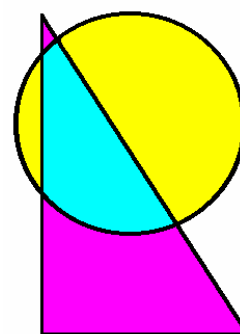


# A

## 2<sub>o</sub> - Day

## Introduction to

# APL 3 & 4



Graeme Robertson Ltd.

Featuring **DYALOG** Versions 10 & 11

### Day 1: Third Generation APL – Objects

Modules 1 to 10

Graphical User Interface  
Object Linking and Embedding

### Day 2: Fourth Generation APL – The Internet

Modules 11 to 20

Transmission Control Protocol / Internet Protocol  
Microsoft .NET Framework

## Learning Modules

Module0: Notation and Conventions ..... 9

Module1: Objects and their Properties ..... 13

Module2: Methods and Events ..... 17

Module3: Dot Syntax ..... 23

Module4: The Session Object..... 30

Module5: Control Structures ..... 36

Module6: In-Process OLE Servers ..... 40

Module7: OLE Clients..... 46

Module8: ActiveX Controls ..... 53

Module9: C Function Access..... 59

Module10: Stand-Alone Applications..... 70

Module11: Advanced Dot Syntax ..... 77

Module12: Dynamic Programs ..... 103

Module13: APL Threads..... 117

Module14: TCP/IP Sockets..... 129

Module15: APL Web Servers ..... 136

Module16: APL Web Clients ..... 146

Module17: Dyalog.Net ..... 148

Module18: Dyalog.Net Classes..... 165

Module19: Dyalog.Asp.Net ..... 169

Module20: Dyalog APL Classes ..... 183

re	what	at	when	is	the thing	TOPIC
for	APL 1	in the	70's	it be	Vars, Fns & Primitive Ops	CORE APL
for	APL 2	in the	80's	it be	Nested Arrays and Applications	2nd Generation
for	APL 3	in the	90's	it be	Object Orientation (GUI Apps)	GUI & OLE
for	APL 4	in the	00's	it be	Internet Orientation (.NET Apps)	TCP/IP & .NET
for	APL 5	in the	10's	it be	Thin Client Applications?	?????

```
makeGrid
File Edit View Options
makeGrid:vals
  'F'DHC'Form' 'The Evolution of APL in a Nut Case'
  'F.G'DHC'grid'(5 7p'')(size' 100 100)(titlewidth' 0)(resizcols' 1)
  'F.G'DHC'coltitles' 're' 'what' 'at' 'when' 'is' 'the thing' 'TOPIC'
  'F'DHC'iconobj' 'BICO'
  vals+(5p'for',[1.5](c'APL '),''12345'
  vals,+(5p'in the',((78901''),'c'0's'),[1.5]5p'it be'
  vals+5 7↑vals
  vals[1:6]←'Vars, Fns & Primitive Ops'
  vals[2:6]←'Nested Arrays and Applications'
  vals[3:6]←'Object Orientation (GUI Apps)'
  vals[4:6]←'Internet Orientation (.NET Apps)'
  vals[5:6]←'Thin Client Applications?'
  vals[7]←'CORE APL' '2nd Generation' 'GUI & OLE' 'TCP/IP & .NET' '?????'
  'F.G'DHC'values'vals
  F.G.CellWidths+5.44 9.63 8.89 8.89 7.41 38.77 20.5
  F.Size-12.31 31.65
  F.G.ColTitleFCol←(7p=255 0 255)
  F.G.CellTypes+5 7p1
  F.G.CellTypes[2]←1+5
  F.G.BCol←1(255 0 0)(255 127 0)(0 255 0)(0 0 255)(255 0 255)
  F.G.FCol←0 0 0(255 255 0)(3p255)(255 255 0)
  'Fnt'DHC'Font' 'comic sans ms' 16
  F.G.CellFonts←'Fnt'
  'Fnt2'DHC'Font' 'comic sans ms' 16('weight' 700)
  F.G.FontObj←'Fnt2'
```



**ROBERTSON (Publishing)**  
15 Little Basing, Old Basing,  
Basingstoke, RG24 8AX, UK.

Copyright © 2007 Graeme Robertson Ltd.

*This publication may be used, reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the permission of the publisher.*

*This document is distributed subject to the condition that it shall not, by way of trade or otherwise, be sold or hired out without the publisher's prior consent. It may however be used in APL classes and circulated in any form of binding or cover with a similar condition, including this condition, being imposed on the subsequent owner.*

Published January 2007 as APL3\_4\*.PDF.

### **Dyalog APL is worth learning because it:**

- strictly complies with APL ISO **Standard** 8485
- has a brilliant **tracer** for the interpreted APL environment
- runs on **Windows** (9x, NT, 2000, CE, XP), AIX, Solaris, Linux and more...
- affords **multilingual** development and runtime environments
- has 71 built-in **GUI classes** (based on Windows API), each with 10 to 80 properties...
- is **.NET enabled**, ready for Windows Vista (the 2007 successor of XP)
- has excellent printed **manuals** and detailed **help** files (created using APL)
- has many useful supporting sample **workspaces** and files
- provides DSS, the *Dyalog Support Service* via [www.dyalog.com](http://www.dyalog.com) and [support@dyalog.com](mailto:support@dyalog.com)

A **Level 1** course in **APL1\_2.PDF** introduces mainstream 1<sup>st</sup> and 2<sup>nd</sup> generation APLs such as IBM APL2, Dyalog APL, STSC APL\*PLUS and MicroAPL APL.68000. **APL1\_2.PDF** is freely available from MicroAPL at [http://www.microapl.co.uk/apl/APL1\\_2.PDF](http://www.microapl.co.uk/apl/APL1_2.PDF) and may be given by instruction.

This **Level 2** course in **APL3\_4\*.PDF** introduces Dyalog APL versions 7-11 under Microsoft XP Pro. This course, when delivered by Graeme Robertson Ltd., comes bundled with supporting materials:

- **varChar**, a Dyalog APL addin for studying Dyalog APL arrays, (a free stand-alone version is available from [www.dyalog.com](http://www.dyalog.com) download zone)
- a handy, short, fold-up Dyalog APL **Reference Card**
- a sample Pocket Dyalog application workspace called **AddrBk.DWS**

**Conduct of this course:** The suggested conduct of the course is similar to that stated in **APL1\_2.PDF** and reproduced below. The course notes are provided in Portable Document Format, **one APL3\_4Module\*.PDF file at a time**, to enable cut & paste, to encourage experimentation and remote learning, and to increase the likelihood of obtaining feedback ☺.

After short introductions, the group is invited to divide up into pairs. Each pair works on one computer for the duration of the course. Each pair is given the first module and asked to work through it on their computer at their own pace. Pairs are encouraged to help each other with new concepts and difficulties as they arise and to experiment on the computer with any ideas that they think they can express in APL statements. Tuition is given when the pair cannot resolve problems. Questions may be answered directly on matters of fact, or indirectly by way of a suggestion as to how the problem might be tackled. Each day could cover about 10 modules, depending upon the pace of each pair. There is no pressure to complete all modules (remaining modules are given out at the end of the course). At the discretion of the tutor, modules may be skipped or assigned for private study after the course. Introductions and synopses are given with an overhead projector at suitable intervals throughout the course to the group as a whole.



# APL 3 & 4

## Course Contents

<b>Module0: Notation and Conventions .....</b>	<b>9</b>
§ 0.1 New Symbols.....	9
§ 0.2 Naming Conventions .....	10
§ 0.3 Variable <i>dataTypes</i> .....	11
<b>Module1: Objects and their Properties .....</b>	<b>13</b>
§ 1.1 Object Spaces.....	13
§§ 1.1.1 Creating vanilla Namespaces with $\square NS$ .....	13
§§ 1.1.2 Creating GUI Object Spaces with $\square WC$ .....	13
§§ 1.1.3 Changing Space with $\square CS$ .....	14
§ 1.2 Properties of Object Spaces .....	14
§§ 1.2.1 Examining Properties of an Object with $\square WG$ .....	14
§§ 1.2.2 Setting Properties of an Object with $\square WS$ .....	14
§§ 1.2.3 Building complex Objects .....	15
§ 1.3 Property Variables .....	15
§§ 1.3.1 Exposing Object Properties with $\square WX$ .....	16
§§ 1.3.2 Assigning Properties with $\leftarrow$ .....	16
§§ 1.3.3 Rebuilding complex Objects.....	16
<b>Module2: Methods and Events .....</b>	<b>17</b>
§ 2.1 Object Methods.....	17
§§ 2.1.1 Enqueuing Object Methods with $\square NQ$ .....	17
§§ 2.1.2 Invoking Event default Action using $\square NQ$ .....	17
§§ 2.1.3 Method Functions .....	17
§ 2.2 Object Events and callback Functions .....	18
§§ 2.2.1 Firing Events by User Actions.....	18
§§ 2.2.2 Attaching callback Functions to Events .....	18
§§ 2.2.3 Bringing Objects to Life .....	19
§ 2.3 The Event Queue .....	19
§§ 2.3.1 Dequeuing Events with and without $\square DQ$ .....	19
§§ 2.3.2 Tracing $\square DQ$ .....	20
§§ 2.3.3 Defining complex Behaviour.....	21
<b>Module3: Dot Syntax.....</b>	<b>23</b>
§ 3.1 Object References .....	23
§§ 3.1.1 Making References with $\mathfrak{A}$ and $\leftarrow$ .....	23
§§ 3.1.2 Parent.Child Hierarchy .....	24
§§ 3.1.3 Object.Object. .. Object.Object Rationale .....	25
§ 3.2 Direct Property Access .....	26
§§ 3.2.1 Object.Variable Syntax.....	26
§§ 3.2.2 Object.Object. .. Object.Property Rationale .....	27
§§ 3.2.3 Using Object.Object. .. Object.Property Constructions.....	27
§ 3.3 Direct Method Invocation.....	27
§§ 3.3.1 Object.Function Syntax .....	27
§§ 3.3.2 Object.Object. .. Object.Function Rationale.....	28
§§ 3.3.3 Defined Operators in Object Space .....	28
Semantics (Meaning).....	29



<b>Module4: The Session Object .....</b>	<b>30</b>
§ 4.1 Using the Session Object .....	30
§§ 4.1.1 Immediate Execution Mode of <code>□SE</code> .....	30
§§ 4.1.2 Tracer and Editor of <code>□SE</code> .....	30
§§ 4.1.3 Choosing syntax Colours .....	31
§ 4.2 Inside the Session Object .....	33
§§ 4.2.1 Exploring the Workspace and <code>□SE</code> .....	33
§§ 4.2.2 Examining Session Menus and Buttons .....	33
§§ 4.2.3 Miscellaneous Properties and Methods .....	34
§ 4.3 Building the Session Object .....	34
§§ 4.3.1 Tracing <code>▽BUILD_SESSION▽</code> .....	34
§§ 4.3.2 Foreign Language Support .....	35
§§ 4.3.3 Adding useful Extensions .....	35
<b>Module5: Control Structures.....</b>	<b>36</b>
§ 5.1 Logical Decisions and Jumps .....	36
§§ 5.1.1 The <code>:If</code> Statement .....	36
§§ 5.1.2 Further truth Conditionals.....	36
§§ 5.1.3 The <code>:Select</code> Statement .....	37
§ 5.2 Looping Constructs.....	37
§§ 5.2.1 The <code>:For</code> Statement.....	37
§§ 5.2.2 Generalised <code>:For</code> Statements .....	38
§§ 5.2.3 <code>:Repeat</code> and <code>:While</code> Loops .....	38
§ 5.3 Digging .....	38
§§ 5.3.1 The <code>:With</code> Statement .....	38
§§ 5.3.2 Digging into SubSpaces.....	39
§§ 5.3.3 <code>:Trap</code> versus <code>□TRAP</code> .....	39
<b>Module6: In-Process OLE Servers.....</b>	<b>40</b>
§ 6.1 Creating an OLE Server in a DLL .....	40
§§ 6.1.1 The <code>OLEServer</code> Object .....	40
§§ 6.1.2 Exporting Variables and Functions as Properties and Methods .....	40
§§ 6.1.3 Saving and registering your <code>OLEServer</code> .....	41
§ 6.2 Variable type Information.....	41
§§ 6.2.1 Setting Method Information .....	41
§§ 6.2.2 Setting Property Information .....	41
§§ 6.2.3 Exploring the Registry Entry .....	42
§ 6.3 Using your OLE Server .....	42
§§ 6.3.1 The <code>OLEClient</code> Object .....	42
§§ 6.3.2 Examining Type Libraries .....	43
§§ 6.3.3 Calling an OLE Server from VB .....	44
<b>Module7: OLE Clients .....</b>	<b>46</b>
§ 7.1 Inside Microsoft Word.....	46
§§ 7.1.1 Registry Entries, Object Models and Type Libraries .....	46
§§ 7.1.2 Digging into Word .....	47
§§ 7.1.3 Demonstrating the Power of OLE .....	48
§ 7.2 Manipulating Microsoft Excel from the Inside .....	49
§§ 7.2.1 Recognising the Object Model .....	49
§§ 7.2.2 Digging into Excel.....	49
§§ 7.2.3 Gaining full Control of Excel .....	50
§ 7.3 Linking to other Servers .....	50
§§ 7.3.1 Outlook .....	50
§§ 7.3.2 Microsoft Internet Explorer .....	51
§§ 7.3.3 Beyond.....	52



<b>Module8: ActiveX Controls .....</b>	<b>53</b>
§ 8.1 Creating an ActiveX Control in an OCX.....	53
§§ 8.1.1 The <i>ActiveXControl</i> Object.....	53
§§ 8.1.2 The <i>Create</i> Callback.....	53
§§ 8.1.3 Creating the OCX on <i>)SAVE</i> .....	54
§ 8.2 Using your ActiveX Control.....	55
§§ 8.2.1 Creating an Instance from an <i>OCXClass</i> .....	55
§§ 8.2.2 Using Controls in IE 6.0 .....	55
§§ 8.2.3 Using Controls in Visual Basic and VBA .....	56
§ 8.3 Browsing registered OLE Controls .....	56
§§ 8.3.1 Having a quick Look .....	56
§§ 8.3.2 Having a deeper Look.....	56
§§ 8.3.3 Trying some Examples .....	57
<b>Module9: C Function Access .....</b>	<b>59</b>
§ 9.1 Declaring <i>dataTypes</i> of Arguments and Results .....	59
§§ 9.1.1 Quick View of DLLs and their Contents.....	59
§§ 9.1.2 The Meaning of the right Argument of <i>□NA</i> .....	59
§§ 9.1.3 Discovering C Function Syntax.....	60
§ 9.2 Examples of C Function Calls .....	60
§§ 9.2.1 Simple Examples .....	60
§§ 9.2.2 More complex Examples .....	62
§§ 9.2.3 Other API Calls.....	64
§ 9.3 Harnessing large C Libraries .....	65
§§ 9.3.1 Fastest Fourier Transform in the World .....	65
§§ 9.3.2 Open Graphics Library .....	66
§§ 9.3.3 Linear Algebra Package.....	66
<b>Module10: Stand-Alone Applications.....</b>	<b>70</b>
§ 10.1 Building GUI Applications.....	70
§§ 10.1.1 The bare Minimum .....	70
§§ 10.1.2 Completing the Address Book Application.....	71
§§ 10.1.3 Enhancing your Address Book .....	71
§ 10.2: Making runtime Executables .....	71
§§ 10.2.1 Files to Include .....	71
§§ 10.2.2 Inifiles and the Windows Registry .....	72
§§ 10.2.3 The [File][Export] MenuItem .....	74
§ 10.3 Aspects of Pocket APL.....	74
§§ 10.3.1 Pocket Platforms.....	74
§§ 10.3.2 Creating the executable Program.....	76
§§ 10.3.3 Building a distributable Application.....	76
<b>Module11: Advanced Dot Syntax.....</b>	<b>77</b>
§ 11.1 Object Variables .....	77
§§ 11.1.1 Stranding Object Properties .....	77
§§ 11.1.2 Stranding Objects.....	80
§§ 11.1.3 Arrays of .. Arrays of Objects.....	83
§ 11.2 Understanding <i>(...).( ...)</i> .....	87
§§ 11.2.1 Expanding Array.Strand .....	87
§§ 11.2.2 Expanding Array.Array .....	94
§§ 11.2.3 Expanding Array.Function .....	96
§ 11.3 Arrays of Programs.....	99
§§ 11.3.1 Interpreting <i>... ( ... ) ... ( ... ) . f<sub>1</sub></i> .....	99
§§ 11.3.2 Arrays of .. Arrays of defined Functions .....	100
§§ 11.3.3 Arrays of .. Arrays of defined Operators .....	101



<b>Module12: Dynamic Programs.....</b>	<b>103</b>
§ 12.1 Direct Definition.....	103
§§ 12.1.1 Programming DFns.....	104
§§ 12.1.2 MultiLine DFns .....	106
§§ 12.1.3 Guards and Error Guards .....	107
§ 12.2 Extended direct Definition.....	109
§§ 12.2.1 Programming DOps .....	109
§§ 12.2.2 Idioms and Utilities.....	110
§§ 12.2.3 Object.Object..Object.Operator Rationale.....	111
§ 12.3 Recursion .....	111
§§ 12.3.1 Recursive Functions.....	111
§§ 12.3.2 Recursive Operators.....	113
§§ 12.3.3 Biological Beauties .....	114
<b>Module13: APL Threads.....</b>	<b>117</b>
§ 13.1 Spawning a new Thread.....	117
§§ 13.1.1 The Spawn Operator, &.....	117
§§ 13.1.2 Thread Identity from $\square TID$ and $\square TNAME$ .....	119
§§ 13.1.3 Thread Numbers with $\square TNUMS$ and $\square TCNUMS$ .....	120
§ 13.2 MultiThread Interactions .....	122
§§ 13.2.1 Thread Synchronisation with $\square TSYNC$ .....	122
§§ 13.2.2 Holding Tokens with $:Hold$ .....	122
§§ 13.2.3 Pooling Tokens with $\square TPUT$ and $\square TGET$ .....	123
§ 13.3 General Thread Programming .....	124
§§ 13.3.1 Thread Switching.....	124
§§ 13.3.2 External Threads with $\square NA$ .....	125
§§ 13.3.3 Threading callback Functions.....	126
<b>Module14: TCP/IP Sockets.....</b>	<b>129</b>
§ 14.1 The <i>TCPSocket</i> Object.....	129
§§ 14.1.1 IP Addresses and Ports .....	129
§§ 14.1.2 <i>SocketType</i> and <i>Style</i> Properties.....	129
§§ 14.1.3 Workspace to Workspace Communications .....	130
§ 14.2 A simple character Socket .....	131
§§ 14.2.1 Connecting to a server Socket with <i>TCPConnect</i> .....	131
§§ 14.2.2 Sending to the server Socket using <i>TCPSend</i> .....	132
§§ 14.2.3 Receiving from the server Socket with <i>TCPRecv</i> .....	132
§ 14.3 Some Complications.....	133
§§ 14.3.1 HTTP and HTML .....	133
§§ 14.3.2 Buffering received Data.....	133
§§ 14.3.3 Servicing multiple Connections.....	134
<b>Module15: APL Web Servers.....</b>	<b>136</b>
§ 15.1 Making a simple Server .....	136
§§ 15.1.1 Creating a listening Socket .....	136
§§ 15.1.2 Cloning a listening Socket on <i>TCPAccept</i> .....	137
§§ 15.1.3 Sending an HTML File on <i>TCPRecv</i> .....	138
§ 15.2 Making a realistic Server .....	139
§§ 15.2.1 Threading multiple Connections.....	140
§§ 15.2.2 Communicating through HTTP.....	141
§§ 15.2.3 Running APL Functions on a Server .....	142
§ 15.3 Internet Practicalities .....	143
§§ 15.3.1 Domain Name Servers .....	143
§§ 15.3.2 Firewalls and proxy Servers .....	144
§§ 15.3.3 An ISP running Dyalog.DLL.....	145





<b>Module16: APL Web Clients</b>	<b>146</b>
§ 16.1 Getting to the outside World	146
§§ 16.1.1 Direct Connection through your Internet Service Provider	146
§§ 16.1.2 Proxy Servers and Firewalls	146
§ 16.2 Asking the Web	146
§§ 16.2.1 Connecting and sending the Question	146
§§ 16.2.2 Receiving and interpreting the Answer	146
<b>Module17: Dyalog.Net</b>	<b>148</b>
§ 17.1 Revealing the .NET Framework	148
§§ 17.1.1 Getting Microsoft .NET	148
§§ 17.1.2 Assemblies (à), Namespaces (ñ) and Classes (¢)	148
§§ 17.1.3 Using <code>USING</code>	152
§ 17.2 Exploring the .NET Interface	153
§§ 17.2.1 Examining Classes	153
§§ 17.2.2 Examining Methods	154
§§ 17.2.3 Examining Properties	156
§ 17.3 Digging into .NET	158
§§ 17.3.1 Windows Forms	158
§§ 17.3.2 Communications	161
§§ 17.3.3 Generalising APL Primitives	163
<b>Module18: Dyalog.Net Classes</b>	<b>165</b>
§ 18.1 Writing Dyalog.Net Classes	165
§§ 18.1.1 Dyalog Namespaces and .NET Namespaces	165
§§ 18.1.2 Creating a <i>NetType</i> Object	165
§§ 18.1.3 Writing Functions and defining Variables	165
§ 18.2 Exporting Methods and Properties	165
§§ 18.2.1 Arguments and Result <i>dataTypes</i>	165
§§ 18.2.2 Making an Assembly	166
§§ 18.2.3 Checking the MetaData	167
§ 18.3 Calling Dyalog.Net Classes	167
§§ 18.3.1 Calling your Dyalog.Net Class from Dyalog APL	167
§§ 18.3.2 Calling your Dyalog.Net Class from C# and VB.NET	168
§§ 18.3.3 Complications	168
<b>Module19: Dyalog.Asp.Net</b>	<b>169</b>
§ 19.1 Dynamic Web Pages	169
§§ 19.1.1 Active Server Pages in VBScript or JScript	169
§§ 19.1.2 The <i>System.Web.UI.Page</i> Class	171
§§ 19.1.3 The <i>System.Web.UI.WebControls</i> Namespace	173
§ 19.2 Dyalog Script Language	174
§§ 19.2.1 Callbacks in Dyalog APL	174
§§ 19.2.2 Workspace behind ...	175
§§ 19.2.3 The TextBox Control	176
§ 19.3 Remote Applications	178
§§ 19.3.1 The C:\Inetpub\wwwroot\ Directory	179
§§ 19.3.2 The <i>System.Drawing</i> Namespace	180
§§ 19.3.3 The <i>System.Web.Services</i> Namespace	181
<b>Module20: Dyalog APL Classes</b>	<b>183</b>
§ 20.1 User defined Classes	183
§§ 20.1.1 The <code>:Class</code> Structure	183
§§ 20.1.2 The <code>:Field</code> Statement	185
§§ 20.1.3 Name sub Classifications of <code>NC</code>	186
§ 20.2 Methods and Properties in Classes	187



§§ 20.2.1 The $\nabla$ (Method) Structure .....	187
§§ 20.2.2 The <b>:Implements</b> Statement.....	188
§§ 20.2.3 The <b>:Property</b> Structure.....	190
§ 20.3 Architecture with Class Factories .....	192
§§ 20.3.1 Designing an Object Model.....	192
§§ 20.3.2 Building with Objects .....	194
§§ 20.3.3 Encapsulating, Inheriting and Morphing .....	194

Some innocent facts ☺

- There is no exact finite Boolean representation of the decimal number 0.1 as a binary floating-point number.
- The density of rational numbers on the real line is infinitesimal compared to the density of transcendentals.
- One Jupiter day equals 0.416.. Earth days.

Question ☺ Is there a planet somewhere whose day is approximately equal to 10 Earth days?

Answer ☺ Very likely! (On that planet, this is a two day teach-yourself course ☺.)





## Module0: Notation and Conventions

### § 0.1 New Symbols

It shouldn't worry an APLer to have a few new suggestive symbols kicking around. Of course they ought to be well defined, although their meaning should be 'onomatopoeically' self-evident from their shape. Most of our new symbols are not to be found in the □*AV* of any APL font and are simply introduced to clarify explanation and writing in the spirit of the original *Iverson Notation*. Some of these symbols are to be found in some APL fonts, but they are not here coloured *green* and are, by implication, not available as part of an executable Dyalog APL statement. Executable APL keywords are written in *green*. An executable input expression is written in *green* APL font and indented by 6 spaces. Any resulting output from the expression is coloured *red* and is placed unindentedly on the subsequent line or lines. Or, it is placed on the same line as the input expression, separated from it by the new symbol  $\hookrightarrow$ .

*Expr*  $\hookrightarrow$  *Result*

Error unless expression *Expr* returns result *Result*

The symbol  $\hookrightarrow$  (*returns*) is black to identify it as non-executable, and the definition is in black to emphasise that is not genuine (current) APL. The *Result* may also be an executable expression which itself would, if executed, return a result identical to that returned by the original expression, *Expr*. So, a valid example would be  $3 \times 4 \hookrightarrow +/3 \rho 4$

*Expr*  $\mapsto$  *Obj*

*Expr* instantiates object *Obj* (as a side effect)

This is used to comment functions that internally create global GUI objects.

*Expr*  $\hookrightarrow$  *Function*

*Expr* can fire callback function *Function*

This is used to document functions that internally assign callbacks to global objects.

*APLCode*  $\leftarrow$

Code continues on next line ...

$\_APLCode$   $\mathbb{I}$

.. end of line of code

The pair ( $\leftarrow, \mathbb{I}$ ) is used to wrap 2 or more input lines while maintaining their syntactic unity and integrity.

$\vdash$  *Prop*

Error unless proposition *Prop* is contingently true

Symbol  $\vdash$  (*contingently*) means factually true, so here *Prop*  $\hookrightarrow$  1.  $\vdash a \vee b$  implies, *a posteriori*, that  $a \vee b \hookrightarrow$  1

$\models$  *Prop*

Proposition *Prop* is necessarily true, so *Prop*  $\hookrightarrow$  1

Symbol  $\models$  (*logically*) establishes the logical truth of a proposition, so  $\models a \vee b$  implies, *a priori*, that  $a \vee b \hookrightarrow$  1, and further, that ( $\models a$ ) or ( $\models b$ ), where *or* is inclusive rather than exclusive.

*Code* .. *Code*

Means fill in the gap with the obvious code

This single character double-dot (*gap*) can also be used at the very start of a line to mean fill in the obvious start.

Note in the modules to follow, boxed syntax definitions usually just apply to one of a number of possible syntaxes. For example, ambivalence and shy results are not usually left undetermined.

*Code* ...

Means finish the statement with the obvious ending

This single triple-dot character (*ending*) asks you, the reader, to complete the extreme end of any expression or suitably parenthesised expression. It asks more of you than *dittos* which just imply repetition, and more than gaps (..) which are usually essentially repetition too, given the clues from the code on either side. An author who uses this ending should feel that the code to the left is sufficient for the reader to be able to fill in the ending without undue pain. Suitable fill code should spring to mind.



You may consider .. or ... as notation for  $x$  or  $y$  in some 'literal algebra'. eg "Hello .., this is .., how are you ... ?" where .. and ... are any suitable strings.

$$() \quad \vdash () \equiv \theta$$

This implies that  $() \equiv \theta \hookrightarrow 1$  and that  $\vdash (, 0) \equiv \rho()$

Other symbols that may be introduced in the following modules include:

$f_1$  as a generic monadic function (distinguish *monadic* function from *monistic* operator),

$g_2$  as a generic dyadic function (distinguish *dyadic* function from *dualistic* operator),

$\hat{o}_a$  to represent a typical operator and

$\hat{n}_a$  to represent a typical namespace.

## § 0.2 Naming Conventions

The following naming conventions have been followed where possible and where appropriate:

Variable names are regarded as proper nouns and therefore start with a capital letter, an exception being local *temporary variables* used near to their source. These may be regarded as pronouns for which we may use small letters.

Functions and operators play the roles of verbs, adverbs, adjectives, conjunctions or prepositions depending on the details of their syntax. Their names all start with a small letter as they are mid-sentence words. However, *niladic functions* may start with a capital, because if they return a result they are syntactically like variables, and if they do not return a result they are like complete imperative intransitive verb sentences. Likewise, *monadic functions that do not return a result* may also begin with a capital letter because they are like imperative transitive verbs and therefore usually begin a sentence.

To give you the idea, an attempt has been made to compose the Course Contents roughly in this style. Module titles are noun phrases that represent the subject matter. Section and subsection headings are subjects or predicates or object noun phrases. An attempt is made to capitalize the words according to their contextual parsing character (see APL Linguistics in *Vector Vol.2 No.2 p118*).

Sentence Syntactic Elements	Grammatical Rôle	Example Names
<i>Variable</i>	Noun/Pronoun	<i>Data</i> $\diamond$ <i>DATA</i> $\diamond$ <i>d</i>
<i>Niladic</i>	Intransitive imperative	<i>Run</i> $\diamond$ <i>RunApplication</i>
$R \leftarrow \text{Niladic}$	Noun	<i>.. is Temperature</i>
<i>Monadic</i> $\omega$	Transitive imperative	<i>RunOn</i> ...
$R \leftarrow \text{monadic } \omega$	Adjective/Participle	<i>.. is dataType</i> ...
$\alpha \text{ dyadicFn } \omega$	Verb	<i>.. takes</i> ...
$R \leftarrow \alpha \text{ dyadic } \omega$	Preposition/Conjunction	<i>.. is .. tiedTo</i> ...
$R \leftarrow (\alpha \alpha_1 \text{ monistMonadOpr}) \omega$	Adverb	<i>.. completely</i> ...
$R \leftarrow \alpha (\alpha \alpha_1 \text{ monistDyadOpr}) \omega$	Adverb	<i>.. overNight</i> ...
$R \leftarrow (\alpha \alpha_1 \text{ dualMonadOpr } \omega \omega_1) \omega$	Conjunction/Preposition	<i>.. togetherWith</i> ...
$R \leftarrow \alpha (\alpha \alpha_1 \text{ dualDyadOpr } \omega \omega_1) \omega$	Conjunction/Preposition	<i>.. is .. and</i> ...
etc...		

This is a heuristic pragmatic association with ordinary language for the purpose of meaningful readability. Note the convention of capitalizing the first letter of each word or syllable after the first.



Object names follow variable naming conventions. Object method names should follow function conventions but are predetermined by their author. From a glance at IE6 method names (or OpenGL method names) Microsoft may be moving in the above direction. Object property names are nouns and are generally ‘correctly’ (by this convention) predetermined.

An effort has been made to (capitalize and colour) keywords in the text when used as keywords rather than as general concepts. *eg* an object initiates an event, *cf* an object has an *Event* property. Don't be too reluctant to choose long meaningful names. AutoComplete springs into action in version 10 to ease the pain.

Pre-empting a Unicode (see <http://www.unicode.org>)  $\square AV$ , letters in names assume all the characteristics of modern fonts, *ie* you can have  $\mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A}$  etc... all for the price of  $\square AV[66]$  and all equivalent, just as an ordinary word is assumed to have the same meaning in any colour. The three alphabets in  $\square AV$  are taken to be  $\mathbf{A} \dots \mathbf{Z} \mathbf{a} \dots \mathbf{z} \mathbf{^a} \dots \mathbf{^z}$  *ie* capital letters, small letters (possibly subscripted) and superscripted letters of various styles. This is done to encourage full tensor notation such as  $(X_1, X_2, X_3)$  or

$$T_{ijkl} \leftarrow G_{im} + . \times T^m_{jkl}$$

(as used in *eg* <http://arxiv.org/ftp/hep-th/papers/0304/0304244.pdf>) and also to encourage traditional British shorthand forms such as  $W^m$  Robertson &  $C^o L^{td}$  or  $Sci^{ntfc}$  &  $Med^{cl}$  Net<sup>wk</sup>. We also suppose, in these notes, that *any* linear Rich Text may follow a comment symbol in the APL Session and we assume that any number of new symbols will one day be available to APL programmers for names.

Menus and other one-click options relating to the application under discussion are placed in square brackets. Thus [File][Load] and [Tools][Options] refer in an obvious way to Session menus.

### § 0.3 Variable *dataTypes*

Different environments use different notations to specify the number of bytes being used and the interpretation of the bits. For example, the C programming language, which we meet in  $\square NA$  in Module 9, describes numbers as integer, double, long double, float, .. whereas  $\square NA$  itself uses notation similar to that used in the Larg (left argument) of  $\square FMT$  which itself originates from FORTRAN. More modern definitions of numeric “dataTypes” are written VT\_I2, VT\_I4, VT\_DECIMAL, VT\_R4, VT\_R8, ...

APL, thank goodness, does not generally demand that the programmer be aware in advance of exactly what type of binary representation is being used for any particular piece of data. Mathematica, J and some other modern computer languages are even more forgiving - numbers may even be infinite!

In pure mathematics, occasionally the domain of numbers in which one is thinking has to be generalised to the next set. Whole numbers are a subset of natural numbers which are a subset of the group of integers which is a subgroup of the group of rational numbers which is a subgroup of the field of real numbers which is a subfield of the complex field which is a subfield of the non-commutative quaternion ring which is a subring of the non-commutative and non-associative octonion ring.

$$\mathbf{N} \subset \mathbf{Z} \subset \mathbf{Q} \subset \mathbf{R} \subset \mathbf{C} \subset \mathbf{H} \subset \mathbf{O}$$

There is a natural hierarchy that is embraced at each level, as and when required, when learning mathematics. Thus when learning, at an early age, that division of integers can land you in the rationals, you are not generally obliged thenceforth to predetermine whether the result of any particular division will result in a number which is integer or one which is non-integer (but necessarily rational).

In APL the actual representation employed for storing a number can change from line to line and from function call to function call without the programmer having to know. The interpreter itself attempts to



discover the most efficient storage representation. However, when communicating through APL with the 'real' computer world beyond APL 1 & 2 you have to start thinking about the machine a bit more. You sometimes have to tell functions in advance what sort of arguments to expect and what sort of results to generate. This is the function *dataType* signature. It helps hardware to know how to store stuff.

### Loosely Correlated *dataType* Definitions (internal memory formats)

APL if you had to say...	NA	C	Interface Definition Language (OLE/COM)	Visual Basic	C#	.NET Data Types
CharSc	C1, T,	uchar		Byte	byte, char	System.Byte, System.Char
CharVec	C, T[]	char[] char*	VT_BSTR, VT_PTR TO VT_BSTR	String	string	System.String
VecCharVec	{T[4]}[5]	(char* )[]	VT_ARRAY OF VT_BSTR, VT_STRING[]	String(0 to 12)	string[]	
NumSc, BoolSc, IntSc	I2, I4, F8, U1, U4	short, uint, int, long, float, dword, lptstr double	VT_BOOL, VT_DECIMAL, VT_I1, VT_I2, VT_I4, VT_R4, VT_F8, VT_R8, VT_PTR TO VT_UI4	Integer, Long, Single, Double, Decimal Boolean, Date	sbyte, short, int, uint, ulong, float, double, bool, decimal	System.SByte, System.Int16, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64, System.Decimal
NumVec, IntVec	{F8 I2}, I[], F8[]	Int[]	VT_CY, VT_DATE, VT_COLOR, VT_I4[]	Integer(1 to 3)	int[], double[]	
Arr	A	void[], int[]	VT_VARIANT, VT_ARRAY	Variant()	object	System.Object, System.Array
ArrEncArr	{I4 T[9]}, {I4 U4 {U1[4]} I4 I4}	struct	VT_ARRAY OF VT_VARIANT, VT_SAFEARRAY	Variant( 1 to 3, 1 to 4, 1 to 2)	object	System.Object
MatEncArr			VT_VARIANT[;]		int[][,]	System.Object
RefSc			VT_DISPATCH, VT_PTR TO VT_COCLASS	Object		System.Object, etc...
Empty, NULL		void	VT_VOID,	Empty, Nothing, Null	void	System.Void
OR of space			VT_DISPATCH			

0.3.1 Review **APL1\_2.PDF** which introduces mainstream 1<sup>st</sup> and 2<sup>nd</sup> generation APLs such as *IBM APL2*, *Dyalog APL*, *STSC APL\*PLUS* and *MicroAPL APL.68000*.

Note: You can download **APL1\_2.PDF** free from [http://www.microapl.co.uk/apl/APL1\\_2.pdf](http://www.microapl.co.uk/apl/APL1_2.pdf).