Dyalog North America – 29-30 September 2025

Introduction to Link

Morten Kromberg
CTO

Goals

- Introduce you to Link
- Give you time to practice
  - Exporting a workspace to text files
  - Working with your new source format
  - Being hit by some of the "gotchas"
  - Creating a runtime environment / distribution
- Demo by Morten of GitHub + VS Code

Workshop Overview

14:00-15:00 Getting Started with Link

- What is Link?
- Starting a new project
- Converting an existing project

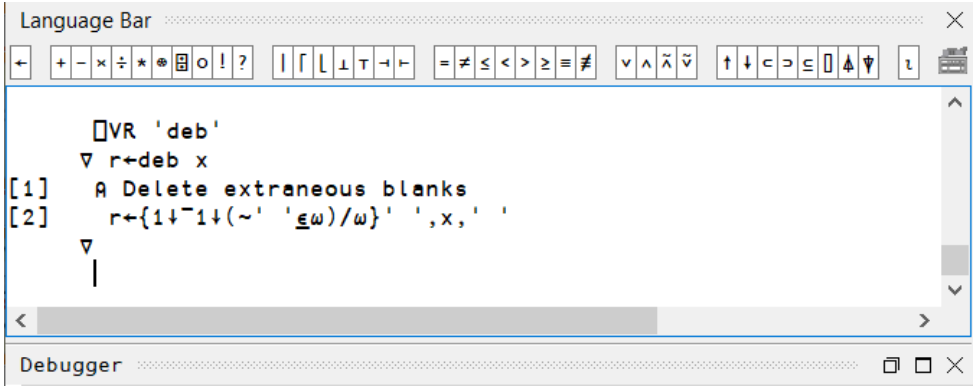
15:15-16:15 Configuration, Text Files, "Legacy" Features

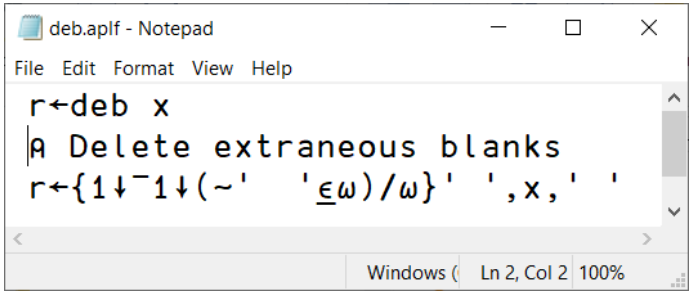
- Configuration files, Text representations, Saved workspaces with Links
- Legacy Features: Case insensitivity, Flat workspaces

16:30-17:30 The Final Touches – Distribution & Code Sharing

- Launching from Text Source
- Saving workspaces with Active Links
- Building & Deploying a real application
- Demo: How Morten uses Link

But first - What exactly is Link?





- Each code item in the active workspace is **link**ed to a file
  - [Unscripted] Namespaces map to a directory structure containing these files
- By default, the link is bi-directional:
  - If the item is modified using the APL editor, the source file is updated
  - If the source file is modified using an external tool, the workspace is updated

File System Watcher

- External source changes are detected using a "File System Watcher" - .NET is required
- Works fine for synchronising the WS with changes made in an external editor
- **NOT** appropriate for handling "bulk" changes
  - **DO NOT** unzip or copy lots of files into a watched folder
  - **DO NOT** do a large checkout/revert which changes many files
- **DO NOT** use live links to deploy code changes via shared drives to your production server!

Why is Link  ?

- With source code in text files we can use ***extremely*** attractive tools developed outside the APL community
  - Tools for editing, comparing, mergeing, refactoring, sharing, building, testing, computing statistics, ...
- ... in addition to all our own tools
- ... without losing any of what is good about interactive development

Why is Link  ?
IMPORTANT


MuleSoft


eclipse


SUBVERSION


mercurial


Jenkins


Visual Studio Code


GitHub


git


circleci


Emacs


Vim

And now also "AI" / LLMs ...

task-1.md M | test_solution.apln M | README.md | PLAN.md U | day9.aplf M | CLAUDE.md

PLAN.md > # Solution Plan for Task 1 - Traveling Salesman Problem (TSP)

```
# Solution Plan for Task 1 - Traveling Salesman Problem (TSP)

## Problem Analysis
This is a classic Traveling Salesman Problem (TSP) where we need to:
1. Visit each city exactly once
2. Find the shortest possible route
3. Can start and end at any two different cities

## Solution Approach

### Data Parsing
1. Parse input lines in format "City1 to City2 = Distance"
2. Extract unique city names
3. Build a distance matrix or lookup structure

### Algorithm Options

#### Option A: Brute Force with dfns Permutations (Recommended for small datasets)
- Use `pmat` from dfns workspace to generate all permutations
- Calculate total distance for each permutation
- Return minimum distance
- **Pros**: Simple, guaranteed optimal, leverages existing dfns
- **Cons**: O(n!) complexity, only viable for small n (< 10-12 cities)

#### Option B: Build Custom TSP Solver
- Implement dynamic programming with bitmasks
- Or use nearest neighbor heuristic
- **Pros**: More scalable
- **Cons**: Complex implementation, may not be needed for small dataset

### Recommended Implementation Steps

1. **Parse input data**
   - Use regex `'(\w+) to (\w+) = (\d+)'` to extract cities and distances
   - Create bidirectional edge map (A→B and B→A have same distance)

2. **Build distance matrix**
   - Map city names to indices
   - Create n×n matrix where matrix[i,j] = distance from city i to city j
   - Use 0 or ∞ for non-connected pairs (though problem implies all pairs are
   given)
```

Claude Code

● Now let's test with pmat to generate permutations:

● dyalog-script - execute_apl (MCP)(code: "⍝ Import pmat from dfns\n_←'pmat'⎕CY'dfns'\n\n⍝ Generate all permutations of 3 cities\nperms ← pmat 3\n⎕←'All permutations of 1 2 3:'\n⎕←perms\n\n⍝ Test calculating route distance\ndistMatrix ← 3 3⍴0 464 518 464 0 141 518 141 0\n⎕←'Distance matrix:'\n⎕←distMatrix\n\n⍝ Calculate distance for route 1 2 3\nroute ← 1 2 3\npairs ← 2,/route ⍝ Consecutive pairs\n⎕←'Route:' route\n⎕←'Pairs:' pairs\n\n⍝ Get distances for each pair\ndistances ← {distMatrix[⍵[1];⍵[2]]}¨pairs\n⎕←'Distances:' distances\n⎕←'Total:' (+/distances)")

⎿ {
    "output": "All permutations of 1 2 3:\n1 2 3\n1 3 2\n2 1 3\n2 3 1\n3 1 2\n3 2 1\nDistance matrix:\n 0 464 518\n464   0 141\n518 141   0\n Route: 1 2 3 \n Pa
  … +11 lines (ctrl+o to expand)

● Perfect! Now let's implement the complete solution in a more terse APL style:

● Update(src/day9.aplf)
⎿ Updated src/day9.aplf with 8 additions

```
     Day9←{
         ⍝ This is the solution to:
         ⍝     https://adventofcode.com/2015/day/9
+
+        _←'pmat'⎕CY'dfns'
+        parsed←'(\w+) to (\w+) = (\d+)'⎕S{⍵.(1↓Lengths↑¨Offsets↓¨⊂Block)}⊢⍵
+        cities←∪⊃,/2↑¨parsed
+        n←≢cities
+        dist←n n⍴0
+        {from to d←⍵ ⋄ i j←cities⍳from to ⋄ dist[i;j]←dist[j;i]←⍎d}¨parsed
+        ⌊/+/{dist[⍵[1];⍵[2]]}⍨¨2,/¨pmat n
     }
```

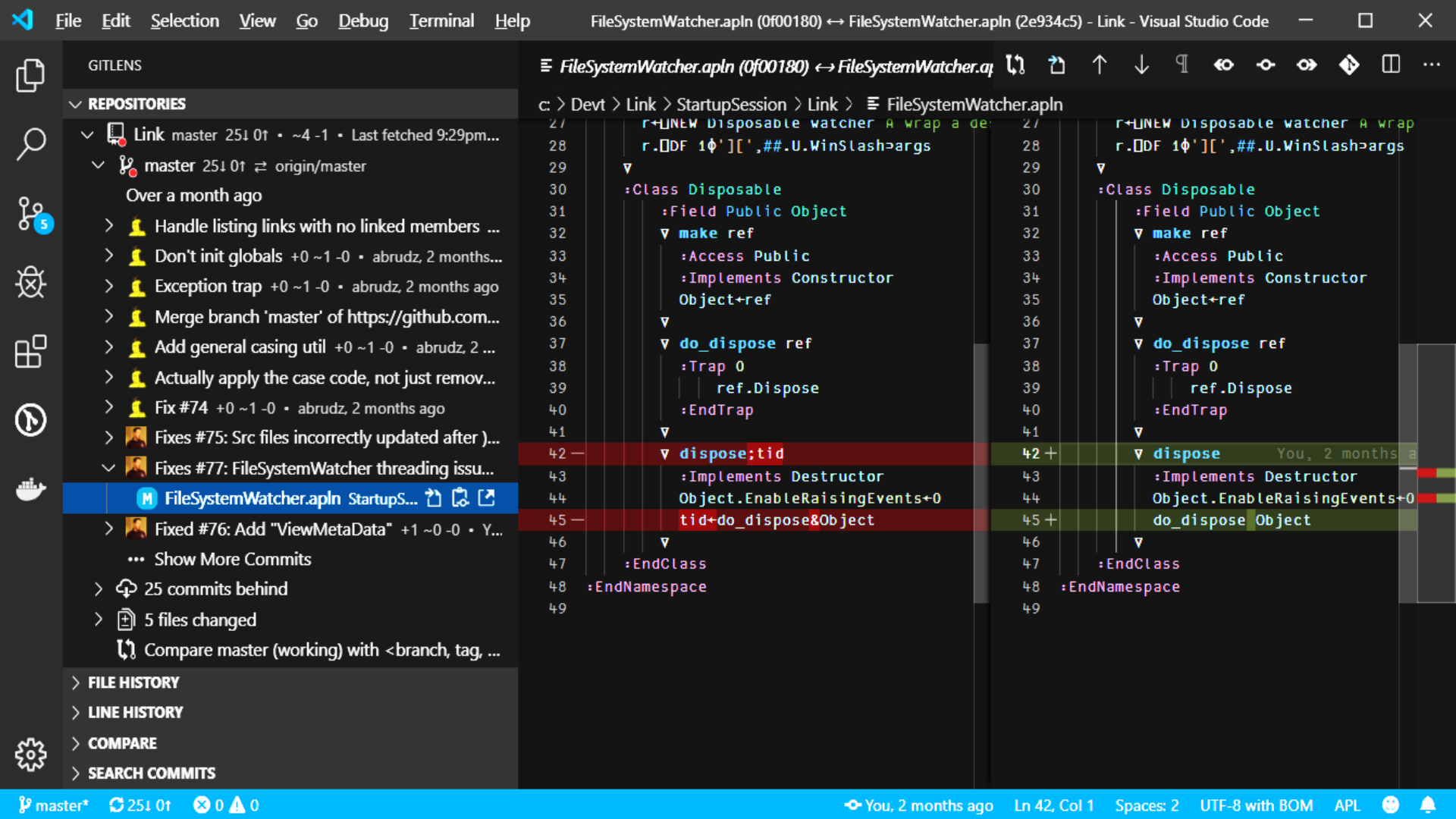
● Now let's run the tests to verify our solution:

✱ Implementing Day9 function to pass tests… (esc to interrupt · ctrl+t to hide todos)
⎿ ☒ ~~Examine existing skeleton and test files~~
☒ ~~Write tests for parsing input data~~
☐ Implement Day9 function to pass tests
☐ Run full test suite

>

▸▸ accept edits on (shift+tab to cycle)

main* ⊗0 ⚠0 ⓘ81 Live Share Git Graph


File Edit Selection View Go Debug Terminal Help
FileSystemWatcher.apln (0f00180) ↔ FileSystemWatcher.apln (2e934c5) - Link - Visual Studio Code
GITLENS
REPOSITORIES
Link master 25↓ 0↑ • ~4 -1 • Last fetched 9:29pm...
master 25↓ 0↑ ⇄ origin/master
Over a month ago
Handle listing links with no linked members ...
Don't init globals +0 ~1 -0 • abrudz, 2 months...
Exception trap +0 ~1 -0 • abrudz, 2 months ago
Merge branch 'master' of https://github.com...
Add general casing util +0 ~1 -0 • abrudz, 2 ...
Actually apply the case code, not just remov...
Fix #74 +0 ~1 -0 • abrudz, 2 months ago
Fixes #75: Src files incorrectly updated after)...
Fixes #77: FileSystemWatcher threading issu...
FileSystemWatcher.apln StartupS...
Fixed #76: Add "ViewMetaData" +1 ~0 -0 • Y...
Show More Commits
25 commits behind
5 files changed
Compare master (working) with <branch, tag, ...
FILE HISTORY
LINE HISTORY
COMPARE
SEARCH COMMITS
FileSystemWatcher.apln (0f00180) ↔ FileSystemWatcher.ap
c: > Devt > Link > StartupSession > Link > FileSystemWatcher.apln
r←⎕NEW Disposable watcher ⍝ wrap a de
r.⎕DF 1⌽'][',##.U.WinSlash⊃args
:Class Disposable
:Field Public Object
∇ make ref
:Access Public
:Implements Constructor
Object←ref
∇
∇ do_dispose ref
:Trap 0
ref.Dispose
:EndTrap
∇
∇ dispose;tid
:Implements Destructor
Object.EnableRaisingEvents←0
tid←do_dispose&Object
∇
:EndClass
:EndNamespace
r←⎕NEW Disposable watcher ⍝ wrap
r.⎕DF 1⌽'][',##.U.WinSlash⊃args
:Class Disposable
:Field Public Object
∇ make ref
:Access Public
:Implements Constructor
Object←ref
∇
∇ do_dispose ref
:Trap 0
ref.Dispose
:EndTrap
∇
∇ dispose
You, 2 months a
:Implements Destructor
Object.EnableRaisingEvents←0
do_dispose Object
∇
:EndClass
:EndNamespace
master*
25↓ 0↑
0 0
You, 2 months ago
Ln 42, Col 1
Spaces: 2
UTF-8 with BOM
APL

Fixes #77: FileSystemWatcher thr

GitHub, Inc. [US] | github.com/Dyalog/link/commit/2e934c527ce73dd77de79477fcba028ce2b56506

Apps | mkromberg (Morte... | APL | kdb - Interprocess... | The APL Orchard | c... | Git | Flying | D19

Search or jump to... / Pull requests Issues Marketplace Explore

Dyalog / link

Unwatch 9 | Star 4 | Fork 2

Code | Issues 19 | Pull requests 0 | Projects 0 | Wiki | Security | Insights | Settings

Fixes #77: FileSystemWatcher threading issue

master

Browse files

mkromberg committed on Jun 19

1 parent 767c3d0 commit 2e934c527ce73dd77de79477fcba028ce2b56506

Showing 1 changed file with 2 additions and 2 deletions.

Unified | Split

4 StartupSession/Link/FileSystemWatcher.apln

@@ -39,10 +39,10 @@

```
                ref.Dispose
            :EndTrap
        ∇
-       ∇ dispose;tid
          :Implements Destructor
          Object.EnableRaisingEvents←0
-         tid←do_dispose&Object
        ∇
    :EndClass
```

```
                ref.Dispose
            :EndTrap
        ∇
+       ∇ dispose
          :Implements Destructor
          Object.EnableRaisingEvents←0
+         do_dispose Object
        ∇
    :EndClass
```

Other Benefits of Text Source

- Easily share code between APL versions
  - Text files are backwards **and** forwards compatible

Drawbacks of Text Source

- This space intentionally left blank

Current Link Versions

Link Version	Shipped with	Also supported on
4.1	20.0	19.0
4.0	19.0	18.2
Earlier	Never Mind	

- Since 4.0, Link is stable and well documented
- Move on from earlier versions as soon as possible

Link 4.0 · Milestone #2 · Dyalog

github.com/Dyalog/link/milestone/2?closed=1

Apps | Link | EWC | METSIM | APL | Flying &

Dyalog / link

Code | Issues 87 | Pull requests | Discussic

Milestones / Link 4.0

Link 4.0

Closed Due by December 7, 2023 Closed last year

Companion of Dyalog 19.0

Open 0 **Closed** 52

- Link.Test should use new extensions
  #370 · by mkromberg was closed on Jun 29, 2023
- Allow linking a single namespace, cla
  #245 · by dyavc was closed on Sep 5, 2023
- Use Link to load other things into □S
  #377 · by abrudz was closed on Jun 25, 2023
- Link should support storing multi-lin
  #210 · by mkromberg was closed on Jan 10, 2024
- Preserve □AT information enhanceme
  #224 · by abrudz was closed on Feb 6, 2024
-]link.status does not report on "watcl
  #412 · by aplteam was closed on Mar 7, 2022
- Exportable system variables should ir
  #419 · by RonM-Dyalog was closed on Sep 5, 2023
- Relocate the Test namespace refactor
  #443 · by mkromberg was closed on Jun 29, 2023

Link 4.1 (for 20.0) · Milestone #

github.com/Dyalog/link/milestone/5?closed=1

Apps | Link | EWC | METSIM | APL | Flying & Sailing | Car | Dyalog | Cloud | Travel | Linux | Sport | Productivity | Ferie 2022 | Sommerferie 2025 | All Bookmarks

Dyalog / link

Type / to search

Code | Issues 87 | Pull requests | Discussions | Actions | Projects | Wiki | Security | Insights | Settings

Milestones / Link 4.1 (for 20.0)

Edit | Close Milestone | New issue

Link 4.1 (for 20.0)

Open Overdue by 5 month(s) • Due by March 31, 2025 Last updated 4 months ago

100% complete

To go out with 20.0 ETA 2024Q2

Open 0 **Closed** 27

- Improve error reporting on failed file import enhancement MID
  #421 · by mkromberg was closed on Feb 4
- Link can miss updates when a Git Revert (or other mass-operation) is done bug HIGH
  #492 · by mkromberg was closed on Feb 13 — 1
- Better error message when a path contains an invalid name enhancement MID
  #474 · by abrudz was closed on Jan 29 — 1
- Class and directory worked with :Include in Link 2.0.6 but not Link 3.0 bug HIGH
  #525 · by dyavc was closed on Feb 13 — 4
- Quickly editing can lead to loss of stop bits bug MID
  #514 · by abrudz was closed on Feb 13 — 2
- link with flatten: getFilename hook called when new file is added to a linked folder, but with wrong filename in args bug HIGH
  #536 · by e9gille was closed on Feb 14 — 1
- link w/o flatten: getFilename hook called when new file is added to a linked folder, but with wrong apl name in args bug HIGH
  #537 · by e9gille was closed on Feb 14 — 1
- Allow linking to a non-empty dir that only contains irrelevant files enhancement HIGH
  #554 · by abrudz was closed on Feb 3
- Silent failures for Windows file names that represent devices bug HIGH
  #653 · by abrudz was closed on Feb 3 — 7

Exercise 0 - Documentation

Open the documentation in a browser:

- https://dyalog.github.io/link/4.1/

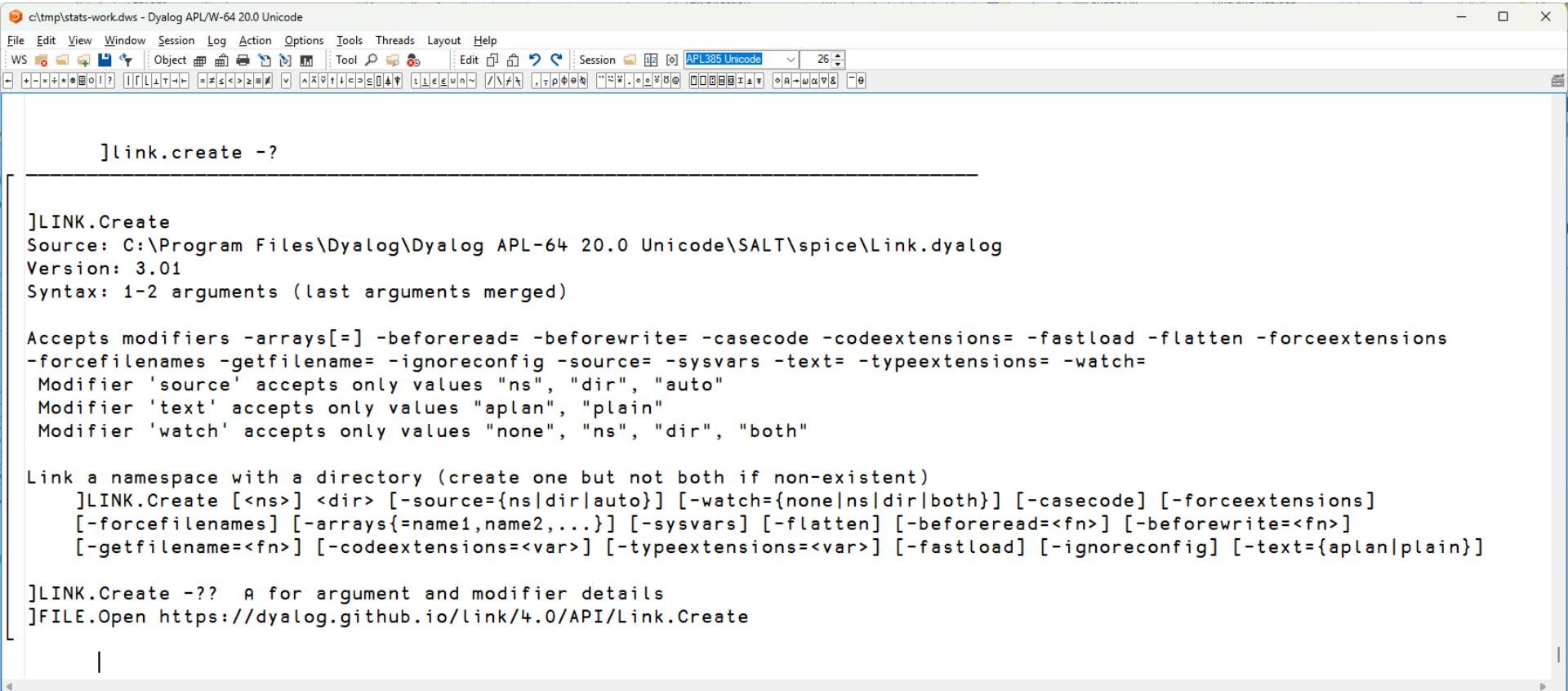
or

- Use -? or -?? with user commands

  ```
  ]link.create -?
  ```

  … Look at the last line of output

Documentation



Introduction

Link allows you to use Unicode text files to store APL source code, rather than "traditional" binary workspaces. The benefits of using Link and text files include:

- It is easy to use source code management (SCM) tools like Git or Subversion to manage your code. Although an SCM is not a requirement for Link, Dyalog ***highly*** recommends using Git or similar systems to manage source code that Link will load into your APL session.
- Changes to your code are **immediately** written to file: there is no need to remember to save your work. The assumption is that you will make the record permanent with a *commit* to your source code management system, when the time is right.
- Unlike binary workspaces, text source can usually be shared between different versions of APL - or even with human readers or writers who don't have APL installed at all.

Link is NOT...

- **A source code management system**: Link itself has no source code management features. As mentioned above, you will need to use a separate tool like Git to manage the source files that Link will allow you to use and modify from Dyalog APL.
- **A database management system:** although Link is able to store APL arrays using *array notation*, this is only intended to be used for constants which you consider to be part of the source code of your applications. Although all functions and operators that you define using the editor will be written to source files by default, source files are only created for arrays by explicit calls to Link.Add or by specifying optional parameters to Link.Export. Application data should be stored in a database management system or files managed by the application.

Basic Usage

These sections cover the most commonly used commands. For more advanced usage, please consult the API documentation.

Starting from an existing folder containing text files

Use Link.Create to Link a directory containing text source to a namespace in the active workspace.

The following example loads APL code from the folder **/users/sally/myapp** into a namespace called `myapp`.

```
      ⎕SE.Link.Create myapp '/users/sally/myapp'
```

For ad hoc use in an interactive session, it might be more convenient to use the user command:

```
      ]LINK.Create myapp /users/sally/myapp
```

Linking a directory on startup

If you are using Dyalog version 18.2 or later, you can cause a link to be created to the root of your workspace (`#`) as APL starts, by setting the `LOAD` parameter on the command line or as an environment variable. After establishing the link, the system will call the function `Run` with a right argument containing the name of the directory. You can disable the call to `Run` by including the `-x` switch on the command line (in the same way that the `-x` switch inhibits the execution of the latent expression when loading a workspace).

Starting a New Project

- To start a new project

```
)ns myns
]link.create myns /my/dir
```

- At least one of `myns` (the namespace) or `/my/dir` (the directory) must exist
- If both exist, only one may be populated
- You can also create a Link to #

```
]link.create # /my/dir
```

Starting a new project

If you are starting a completely new project, you can either create a namespace in the active workspace, or a folder on the file system (or both), and use Link.Create, naming the namespace and the folder, as in the example at the start of this page.

- If neither of them exist, Link.Create will reject the request on suspicion that there is a typo, in order to avoid silently creating an empty directory by mistake.
- If both of them exist AND contain code, and the code is not identical on both sides, Link.Create will fail and you will need to specify the `source` option, whether the namespace or the directory should be considered to be the source. Incorrectly specifying the source will potentially overwrite existing content on the other side, so use this with extreme caution!

To illustrate, we will create a namespace and populate it with two dfns and one tradfn, in order to have something to work with. In this example, the functions are created using APL expressions; under normal use the functions would probably be created using the editor, or perhaps loaded or copied from an existing workspace.

```
      'stats' ⎕NS ⍬ ⍝ Create an empty namespace
      stats.⎕FX 'mean←Mean vals;sum' 'sum←+⌿,vals' 'mean←sum÷1⌈⍴,vals'
      stats.Root←{⍺←2 ⋄ ⍵*÷⍺}
      stats.StdDev←{2 Root(+.×⍨÷⍴),⍵-Mean ⍵}
```

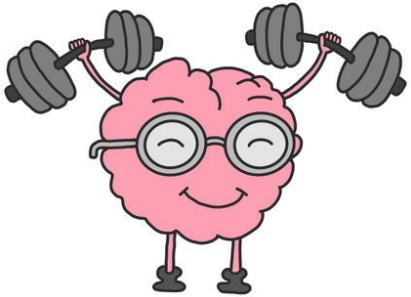
We could now create a source directory using Link.Export, and then use Link.Create to create a link to it. However, Link.Create can do this in one step: assuming that the directory `/users/sally/stats` is empty or does not exist, the following command will detect that there is code in the namespace but not in the directory, and create a link based on the namespace that we just populated with our functions:

```
      ]LINK.Create stats /users/sally/stats
Linked: #.stats ↔ C:\tmp\stats
```

Source File Extensions

- Link creates files with extensions that identify the type of APL item exported
  - `.aplf` and `.aplo` for functions and operators
  - `.apla` for arrays
  - `.aplc` and `.apli` for classes and interfaces
  - `.apln` for namespaces that have source
    - namespaces w/out source become directories

Exercise 1 – Your first Link

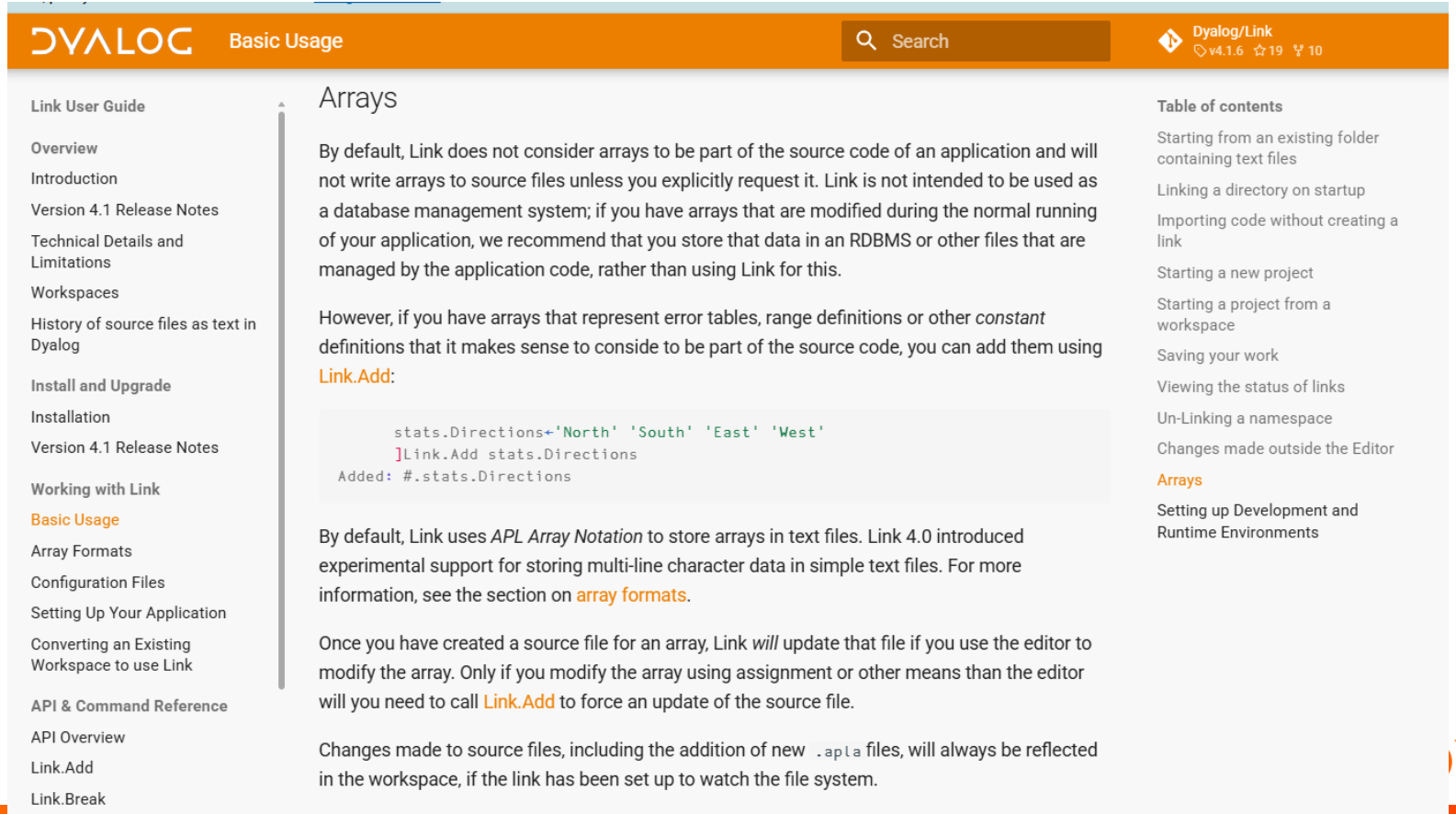


- Create a namespace with at least one function in it
- Create a link between the namespace and a new or empty directory
- Verify that source files were created in the directory
- Edit a function and verify that the source file is updated
- Edit a source file using notepad or another external editor
  - Verify that the function is updated in the WS
- `)CLEAR` and re-create the link
- Verify that your code is loaded into the workspace

Variables

- By default, variables are **NOT** linked
  - Most variables are not part of the "source code"
- The `-arrays` switch will cause ALL arrays to be exported on Link.Create
- Once a source file exists for an array, Link will respond to source changes
- Link.Add can be used to create a source file, or update it

Variables



DYALOG Basic Usage

Search

Dyalog/Link
v4.1.6 ☆19 10

Link User Guide

Overview
Introduction
Version 4.1 Release Notes
Technical Details and Limitations
Workspaces
History of source files as text in Dyalog

Install and Upgrade
Installation
Version 4.1 Release Notes

Working with Link
Basic Usage
Array Formats
Configuration Files
Setting Up Your Application
Converting an Existing Workspace to use Link

API & Command Reference
API Overview
Link.Add
Link.Break

Arrays

By default, Link does not consider arrays to be part of the source code of an application and will not write arrays to source files unless you explicitly request it. Link is not intended to be used as a database management system; if you have arrays that are modified during the normal running of your application, we recommend that you store that data in an RDBMS or other files that are managed by the application code, rather than using Link for this.

However, if you have arrays that represent error tables, range definitions or other *constant* definitions that it makes sense to conside to be part of the source code, you can add them using Link.Add:

```
      stats.Directions←'North' 'South' 'East' 'West'
      ]Link.Add stats.Directions
Added: #.stats.Directions
```

By default, Link uses *APL Array Notation* to store arrays in text files. Link 4.0 introduced experimental support for storing multi-line character data in simple text files. For more information, see the section on array formats.

Once you have created a source file for an array, Link *will* update that file if you use the editor to modify the array. Only if you modify the array using assignment or other means than the editor will you need to call Link.Add to force an update of the source file.

Changes made to source files, including the addition of new `.apla` files, will always be reflected in the workspace, if the link has been set up to watch the file system.

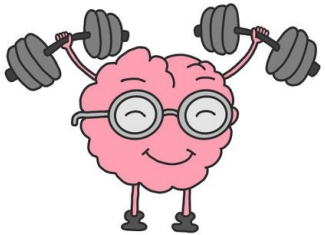
Table of contents

Starting from an existing folder containing text files
Linking a directory on startup
Importing code without creating a link
Starting a new project
Starting a project from a workspace
Saving your work
Viewing the status of links
Un-Linking a namespace
Changes made outside the Editor
Arrays
Setting up Development and Runtime Environments

Assignment

- Link will create or update source files for functions, operators and variables if they are **EDITED**
  - For variables, **existing** source files will be updated
- Assigning new values to functions or variables using ← will **NOT** update the source file
  - The same is true for]NS,]CY,]FX or any system function that modifies or creates definitions
- Use Link.Add to cause files to be created or updated

Exercise 2 - Variables



- Create a variable containing a constant that your application needs
- Write it to file using `]Link.Add`
- Inspect the file, edit it, and verify that the new value appears in the workspace
- Change the value in the workspace
  - Note that the file is NOT updated
  - Update it using]Link.Add
- Q: Can you write system variables to file?
  - (check the docs)

Link.Create - Recap

Link.Create creates a "live" link between a namespace and a directory

- One end of the link must be empty, code is replicated at the empty end of the link
- By default, changes made on either side are immediately replicated on the other
  - You can set `-watch=ns|dir` if you only want changes "watched for" on one side of the link
- Variables are not exported by default
- Only the editor (or Link.Add) will trigger file updates

Import and Export

If you do not want a live link after the operation:

- **Import** brings objects into a namespace from a directory
- **Export** "saves" the contents of a namespace to files

Most of the switches/options that apply to Create also apply to Import and Export

Namespace = Directory

- Each function, operator or array is linked to a file
- Namespaces which are defined using source text are also linked to a single file with extension `.apln`
  - If such a namespace contains "free" functions or variables, Link will ignore them
- Namespaces **without source** map to directories
- If an exported namespace contains sub-namespaces
  - Each one becomes a subdirectory
- If an imported directory contains subdirectories
  - Each one becomes a namespace

myns.apln

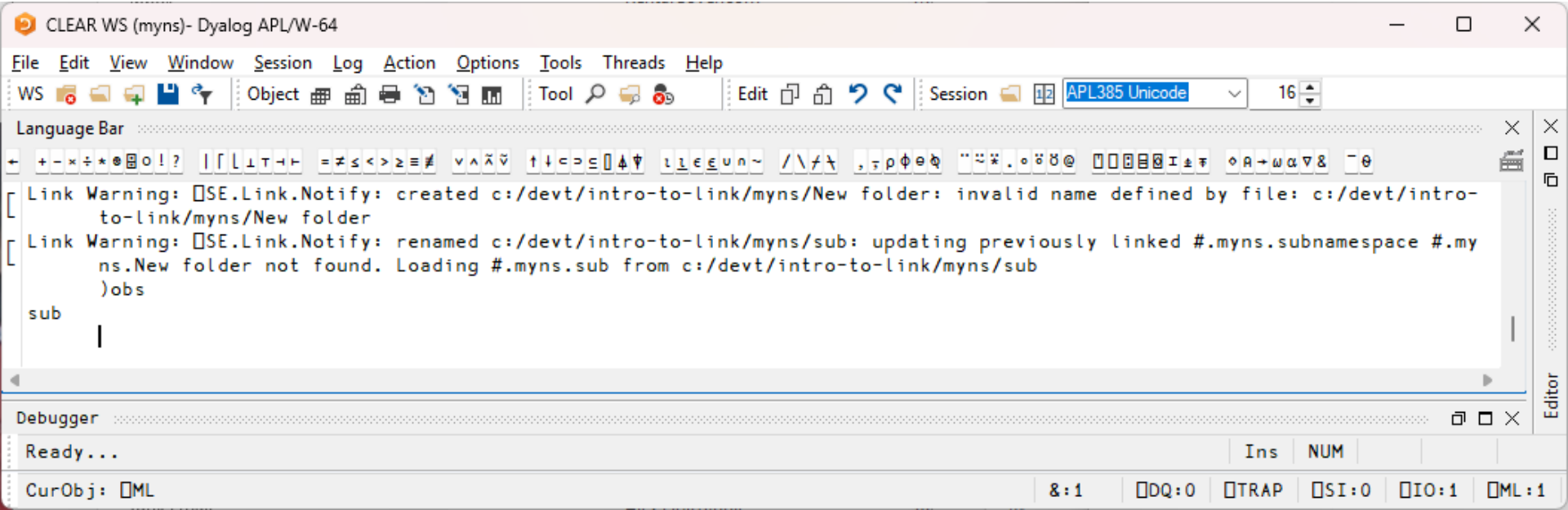
```
:Namespace myns
    ...
:EndNamespace
```

Exercise 3 - Subdirectories

- Create a subdirectory in your source structure
  - The name must be a valid APL namespace name
- Verify that a corresponding namespace is created in the workspace
-)ED a function in the namespace
- Rename the subdirectory
- Verify the effect in the workspace

Ugh

- If you started creating "New folder", Link will complain



Converting an Existing Workspace to use Link

In order to start using Link to maintain code that resides in a workspace, you first need to export the code in the workspace to one or more folders.

The simplest way to do this is to use Link.Export. In principle, it should be possible to write the entire contents of any workspace to an empty folder called `/folder/name` using the following:

```
'options' ⎕NS ⍬
options.(arrays sysVars)←1
options ⎕SE.Link.Export # '/folder/name'
```

or equivalently, using the user command:

```
]link.export # /folder/name -arrays -sysvars
```

You can also use Link.Create with the same arguments, if you want an active link to exist after the export has been done.

Options

-arrays

By default, Link assumes that the "source code" only consists of functions, operators, namespaces and classes. Variables are assumed to contain data which is transient and thus not part of the source. The `-arrays` causes all arrays in the workspace to be written to source files as well. You can also write selected variables to file, see the documentation for Link.Create for more options.

-sysVars

By default, Link will assume that you do **not** wish to record the settings for system variables, because your source will be loaded into an environment that already has the desired settings. If you want to be 100% sure to re-create your workspace exactly as it is, you can use `-sysVars` to record the values of system variables from each namespace in source files.

Beware that this will add a *lot* of mostly redundant files to your repository. It is probably a better idea to analyse your workspace carefully and only write system variables to file if you really need them, using Link.Add.

Case Coding

- Case-insensitive file systems (like Windows) do not distinguish between FOO.aplf and Foo.aplf
- Link avoids this issue using "case coding"

Case Coding

- Avoid file name clashes in case-insensitive file systems by adding an octal representation of the case to the file name:

```
      ]link.create # c:\tmp\foos
ERRORS ENCOUNTERED: ⎕SE.Link.Create: File name case clash -
try using caseCode←1:
#.FOO
#.foo

     ]link.create # c:\tmp\foos -casecode
Linked: # ↔ "c:\tmp\foos"

     0 (⎕NINFO⍠1) 'c:\tmp\foos\*.aplf'
c:/tmp/foos/foo-0.aplf c:/tmp/foos/FOO-7.aplf
```

```
     ∇foo
[1]  2+2
     ∇

     ∇FOO
[1]  3+3
     ∇
```

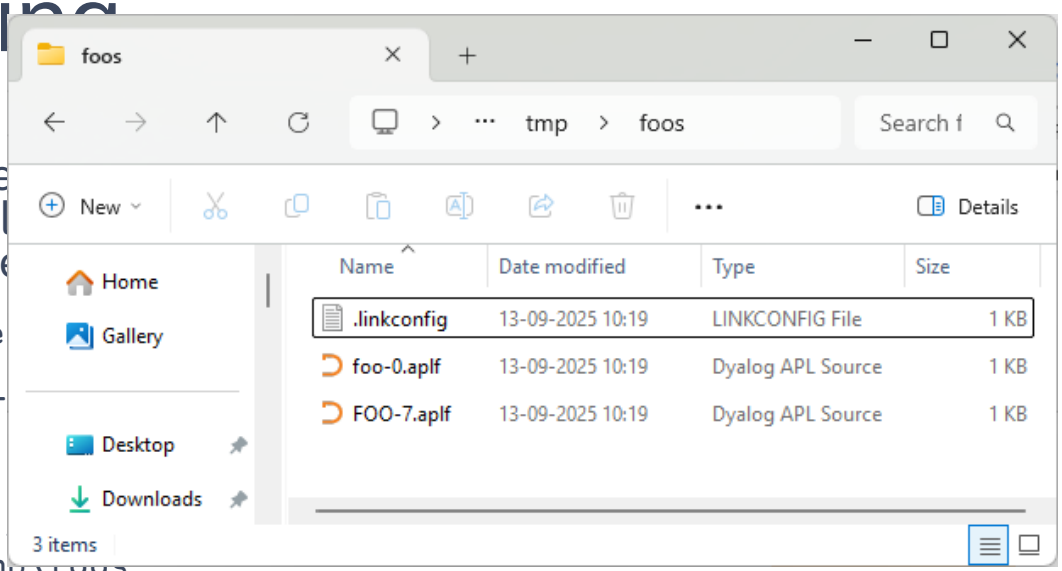
Case Coding

- Avoid file name
  adding an octal
  to the file name

```
      ]link.create
ERRORS ENCOUNTERED:
try using caseCode←
#.FOO
#.foo

      ]link.create
Linked: # ↔ "c:\tmp\foos

      0 (⎕NINFO⍠1) 'c:\tmp\foos\*.aplf'
c:/tmp/foos/foo-0.aplf c:/tmp/foos/FOO-7.aplf
```




```
      ∇foo
[1]   2+2
      ∇

      ∇FOO
[1]   3+3
      ∇
```

API vs User Commands?

All user commands have an API equivalent. Or as Adám would put it, most API functions have a user command for interactive use.

```
]link.export # whatever -casecode

(caseCode:1) □SE.Link.Export # 'whatever'
```

When you automate Link operations, **PLEASE DO NOT:**

```
□UCMD 'link.export # whatever -casecode'
```

NB: that all user command switches are lowercase (`casecode`), while option namespace names are camelcase (`caseCode`).

Documentation is for the API

caseCode

Default: **off**

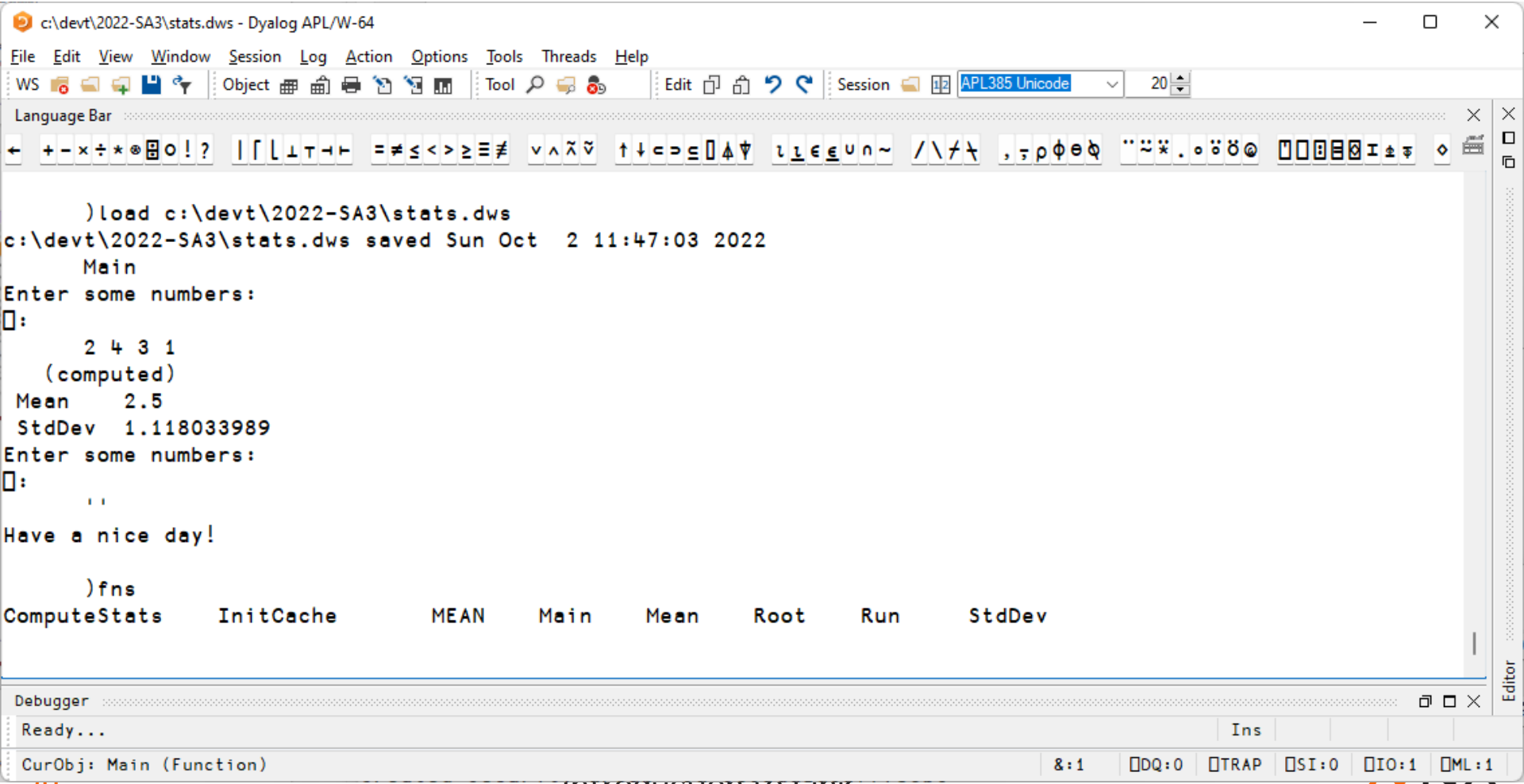
The **caseCode** flag adds a suffix to file names on write.

If your application contains items with names that differ only in case (for example `Debug` and `DEBUG`), and your file system is case-insensitive (for example, under Microsoft Windows), then enabling **caseCode** will cause a suffix to be added to file names, containing an octal encoding of the location of uppercase letters in the name.

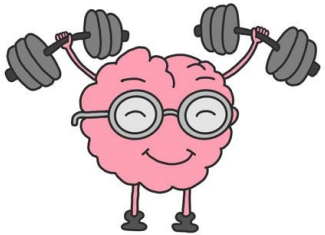
For example, with caseCode on, two functions named `Debug` and `DEBUG` will be written to files named `Debug-1.aplf` and `DEBUG-37.aplf`.

Note

Dyalog recommends that you avoid creating systems with names that differ only in case. This feature primarily exists to support the import of applications which already use such names. You will probably also want to enable **forceFilenames** if you enable **caseCode**.


c:\devt\2022-SA3\stats.dws - Dyalog APL/W-64
File Edit View Window Session Log Action Options Tools Threads Help
WS Object Tool Edit Session APL385 Unicode 20
Language Bar
)load c:\devt\2022-SA3\stats.dws
c:\devt\2022-SA3\stats.dws saved Sun Oct 2 11:47:03 2022
Main
Enter some numbers:
⎕:
2 4 3 1
(computed)
Mean 2.5
StdDev 1.118033989
Enter some numbers:
⎕:
''
Have a nice day!
)fns
ComputeStats InitCache MEAN Main Mean Root Run StdDev
Debugger
Ready...
Ins
CurObj: Main (Function)
&:1
DQ:0
TRAP
SI:0
IO:1
ML:1
Editor

Exercise 4

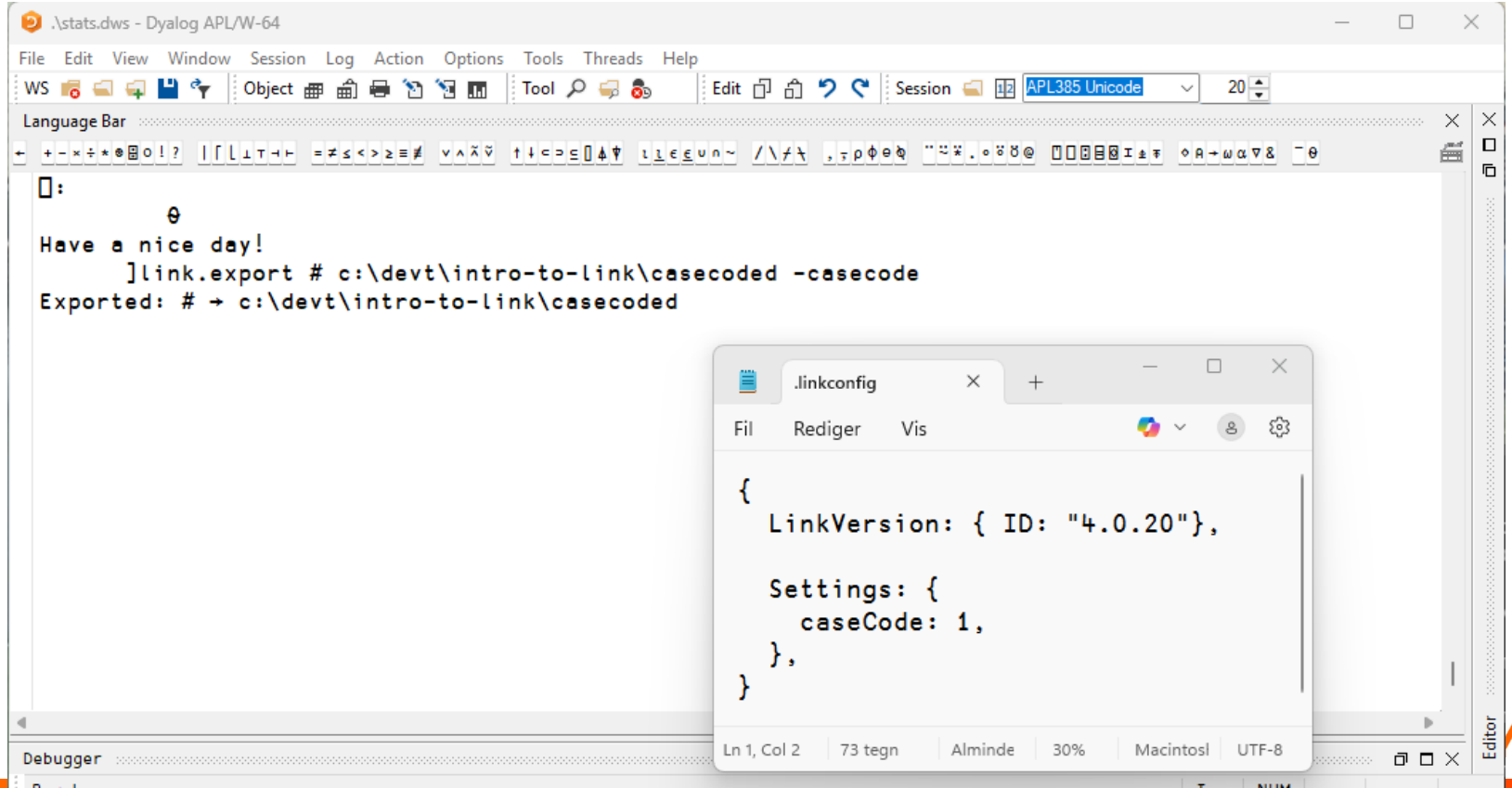


- Export the workspace `stats.dws` to a directory
- Note that
  - It contains two variables
  - Has a non-default ML
  - Has two names which differ only in case

Exercise 4 - Discussion

- Use `-casecode` or rename?
- Which variables should be considered "source"
  - `STATFNS` ?
  - `RESULTS` ?
- `⎕ML=3`:
  - Use –sysvars switch,]link.add `⎕ML`
  - … or refactor to use `⎕ML←1`

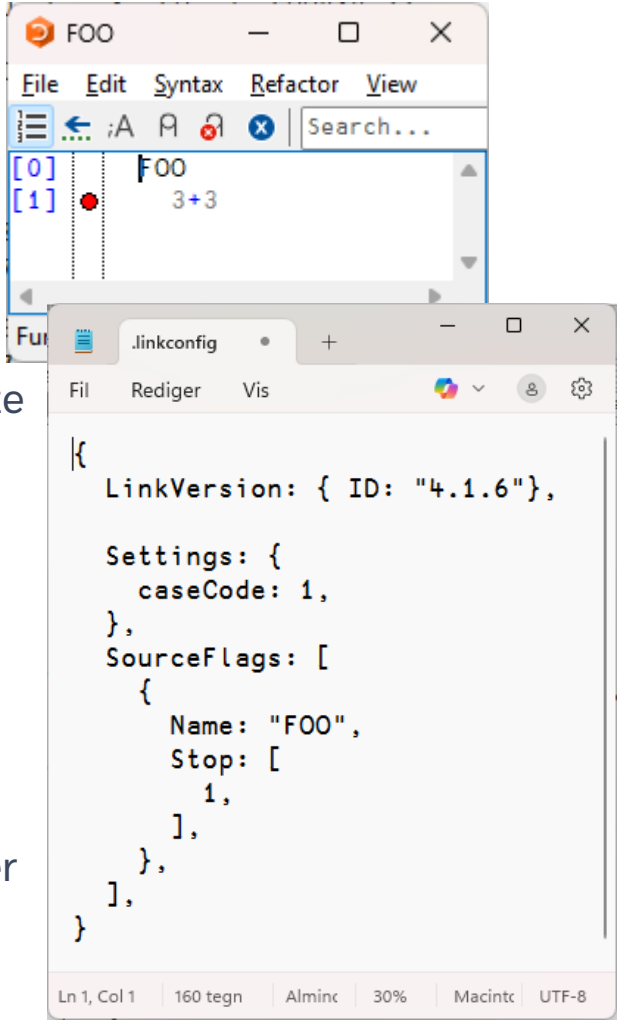
- If you used –casecode, a .linkconfig file is created



The .linkconfig file

Introduced in Link 4.0

- Records non-default link options
  - So you don't need to say –casecode each time you create a link to that folder
- Tracks - and reinstates - Stop+Trace flags on Create
  ```
  ]link.stop FOO 1
  ```
  - "Automatic" registration of stop bits in a file in the application direction is a bit controversial, may become an option
  - NB: A bit experimental; flags are not recorded if no other change is made to the source ☹



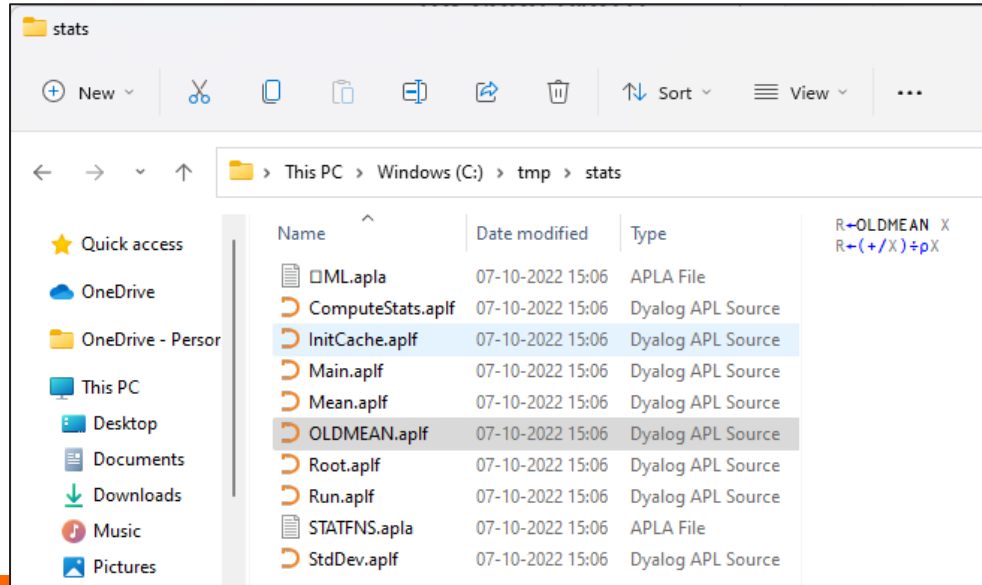
Exercise 4 – Morten's Solution

```
      )load c:\devt\intro-to-link\stats
c:\devt\intro-to-link\stats.dws saved Tue Oct 4 22:59:08 2022
      )fns
ComputeStats InitCache MEAN Main Mean Root Run StdDev
      )ed MEAN ⍝ rename to OLDMEAN
      )erase MEAN

      )vars
RESULTS STATFNS
      ]link.export # c:\tmp\stats
Exported: # → c:\tmp\stats

      ]link.export ⎕ML c:\tmp\stats
Exported: #.⎕ML → c:/tmp/stats/⎕ML.apla

      ]link.export STATFNS c:\tmp\stats
Exported: #.STATFNS → c:/tmp/stats/STATFNS.apla
```



Issues with old applications

Link & Dyalog APL offer some help:

- -casecode for names that differ only in case
- -flatten for workspaces that are not divided into namespaces
- Underscores: "do not exist" in Unicode

You're on your own:

- Objects that have no text representation
- Derived functions

The `-flatten` switch

Even if everything in your old workspace is in #, it might benefit from being organised into separate directories (utilities, etc)

- `-flatten` allows you to load the contents of multiple directories into a flat workspace
- The link between individual functions and files is maintained

Advanced Options

ignoreconfig

Default: **off**

The **ignoreconfig** allows you to ignore the `.linkconfig` file in the linked directory. This can be useful during debugging, especially if you have a damaged configuration file.

flatten

Default: **off**

The **flatten** flag prevents the creation of sub-namespaces in the active workspace.

The **flatten** option will load all items into the root of the linked namespace, even if the source code is arranged into sub-directories. This is typically used for applications that have source which is divided into modules, but still expects to run in a "flat" workspace.

Note that if **flatten** is set newly created items need special treatment:

- If a function or operator is renamed in the editor, the new item will be placed in the same folder as the original item.
- If a new item is created, it will be placed in the root of the linked directory.
- It is also possible to use the **getFilename** setting to add application-specific logic to determine the file name to be used (or prompt the user for a decision).

A suggested workflow is to always create a stub source file in the correct directory and edit the function that appears in the workspace, rather than creating new functions in the workspace.

This option takes effect only when **source** is **dir**.

DYNA

Underscores

- There are no Underscored characters in Unicode
  - Underlining is seen as a style, like bold or italic
- The APL385 Unicode Font tells little white lies:

```
      □UCS 9397+⍳26 ⍝ From U+24B6
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

- Using a normal Unicode font, these display as:

ⒶⒷⒸⒹⒺⒻⒼⒽⒾⒿⓀⓁⓂⓃⓄⓅⓆⓇⓈⓉⓊⓋⓌⓍⓎⓏ

- Dyalog recommends that you try to eliminate these characters from your applications ASAP.
- The "font trick" is a temporary measure.

Non-Representable Objects

- Some objects that CAN be saved in a Dyalog workspace have no meaningful textual representation
  - GUI & COM objects
- You need to write code which creates these objects at run or build time
- Such "binary" objects are also non-transferrable between 32/64 or classic/unicode; they can only be loaded by the same architecture

Example: OLE Server...

Example of explicit creation of an otherwise un-representable object:

```
     ∇ R←MakeOLEServer;ns;spec
[1]   ⍝ Recreate the OLE Server before WS is built
[2]    ns←#.StatsServer
[3]    ns.⎕WC'OleServer'('ClassID' '{395E64DF-6B44-4515-B409-6A0A2E1ACD9B}')
                        ('RunMode' 'SingleUse')
[4]    spec←⊂'This function returns the mean' 'VT_VARIANT'
[5]    spec,←⊂'InputNumbers' 'VT_VARIANT'
[6]    ns.SetFnInfo'Mean'spec
[7]    R←0
     ∇
```

Derived Functions

Derived functions are currently not supported:

```
      matmult←+.×
      ]link.add matmult
ERRORS ENCOUNTERED: ⎕SE.Link.Add:
   Cannot get source of #.myns.matmult: Invalid name class (3.3)
```

Solutions:

- Define a dfn or tradfn, e.g `matmult←{⍺+.×⍵}`
- Define a Namespace Script (or an APL Script) which creates your little dervs:

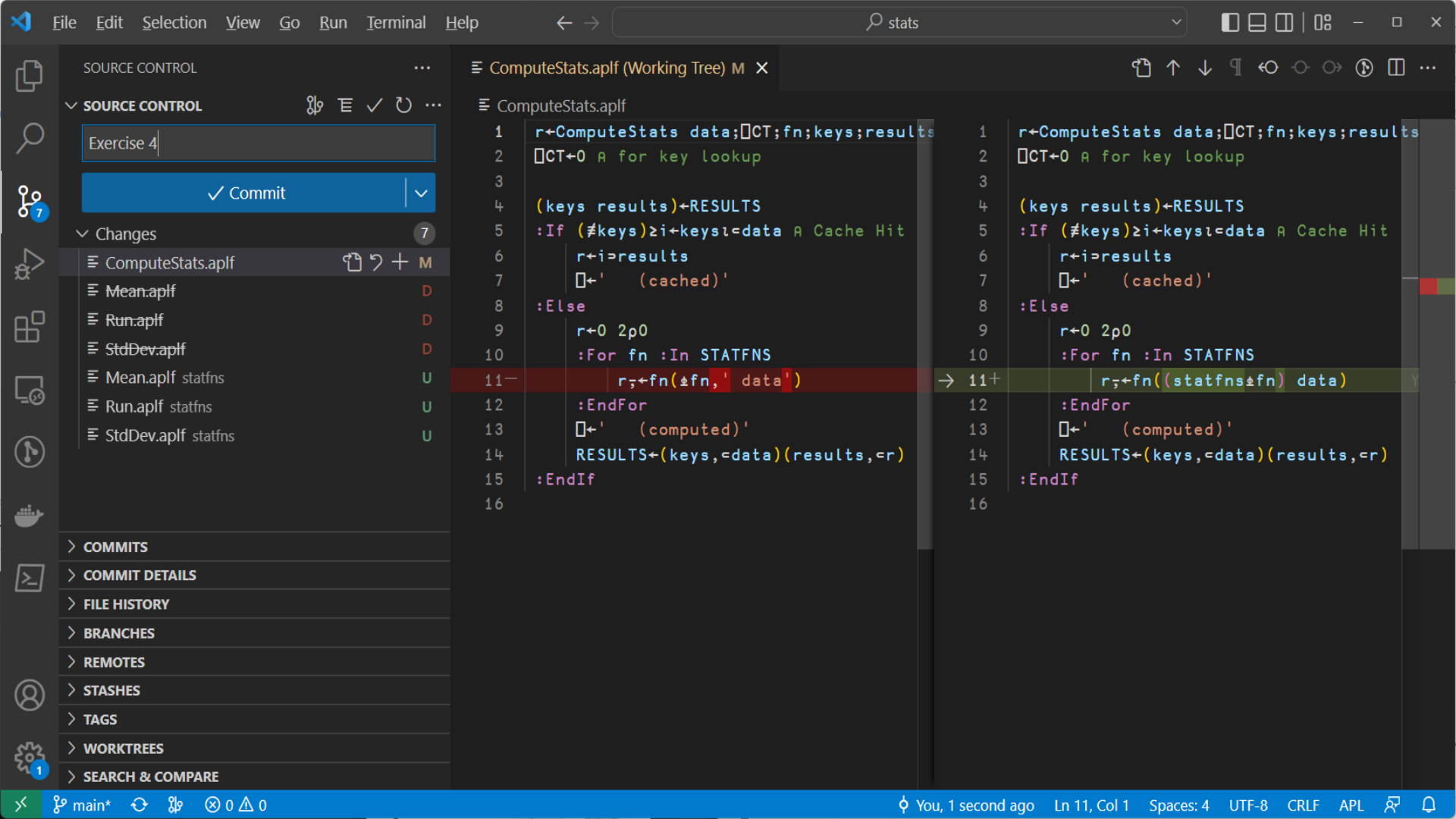
```
:Namespace Utils
   matmult←+.×
:EndNamespace
```

Exercise 5: flatten (or do not)

- Create a subdirectory called "statfns"
- Move the source files for the statistical functions (Mean, StdDev, Root) into it
- Get the application to run again

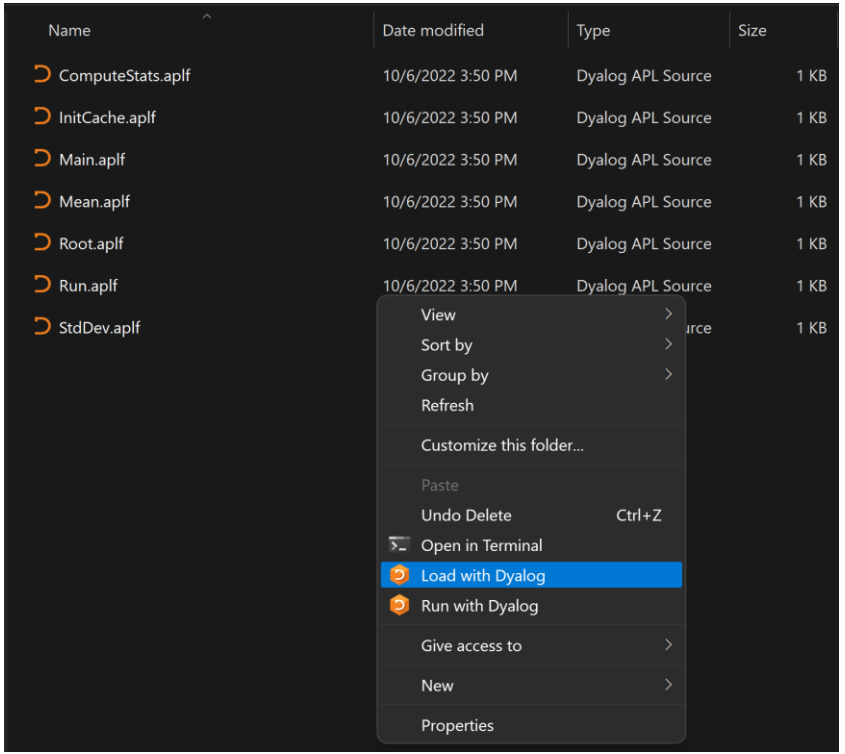
Exercise 5 - Discussion

- `-flatten` or refactor?


File Edit Selection View Go Run Terminal Help
stats
SOURCE CONTROL
SOURCE CONTROL
Exercise 4
Commit
Changes 7
ComputeStats.aplf M
Mean.aplf D
Run.aplf D
StdDev.aplf D
Mean.aplf statfns U
Run.aplf statfns U
StdDev.aplf statfns U
COMMITS
COMMIT DETAILS
FILE HISTORY
BRANCHES
REMOTES
STASHES
TAGS
WORKTREES
SEARCH & COMPARE
ComputeStats.aplf (Working Tree) M
ComputeStats.aplf
r←ComputeStats data;⎕CT;fn;keys;results
⎕CT←0 ⍝ for key lookup
(keys results)←RESULTS
:If (≢keys)≥i←keys⍳⊂data ⍝ Cache Hit
r←i⊃results
⎕←' (cached)'
:Else
r←0 2⍴0
:For fn :In STATFNS
r,←fn(⊂fn,' data')
:EndFor
⎕←' (computed)'
RESULTS←(keys,⊂data)(results,⊂r)
:EndIf
r←ComputeStats data;⎕CT;fn;keys;results
⎕CT←0 ⍝ for key lookup
(keys results)←RESULTS
:If (≢keys)≥i←keys⍳⊂data ⍝ Cache Hit
r←i⊃results
⎕←' (cached)'
:Else
r←0 2⍴0
:For fn :In STATFNS
r,←fn((statfns⍳⊂fn) data)
:EndFor
⎕←' (computed)'
RESULTS←(keys,⊂data)(results,⊂r)
:EndIf
main*
0 0
You, 1 second ago
Ln 11, Col 1
Spaces: 4
UTF-8
CRLF
APL

"Boot" from Source

- Right click in the file explorer
  - "Load with Dyalog" will do a Link.Create on a selected folder, or import a selected file
  - "Run with Dyalog" will look for a function called Run and invoke it if it exists after the link has been created.
- `]FileAssociations` can be used to select APL version



LOAD= parameter

- Point to a file, or a directory
- Can be specified on the command line, or in a .dcfg file
- Add –x to disable startup (just setting LOAD is actually equivalent to "Run with Dyalog")

Example .dcfg file:

```
{
 Settings: {
        AutoPW: 1,
        MaxWS: "512M",
        DadoProjectsFolder: "C:/Git",
        PropertyExposeRoot: 1,
        LOAD: "C:/Git/stats"
        }
}
```

The Run function

- The stats workspace contains a function Run:

```
    ∇ Run dir
[1] Main
    ∇
```

- When launched by LOAD, the right argument will be the name of the folder that was loaded. Also:

```
    ⎕SE.Link.LaunchDir
C:/devt/intro-to-link/old-export
```

Is the Workspace Dead?

With Link, you no longer need to save workspaces (?)

- Well, except for distribution of your application
- Or saving WIP including live data that is not stored in Link, in order to resume work later
- Or saving "crash workspaces"
- Anything else?

Saved Workspaces with Links

From v4.0, Link handles workspaces with "live" links kinda OK

- This is not a very well tested feature

- Do NOT rely on this as a way to manage multiple variations over your source code.
- The correct way to manage branches of your source code is to use something like Git[Hub], we'll look at that soon.

Saved Links

```
      ]link.create # c:\devt\intro-to-link\export
Linked: # ↔ c:\devt\intro-to-link\export

      data←?100⍴100
```

Lunch time!

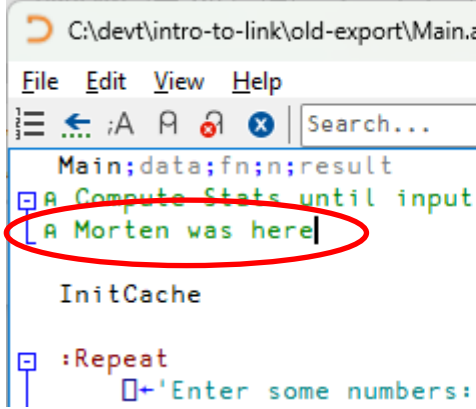
```
      )save c:\tmp\stats-work.dws
c:\tmp\stats-work.dws ⍝ saved Tue Sep 23 18:20:43 2025
```

Back from lunch!

```
      )load c:\tmp\stats-work.dws
c:\tmp\stats-work.dws ⍝ saved Tue Sep 23 18:20:43 2025
Link Warning: 1 link restored: #
```

IF source files match WS contents, the link is just restored. If not, see next slide.

If source does not match WS




```
      )load c:\tmp\stats-work.dws
c:\tmp\stats-work.dws A saved Tue Sep 23 18:20:43 2025
Link Warning: IMPORTANT: 1 namespaces linked in this workspace:
Link Warning: IMPORTANT: Link.Resync is required

      ]link.resync
1 update required: use -proceed option to synchronise

Name    Direction  File                                          Comments
#.Main  ←          c:/devt/intro-to-link/old-export/Main.aplf  File is dated 1 minute ago, WS copy is dated 23 Sep 2025

      ]link.resync  -proceed
1 file read
```

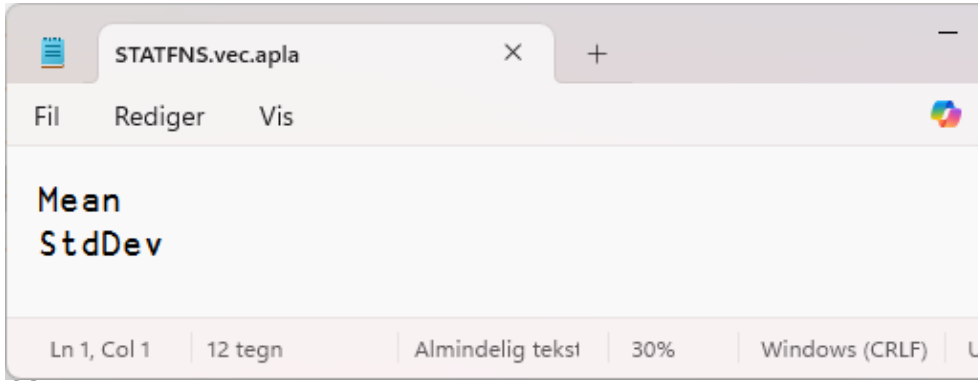
IF you do not agree with the recommended "Direction", you must manually resolve the differences by copy/pasting source etc.

Exercise 6 – Saved Links

- Save a workspace with a Link in it
- Edit a function source file using the editor of your choice
- Reload the workspace and sort things out

"Simple" Text Arrays

- By default, APL Array Notation is used to store all arrays
- This is not always the best format if your data is
  - A character vector with completely regular line endings
    - CR (UCS 13) , LF (10) or CRLF (13 10)
  - A vector of character vectors with no line endings (like STATFNS)
  - A character matrix

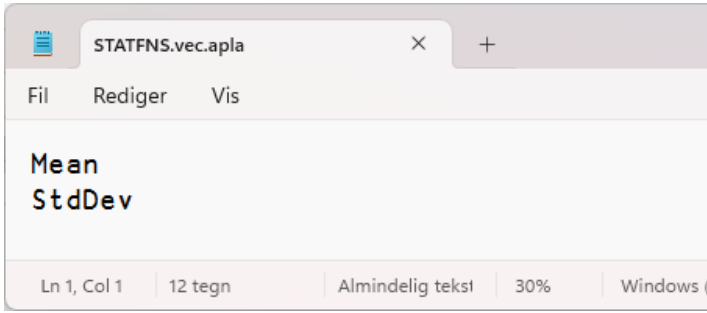


-text=plain|aplan

- The –text switch allows you to select "plain" text formats
  - The default is APL Array Notation: –text=aplan

```
]link.export STATFNS c:\tmp\stats -text=plain
```

- Note that the penultimate segment of the name controls the form of the APL array that will be created from the file.
  - In this case, `.vec.` tells Link that the file should be loaded into a vector of vectors



Text files which are not in APLAN format will have a penultimate "sub-extension" section in the file name which records the format of the original array in the workspace. The below table describes the array file extensions, what the content represents, and the specific criteria for storage in plain text file.

For all plain text types, the array must be non-empty.

File Extension	Array Characteristics	Prohibited characters (⎕UCS)
.CR.apla	Simple vector with each line terminated by `⎕UCS 13`	`10 11 12 133 8232 8233`
.LF.apla	Simple vector with each line terminated by `⎕UCS 10`	`11 12 13 133 8232 8233`
.CRLF.apla	Simple vector with each line terminated by `⎕UCS 13 10`	`11 12 133 8232 8233` *
.vec.apla	Vector of simple character vectors (no scalar elements)	`11 12 13 133 8232 8233`
.mat.apla	Simple character matrix	`10 11 12 13 133 8232 8233`

Array Formats

By default, Link uses *APL Array Notation (APLAN)* to store arrays in text files. While APLAN is a good format for describing numeric data, nested arrays and many high rank arrays, it is not ideal for storing text data. Link 4.0 introduced experimental support for storing multi-line character data in simple text files.

The configuration setting `text` can be used to enable this feature: If `text` is set to `'aplan'` (the default) then all arrays will be store using APLAN. If `text` is set to `'plain'` then text arrays that adhere to a set of very specific criteria will instead be stored in plain text files. You can set this option when a link is created, or using `Link.Configure`.

Text files which are not in APLAN format will have a penultimate "sub-extension" section in the file name which records the format of the original array in the workspace. The below table describes the array file extensions, what the content represents, and the specific criteria for storage in plain text file.

For all plain text types, the array must be non-empty.

The middle segment of the name

- Declares the form of the APL array, **NOT** the file
  - The file will have the separator decided by ⎕NPUT
- Once a file has e.g. `.vec.` in the name, the array format is fixed.
  - You need to delete the file if you want to change th
    - Or rename it and re-create the link
  - **NB:** If you delete the file while watch=both, the variable will be expunged!
- You can set the default for future files

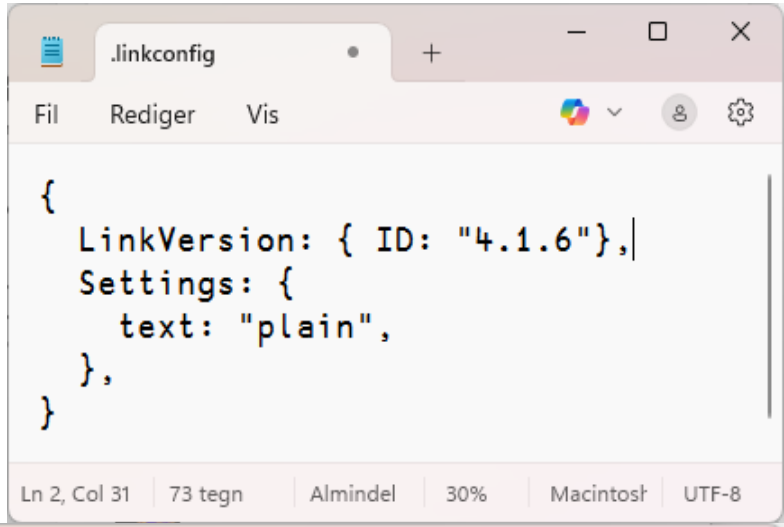
```
]link.configure myns text:plain
```

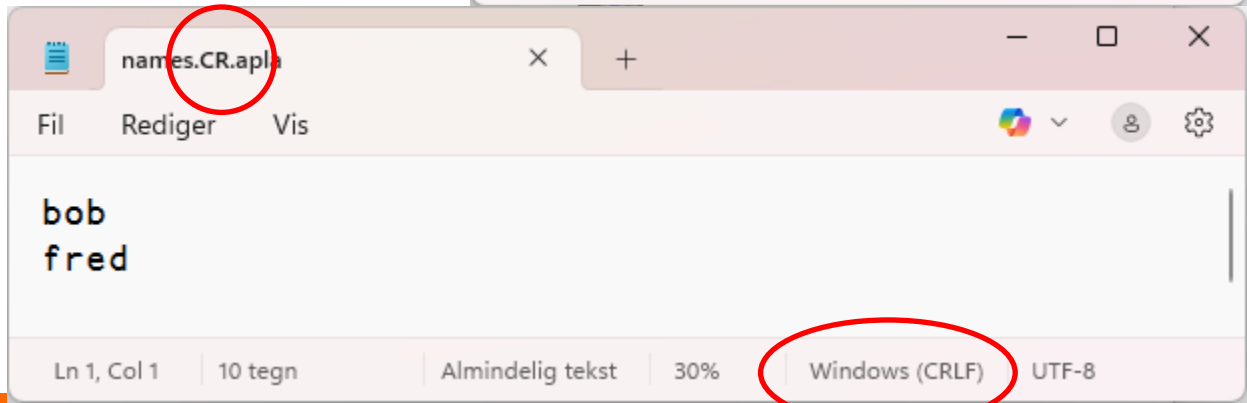
text:plain, continued

Once the default is set to "plain":

```
      NL←⎕UCS 13 ⍝ "Carriage Return"
      names←'bob',NL,'fred',NL

      ]link.add names
Added: #.myns.names
```





Text:plain, continued

If watching the directory, then after saving the edited file:

```
    ↑('.'@(NL∘=)names)(⎕UCS names)
  j   i   l   l   .   b   o   b   .   f   r   e   d   .
106 105 108 108 13 98 111 98 13 102 114 101 100 13
```

Text:plain, continued

Renaming the file to `*.LF.apla` gives a nasty warning:

```
Link Warning: ⎕SE.Link.Notify:
renamed c:/devt/intro-to-link/myns/names.LF.apla:
ignoring attempt to redefine #.myns.names
```

However, if you re-create the Link, note that the line endings are LF's:

```
      ↑('.'@(=∘(⎕UCS 10))names)(⎕UCS names)
  j   i   l   l   .   b   o   b   .   f   r   e   d   .
106 105 108 108 10 98 111 98 10 102 114 101 100 10
```

Final words about "Data"

- As a general rule, Link is not for saving "data"
  - It is for "source code"
- However, this is a grey area
  - Are mortality tables or other assumptions code or data?
  - If they need to be tracked and audited, using text source brings many useful tools to bear
  - One persons data is another ones code

Oh, and ...

- Did I say "text:plain" is not very well tested ...
-]Link.Resync doesn't work with text:plain
  - Hopefully fixed before v20.0 is shipped
- Otherwise, Link 4.1.7 will be in a DSS update (and on GitHub)

Exercise 7 – text:plain

Switch to using `text:plain` in the stats app

- Use `]link.configure` to change the default to `text:plain`
- Verify the contents of the .linkconfig file
- Replace STATFNS.apla with STATFNS.vec.apla
- Verify that the application still works

Boot or Build?

- It is fine (even encouraged!) to dynamically load text source during development
- For small or open-source applications, it is also fine for distribution / runtime if performance is not an issue
- It is NOT recommended to dynamically load source from large numbers of text files in production environments

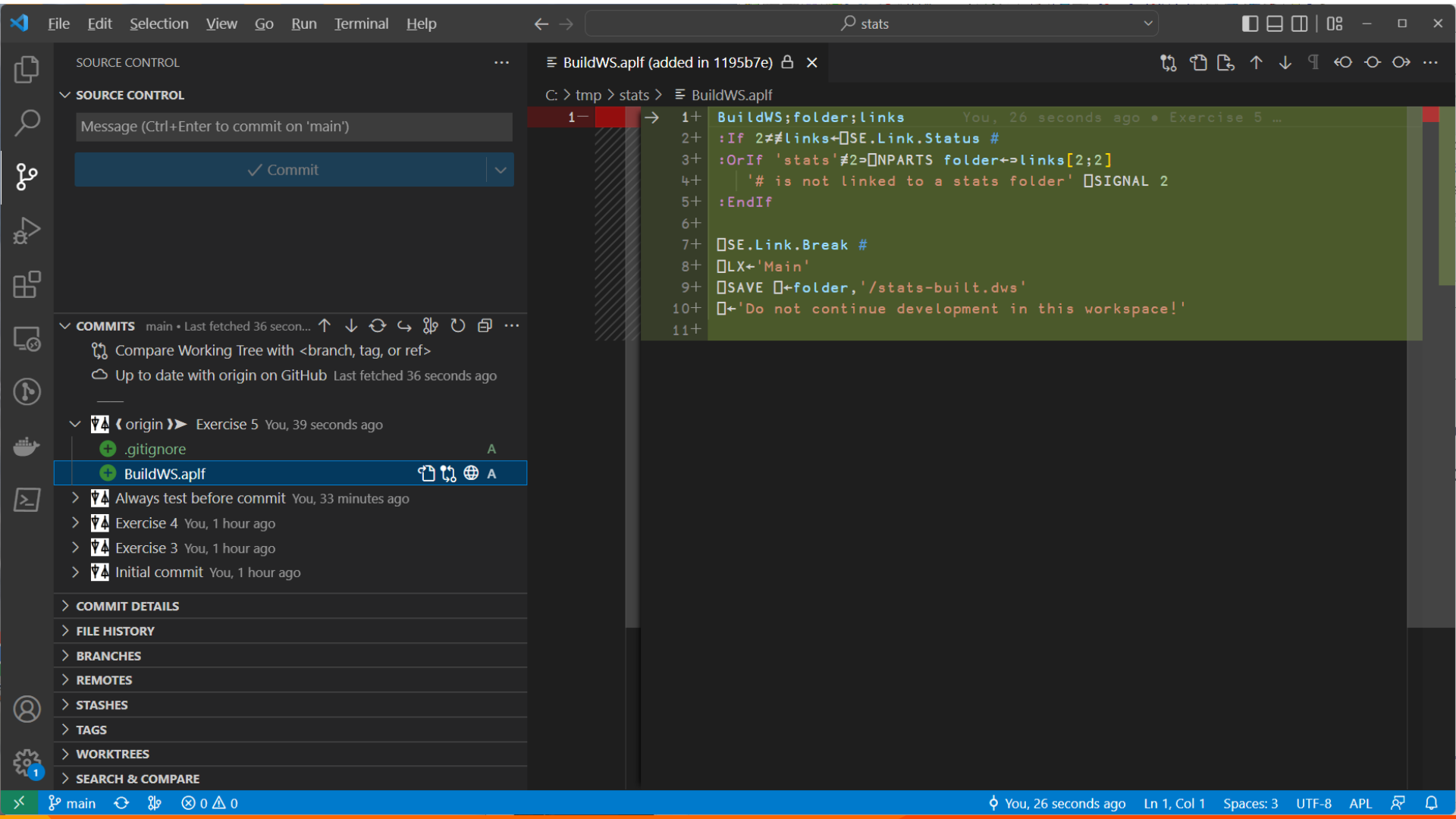
Exercise 8 - Distribution

- Write a "BuildWS" function which
  - Load the source into the workspace
  - If source is already Linked, see `Link.Break`
  - Uses `]SAVE` to create a workspace that will run when loaded

Exercise 8 - Solution

```
      ∇ BuildStats;home;wsid
[1]     :If 0=≢home←1⊃⎕NPARTS 4⊃5179⌶⊃⎕SI
[2]         →0⊣⎕←'Unable to determine source directory!'
[3]     :EndIf
[4]     ⎕SE.Link.Import #(home,'/export')
[5]     ⎕LX←'Main'
[6]     0 ⎕SAVE wsid←home,'/runstats.dws'
[7]     ⎕←'Saved: ',wsid
      ∇
```

```
dyalog.exe LOAD=C:\devt\intro-to-link\BuildStats.aplf
```


File Edit Selection View Go Run Terminal Help
stats
SOURCE CONTROL
SOURCE CONTROL
Message (Ctrl+Enter to commit on 'main')
Commit
COMMITS main • Last fetched 36 secon...
Compare Working Tree with <branch, tag, or ref>
Up to date with origin on GitHub Last fetched 36 seconds ago
origin Exercise 5 You, 39 seconds ago
.gitignore A
BuildWS.aplf A
Always test before commit You, 33 minutes ago
Exercise 4 You, 1 hour ago
Exercise 3 You, 1 hour ago
Initial commit You, 1 hour ago
COMMIT DETAILS
FILE HISTORY
BRANCHES
REMOTES
STASHES
TAGS
WORKTREES
SEARCH & COMPARE
BuildWS.aplf (added in 1195b7e)
C: > tmp > stats > BuildWS.aplf
BuildWS;folder;links You, 26 seconds ago • Exercise 5 …
:If 2≠≠links←⎕SE.Link.Status #
:OrIf 'stats'≠2=⎕NPARTS folder↔links[2;2]
'# is not linked to a stats folder' ⎕SIGNAL 2
:EndIf
⎕SE.Link.Break #
⎕LX←'Main'
⎕SAVE ⎕←folder,'/stats-built.dws'
⎕←'Do not continue development in this workspace!'
main
0 0
You, 26 seconds ago
Ln 1, Col 1
Spaces: 3
UTF-8
APL

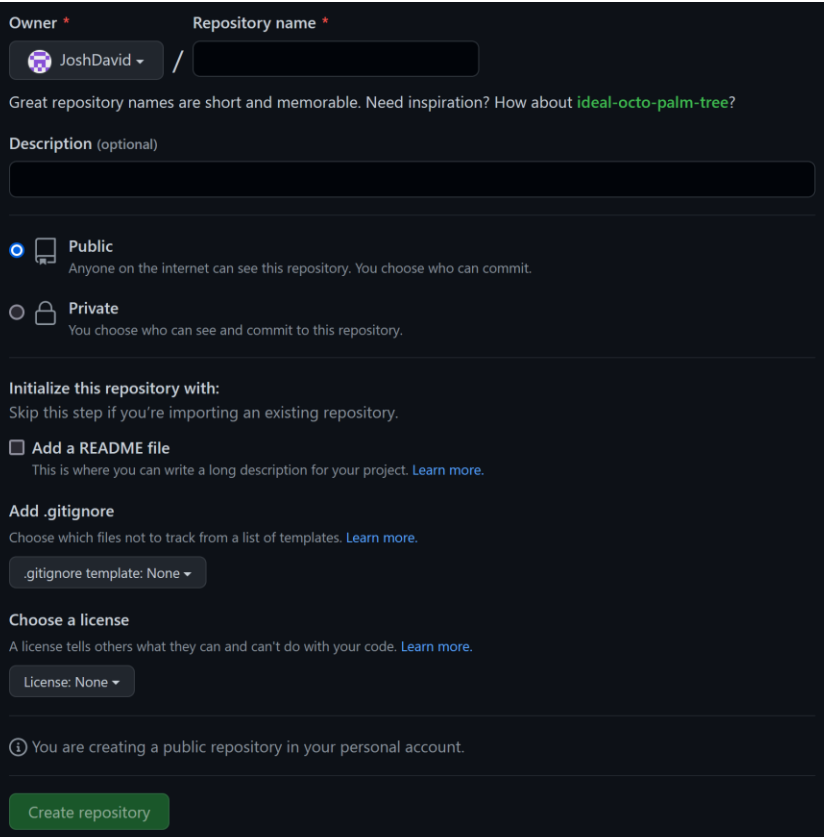
A Day in the Life …

… Of a text-based APL developer

- Create GitHub repo
- Clone it locally
- Populate it
- Make a change, roll back
- Roll forward,
- Make branch, commit, pull request

- Finally, look at METSIM Build & Run
- Talk about -preload

Creating a GitHub Repository



Picking a License

I need to work in a community.

Use the **license preferred by the community** you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to **add a license**.

I want it simple and permissive.

The **MIT License** is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

Babel, **.NET**, and **Rails** use the MIT License.

I care about sharing improvements.

The **GNU GPLv3** also lets people do almost anything they want with your project, *except* distributing closed source versions.

Ansible, **Bash**, and **GIMP** use the GNU GPLv3.

Cloning your repo

- You can set your Git client up to communicate via HTTPS or SSH
- HTTPS is easier to setup, and with Personal Access Tokens (PAT) and 2FA (Two-factor Authentication) it probably satisfies your security needs
- Once set up, try cloning your new "repo", or:

  git clone https://github.com/dialog-training/2022-SA3 /some/folder

Creating a personal access token

You can create a personal access token to use in place of a password with the command line or with the API.

In this article

Notes:

- If you use GitHub CLI to authenticate to GitHub on the command line, you can skip generating a personal access token and authenticate via the web browser instead. For more information about authenticating with GitHub CLI, see `gh auth login`.
- Git Credential Manager is a secure, cross-platform alternative to using personal access tokens (PATs) and eliminates the need to manage PAT scope and expiration. For installation instructions, see Download and install in the GitCredentialManager/git-credential-manager repository.

Personal access tokens (PATs) are an alternative to using passwords for authentication to GitHub when using the GitHub API or the command line.

If you want to use a PAT to access resources owned by an organization that uses SAML SSO, you must authorize the PAT. For more information, see "About authentication with SAML single sign-on" and "Authorizing a personal access token for use with SAML single sign-on" in the GitHub Enterprise Cloud documentation.

As a security precaution, GitHub automatically removes personal access tokens that haven't been used in a year. To provide additional security, we highly recommend adding an expiration to your personal access tokens.

A token with no assigned scopes can only access public information. To use your token to access repositories from the command line, select `repo`. For more information, see "Available scopes".

How Morten Works – Live Demo

- VS Code & Git Lens
- METSIM build

```
    ∇ Build;GITPATH;gitrepos;warn;ai3;dirs;nss;ns;dir;z;type;filename;flags;resource;icon;cmdline;details;DEV;m;P
[1]   DEV←∨/'yY1'∈2 ⎕NQ'.' 'GetEnvironment' 'DEV'
[2]   PULL←∨/'pull'⍷⎕C 2 ⎕NQ'.' 'GetCommandLine'
[3]
[4]   ⎕←'Building METSIM ',((DEV+1)⊃'runtime' 'development'),' workspace...'
[5]   :If 0=≢GITPATH←1⊃⎕NPARTS ¯1↓1⊃1 ⎕NPARTS 4⊃5179⌶1⊃⎕SI
[6]       GITPATH←∈1 ⎕NPARTS{ω,'/'/⍨~(¯1↑ω)∈'/\'}{0=≢ω:'c:\devt\' ⋄ ω}2 ⎕NQ'.' 'GetEnvironment' 'GITPATH'
[7]   :EndIf
[8]   :If ~∧/m←⎕NEXISTS dirs←GITPATH∘,¨'metsim/dyalog' 'AtfIn/APLSource/A2K' 'deltawi/code/DWI'
[9]       ⎕←'Unable to build METSIM® for Dyalog APL, cannot find:'
[10]      ⎕←⍪'      '∘,¨(~m)/dirs
[11]      →0
[12]  :EndIf
[13]  nss←'#' 'A2K' 'dwi'
[14]
[15]  :For (ns dir) :InEach nss dirs
[16]      ai3←⎕AI[3]
[17]      ⎕←'Processing ',dir,'... '
[18]      :If PULL
[19]          ⎕←dir(1⊃pull←⎕SHELL'git' '-C'dir'pull')
[20]      :EndIf
[21]      :If DEV
[22]          z←(fastLoad:1 ⋄ flatten:∨/'metsim'⍷dir)⎕SE.Link.Create ns dir
[23]      :Else
[24]          z←(fastLoad:1 ⋄ flatten:∨/'metsim'⍷dir)⎕SE.Link.Import ns dir
[25]      :EndIf
[26]      ⎕←(1⍕0.001×(3⊃⎕AI)-ai3),' seconds'
[27]  :EndFor
```

```
[29]   ⎕←'Verifying git status...'⊣warn←0
[30]   gitrepos←GitStatus¨GITPATH∘,¨{(¯1+⍵⍳'/')↑⍵}¨(≠GITPATH)↓¨dirs
[31]   ⎕←⍪gitrepos
[32]
[33]   :If ∨/~gitrepos.branch∊'main' 'main...origin/main'
[34]       ⎕←'** Warning: not building exclusively from main branches'⊣warn←1
[35]   :EndIf
[36]   :If ~∧/gitrepos.clean
[37]       ⎕←'** Warning: repos contain uncommited changes: ',(≠GITPATH)↓¨(~gitrepos.clean)/gitrepos.path⊣warn←1
[38]   :EndIf
[39]
[40]   :If 0≠⎕NC'RIDE'←2 ⎕NQ'.' 'GetEnvironment' 'RIDE' ⍝ LEAVE THIS AS GLOBAL for Run to inspect!
[41]       ⎕←'*** NB NB RIDE is set: ',RIDE,' ***'
[42]   :EndIf
[43]
[44]   :If warn
[45]       ⎕←'Proceed (Y/N)?'
[46]       →(∨/'yY'∊⍞)↓0
[47]   :EndIf
[48]
[49]   ⎕←'Copying SQAPL and Conga'
[50]   'Conga'⎕CY'conga'
[51]   'SQA'⎕CY'sqapl'
[52]
[53]   A2K.ΔWI←dwi.WI
[54]   GITREPOS←gitrepos
[55]
[56]   ⎕WSID←'C:\METSIMD\METSIM'
[57]   ⎕LX←'Run'
```

```
[59]   :If DEV
[60]       0 ⎕SAVE ⎕WSID
[61]       ⎕←'Saved: ',⎕WSID
[62]
[63]   :Else ⍝ Build a stand-alone executable
[64]       ⎕RTS←1
[65]       filename←'C:\METSIMD\METSIM.exe'
[66]       type←'StandaloneNativeExe'
[67]       flags←8 ⍝ Runtime
[68]       resource←''
[69]       icon←'c:\METSIMD\METSIM.ico'
[70]       cmdline←'METSIM.exe MAXWS=512M METSIM_FORMSIZE="1000 1600"'
[71]       details←['Comments' 'METSIM® for Dyalog APL'
[72]           'CompanyName' 'MSI Inc.'
[73]           'FileDescription' 'METSIM® executable for Dyalog APL'
[74]           'FileVersion'(↑'Mmm YYYY'(1200⌶)1 ⎕DT'J')
[75]           'ProductVersion'(↑'Mmm YYYY'(1200⌶)1 ⎕DT'J')
[76]           'LegalCopyright'('Copyright MSI Inc. ',⍕1⊃⎕TS)
[77]           'ProductName' 'METSIM® for Dyalog APL']
[78]
[79]       :If 1=2 ⎕NQ'.' 'bind'filename type flags resource icon cmdline details
[80]           ⎕←'Created executable with command line ',cmdline
[81]           :If ⎕NEXISTS buildfile←GITPATH,'metsim/installer/METSIM/METSIM.exe'
[82]               buildfile(⎕NCOPY⍠'IfExists' 'Replace')filename
[83]               ⎕←'Also updated ',buildfile
[84]           :EndIf
[85]       :Else
[86]           ...
[87]       :EndIf
[88]   :EndIf
```

```
    ∇ Run;GITPATH;dirs;nss;z;ns;dir;repo;changed;files;prefix;names;parts;m;file;ext;name;path;build;loadsrc;⎕TRA
[1]   ⍝ Start Dyalog METSIM
[2]    ⎕TRAP←0 'C' '→ERROR'
[3]
[4]    config←ReadConfig
[5]    loadsrc←4⊃5179⌶1⊃⎕SI
[6]
[7]    :If config.DEV ⍝ In developer mode, patch changes since last Build
[8]        ⎕←'Setting up development environment - Patching and Linking'
[9]        :If 0=≢GITPATH←{(¯1+1⍳⍨'metsim/dyalog'⍷⎕C ⍵)↑⍵}∊1 ⎕NPARTS loadsrc
[10]           GITPATH←∊1 ⎕NPARTS{⍵,'/'/⍨~(¯1↑⍵)∊'/\'}{0=≢⍵:'c:\devt\' ⋄ ⍵}config.GITPATH
[11]       :EndIf
[12]       dirs←GITPATH∘,¨'metsim/dyalog' 'AtfIn/APLSource/A2K' 'deltawi/code/DWI'
[13]       nss←'#' 'A2K' 'dwi'
[14]
[15]       :For (ns dir) :InEach nss dirs
[16]           path←GITPATH,{(¯1+⍵⍳'/')↑⍵}(≢GITPATH)↓dir
[17]           build←(GITREPOS.path⍳⊂path)⊃GITREPOS
[18]           ⎕←repo←build.hash GitStatus path
[19]           :If build.branch≠repo.branch
[20]               ⎕←'*** Unable to patch - this workspace was built from branch [',build.branch,'] - rebuild requ
[21]               :Continue
[22]           :EndIf
[23]           saved←13 ⎕NINFO ⎕WSID,'.dws'
[24]           (names times)←0 13 ⎕NINFO⍠('Wildcard' 1)('Recurse' 1)⊢dir,'/*'
[25]           parts←⎕NPARTS(times>saved)/names
[26]           names←(m←((3⊃¨parts)∊⊂'.dyalog')∨(4↑¨3⊃¨parts)∊⊂'.apl')/2⊃¨parts
[27]           ⎕EX names~⊂'Run
```

```
[27]          ⎕EX names~⊂'Run'
[28]          :For (path name ext) :In m/parts
[29]              ⎕←'Patching ',path,name,ext
[30]              :If ext≡'.apla'
[31]                  ns ⎕SE.Link.Fix 1↓⎕NGET(path,name,ext)1
[32]              :Else
[33]                  2(⍎ns).⎕FIX'file://',path,name,ext
[34]              :EndIf
[35]          :EndFor
[36]          z←(fastLoad:1 ⋄ preloaded:1 ⋄ flatten:v/'metsim'⍷dir)⎕SE.Link.Create ns dir
[37]      :EndFor
[38]      :If v/m←~⎕NEXISTS ⎕SE.Link.Links.dir
[39]          ⎕←'*** WARNING: source folder(s) not found: ',⍕m/⎕SE.Link.Links.dir
[40]      :EndIf
[41]      :If 0=≢loadsrc
[42]          ⎕←'*** WARNING: this is not a development workspace - no file links ***'
[43]      :EndIf
[44]      ⎕←'Config:'
[45]      ⎕JSON⍠'Compact' 0⊢config
[46]  :ElseIf 0≠≢loadsrc
[47]      ⎕←'*** WARNING: development workspace - file links exist ***'
[48]  :EndIf
```

```
[50]    A2K.ΔWI←dwi.WI
[51]    dwi.Init #
[52]    dwi.DEBUG←~config.TRAP_DWI_ERRORS
[53]
[54]    ⎕WSID←'C:\METSIMD\METSIM'
[55]    A2K.DEBUG_INTERRUPTS←'#.DEBUG_INTERRUPTS'
[56]    A2K.DISPLAY_ERRORS←config.DISPLAY_TRAPPED_ERRORS/'#.DISPLAY_ERRORS'
[57]    ΔDEBUG←isRuntime
[58]    silent←∨/'1yY'∈2 ⎕NQ'.' 'GetEnvironment' 'SIL'
[59]
[60]    :If config.DEV∧~isRuntime∨silent
[61]        ⎕←'      LOAD'
[62]    :Else
[63]        :If 0≠≢RIDE
[64]            3502⌶RIDE ⋄ 3502⌶1
[65]            Wmsg'RIDE enabled - disable before shipping!' 48
[66]        :EndIf
[67]        LOAD
[68]        ⎕OFF
[69]    :EndIf
[70]    →0
[71]
[72]  ERROR:
[73]    ⎕TRAP←0 'S'
[74]    'EF'⎕WC'Form' 'METSIM® Startup Error'('Size' 300 1000)('Coord' 'Pixel')
[75]    'EF.TXT'⎕WC'Text'(⊂⍕⍪⎕DMX.DM)(10 10)('Font' 'APL385 Unicode')
[76]    ⎕DQ'EF'
      ∇
```