**Introduction:** This is intended to help users unfamiliar with the *APL* language to get started using *APLSE, MARTHA* and *LLAMA*. It isn’t meant as a complete reference to *APL*, but it should provide the basics. It also covers features specific to *APLSE*, especially the editing system. There are directions on finding more complete documentation and built in help information provided with *APLSE* and in the *PLUSDEMO* programs. An *APLSE* keyboard map and command reference is included on the last page that you can print out.

**Workspaces:** *APL* stores collections of “functions” (programs) in “workspaces” (program files). The workspace files (with the extension “.AWS”, for *APL Workspace*) are organized into “libraries” (directories) that are referred to by number. For example, when you start *APLSE*, it will automatically load the workspace *INI0* from Library #4. The standard configuration has the *LLAMA* workspaces in Library #8. If you want to work with the plotting software, you “load” the workspace by typing:

```apl
)load 8 llplot
```

*APL* responds with:

```
8 LLPLOT SAVED 8/26/2007 18:03:00 (along with the opening info for the workspace).
```

This simple operation points out several important things. When *APL* is waiting for user input, it normally parks the cursor 6 spaces over from the left. When *APL* produces a result, it starts at the left edge of the screen. The “) load” command is not case sensitive, but most other *APL* operations are. Historically, the original mainframe *APL* only had uppercase letters available. As a result, older *APL* code (like *MARTHA* and *LLAMA*) uses uppercase for most function names. The
convention used in MARTHA and LLAMA is that the regular functions are in upper case, and “background” functions, that aren’t typically called by users, are in lower case. Most people find it easiest to turn Caps Lock on at the beginning of an APL session. It used to be possible to automate this, but not in Windows NT or XP.

**Workspace System Commands:** Here is a summary of some basic system commands:

<table>
<thead>
<tr>
<th>Command Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>)LOAD Lib# WSName</td>
<td>Loads workspace from disk (Overwrites existing workspace)</td>
</tr>
<tr>
<td>)SAVE</td>
<td>Saves current workspace to disk</td>
</tr>
<tr>
<td>)OFF</td>
<td>Exits APL session (be sure to save your files first!)</td>
</tr>
<tr>
<td>)CLEAR</td>
<td>Starts a new empty workspace</td>
</tr>
<tr>
<td>)WSID Lib# WSName</td>
<td>Changes current workspace name</td>
</tr>
<tr>
<td>)LIBS</td>
<td>Lists all APL libraries</td>
</tr>
<tr>
<td>)LIB Lib#</td>
<td>Lists all workspaces and files in specified library</td>
</tr>
<tr>
<td>)CMD DOS Command</td>
<td>Executes a DOS command (change directory, etc.)</td>
</tr>
</tbody>
</table>

**System Commands for APL Workspace Objects:** Here are some of the additional commands for working with APL “objects”, i.e. variables and functions:

<table>
<thead>
<tr>
<th>Command Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>)COPY Lib# WSName Object(s)</td>
<td>Copy functions/variables to current workspace (overwrites!)</td>
</tr>
<tr>
<td>)ERASE Object(s)</td>
<td>Erases functions / variables from current workspace</td>
</tr>
<tr>
<td>)FNS</td>
<td>Lists all functions in the current workspace</td>
</tr>
<tr>
<td>)VARS</td>
<td>Lists all variables in the current workspace</td>
</tr>
</tbody>
</table>

**Working in APL:** APL is an interpreted language, and is capable of operating in “immediate” mode, where it performs calculations directly on screen. You can use it as a “super calculator” to perform operations without having to write any APL programs. For example, you can type:

\[ 12 + 36 \]

and when you press <Enter>, it will respond with:

48

This isn’t too exciting until you realize that APL includes mathematical operators like matrix inversion that allow you to perform calculations vastly more complex than simple addition. If you load a workspace that includes APL functions, you can run the functions (programs) directly as well. For example, if you have a function called VOL that computes the product of three numbers,
and another function named \texttt{FtoM} that converts feet to meters, the volume of a 3’ by 4’ by 6’ box in meters is calculated:

\begin{verbatim}
VOL FtoM 3 4 6
2.152
\end{verbatim}

This illustrates a couple more points about \textit{APL}. Calculations are normally performed strictly from right to left, so \texttt{FtoM} operates on the vector of dimensions before the result is passed on to \texttt{VOL}. You can use parentheses to force a different order of execution if you need to, but with a little forethought, you can frequently eliminate many of them. Vectors can be entered as a series of numbers separated by commas or spaces. \textit{APL} is an array based language, and performing operations on vectors and matrices is built in. To expand the first trivial math example above, if you needed to add two series of numbers, you could do it as follows:

\begin{verbatim}
1 2 3+4,5,6
5 7 9
\end{verbatim}

\textbf{Numbers in \textit{APL}:} One quirk of \textit{APL} is how you must enter negative numbers. A negative number is preceded by a “high” minus sign, i.e. \texttt{ý2}. The high minus sign is typed using <Alt> 2. \textit{APL} uses the conventional minus sign “\texttt{-}” as an “operator”. If it is used as a “dyadic” operator (2 arguments), it performs subtraction, i.e. \texttt{6-2}. If it is used as a monadic (one argument) operator, it negates the argument:

\begin{verbatim}
4 5 6-1 ý2 3
3 7 3
-4 5 6-1 ý2 3
-3 -7 -3
\end{verbatim}

\textit{APL} can use scientific notation for large or small numbers, using ‘E’ for the exponent:

\begin{verbatim}
1Eý2 100000000000
0.01 1E11
\end{verbatim}

\textbf{Variables:} Similar to the issues with the minus sign operator, \textit{APL} uses the equal sign as a computational operator to test equality, not for assigning values to variables. The “left arrow” symbol “\texttt{¶}” is used instead. This is typed using <Alt> \[. For example:

\begin{verbatim}
A¶12
B¶6
A-B
6
\end{verbatim}

Variable names can be as long as you want, but can’t contain spaces. Variables can be numbers, letters, vectors, or matrices. Character vectors (strings) are entered using single quotes:

\begin{verbatim}
A¶'APL IS FUN'
\end{verbatim}
System variables: APL includes a number of “system variables” that set certain characteristics. Some can be modified to suit your needs. For example, to control the number of digits displayed, you can use the “Print Precision” system variable, “\(PP\)”. The “quad” symbol is typed using <Alt> L.

\[
\begin{array}{c}
1.23456 \\
1.23456 \\
\quad PP-3 \\
1.23456 \\
1.23
\end{array}
\]

Another sometimes useful variable is the “index origin”, “\(IO\)”. This sets whether APL starts counting from “0” or “1” when doing things like indexing into arrays or matrices. For most work, \(IO\) is usually set to “1”, but occasionally it is useful (especially for boolean & binary operations) to temporarily set it to “0”. The first thing many workspaces & functions do is set \(IO\) to the desired value just to be sure. If it’s wrong, you can get some very surprising results.

Matrices: As was mentioned earlier, APL is very much at home working with matrices. To create a matrix, you typically use the dyadic “reshape” operator “\(\rho\)” (<Alt> R). The left argument is the “shape” you want (rows & columns for a 2 dimensional matrix, but you can create as many dimensions as you can keep track of), and the left argument is the data you want in the matrix. If you specify a larger matrix than the data supplied, it starts over at the beginning, and if you specify too much data, it gets truncated:

\[
\begin{array}{c}
2 \ 3 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \\
1 \ 2 \ 3 \\
4 \ 5 \ 6 \\
\quad 2 \ 3 \ 1 \ 2 \ 3 \ 4 \\
1 \ 2 \ 3 \\
4 \ 1 \ 2 \\
\quad 2 \ 3 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \\
1 \ 2 \ 3 \\
4 \ 5 \ 6
\end{array}
\]

APL Functions: The real power of APL is in writing your own functions. Any useful discussion of how this is done is beyond the scope of this tutorial, but there is detailed info in the APLSE documentation and the PLUSDEMO package on how to do this.

Available APL Operators: The table below (from the PLUSDEMO package) lists the various operators that are available in APLSE. Many of these are fairly straightforward, but a large number require much more explanation than can be covered in a brief table like this. For example, the dyadic “circular” operator “\(\circ\)” can perform all of the standard trigonometric functions (and some not so common ones) depending on the value of the left argument. You can get a complete description by running PLUSDEMO and accessing the detailed information linked to each operator, or from the APLSE documentation.
APLSE also has a large number of “system functions”, which perform a wide range of operations, from file handling, to graphics, to communicating with other computer languages. The table below (also from PLUSDEMO) shows the wide variety of additional features and capabilities available.

These are not described in the APLSE documentation, which is why it’s useful to have access to PLUSDEMO as well. It may seem a little imposing, but most of the system functions are in groups related to specific types of operations, and you may never have to deal with them. For example, unless you get heavily involved with file operations, graphics, or the internal window system, you’ll never see over half of these. I wouldn’t recommend getting too far into any of the APLSE specific functions unless you can find a full set of APL*PLUS/PC manuals.
Holding down <Ctrl> and hitting the “/” key in APLSE opens a command menu at the top of the screen. In DOS, you can use the mouse to move between options. In Windows NT & XP, you have to use the arrow keys. The command menu will open with the Session menu selected at the far left:

This indicates that the only session is the workspace “INI0” in Library 4. If you open a function, variable or file for editing, the screen changes color, and the Session menu will show the object you are working on. You can have multiple objects open at one time, and this command menu allows switching from one to another quickly.

This shows an editing session on the function “test”. You can switch back and forth between the editing session and the main APL session here. This is useful if you are copying or moving text between the main session and your editing session using “tags”, which are described later.

The “Keyboard” menu shows the status of the keyboard. In Windows NT & XP, it shows status only. You can’t use this to change any of the settings.

The “Open” menu allows you to edit APL objects (variables or functions), as well as “native” (DOS) files.

The “Close” menu gives several options for exiting and/or saving the current editing session.
The “Search” menu allows you to find strings in the edit session (case-sensitive or not), and “tokens”, which are strings that are not part of a longer string. For example, using the Token search for “and” would only find instances of “and”, and wouldn’t locate the word “sand”.

The “Edit” menu allows you to modify the type or name of an object, and to toggle line numbers on and off.

The “Printer On” / “Printer Off” selections are equivalent to using <Ctrl> - PrtSc to send screen information to the printer. When on, it will send any new computations, function listings, etc. to the printer until it is toggled off. Note: In Windows NT & XP, the printer output is buffered, and won’t immediately send the data to the printer unless you send a form-feed to the printer. The INI0 workspace assigns a printer form-feed command to the function key F1, for this purpose.

The Status Line is the one-line display at the bottom of the screen indicating the active workspace, cursor location, keyboard settings (insert, caps lock, etc.) and printer status. The image below shows that the workspace is INI0 in library 4, the cursor is 47 lines down from the top of the session (not the visible screen) & 17 characters to the right, and the keyboard insert mode is active:

The “Info” menu also you to access on-line help. The “Keystrokes” menu item brings up a summary of editing and menu commands. Ordinarily, “Help Session” would load Help information from a file, but the required file isn’t supplied with APLSE. We are working on getting permission to include this in the future.
Editing Operations With The Mouse & Keyboard

Editing Operations With The Mouse: There are a number of editing features in APLSE that can use the mouse. The basic process consists of “tagging” a block of code or text, which can then be copied, moved or deleted. Warning!: the normal Window shortcut keys for cutting & pasting do NOT work. In fact, <Ctrl> C does the equivalent of a paste instead of a copy!

To tag a collection of entire lines, you merely hold down the right mouse button while dragging down over the desired material. When you release the mouse button, the text will remain hi-lighted, and there will be a “T” indicator in the status line that shows the line numbers:

You can now move elsewhere in your session, or open up an editing window with a new or existing variable. When you left click the mouse at the desired location, a small menu pops up to ask what you want to do with the tagged text:

You can either use one of the indicated shortcut keys (the “^” indicates holding down the <Ctrl> key), or click on the appropriate option with the mouse. To clear the tagged selection, just right click anywhere on the screen.

To tag part of a line or an arbitrary section of the screen, click BOTH mouse buttons simultaneously and hold them down as you drag over the text you want. After that, the process for working with the tagged information is the same as before.

Tagging Blocks With the Keyboard: You can also use the keyboard to tag blocks. You use <Ctrl> T to start and end the tagging process. After pressing <Ctrl> T at the beginning of the block you want, move the cursor to where you want the block to end using the up & down keypad arrows and then press <Ctrl> T again.

To tag blocks containing partial lines, use <Ctrl-Shift> T to initiate and end the tagging process. Use the arrow keys, PgUp, PgDn, Home or End to move the cursor around the screen to select the block desired. You can end the tagging process by just hitting <Enter>, although <Ctrl> T or <Ctrl-Shift> T will also work.
What To Do If Things Go Wrong

**Don't Panic!** If you are working in the editor and accidentally mess things up, <Ctrl> Q will exit without saving anything. If you accidentally write a function that seems to be stuck or is in an infinite loop, <Ctrl> Esc should get you out. It also works if a function is asking for input & you have no idea what it wants.

One thing to watch out for is the “State Indicator”. This keeps track of where you were when a function crashes. It can be a handy debugging tool, because it saves everything in the state it was in when it encountered a problem. However, this means there is also a lot of memory being used to hold this information, and there are variables and settings present that you don’t expect. The status line at the bottom of the screen will show the word “Susp” to indicate that something is suspended:

![Status Indicator](image)

Here is an example using the LLAMA mixer spur program. The program was run with a pair of single quotes as the right argument for the function `MIXSPUR`. This creates an “empty” input vector to the program, and it prompts the user for the required data. In this case, the user typed in a question mark, which the program didn’t expect.

```
MIXSPUR ''
LO, RF(low), RF(high), IF(low), IF(high), Max Order: ?
SYNTAX ERROR
MIXSPUR[17] 🍀 ?
```

The system indicates that `MIXSPUR` has a problem on line [17]. The status line also indicates that there is a suspended operation present. To see what is going on, you can check the State Indicator by typing “)` SI”`. In this case, only one function was running, and it verifies that MIXSPUR is hung up at line [17]

```
)` SI
MIXSPUR[17] ★
```

Before you try to go back to work, you should clear the State Indicator. This is done by typing a right arrow “→” (<Alt> ] ). To check the State Indicator, type “)` SI” again:

```
→
)` SI
```

The State Indicator is now empty, and you can continue normally. Sometimes you can lose track, and there will be several suspended operations piled up. You need to keep entering right arrows until the State Indicator is empty and the “Susp” indicator in the status line goes away.
Additional Documentation in **APLSE** and **PLUSDEMO**

**INITIAL** Workspace in **APLSE**: **APLSE** provides some documentation in the workspace **INITIAL**, which should be in Library 3. If you execute:

```
)load 3 initial
```

You will get the following screen:

You can view the documentation chapters on-line by just typing the name of the chapter, or if you have a printer set up, you can print the chapters using the instructions provided. All 6 chapters have been collected in a single PDF document that is included in the **APLSE** zip file.

The information on standard **APL** functions is quite good, but as was mentioned earlier, there is no documentation on “system functions” that allow you to work with files, graphics, etc. These typically start with the “quad” symbol “□” (<Alt> L on the keyboard). Most are system specific, but some are standard to many **APL** systems. For example, there are “quad” versions of many of the “)” system commands so that you can perform the required operation with in a function. Fortunately, the documentation **APLSE** is lacking in this area is covered to some extent by the **PLUSDEMO** package.

**PLUSDEMO** Software Package: Before **APLSE** was made available, STSC/Manugistics had a software package called **PLUSDEMO** to demonstrate the capabilities of their **APL*PLUS/**PC software. Unlike **APLSE**, **PLUSDEMO** is not a fully functional **APL** package. However, it contains some very nice documentation on **APL** and its usage. For some reason they disabled the mouse, but you can navigate easily using the arrow keys. You can type <Esc> at any time to return to the main screen, or to exit the program from there. Here is the opening screen:
In addition to the instructional material, it has a very nice keyboard layout that can be accessed at any time by pressing F3. What is really useful is that hi-lighting any particular APL symbol and pressing <Enter> gives you a quick tutorial on what that APL operator does and its usage. If you are working in a newer Windows OS, you can run PLUSDEMO along side an APLSE session to have ready access to this feature. You can run one or both in a windowed mode by using <Alt-Enter> to switch back & forth from full-screen.
PLUSDEMO also includes a number of example programs that demonstrate a wide range of features & capabilities. The screen shot below shows the sample program page accessible from the “APL Programs” tab in the PLUSDEMO startup screen.

In addition to the various documentation features, PLUSDEMO allows you to experiment with APL expressions and even small programs with a built in interpreter. This is accessed from the main PLUSDEMO screen from the “Try APL*PLUS” tab:
Most Special APL Characters Are Typed Using The < Alt > Keys
"Shifted" APL Characters Require Using < Shift > & < Alt > Together

---- Menu Operations ----
Ctrl-/: Invoke menus
Esc: Exit menus
←→/↑/↓/Tab/Backtab: Move highlight
Enter or Ins: Select option
Del: Deselect option

---- Session Management ----
Ctrl-I: Initialize new editing session
Ctrl-Shift-I: Initialize new session with name at cursor
Ctrl-E: save edit object & End session
Ctrl-Q: Quit this session without saving
Ctrl-S: Save edit object & keep session
Ctrl-X: eXit to APL session - keep edit session
Ctrl-Shift-P: move to Previous session
Ctrl-Shift-N: move to Next session
Ctrl-A: Alter session type (character only)
Ctrl-Shift-R: Rename session

---- Page Navigation ----
PgUp/PgDn: Scroll screen up / down 1 line
Ctrl-PgUp/ PgDn: Scroll up / down 1 page
Ctrl-Home/End: Cursor to top / bottom of screen