# 2014 Dyalog User Meeting

# Data Workshop Exercises 21 September 2014

Exercises

|  |  |
| --- | --- |
| Text Files |  |
| CSV Files |  |
| XML Files |  |
| Excel Files |  |
| Component Files |  |
| Databases |  |
| JSON Data |  |
| XML Data |  |
| Web Services |  |
| Data Visualisation/Syncfusion |  |
| Solutions to Selected Exercises |  |

Text files

Text files come in many forms. In the *Unicode* version of *Dyalog* they can contain characters that fit on a single byte (8 bits) but also on more than one. For example, Russian characters and APL characters all use two bytes. All characters can be represented in *Unicode[[1]](#footnote-1)*. The ASCII character set is the first 128 *Unicode* characters. Some accented letters and characters like ¢£¤¥¦§ are in the range 128-255 range. APL, Russian et al are beyond 255.

An easy way to determine how many bytes a character will use is by using the system function ⎕DR. For example:

⎕DR 'a'

80

⎕DR '⌽'

160

Here 80 means “8 bits characters” and 160 means “16 bits characters”. There are also 32 bits characters. 𠃿 is an example.

Each character of a string will use as many bytes as the character that uses the most bytes. For example:

⎕DR 'abcd' ⍝ four 8 bits characters

80

⎕DR 'abcd⌽𠃿' ⍝ four 8 bits, one 16 bits and one 32 bits

320

When reading or writing to a native file it is important to take that into consideration.

When dealing with a file you should remember what type of character you wrote it with for when you have to read it back.

APL will write a native with type 80 by default.

tie←'\tmp\test1.txt' ⎕NCREATE 0

'abcd' ⎕NAPPEND tie

⎕NSIZE tie

4

⎕NREAD tie 80 4 0

abcd

Here APL appended the text using 80 as type by default. If we try with some text whose characters do not fit on 1 byte we get an error:

'abc⌽' ⎕NAPPEND tie

DOMAIN ERROR: Left argument could not be converted 'abc⌽'⎕NAPPEND tie

∧

'abc⌽' ⎕NAPPEND tie **160**

⎕NSIZE tie

12

⎕NREAD tie **80** 12 0

abcda b c =#

Here the file contains 12 characters: 4 from the initial append and 8 (4 x 2 bytes) from the second append. Each of the characters from the second append has been added using 2 bytes. While **a**, **b** and **c** were padded with a null (0), the ⌽ used effectively 2 bytes. Because we are reading each byte individually using type 80 they appear as **=** and **#**.

If we try to read the file using 160 instead we get our characters back:

⎕NREAD tie 160 6 0

扡 摣 abc⌽

We tried reading 6 characters from the beginning using type 160 (2 bytes characters) and APL did so. The last 4 characters are 'abc⌽' but our initial ‘abcd’ was transformed into扡 摣because the internal 16 bits representation of **ab** is扡 and that of **cd** is 摣.

This can be better understood if we use ⎕DR and ⎕UCS[[2]](#footnote-2) to change the type of ‘abcd’:

163 ⎕DR 'abcd'

25185 25699

⎕UCS 25185 25699

扡 摣

160 ⎕DR 'abcd'

扡 摣

The lesson to retain here is: you should always specify the type when writing (append or replace) and it should in general be 160 if you intend to use any character outside the Windows extended ASCII character set (the first 255 Unicode characters).

More generally, if you are to write out text that only occasionally go beyond the 255 *Unicode* range you should *encode* the text using UTF-8.

UTF-8 (UCS[[3]](#footnote-3) Transformation Format on 8 bits) is a variable-width encoding that can represent every character in the Unicode Character Set.

⎕UCS accepts an optional left argument specifying the encoding to perform. UTF-8 is one of them.

Reusing our examples above:

'UTF-8' ⎕UCS 'abcd'

97 98 99 100

⎕UCS 'abcd' ⍝ same thing because no encoding occurs

97 98 99 100

'UTF-8' ⎕UCS 'abcd⌽'

97 98 99 100 226 140 189

We can see that the 4 characters **abcd** were left as is but that the character ⌽ was encoded on 3 bytes. We can bring **abcd** directly:

⎕UCS 'UTF-8' ⎕UCS 'abcd'

abcd

but we cannot do that for the second example:

⎕UCS 'UTF-8' ⎕UCS 'abcd⌽' ⍝ no decoding the encoded string

abcdâ½

for that we need to use UTF-8 again:

'UTF-8' ⎕UCS 'UTF-8' ⎕UCS 'abcd⌽'

abcd⌽

## Reading and writing encoded text files

If you are to read and write text files to be used outside APL you should mark the file so that other programs know what is in the file.

There is a convention to use when using text files:

* If the file starts with the 3 characters 239 187 191 then the file is deemed to be a UTF-8 encoded file
* If the file starts with the characters 255 254 then the file is written as a series of 16 bits for each character (our 160 type). This encoding is known as UCS2 for Unicode Character Set on 2 bytes.

So to create a new UTF-8 file you should always write the characters ⎕UCS 239 187 191 before anything else.

Likewise, when reading a text file you should check the first characters to detect the encoding method.

# Cover functions

Trying to remember what to do when encoding text can be a tedious task.

Dyalog has written cover functions to deal with everyday situations.

In workspace *LoadDATA* reside a namespace, *fileUtilities*, which contains functions to read and write files called *ReadFile* and *WriteFile*. They handle most common situations.

SALT[[4]](#footnote-4) also contains the same functions and other utilities to deal with files in file “[SALT]/tools/code/fileUtils.dyalog”

# Exercices

1. On your USB stick there should be a file named UCS2file.txt  
   Try to read it in APL using ⎕NREAD. Can you open that file using Notepad[[5]](#footnote-5)?
2. The file “aplcode.dyw” contains some APL characters that Notepad should read without problems. Read the file in and make sure you can see the same thing in APL. You can use the utilities.
3. Use APL to store a program’s Canonical Representation (⎕CR of a program) in a text file.

CSV Files

Comma Separated Values files are text files containing values separated by a delimiter, usually a comma, a semi-colon or a tab. They are common throughout the industry because they are portable.

Each file is a table. Each line of the file represents one line (record) of the table. Lines are separated by another delimiter, usually a CR and/or NL (ASCII or Unicode 13 and 10). Lines don’t have to have the same number of fields (values).

An example of 2 rows, 3 fields each:

Field 1 in row1,field 2 in row 1,field 3 in row 1

Field1inrow2, fiels-2 in row 2,f3/r2

If a field contains the delimiter then the field is surrounded by another delimiter, usually a double quote, like this:

“Field 1, in row1”,field 2 in row 1,field 3 in row 1

Field1inrow2,” fiels-2, in row 2”,”f3/r2”

Note that surrounding a field with double quotes is always permitted as for “f3/r2”.

If the field contains the double quote then this one is doubled and the field surrounded by a double quotes if not already present.

“Field 1, in row1”,field 2 in row 1,field 3 in row 1

“row2””s field1”,” fiels-2, in row 2”,”f3/r2”

## Reading and writing CSV files

Reading and writing is the same as any text file, including Unicode text files.

In order to separate lines you cut on CR and/or NL. In order to separate fields you need to know the field separator (usually a comma) and split values on it, taking into account the double quotes, if present, to mask out those separators inside them.

# Exercises

On the USB stick you will find 2 CSV files: “Fiscal\_and\_Monetary\_Data\_for\_2008-2012.csv” and “Fiscal\_and\_Monetary\_Data\_for\_2008-2012.txt” almost identical in values.

1. Try to read them both. The .csv file uses the comma as field delimiter and the .txt file uses Tab (ASCII 9) instead. You can use utilities (e.g. in workspace *LoadDATA*).
2. Make up a matrix of values and write them out to a CSV file of your choice. Use ‘;’ as field delimiter. Look at the result in Notepad.

XML Files

Extensible Markup Language (XML) is a [markup language](http://en.wikipedia.org/wiki/Markup_language" \o "Markup language) that defines a set of rules for encoding documents in a [format](http://en.wikipedia.org/wiki/File_format) that is readable for both [human](http://en.wikipedia.org/wiki/Human-readable_medium)s and [machines](http://en.wikipedia.org/wiki/Machine-readable_data).

It is a textual data format with strong support via [Unicode](http://en.wikipedia.org/wiki/Unicode) for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures.

For example, the APL variable *Mat* containing this:

⍴⎕←Mat

COLUMN\_NAME DATA\_TYPE TYPE\_NAME COLUMN\_SIZE

ID 4 COUNTER 10

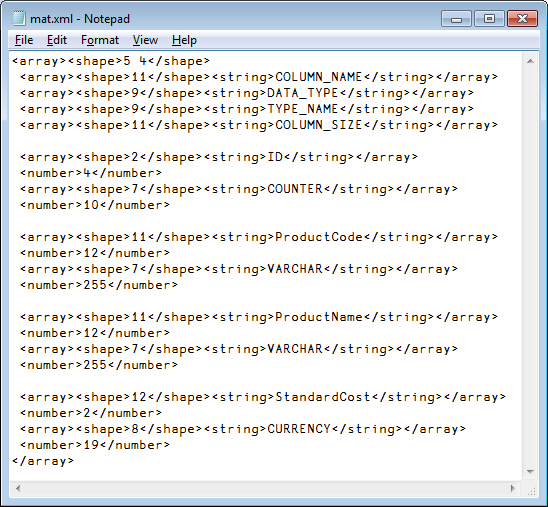
ProductCode 12 VARCHAR 255

ProductName 12 VARCHAR 255

StandardCost 2 CURRENCY 19

5 4

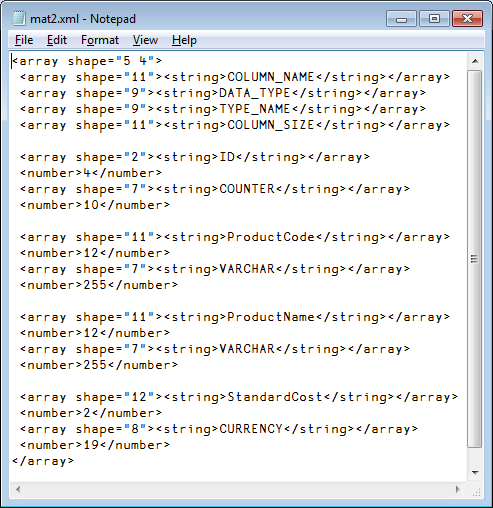
Could be represented by the following XML code:



Which you can find on the USB stick.

In this example we define a structure called ‘array’ which consists of a series of ‘array’ made of a ‘shape’ followed by a ‘string’, or a ‘number’.

We could have done this differently and used an attribute to represent the shape as in



Since XML can represent any arbitrary structure we can store and retrieve almost anything we want.

Very often we store records as in

<?xml version="1.0"?>

<payroll>

<employee id="001">

<firstname>Sue</firstname>

<salary>13000</salary>

</employee>

<employee id="002>

... etc ...

This is a payroll table made of employees each having an id, a first name and a salary. Here **id=”…”** really represents a field in the table and if we were to read this data in it would probably look like this:

id firstname salary

001 Sue 13000

002 Pete 12500

The program LoadXML in workspace LoadDATA will read in such XML files. The program SaveXML will do the reverse.

# Exercises

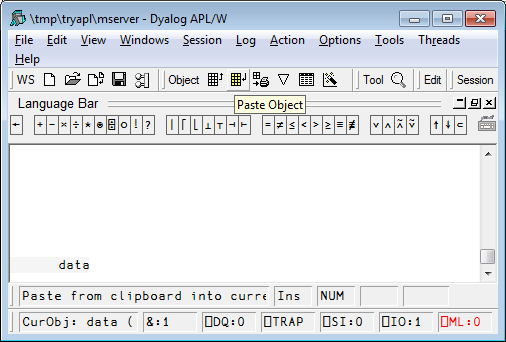
1. Create an APL matrix containing the same type of data in each column except for the first row containing column headers. Store this variable as XML in a file of your choice. Use Notepad to view the file. If your file extension is .xml double click on it to view it in a browser.
2. View file book.xml on the USB stick and bring its content in APL. You can use ⎕NREAD and ⎕XML or use the program *LoadXML* in the *LoadDATA* workspace.

Excel

Data in Microsoft’s Excel is easy to get at and it is as easy to store data into Excel using the Dyalog environment.

## Manual copy

You can manually copy the data in Excel and bring it in APL by pasting it as text into the editor or the session. You can also bring each copied cell individually as a nested array into a variable using the Paste Object menu item in the menu bar. To do so position the cursor over or just past the name of a variable and click on the Paste Object icon in the menu:



The variable will then contain the pasted cells.

## Controlled copy

To read/write under program control you can use the “LoadXL” and “SaveXL” programs in the *LoadDATA* workspace.

These 2 functions need the file name, the sheet name if more than one, and the range to read/write. The comments in the programs give examples.

# Exercises

1. Read all the data in file **FMD2008-2012(subset).xlsx** on the USB stick
2. In file **clients.xlsx** on the same stick fetch all the names in sheet *countries*
3. Create an Excel file named **phoneBook** containing records of name, surname and tel # in sheet ‘numbers’. If you need to generate records at random look at the script names.dyalog. There you will find a program, <genNames>, that you can use or modify to produce random names.
4. On the USB stick you will find a folder in which 12 Excel files representing timesheets for each of the 12 months in 2013 are located (it is named **xcel**).  
   In columns **B, C** and **F**, starting at row **2**, is found Company, Task and time taken for that task in fraction of a day (0.5=12 hours). NULLs are ignored.  
   You should read each sheet’s column B, C and F and find the time taken for each task **in hours** for the **ACME** Company for the entire month of January. This should be a 2 column matrix with the task in column 1 and the total time taken for that task in column 2.
5. Do the same as #3 but for the entire year. Again a 2 column matrix is returned. Bonus points for turning the result into a 13 column matrix where the first column is the task and the other columns are for the total of each month. Extra bonus points for using the Key operator.

Component Files

Component files are files containing APL arrays.

They are unique to APL. Most APL versions have had a version of component files available for years.

Dyalog’s component files are a little bit special. They can recover from most errors thrown at them, even system crashes.

There are many Quad-functions to perform creation, deletion, adding, removing, reading, replacing, etc. There are functions to synchronize (hold) operations and check the file too.

The files may be viewed as a vector of APL data, each element or component holding a single array.

A quick example:

tie←'\tmp\cfex1' ⎕FCREATE 0

⎕FSIZE tie ⍝ the file is initially empty

1 1 304 1.844674407E19

⍝ File starts at 1, next component will be 1, 304 bytes

⍝ Append 2 new components

⎕←'This is a string' ⎕FAPPEND tie ⍝ 1st component

1

⎕←(100×⍳10) ⎕FAPPEND tie ⍝ 2nd component

2

⎕FSIZE tie ⍝ File starts at 1, next component will be 3

1 3 1236 1.844674407E19

⎕←(10 20 30?¨1000) ⎕FAPPEND¨tie

3 4 5

⎕FSIZE tie ⍝ File starts at 1, next component will be 6

1 6 1704 1.844674407E19

⎕FREAD tie 3 ⍝ read 3rd component

132 756 459 533 219 48 679 680 935 384

(3 4⍴⎕A) ⎕freplace tie 3 ⍝ replace it by a matrix

⎕FSIZE tie ⍝ File starts at 1, next component will be 6

1 6 1908 1.844674407E19

⍝ the size of file has increased slightly (+204 bytes)

⎕FUNTIE tie

⎕FLIB '\tmp' ⍝ list of files in ‘\tmp’

C:\tmp\cfex1

You can create all sorts of applications involving component files.

The files, like any other native file, are stored locally or on a network and can be visible to anyone with access to the computer/network. Applications requiring security should use other methods. Dyalog offers DFS, the Dyalog File System, which offers security and other features like backup/restores. This will not be seen here.

# Exercises

1. Create a component file containing 3 components:
   1. A text vector
   2. A matrix of numbers
   3. An array of mixed elements
2. Create a file containing 100 components and
   1. Read the first 10 components, all at once, atomically
   2. Replace components 11..20 with the letters ‘A-Z’ in a single statement
   3. How can you ensure replacement will be atomic?
3. Create a series of program to add records to a component file such that the records are kept sorted. Bonus points: if the component becomes too big it should split. For simplicity sake consider the records to be a simple string of 50 characters and split the records when they go beyond 20.  
   So the task is:
   1. Create a file with one empty (0 rows) matrix
   2. Write a program to
      1. Read the component
      2. Add the new record
      3. Sort the matrix
      4. Replace the component
   3. Write a program to delete a record in the same fashion
   4. Bonus: consider a multi-user application of this program: how do you coordinate updates?

Databases

Databases come in all kinds of formats.

Dyalog provides a workspace to deal with SQL databases at large.

The workspace contains a single namespace with programs to perform various functions in it.

## ODBC

Since 1992, Open Database Connectivity (ODBC) has been a standard interface for accessing database management systems. In general, ODBC drivers will use Structured Query Language (SQL) to express queries and make updates to data. As a result, the use of an ODBC interface requires some understanding of SQL, unless you can make do with the functionality provided by the *LoadSQL* function described later.

ODBC drivers now exist for a very wide variety of databases, from simple drivers which give limited access to "flat" DOS files, through more sophisticated local database managers such as Microsoft Access, to multi-user DBMS systems such as Microsoft SQL Server, MySQL, Oracle, or DB2 running on a variety of server operating systems. ODBC drivers are even available for data sources which are not databases at all.

ODBC has been the most widely used standard for database access under Microsoft Windows since the mid-1990’s – and the ODBC interface (known as SQAPL) is bundled with Dyalog APL at no extra cost under Windows.

## SQAPL

SQAPL provides an interface between APL and database drivers which conform to the Microsoft ODBC specification. SQAPL consists of an APL workspace containing a set of interface functions which make calls to a DLL which is written in C and provides a high-performance gateway to ODBC.

To be precise, SQAPL communicates with a component known as the Driver Manager, which is responsible for loading and managing database drivers. Under Microsoft Windows, Microsoft provides a standard Driver Manager, and a version of SQAPL which is compatible with this is bundled with Dyalog APL.

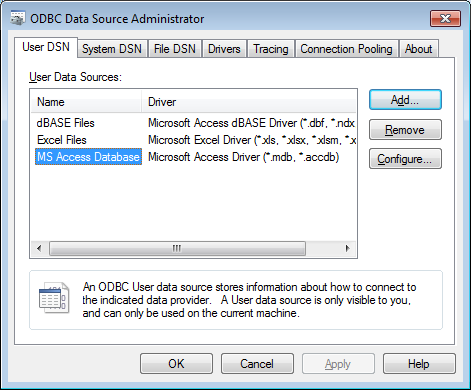
Using SQAPL, you can:

* Retrieve a list of available data sources, and connect to one or more of them.
* Query the database catalogue, to determine which tables, view and columns exist in a data source.
* Prepare and then repeatedly execute SQL statements, or execute SQL statements immediately without first preparing them. Multiple statements may be active simultaneously.
* Retrieve data from a result set. Retrieve a description of the contents of a result set.
* Execute SQL statements multiple times using a matrix containing a row of data for each execution (known as Bulk Input).
* Commit or roll back transactions.
* If connected with sufficient privileges, execute any SQL statement supported by the database management system, including the creation of tables or views (Data Definition Language – or DDL), indexes, stored procedures, or GRANT statements (etc).
* Retrieve a list of data types supported by a data source.

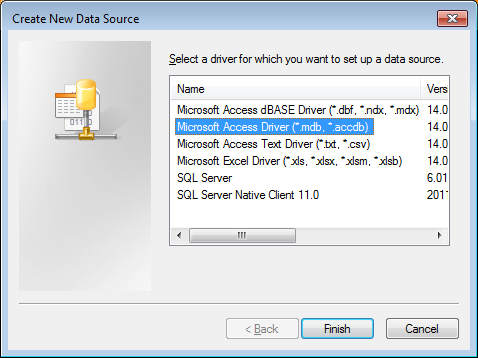
## Setting up an ODBC source

(Under Windows)

Look under administrative tools: ODBC drivers



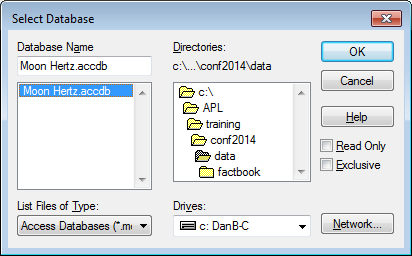
Select “Microsoft Access Driver” for an Access database:



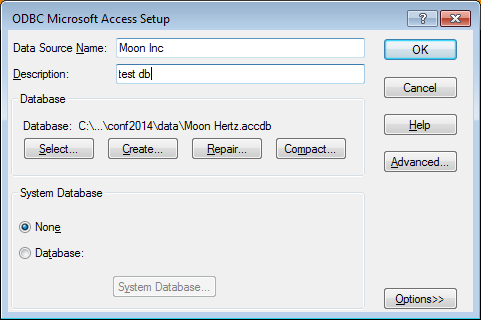
Select the file…



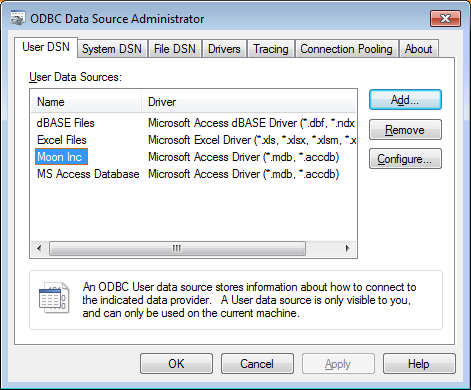
Browse to the database file:



Give it a source name and description:



### Click OK to confirm



### From now on the database chosen (here “Moon Hertz.accdb”) will be available under the name chosen (here “Moon Inc”) also known as DSN (Data Set Name).

### The LOADDATA Workspace

If all you need to do is load some data from an ODBC data source – or populate a table with data, the functions *LoadSQL* and *SaveSQL* from the utility workspace *LoadDATA* may be all that you need. The following example loads data from Microsoft’s Access Database called *Moon Inc*. The arguments to *LoadSQL* are [1] a DSN optionally followed by a password and user id if necessary, [2] a table name, [3] optionally a list of column names to be retrieved:

)LOAD LOADDATA

Saved…

To get a list of all the tables available do

LoadSQL 'Medical' '' 'TABLE\_NAME'

TABLE\_NAME

Admissions

Diagnostics

Patients

⍴r←LoadSQL 'Medical' 'Patients'

3 7

r

12 Alicia Parks 1983-01-02 8 Coro rd L.A. CA

13 Mary Chavez 1985-02-03 818 R str Santa Ana CA

14 Carl Thomas 1995-03-04 12 Cab Av. Tempe AZ

# Exercises

1. On the USB stick provided you will find a database named <ACME Coyote.accdb>.  
   Setup an ODBC connection to it and find all the tables in it.
2. Read all the records from the *Products* table.
3. Find all the column names in table *Products*. Hint: you will need to connect to a database before finding the columns of the table.
4. Read only columns *ProductName* and *ListPrice* in the *Products* table.
5. Create a new table *Partners* in the same database with columns *Name* and *Share* (an integer) and add 3 new partners at the same time.

XML Data Exercises

The XML namespace contains functions to:

* convert XML into a namespace-oriented form for easier manipulation from within APL
* convert a namespace to XML
* attempt to convert HTML into XHTML so that it can be processed more easily under program control.

First load the DataWorkshop workspace on the USB stick and run the xml demo to prepare the environment.

)load {your location}/dataworkshop/dataworkshop  
... Saved ...

]demo [ws]/exercises/xml.txt

### Exercise 1

Read the file cds.xml into the workspace.

Determine the distribution of countries in the list of CDs.

### Exercise 2

Use conga's Samples.HTTPGet to grab the content of the main Dyalog.com web page.

How many hyperlinks does it contain? What are the links?

JSON Data Exercise

The JSON class contains functions to:

* convert an APL array or namespace to a JSON representation. When using the option to serialize, the representation is lossless
* convert JSON into an APL namespace or array
* convert an XML representation to a JSON representation
* convert a JSON representation to an XML representation
* format MiServer's APLJAX commands
* format parameters suitable for use with jQuery-like tools

)load {your location}/dataworkshop/dataworkshop  
... Saved ...

]demo [ws]/exercises/json.txt

### Exercise 1

Create an arbitrarily complex APL array.

Use JSON.fromAPL to serialize it.

User JSON.toAPL to deserialize it.

Web Service/Google Exercises

)load {your location}/dataworkshop/dataworkshop  
... Saved ...

]demo [ws]/exercises/json.txt

### Exercises

1. Use Samples.HTTPGet to access the Google Directions API to get directions from Eastbourne to Basingstoke, using the following URI:  
     
   <http://maps.googleapis.com/maps/api/directions/json?origin=Basingstoke,UK&destination=Eastbourne,UK>  
     
   Get the directions in both JSON and XML formats and examine them to determine which is easier to manipulate.
2. Use Samples.HTTPGet to access the Google Directions API to get directions from Eastbourne to Basingstoke, using the following URI:  
     
   <http://maps.googleapis.com/maps/api/staticmap?center=Eastbourne,UK&size=600x300&zoom=15>
3. Get the list of all US Federal Agencies from: <http://www.federalregister.gov/api/v1/agencies>

Data Visualisation/Syncfusion

Under a licensing agreement with *Syncfusion*, Dyalog includes the *Syncfusion* library of WPF controls. These may be used by Dyalog APL users to develop applications, and may be distributed with Dyalog APL run-time applications.

The *Syncfusion* libraries comprise a set of .NET assemblies which are supplied in the Syncfusion/4.5 sub-directory of the main Dyalog APL installation directory (for example: c:\Program Files\Dyalog\Dyalog APL-64 14.0 Unicode\Syncfusion\4.5.

# Requirements

To use the Syncfusion libraries you must be using Microsoft .NET Version 4.5.

In addition, to use the controls contained in these assemblies it is necessary to perform one or both of the following steps.

# Using XAML

If using XAML, the XAML must include the appropriate xmlns statements that specify where the Syncfusion controls are to be found. For example:

xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Gauge;

assembly=Syncfusion.Gauge.WPF"

The above statement defines the prefix syncfusion to mean the specified Syncfusion namespace and assembly that contains the various Gauge controls. When the prefix syncfusion is subsequently used in front of a control in the XAML, the system knows where to find it. For example:

<syncfusion:CircularGauge Name="fahrenheit" Margin="10">

# ⎕USING

In common with all .Net types, when a Syncfusion control is loaded using XAML or using ⎕NEW it is essential that the current value of ⎕USING identifies the .Net namespace and assembly in which the control will be found. For example:

⎕USING←,⊂'Syncfusion.Windows.Gauge,Syncfusion/4.5/Syncfusion.Gauge.WPF.dll'

This statement tells APL to search the .Net namespace named Syncfusion.Windows.Gauge, which is located in the assembly file whose path (relative to the Dyalog installation directory) is Syncfusion/4.5/Syncfusion.Gauge.WPF.dll.

There are dozens of Syncfusion controls that can be modified at will. You can get a better idea of what is possible by visiting Syncfusion's web site (http://syncfusion.com).

# Exercises

On the USB stick you should find a workspace named gauge.dws which you can load and inspect and run program <FCtempGauges>. Move the needles.

Look at the XAML and how callback functions are used. The code simply attaches the callback function <TempChanged> to this, and disables any callback on the other CircularPointer object.

Note that if both CircularPointer objects had callbacks on <TempChanged> at the same time, the system would enter a callback loop.

Solutions to Selected Exercises

# Text files

1. size←⎕NSIZE tie←'…\UCS2file.txt' ⎕ntie 0  
   ⎕ucs 1↓⎕nread tie 163 size  
   or  
   fileUtilities.ReadFile '…\UCS2file.txt'
2. fileUtilities.ReadFile '…\aplcode.dyw'
3. fileUtilities.WriteFile'…\fn.cr' (2↓,(⎕ucs 13 10),⍤1⊢⎕cr 'TestXML')

# Component files

1. tie←'\tmp\cf1'⎕fcreate 0  
   'text vector' (2 3⍴⍳9) (2 4⍴2 '⎕' 'txtv' ⎕null ⎕a) ⎕fappend¨tie
2. (⍳100) ⎕fappend ¨ tie←'\tmp\cf2'⎕fcreate 0
   1. ⎕fread tie (⍳10)
   2. ⎕a∘⎕freplace¨tie,¨10+⍳10
   3. Use a cover function and use ⎕FHOLD

# CSV files

1. LoadTEXT '…\Fiscal\_and\_Monetary\_Data\_for\_2008-2012.csv'  
   LoadTEXT '…\Fiscal\_and\_Monetary\_Data\_for\_2008-2012.txt'(⎕ucs 9)
2. SaveTEXT (?3 4⍴1e6) '\tmp\t1.txt' ';'  
   SaveTEXT (2 3⍴ ';' 'abc' 123 '"I''m"' 'O;K' '!') '\tmp\t2.txt' ';'

# XML files

1. M←'ID' 'Name' ⍪ (⍳4), ⍪'Sue Yoo' 'Dyl Pickle' 'Bear Trapp' 'Moe Lester'  
   SaveXML M '\tmp\x1.xml'
2. xml←⎕xml ⎕NREAD t 80, ⎕NSIZE t←'…\book.xml'⎕ntie 0  
   then extract data manually, or  
   LoadXML '…\book.xml'

# Excel files

1. LoadXL '…\FMD2008-2012(subset).xlsx'
2. LoadXL '…\clients.xlsx' 'countries'
3. ]Load …\names  
   nms←#.names.genNames 20 ⋄ nms[;3]←2e9+?20⍴8e9   
   SaveXL nms '\tmp\phonebook' 'numbers'
4. file←'…\xcel\ts1301'  
   all← (LoadXL file '\*' 'B2:C20'), LoadXL file '\*' 'F2:F20'  
   keep←(~all[;2]∊⎕NULL)∧all[;1]∊⊂'ACME'  
   (task time)←↓⍉ 0 1↓keep⌿all  
   task { ⍺, 24 × +/ ⍵ } ⌸ time
5. xlf←'…\xcel\ts13' ∘,¨↓'ZI2'⎕fmt ⍳12  
   KGD←{((⍵[;2]∊⎕NULL) < ⍵[;1]∊⊂'ACME')⌿ 0 1↓⍵} ⍝ Keep Good data  
   all←↑⍪/md←{KGD ↑⍪/ (LoadXL ⍵ '\*' 'B2:C20') , ¨ LoadXL ⍵ '\*' 'F2:F20'} ¨ xlf  
   (task time)←↓⍉all ⋄ task { ⍺, 24 × +/ ⍵ } ⌸ time ⍝ yearly data  
   BONUS: md2←↑⍪/(⍳12) { ⍵[;1], ⍺, 24×⍵[;,2] } ¨ md  
   ]pivottable md2

# Databases

1. (see section instructions)
2. LoadSQL ' ACME Coyote ' 'Products'
3. )COPY SQAPL SQA  
   SQA.Connect 'c1' 'ACME Coyote'  
   SQA.Columns 'c1' 'Products'
4. LoadSQL ' ACME Coyote' 'Products' ('ProductName' 'ListPrice')
5. P ←3 2⍴ 'IBMX' 30 'Microhard' 25 'Goggle' 20  
   SaveSQL P 'ACME Coyote' 'partners' 'create table partners (name char(10), share integer)'

1. We usually talk about *Unicode* points instead of *Unicode* characters as each character is given a numeric value [↑](#footnote-ref-1)
2. The system function ⎕UCS turns a character into a Unicode point (number) and vice-versa. [↑](#footnote-ref-2)
3. UCS=Unicode Character Set [↑](#footnote-ref-3)
4. SALT is a repository of utilities provided with APL [↑](#footnote-ref-4)
5. If Notepad does not display APL characters nicely use another font like *APL385 Unicode* [↑](#footnote-ref-5)