polak_model) a 1968 1977`
  - `wfcreate(wf=semiannual, page=data) s 05:1 06:2`
  - `wfcreate(wf=semiannual, page=data) s 2005s1 2006s2`
  - `Wfcreate(wf=macrofile,page=qdata) q 1995q1 2005q4`
  - `wfcreate(wf=macrofile, page=qdata) q 95:1 05:4`
  - `wfcreate(wf=monthfile, page=monthdata) m 49:02  
05:02`
  - `wfcreate(wf=monthfile, page=monthdata) m 1949m2  
2005m2`
  - `wfcreate(wf=india, page=rural) u 13`

(s1 stands for first semester, s2 for second)

# Workfile

---

- Creating a workfile
  - The command **WFCREATE** creates a workfile by describing its structure.
  - Syntax for regular dated data:
    - **WFCREATE(OPTIONS) FREQUENCY  
START\_DATE END\_DATE**
    - Options provide names for workfile and workfile page:  
**WFCREATE(WF=WF\_NAME, PAGE=PAGE\_NAME)**

# Workfile

---

- A panel data set has a **balanced structure** when each group (or cross-sectional unit) has the same regular frequency containing the same date observations.
- A file to contain such data can be created in two steps.
- Create a standard workfile that could hold time series for each cross-sectional unit. The range of this file should include the earliest and latest dates pertaining to any cross-section.

# Workfile

---

- Provide information on the number of cross-sections
- `WFCREATE(WF=DATAFILE, PAGE=BPANEL) Q 1970Q1 2020Q4 200`
- To create a new page:
  - `PAGECREATE(PAGE=PAGE_NAME) FREQUENCY START_DATE END_DATE [NUM_CROSS_SECTIONS]`
  - `PAGECREATE(PAGE=PAGE_NAME) U NUM_OBS` (Number of observations)

# Workfile

---

- To open workfile:
  - `WFOPEN[PATH\]WORKFILE_NAME`
- To save:
  - `WFSAVE(OPTIONS) [PATH\]WORKFILE_NAME`
  - `OPTIONS:`
    - `1` Single precision
    - `2` Double precision
    - `C` Compressed
- To make a workfile active:
  - `WFSELECT WORKFILE_NAME[\PAGE_NAME]`

# Workfile

---

- Workfile Window
  - Provides access to:
    - All available data via a directory for the objects in a given workfile page.
    - Tools for working with workfiles and their pages.
  - Various elements:
    - Title bar shows workfile designation "**WORKFILE**" followed by the name of the workfile.
    - The full disk path if file has been saved on disk.

# Workfile

---

- Workfile Window (continued)
  - Various elements:
    - Button bar for easy access to useful workfile operations.
    - Two lines of status information below the tool bar where EViews displays the range of the workfile, the current sample, and the display filter i.e. a rule used to select a subset of objects to display in the workfile window.
    - The main portion of the window shows the workfile directory for the contents of the workfile.

# Workfile

---

- Workfile Window (continued)
  - Various elements:
    - View/Details +/- button can be used to toggle between the standard workfile display format and a display of additional information about the date the objects were created or updated, and the label information.
    - The Workfile Summary View provides a description of the current workfile structure, along with the list of the types and numbers in each of the workfile page.

# Database

---

- Like a workfile, a database is an object container.
- Key differences with the workfile:
  - No need to load up the database into memory in order to access content.
  - Objects can be stored to or retrieved directly from the database on disk.
  - Series in the database need not have the same frequency or range.

# Database

---

- Structure

- Capacity limit: 1 million objects per database.
- A set of files on disk consisting of a main file with extension .EDB and a series of index files with following extensions .E0, .E1A, .E1B.

# Database

---

- To create a database:
  - `DBCCREATE DB_NAME`
  - `DB DB_NAME`
  - Example: `DB SHOCKS`
- To open existing database
  - `DB DB_NAME` or `DBOPEN DB_NAME`

# Database

---

- To put objects in database use STORE command:
  - STORE OBJ1 OBJ2 OBJ3 OBJ4 ...
- To retrieve objects, use FETCH
  - FETCH OBJ1 OBJ2 OBJ3 OBJ4 ...
- It is possible to make direct use of series contained in the database without first retrieving them
  - Syntax: DB\_NAME::OBJECT\_NAME

# Structure of an Object

---

- An object can be characterized by the associated views and procedures
  - **Views** provide a way of looking at the data contained in an objec.
  - **Procedures** are tools for altering relevant data.
- Identifying Fields
  - **Name** (up to 16 characters)
  - **Label** (containing extended annotation and commentary)
  - **History** (where EViews automatically records any modification made to the object)

# Types of Objects in EViews 5.1

 Alpha

 Model

 Sym

 Coef

 Pool

 System

 Equation

 Rowvector

 Table

 Graph

 Sample

 Text

 Group

 Scalar

 Valmap

 Logl

 Series

 Var

 Matrix

 Sspace

 Vector

# Types of Objects

---

- Object type is determined by **informational content** and applicable operations.
- Data objects designed to hold numeric information:
  - **Series, scalars, matrices, vectors**
- **Alpha** series designed to contain alphanumeric data.
- Informational content of equation or system:
  - Specification, estimation results, underlying data
- Graphs and tables contain numeric, text and formatting information.

# Object Declaration and Assignment

---

- Declaration statement
  - OBJECT\_TYPE OBJECT\_NAME
    - Examples
      - ALPHA DUMMY
      - COEF(3) BETA
      - EQUATION OLSEQ
      - MATRIX(12, 12) SAM
      - SCALAR B
      - SERIES GDP
      - VALMAP GENDERMAP
      - VECTOR(3) FGT

# Object Declaration and Assignment

---

- Declaration creates an "empty" object as it were (Series are initialized with missing data code **NA**; Other data objects are initialized with **zero**).
- Assignment statements are designed to fill the object with the relevant content.
  - Syntax: **OBJECT\_NAME = EXPRESSION**
- Examples of Assignment:
  - **B=25**
  - **DUMMY=@RECODE(INC<=5000 OR EDU<13, "LOW", "HIGH")**
  - **GPA\_PLOT=2+(4-2)\*@TREND/(@OBSRANGE-1)**
  - **IMILLS=@DNORM(XBHAT)/@CNORM(XBHAT)**

# Object Declaration and Assignment

---

- Declaration and assignment can be combined as in the following examples:
  - `COEF(3) BETA=SELEQ.@COEFS`
  - `GROUP X C GPA TUCE PSI`
  - `SCALAR B=25`
  - `SAMPLE RURAL @ALL IF URBAN=0`
  - `SERIES LGDP=LOG(GDP)`

# Object Commands

- An **object command** allows access to its views or procedures
- Basic Syntax of Object Commands:
  - `[ACTION(act_opt) ]`  
`OBJECT_NAME.VIEW_OR_PROC(v_or_p_opt) ARG_LIST`
  - `ACTION ∈ {DO, FREEZE, PRINT, SHOW}`
    - **DO** executes procedures without opening a window; if object window open then `DO=SHOW`. `DO` is default action for procedures.
    - **FREEZE** creates a table or graph from the object view window.
    - **SHOW** displays object view window (default action for views)

# Object Commands

---

- **ACT\_OPT**: An option that modifies the default behavior of the action.
- **OBJECT\_NAME**: Name of object to be acted upon.
- **VIEW\_OR\_PROC** : The object view or procedure involved.
- **V\_or\_P\_OPT**: An option to modify the default behavior of the view or procedure.
- **ARG\_LIST**: A list of arguments for the view or procedure, generally separated by spaces.

# Managing Objects

---

- Objects in the workfile can be copied, renamed, deleted and stored to disk.
- There are a number of auxiliary commands [**COPY, RENAME, DELETE, STORE**] designed to perform those tasks.
  - In general, auxiliary commands are independent of object views and procedures, or act on objects in a way that is independent of object type or content.

# The Model Object

- Example of a Simple **Input-Output Model**
  - Structure
    - Two sectors: Agriculture and Non-Ag., each producing only one good.
    - Leontief technology:
      - Constant returns to scale
      - Each sector uses inputs in fixed proportions
  - Final demand,  $b_i$ , is exogenously determined
  - Model expressed as a set of material balance equations:

$$x_{ij} = a_{ij}x_j$$

$$x_i = \sum_{j=1}^2 a_{ij}x_j + b_i; \quad i=1,2$$

# The Model Object

## ■ Data for the Input-Output Model

Table 1: Input-Output Transactions Table for a Two-Sector Economy

	Agriculture	Non-Agriculture	Final Demand	Total Output/VA
Agriculture	150	500	350	1000
Non-Agriculture	200	100	1700	2000
Value -Added	650	1400	1100	3150
Total Outlays	1000	2000	3150	6150

Source: Miller and Blair (1985)

Table 2: Technical Coefficients

	Agriculture	Non-Agriculture
Agriculture	0.1500	0.2500
Non-Agriculture	0.2000	0.0500

Source: Author's calculations

Table 3: Leontief Inverse

	Agriculture	Non-Agriculture
Agriculture	1.2541	0.3300
Non-Agriculture	0.2640	1.1221

Source: Author's calculations

# The Model Object

- Model Set-Up
  - Declaration Statement
    - `MODEL IOMOD`
  - Specification
    - `IOMOD.APPEND XS_agr = io_agr_agr*XS_agr + io_agr_nag*XS_nag + FD_agr`
  
    - `IOMOD.APPEND XS_nag = io_nag_agr*XS_agr + io_nag_nag*XS_nag + FD_nag`
  
    - Note: APPEND is used for inline equations (which are specified as text in the model), while MERGE is used for linked equations (which get their specification from an external estimation object). Example: `ISLM.MERGE EQCN`

# The Model Object

---

- Model Calibration
  - Compute values of structural parameters using data and analytical expression of model.
- Baseline Solution
  - IOMOD.SCENARIO BASELINE
  - SLOVE IOMOD
- Simulate Implications of Change in Final Demand
  - SERIES FD\_agr\_chd=600
  - SERIES FD\_nag\_chd=1500
  - IOMOD.SCENARIO(n, a=chd) DEMAND\_SHOCK
  - IOMOD.OVERRIDE FD\_agr FD\_nag
  - IOMOD.SOLVE

# The Model Object

- Simulation Results
  - Total output per sector necessary to meet new levels of demand.

Table 4: Economic Impact of a Demand Shock

	Baseline	Demand Shock
Agriculture	1000	1247.52
Non-Agriculture	2000	1841.58

Source: Author's calculations

# Elements of Programming

---

- Basics
- Program Variables and Arguments
- Control of Execution
- Subroutines

# Basics

---

- A program is a text file with extension **.PRG**, containing a list of EViews commands (each line corresponds to a single command).
- Purpose of Program:
  - Automate repetitive tasks
  - Create a record of a project
- Content of a program comprises two types of commands:
  - Those designed for the command line interface.
  - Program control statements.

# Basics

---

- Command line interface
  - Object Declarations
  - Object Commands
  - Object Assignment Statements
  - Auxiliary Commands
    - Unrelated to a particular object
    - Act on it in a way that is independent of type or content of object: e.g **STORE** or **FETCH**

# Basics

---

- Creating a Program
  - Open a program window with a statement like the following:
    - PROGRAM MINIPAMS
  - Enter text for each command and terminate line by hitting ENTER
    - Autowrap feature for lines longer than window
    - Possible to use underscore continuation character as last character of broken line.

# Basics

---

- Use **SAVE** command to save program on disk.
- Use **OPEN** command to load a previously saved program:
  - **OPEN MINIPAMS.PRG**
- Use the **RUN** command to execute the program.
  - **RUN MINIPAMS**
- By default program will stop running as soon as an error is encountered.
- Use **STOP** sign to run only part of the program (above the stop sign).

# Program Variables and Arguments

---

## ■ Control Variables

- These variables are designed to serve wherever one could use a numerical value.
- The name must start with "!" mark and may contain up to 15 alphanumerical characters
  - Examples:
    - `!N=@ROWS(VNAG)`
    - `MATRIX(!N, !N) IOTAB`
      - Above creates a square matrix called IOTAB , the size of which is equal to that of vector VNAG.
- No need to declare before assignment
- Disappear after program execution

# Program Variables and Arguments

---

- String Variables
  - A string is text enclosed in double quotes e.g. "private consumption"
  - String variable is a variable whose value is a string of text
- String Variables
- Name begins with symbol: "%"
- Assignment: %VR="GDP"
  - Once assigned, it can appear in any expression in lieu of the underlying string.

# Program Variables and Arguments

## ■ Replacement Variables

- Used to refer to an underlying object
- Obtained by enclosing a string or a control variable in curly braces (“{” and “}”)

## ■ Examples

- EQUATION OLSEQ.LS {%VR} C {%VR}(-1)  
means EQUATION OLSEQ.LS GDP C GDP(-1)
- !K=1
- SERIES Y{!K}=GDP means SERIES Y1=GDP

# Program Variables and Arguments

---

## ■ Program Arguments

- Special string variables whose values are passed on to the program during execution.
- Named as "%0", "%1", "%2", "%3" and so on.
- Allow the user to customize program execution by changing the values of these variables as needed.
- The following line comes from a program requires an argument in the form of a workfile name.
- **WFOPEN %0**

# Program Variables and Arguments

---

- Program Arguments
  - The values of the arguments must be supplied when invoking the program.
    - `RUN PAMSDLR DLRHHDATA`
  - Following program, `REGPRG`, has two arguments:
    - `EQUATION OLSEQ`
    - `SMPL 1980Q3 1994Q1`
    - `OLSEQ.LS {%0} C {%1} {%1}(-1) TIME`
    - (Call: `RUN REGPRG LGDP M1`)

# Control of Execution

---

- There are three basic aspects in controlling the way a program runs:
  - Mode of Execution
  - Selective execution of commands
  - Repeated execution of commands under changing conditions.

# Control of Execution

---

## ■ Mode Statement

- Purpose: to change the mode of execution of the program from within.
- Options include:
  - `MODE QUIET`
  - `MODE VERBOSE`
  - `MODE VER4`
- Multiple settings can be invoked in a single line:
  - `MODE QUIET VER4`

# Control of Execution

---

## ■ IF Statements

- Used to execute commands only if some condition is met.
- Start with key word "IF", followed by statement of condition and the word "THEN"
- List commands to be executed if condition is true
- End with the word "ENDIF"

# Control of Execution

---

- IF Statements

```
IF !DEFLATE THEN
    SERIES RGDP=GDP/!DEFLATE
ENDIF
```

- Use "ELSE" to include commands to execute when condition is false

```
IF !DEFLATE THEN
    SERIES RGDP=GDP/!DEFLATE
ELSE
    SERIES RGDP=GDP
ENDIF
```

# Control of Execution

---

- Nested IF Statements

- IF !EPS=0 THEN

- EDE=@SUM(YHW)

- ELSE

- IF !EPS=1 THEN

- EDE=EXP(@SUM(LGYW))

- ELSE

- EDE=(@SUM(YHWT))^(1/(1-!EPS))

- ENDIF

- ENDIF

# Control of Execution

---

## ■ Loops

- The FOR loop allows one to repeat a set of commands for different values of a control or string variable.
- Start with the word "FOR"
- End with "NEXT"
- Example:
  - FOR !K=1 TO !M
  - BETA(!K)=OSLEQ.C(!K)
  - NEXT

# Control of Execution

- In the above example, the **STEP** increase is implicitly set to 1. Here is an explicit treatment:
  - `FOR !NU=1 TO !AVMAX STEP 1/!B`
  - `SERIES QANU=QI^(!NU)`
  - `NEXT`
- Examples of FOR loop with string variable
  - `%S1="agr"`
  - `%S2="nag"`
  - `FOR %S agr nag`
  - `IOMOD.APPEND XS_{%S}=`  
`io_{%S}_{%S1}*XS_{%S1} + io_{%S}_{%S2}*XS_{%S2} +`  
`FD_{%S}`
  - `NEXT`

# Control of Execution

## ■ Nested FOR Loops

### ■ Calibration of an Input-Output Model

- `MATRIX(!N, !N) IOMAT`
- `!i=1`
- `FOR %a agr nag`
- `!j=1`
- `For %b agr nag`
- `SERIES io_{%a}_{%b}= IOTAB(!i, !j)/XS_{%b}`
- `IOMAT(!i,!j)=io_{%a}_{%b}(1)`
- `!j=!j+1`
- `NEXT`
- `!i=!i+1`
- `NEXT`

# Control of Execution

- The **WHILE** Loop
  - Analogous to an IF statement to the extent that it allows repeated execution of a set of commands while one or more conditions are satisfied.
- Provides greater flexibility than the FOR loop in the specification of required conditions
  - It starts with "**WHILE**" and ends with "**WEND**"
  - Example:
    - **!K=1**
    - **WHILE !K<=10**
    - **VECTOR VEC{!K}**
    - **!K=!K +2**
    - **WEND**

# Subroutines

---

- Provide a more general approach to reusing commands and using arguments.
- A collection of commands that can be used repeatedly with minor modifications and without duplicating the commands in question.
- Start with "SUBROUTINE" , end with "ENDSUB"
- Any number of commands can appear in between.

# Subroutines

---

- Example

- SUBROUTINE LOGSUB
  - SERIES Y=LOG(X)
  - ENDSUB

- May include arguments listed within parentheses after name of subroutine, and separated by commas.
- Each argument is specified by object type and by a name.

# Subroutines

---

- Example of Subroutine with arguments:
  - SUBROUTINE POWER(SERIES V,SERIES Y,  
SCALAR P)
  - $V=Y^P$
  - ENDSUB
- Subroutine definitions must be placed at the beginning of the program that will call them.
- Use **INCLUDE** command if subroutines are in separate files

# Subroutines

---

- Use **CALL** statement to execute the commands in the subroutine. **CALL** should be followed by name of subroutine and the list of argument values, if any, enclosed in parentheses and separated by commas.
- Examples
  - **CALL LOGSUB**
  - **CALL POWER(GDP2, GDP, 2)**  
(GDP2=SQUARE OF GDP)

# Subroutines

---

- Global versus Local Objects
  - **Global**: either object exists in workfile at the time the subroutine is called or it is created in the workfile by the subroutine.
  - **Local**: Object is meaningful only within subroutine. It disappears once the subroutine has run.

# Subroutines

---

- Global Subroutine
  - Default category
  - Has access to and can alter global objects.
  - Creates objects that are global.
  - If a global object and an argument have the same name the argument has precedence.

# Subroutines

## ■ Local Subroutine: Gaussian Kernel Matching

- SUBROUTINE LOCAL GAUSS(SERIES CS, VECTOR VP, SERIES PS, SERIES Y, VECTOR MO)
- !BW=0.06 .
- !INT=@ROWS(VP)
- SMPL @ALL IF CS
- FOR !K=1 TO !INT
- SERIES U{!K}= ABS(VP(!K) - PS)/!BW
- SERIES KIJ{!K}=@DNORM(U{!K})
- SERIES YWIJ{!K}=(KIJ{!K}/@SUM(KIJ{!K})) \* Y
- MO(!K)=@SUM(YWIJ{!K})
- NEXT
- ENDSUB

# Subroutines

---

- Local Subroutine

- May not use or update global objects directly from within the subroutine.
- However global objects corresponding to arguments may be used and updated by referring to the arguments. Such objects must be created outside the local subroutine and passed on to the subroutine as arguments.





The End.