

# DYALOG

Glasgow 2024

## Link and the Basics of APL Source in Text Files



Morten Kromberg



Adám Brudzewsky

# Materials

Copy the folder 2024-TP3 from the USB drive

- ◆ One workspace (stats.dws)
- ◆ One Cider project (ReadMail folder)
- ◆ One Powerpoint presentation
- ◆ One zip file containing Link 4.0.20

If you cannot use USB, unzip the latest release from

- ◆ <https://github.com/dyalog-training/2024-TP3/releases/tag/v1.0.0>
- ◆ NB: Exercises assume Link v4.0.20 or later



# Goals




- Give an introduction to Link
- Give you time to experiment with Link
- Focus on the process of moving source from a workspace to text files
  - And rebuilding the runtime environment
- If we have time: Using Cider to manage Tatin and NuGet Packages



Releases / v1.0.0




# Glasgow TP3 release Latest

Compare  

 mkromberg released this 1 minute ago  v1.0.0  ad91cf8

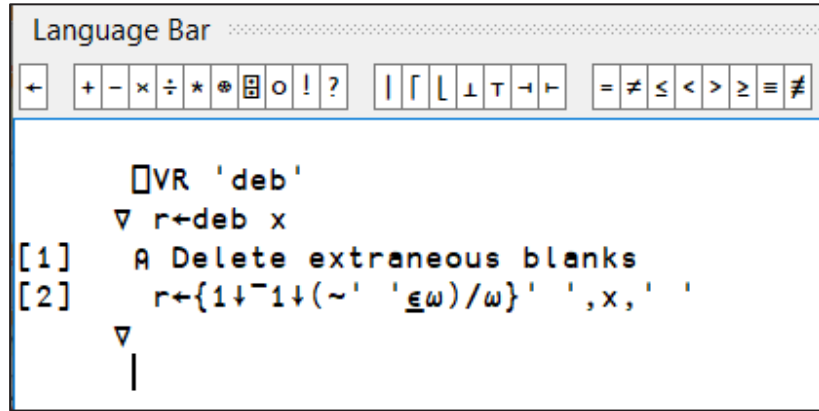
Materials as presented in TP3 workshop at Dyalog'24 in Glasgow.

## Assets 3

 <a href="#">TP3.pptx</a>	8.66 MB	1 minute ago
 <a href="#">Source code (zip)</a>		6 minutes ago
 <a href="#">Source code (tar.gz)</a>		6 minutes ago

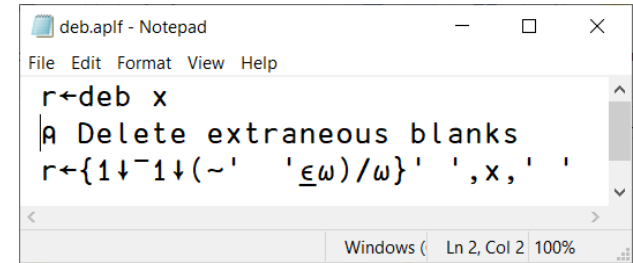


# What exactly is Link?



Language Bar

```
VR 'deb'  
▽ r←deb x  
[1] A Delete extraneous blanks  
[2] r←{1↓~1↓(~' '⊥ω)/ω}' ',x,' '  
▽  
|
```



deb.aplf - Notepad

```
r←deb x  
A Delete extraneous blanks  
r←{1↓~1↓(~' '⊥ω)/ω}' ',x,' '  
Windows ( Ln 2, Col 2 100%
```

- Each code item in the active workspace is **linked** to a file
  - Functions, Operators, Classes and "Scripted" Namespaces – and optionally variables
  - "Unscripted" Namespaces (namespaces with no source text) map to directories
- If the item is edited using the APL editor, the file is updated
- If the file is changed, the workspace is updated

# ”Old Timers”: Link replaces SALT

- Link replaces SALT
  - (SALT will be available until no longer used)
  - Eventually, I hope that most of Link will also disappear and be replaced by functionality in the interpreter
- With Link...
  - The interpreter is tracking the relationships between objects and files
  - A *File System Watcher* responds to external changes (requires .NET, supported under Windows, Mac & Linux)



# About the File System Watcher

- ◆ Runs a callback function in APL each time a file changes in a linked folder
- ◆ Designed to capture changes made in an external editor
- ◆ **Should** be able to handle "small" git actions
  - ◆ Code still being improved
- ◆ Not appropriate for handling "bulk" changes, such as
  - ◆ Unzipping lots of files into a watched folder
  - ◆ Doing a large checkout/revert
  - ◆ Network drives
- ◆ Requires .NET - not available under AIX

# Why is Link **IMPORTANT** ?

- 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** ?



GITLENS

REPOSITORIES

- Link master 251 0t • ~4 -1 • Last fetched 9:29pm...
- master 251 0t ⇌ 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

```

c: > Devt > Link > StartupSession > Link > FileSystemWatcher.apln
27 r+LJNEW Disposable watcher A wrap a de: 27 r+LJNEW Disposable watcher A wrap
28 r.[DF 1φ'] [' , ##.U.WinSlash>args 28 r.[DF 1φ'] [' , ##.U.WinSlash>args
29 29
30 :Class Disposable 30 :Class Disposable
31 :Field Public Object 31 :Field Public Object
32 32
33 :Access Public 33 :Access Public
34 :Implements Constructor 34 :Implements Constructor
35 Object+ref 35 Object+ref
36 36
37 :do_dispose ref 37 :do_dispose ref
38 :Trap 0 38 :Trap 0
39 | ref.Dispose 39 | ref.Dispose
40 :EndTrap 40 :EndTrap
41 41
42 - ▾ dispose;tid 42 + ▾ dispose You, 2 months a
43 :Implements Destructor 43 :Implements Destructor
44 Object.EnableRaisingEvents+0 44 Object.EnableRaisingEvents+0
45 - tid+do_dispose&Object 45 + do_dispose Object
46 46
47 :EndClass 47 :EndClass
48 :EndNamespace 48 :EndNamespace
49 49

```

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

Browse files

master

mkromberg committed on Jun 19

1 parent 767c3d0 commit 2e934c527ce73dd77de79477fcb...2e934c527ce73dd77de79477fcb...2e934c527ce73dd77de79477fcb...2e934c527ce73dd77de79477fcb...2e934c527ce73dd77de79477fcb...

Showing 1 changed file with 2 additions and 2 deletions.

Unified Split

4 StartupSession/Link/FileSystemWatcher.apln

```
@@ -39,10 +39,10 @@
39         ref.Dispose
40         :EndTrap
41         ▾
42 -     ▾ dispose;tid
43         :Implements Destructor
44         Object.EnableRaisingEvents<0
45 -     ▾ tid<do_dispose&Object
46         ▾
47         :EndClass
48         :EndNamespace
```

```
39         ref.Dispose
40         :EndTrap
41         ▾
42 +     ▾ dispose
43         :Implements Destructor
44         Object.EnableRaisingEvents<0
45 +     do_dispose Object
46         ▾
47         :EndClass
48         :EndNamespace
```

# Other Benefits of Text Source

- ◆ Easily share code between APL versions
  - ◆ Text files are backwards **and** forwards compatible
- ◆ Laugh at syserrors

# Drawbacks of Text Source

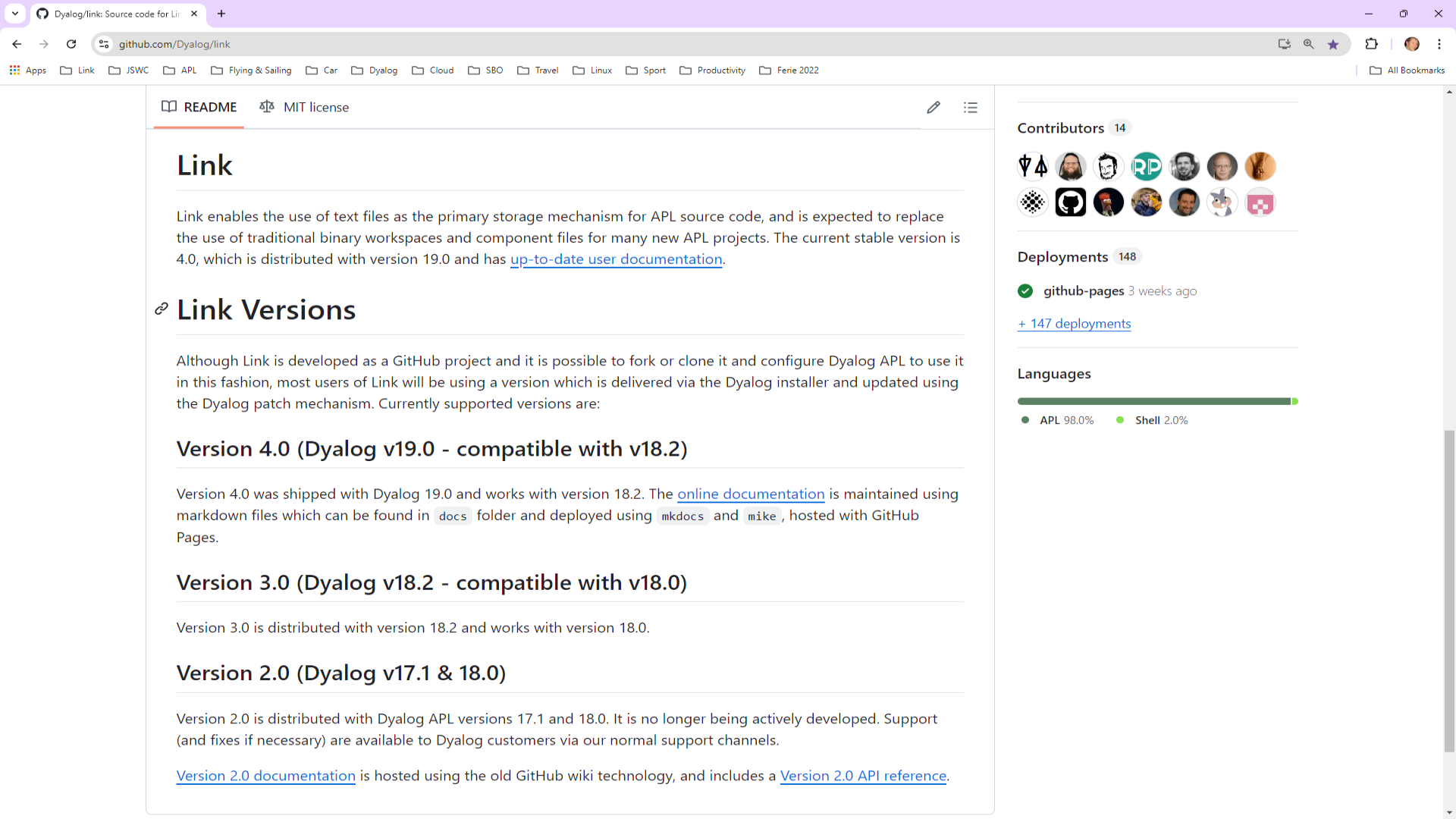
- ◆ This space intentionally left blank

# Link in 2024 – Version 4.0

- ◆ Link 4.0 was shipped with 19.0
  - ◆ Also works with 18.2
  - ◆ Link 3.0 was shipped with 18.2 and works with 18.0
  - ◆ Link 2.0 for 18.0 and 17.1
- ◆ Rapidly growing user base
- ◆ Decent documentation
- ◆ ”Mature”

# Link is a GitHub Project

- <https://github.com/dyalog/link>



README MIT license

# Link

Link enables the use of text files as the primary storage mechanism for APL source code, and is expected to replace the use of traditional binary workspaces and component files for many new APL projects. The current stable version is 4.0, which is distributed with version 19.0 and has [up-to-date user documentation](#).

## Link Versions

Although Link is developed as a GitHub project and it is possible to fork or clone it and configure Dyalog APL to use it in this fashion, most users of Link will be using a version which is delivered via the Dyalog installer and updated using the Dyalog patch mechanism. Currently supported versions are:

### Version 4.0 (Dyalog v19.0 - compatible with v18.2)

Version 4.0 was shipped with Dyalog 19.0 and works with version 18.2. The [online documentation](#) is maintained using markdown files which can be found in `docs` folder and deployed using `mkdocs` and `mike`, hosted with GitHub Pages.

### Version 3.0 (Dyalog v18.2 - compatible with v18.0)

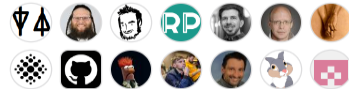
Version 3.0 is distributed with version 18.2 and works with version 18.0.

### Version 2.0 (Dyalog v17.1 & 18.0)

Version 2.0 is distributed with Dyalog APL versions 17.1 and 18.0. It is no longer being actively developed. Support (and fixes if necessary) are available to Dyalog customers via our normal support channels.

[Version 2.0 documentation](#) is hosted using the old GitHub wiki technology, and includes a [Version 2.0 API reference](#).

Contributors 14

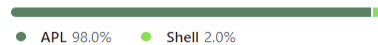


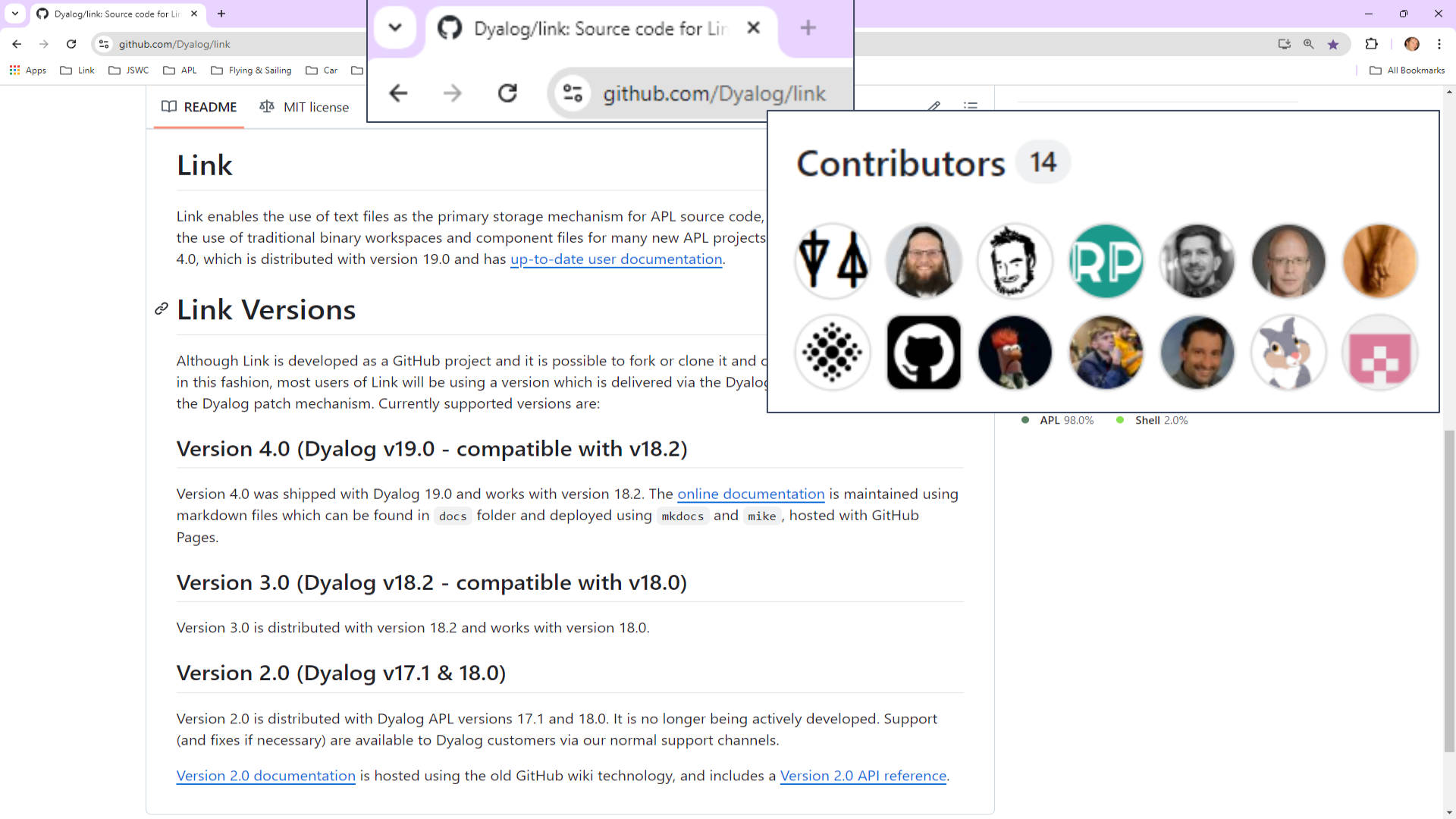
Deployments 148

github-pages 3 weeks ago

[+ 147 deployments](#)

Languages





## Link

Link enables the use of text files as the primary storage mechanism for APL source code, the use of traditional binary workspaces and component files for many new APL projects 4.0, which is distributed with version 19.0 and has [up-to-date user documentation](#).

## Link Versions

Although Link is developed as a GitHub project and it is possible to fork or clone it and use it in this fashion, most users of Link will be using a version which is delivered via the Dyalog patch mechanism. Currently supported versions are:

### Version 4.0 (Dyalog v19.0 - compatible with v18.2)

Version 4.0 was shipped with Dyalog 19.0 and works with version 18.2. The [online documentation](#) is maintained using markdown files which can be found in `docs` folder and deployed using `mkdocs` and `mike`, hosted with GitHub Pages.

### Version 3.0 (Dyalog v18.2 - compatible with v18.0)

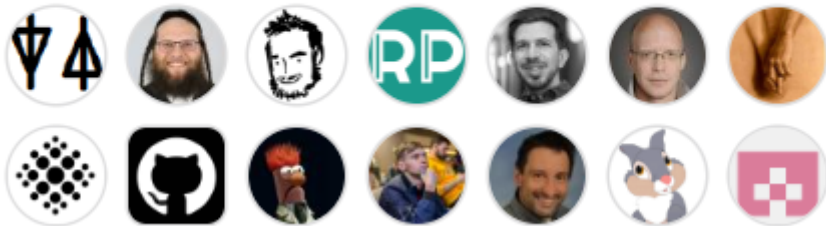
Version 3.0 is distributed with version 18.2 and works with version 18.0.

### Version 2.0 (Dyalog v17.1 & 18.0)

Version 2.0 is distributed with Dyalog APL versions 17.1 and 18.0. It is no longer being actively developed. Support (and fixes if necessary) are available to Dyalog customers via our normal support channels.

[Version 2.0 documentation](#) is hosted using the old GitHub wiki technology, and includes a [Version 2.0 API reference](#).

## Contributors 14



● APL 98.0% ● Shell 2.0%



Filters is:issue is:open

Labels 15

Milestones 2

New issue

84 Open 404 Closed

Author Label Projects Milestones Assignee Sort

Saving an edited APL object resulted in FILE ACCESS ERROR bug #667 opened 3 weeks ago by aplteam 1

Link does not set se.Link.NOTIFY to 1 even though I want it on in my .linkconfig file in the Documents folder bug #663 opened on Jul 24 by dyavc

Empty folders should become empty namespaces bug #658 opened on Jul 5 by aplteam

[Docs]: Description of code extensions in Link.Create docs enhancement #656 opened on Jun 26 by sloorush

An invalid entry in .linkconfig prevents Link from establishing a Link, but it does not add "Error" to the message bug #655 opened on Jun 18 by aplteam

Silent fail to write file if filename is illegal bug #653 opened on Jun 4 by abrudz

The help for ]Link.configure is not good enough bug

# Link is a GitHub Project

- ◆ <https://github.com/dyalog/link>
- ◆ You \*can\* install Link yourself
  - ◆ This was important up to 3.0, when there were regular significant fixes
  - ◆ Still relevant if you want Link 4.0 with Dyalog 18.2
  - ◆ The day after writing the above, I found a couple of bugs related to configuration files, so still relevant 😞
- ◆ We will install Link 4.0.20

# Version 4.0.20 #670

Edit <> Code

Merged mkromberg merged 9 commits into master from fix-654 last week

Conversation 1 Commits 9 Checks 0 Files changed 6 +17 -13

**mkromberg** commented last week Member

Fix #654 "Links not restored in v19.0"  
 Fix #668 "# is not a linked namespace"  
 Fix #669 "Length error in Jlink.configure if stop or trace flags set"  
 Plus a couple of documentation tweaks

Reviewers  
No reviews

Assignees  
No one—assign yourself

Labels  
None yet

Projects  
None yet

Milestone  
No milestone

mkromberg added 9 commits last month

- fa2c441 Add missing "options" argument in calls to SetFlags
- 6abc3c5 Increment version to 4.0.19
- 6607321 Fix #654 - Links not restored in v19.0

## Fix #668 - # is not a linked namespace

[Browse files](#)

master (#670)

v4.0.20

mkromberg committed last week

1 parent 01c0579 commit 385123f

Showing 1 changed file with 1 addition and 1 deletion.

[Whitespace](#)[Ignore whitespace](#)[Split](#)[Unified](#)

StartupSession/Link/Configure.aplf

@@ -15,7 +15,7 @@

```
15     islinked<0
16     container<->[]RSI []XSI U.ContainerNs 0
17
18 -     :If 9=container.[]NC target
19         target<-[]container []target
20         :AndIf 0≠[]links<U.GetLinks
21         :AndIf islinked<(ctarget)∈links.ns           @ If target is a
linked namespace
```

```
15     islinked<0
16     container<->[]RSI []XSI U.ContainerNs 0
17
18 +     :If (9=container.[]NC target)∨(ctarget)∈["'#' '[]SE']
19         target<-[]container []target
20         :AndIf 0≠[]links<U.GetLinks
21         :AndIf islinked<(ctarget)∈links.ns           @ If target is a
linked namespace
```

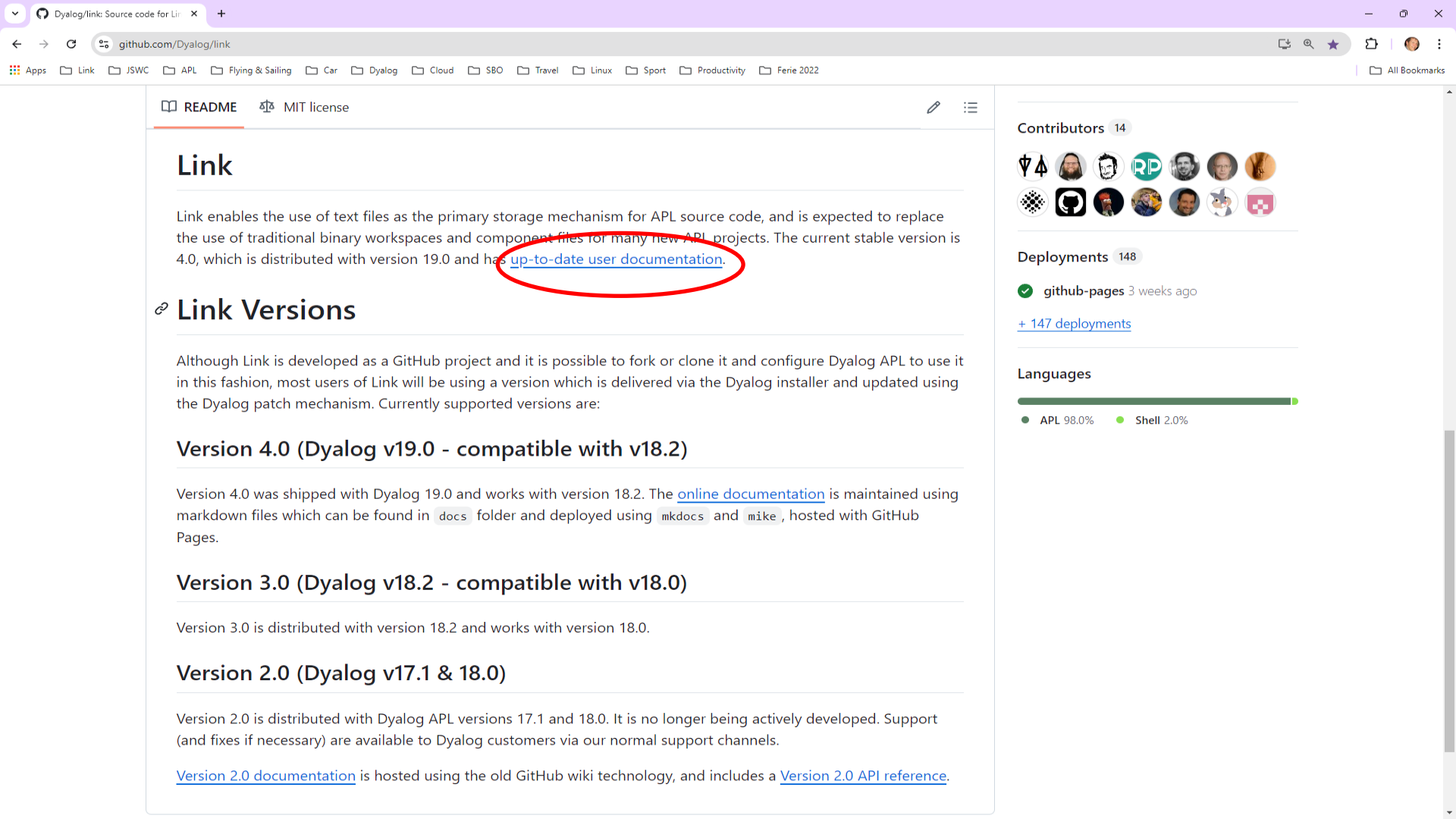
# Verify Link Version

- Start APL, and inspect:

```
□SE.Link.Version  
4.0.20
```

- If you do not have version 4.0.20, follow the instructions at

<https://dyalog.github.io/link/4.0/Usage/Installation/>



README MIT license

# Link

Link enables the use of text files as the primary storage mechanism for APL source code, and is expected to replace the use of traditional binary workspaces and component files for many new APL projects. The current stable version is 4.0, which is distributed with version 19.0 and has [up-to-date user documentation](#).

## Link Versions

Although Link is developed as a GitHub project and it is possible to fork or clone it and configure Dyalog APL to use it in this fashion, most users of Link will be using a version which is delivered via the Dyalog installer and updated using the Dyalog patch mechanism. Currently supported versions are:

### Version 4.0 (Dyalog v19.0 - compatible with v18.2)

Version 4.0 was shipped with Dyalog 19.0 and works with version 18.2. The [online documentation](#) is maintained using markdown files which can be found in `docs` folder and deployed using `mkdocs` and `mike`, hosted with GitHub Pages.

### Version 3.0 (Dyalog v18.2 - compatible with v18.0)

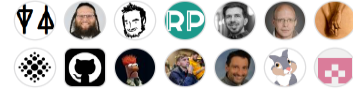
Version 3.0 is distributed with version 18.2 and works with version 18.0.

### Version 2.0 (Dyalog v17.1 & 18.0)

Version 2.0 is distributed with Dyalog APL versions 17.1 and 18.0. It is no longer being actively developed. Support (and fixes if necessary) are available to Dyalog customers via our normal support channels.

[Version 2.0 documentation](#) is hosted using the old GitHub wiki technology, and includes a [Version 2.0 API reference](#).

Contributors 14

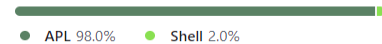


Deployments 148

github-pages 3 weeks ago

[+ 147 deployments](#)

Languages



## Link User Guide

### Overview

#### Introduction

Technical Details and Limitations

Workspaces

History of source files as text in Dyalog

### Install and Upgrade

Installation

Version 4.0 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

Link.CaseCode

Link.Create

Link.Configure

Link.Export

Link.Expunge

# 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 a pre-release of the *literal 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.

## Table of contents

- Link is NOT...
- Link fundamentals
- Functions vs. User Commands
  - User commands
  - API functions
- Further reading
- Frequently Asked Questions

# Starting a New Project

- ◆ To start a new project

```
    )ns myns
    ]link.create myns /my/dir
```
- ◆ At least one of `myns` or `/my/dir` must exist
- ◆ Only one of `myns` or `/my/dir` may be populated



- Link User Guide
  - Overview
    - Introduction
    - Technical Details and Limitations
    - Workspaces
    - History of source files as text in Dyalog
  - Install and Upgrade
    - Installation
    - Version 4.0 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
    - Link.CaseCode
    - Link.Create
    - Link.Configure
    - Link.Export
    - Link.Expunge
    - Link.Fix

## 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 ⍉ A Create an empty namespace
stats.⎓FX 'mean+Mean vals;sum' 'sum++f,vals' 'mean+sum÷1fρ,vals'
stats.Root+{α-2 ⍊ ω÷α}
stats.StdDev+{2 Root+{.×÷÷p},ω-Mean ω}
```

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
```

### 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



## Link User Guide

### Overview

- Introduction
- Technical Details and Limitations
- Workspaces
- History of source files as text in Dyalog

### Install and Upgrade

- Installation
- Version 4.0 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
- Link.CaseCode
- Link.Create
- Link.Configure
- Link.Export
- Link.Expunge

# 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

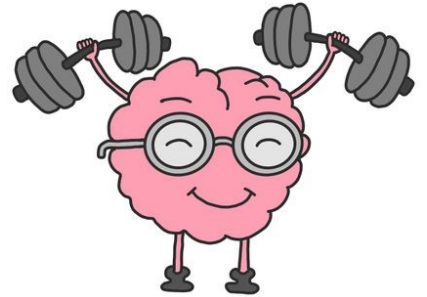
If you are using Dyalog version 18.2 or later, you can cause a link to be created to the root of your workspace (`#`) as APL starts, by setting the `LOAD` parameter on the command line or as an environment variable. After establishing the link, the system will call the function `Run` with a right argument 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).

### 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

# Exercise 1

- Read the general guidelines on the use of Link API Functions and User Commands at <https://dyalog.github.io/link/4.0/API/>
- Create an empty namespace
- Create a link to a directory which does not already exist
- )ED a function in the namespace
- Verify that a source file is created in the directory
- Edit the 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



# Variables

## Link User Guide

### Overview

Introduction

Technical Details and Limitations

Workspaces

History of source files as text in Dyalog

### Install and Upgrade

Installation

Version 4.0 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 consider 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 introduces 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 `.apl a` 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

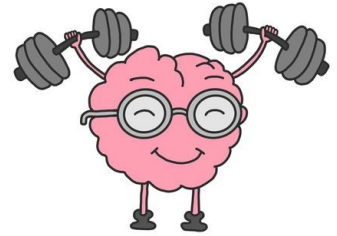
Changes made outside the Editor

### Arrays

Setting up Development and Runtime Environments

# Exercise 2

- ◆ Create a variable containing a constant that your application needs
- ◆ Cause it to be written to a file using `]Link.Add`
- ◆ Inspect the file, edit it, and verify that the new value appears in the workspace
- ◆ Can you write system variables to file?



## Link User Guide

### Overview

- Introduction
- Technical Details and Limitations
- Workspaces
- History of source files as text in Dyalog

### Install and Upgrade

- Installation
- Version 4.0 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

# 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 0  
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

## Table of contents

- Options
  - arrays
  - sysVars
- Workspaces containing Namespaces
- Flat Workspaces and the -flatten Switch
- Recreating the Workspace

Converting an Existing Workspace to use Link

DYALOG Link User Guide 4.0

Search

Dyalog/Link v4.0.17 19 10

Link User Guide

Overview

Introduction

Technical Details

Limitations

Workspaces

History of source files

Dyalog

Install and Upgrade

Installation

Version 4.0 Release Notes

Working with Link

Basic Usage

Array Formats

Configuration Files

Setting Up Your Environment

Converting an Existing Workspace to use Link

API & Command Reference

API Overview

Link.Add

Table of contents

Options

-arrays

```
'options' NS @
options.(arrays sysVars)+1
options SE.Link.Export # '/folder/name'
```

or equivalently, using the user command:

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



Converting an Existing Workspace

dyalog.github.io/link/4.0/Usage/WStoLink/

Apps Link JSWC APL Flying & Sailing Car Dyalog Cloud SBO Travel Linux Sport Productivity Ferie 2022

**DYALOG** Converting an Existing Workspace to use Link

Search

Dyalog/Link v4.0.17 ☆19 🗨10

### Link User Guide

- Overview**
  - Introduction
  - Technical Details and Limitations
  - Workspaces
  - History of source files as text in Dyalog
- Install and Upgrade**
  - Installation
  - Version 4.0 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

## 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](#).

Workspaces containing Namespaces

### Table of contents

- Options**
  - arrays
  - sysVars
- Workspaces containing Namespaces
- Flat Workspaces and the -flatten Switch
- Recreating the Workspace

# ]Link.Add

- Link updates source files when you "fix" in the Editor
- Link will NOT react to changes to source made via other mechanisms, e.g.
  - `life←42 ◇ dup←{ωω}`
  - `)copy dfns cmpx`
  - `□FX 'r←dup x' 'r←x x'`
- You can (must) use `]link.add` to notify Link that source files should be updated
- When writing tools that modify source code, use `□SE.Link.Fix` in place of `□FX`.

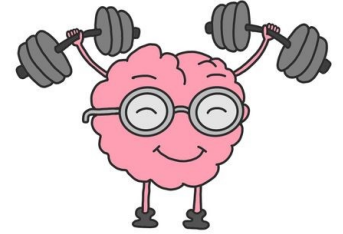
Changes made to source files will always be actioned

*\*IF\** you are using the file system watcher

# Namespace $\leftarrow \rightarrow$ Folder

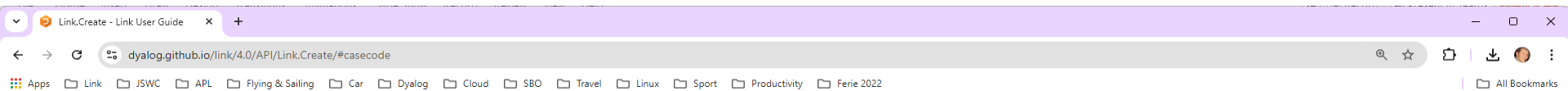
- ◆ Each function, operator or array is linked to a file
  - ◆ "Scripted" namespaces and classes also link to one file per object
- ◆ Each "Unscripted" (aka "Regular") namespace links to a directory
- ◆ If an exported namespace contains sub-namespaces
  - ◆ Each one becomes a sub-directory
- ◆ If an imported directory contains subdirectories
  - ◆ Each one becomes a namespace

# Exercise 2.5



- Create a subdirectory in your source
- 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

# caseCode



Link.Create

Search



Dyalog/Link

v4.0.17 ☆ 19 🗲 10

## Link User Guide

### Overview

Introduction

Technical Details and Limitations

Workspaces

History of source files as text in Dyalog

### Install and Upgrade

Installation

Version 4.0 Release Notes

### Working with Link

Basic Usage

Array Formats

Configuration Files

Setting Up Your Application

## 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**.

## Table of contents

Syntax

Arguments

Result

Common options

source

watch

arrays

sysVars

forceExtensions

forceFileNames

Advanced Options

ignoreconfig

flatten

**caseCode**

beforeWrite

beforeRead

C:\devt\2024-TP3\stats.dws - Dyalog APL/W-64

File Edit View Window Session Log Action Options Tools Threads Help

WS Object Tool Edit Session APL385 Unicode 16

Language Bar

```

Dyalog APL/W-64 Version 19.0.49951
Serial number: 000013
Wed Sep  4 07:13:53 2024
    )load C:\devt\2024-TP3\stats.dws
C:\devt\2024-TP3\stats.dws A saved Fri Oct  7 15:13:42 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

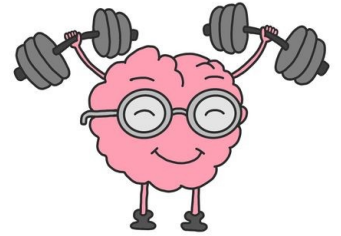
Ready... Ins

CurObj: Main (Function) &:1 [DQ:0 [TRAP [SI:0 [IO:1 [ML:3

Editor

# Exercise 3

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



# Exercise 3 - Discussion

- Use `-caseCode` or `rename`?
- Which variables should be considered "source"?
- `□ML=3:`
  - `-sysvars`
  - `]link.add □ML`
  - Rewrite?



# Exercise 3 – Morten's Solution

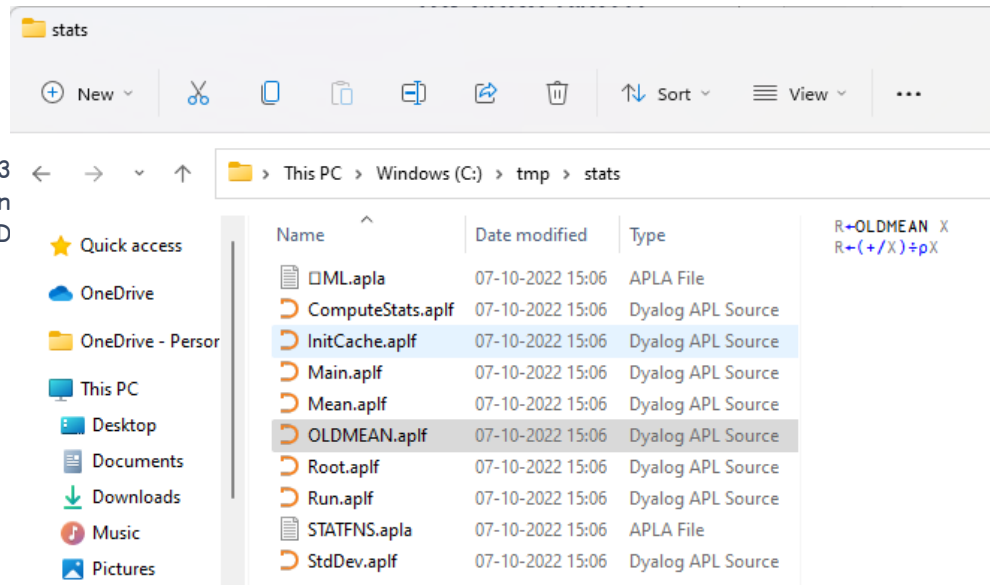
```
)load c:\devt\2024-TP3\stats
c:\devt\2024-SA3\stats.dws A saved Tue Oct 4 22:59:08 2022
)fn
ComputeStats InitCache MEAN Main Mean Root Run StdDev
)ed MEAN A rename to OLDMEAN
)erase MEAN
)vars
RESULTS STATFNS
=>RESULTS
1 2 3          1 2 3 4          2 4 3 1
Mean    2          Mean    2.5          Mean    2.5
StdDev  0.8164965809 StdDev  1.118033989 StdDev  1.118033989
STATFNS
Mean StdDev

]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
```

# Exercise 3 – Morten's Solution

```
)load c:\devt\2024-TP3\stats
c:\devt\2024-SA3\stats.dws A saved Tue Oct 4 22:59:08 2022
)fn
ComputeStats InitCache MEAN Main Mean Root Run StdDev
)ed MEAN A rename to OLDMEAN
)erase MEAN
)vars
RESULTS STATFNS
=>RESULTS
1 2 3          1 2 3 4          2 4 3
Mean      2          Mean      2.5          Mean
StdDev    0.8164965809  StdDev    1.118033989  StdDev
STATFNS
Mean StdDev

]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
```



# Non-Representable Objects

- ◆ Some objects that CAN be saved in a Dyalog workspace have no meaningful textual representation
  - ◆ GUI & COM objects
- ◆ Saving such "binary" objects is a bad idea
  - ◆ They cannot be transferred between 32/64 or classic/unicode
- ◆ You must write code which creates these objects at run or build time

# Create an OLE Server...

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

```
▽ R←MakeOLEServer;ns;spec
[1]  A 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←c'This function returns the mean' 'VT_VARIANT'
[5]  spec,←c'InputNumbers' 'VT_VARIANT'
[6]  ns.SetFnInfo'Mean'spec
[7]  R←0
▽
```

# The -flat switch

- Even if your "legacy" workspace is "flat"...
- It may contain modules that can benefit from being organised into separate directories
- The -flat allows you to load code organised into directories into a flat workspace
- If you edit a function, the editor and Link know from whence it came



## Link User Guide

### Overview

- Introduction
- Technical Details and Limitations
- Workspaces
- History of source files as text in Dyalog

### Install and Upgrade

- Installation
- Version 4.0 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

## 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**.

## Table of contents

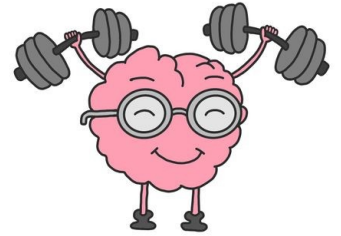
- Syntax
- Arguments
- Result
- Common options
  - source
  - watch
  - arrays
  - sysVars
  - forceExtensions
  - forceFileNames
- Advanced Options
  - ignoreconfig
  - flatten**
  - caseCode
  - beforeWrite
  - beforeRead
  - getFilename
  - codeExtensions

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).

# Exercise 4

- Move the source files for the statistical functions (Mean, StdDev, Root) to a sub-directory called "statfns"
- Get the application to run again





# Exercise 4 - Discussion

- ◆ - flatten or refactor?

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

COMMITTS

COMMIT DETAILS

FILE HISTORY

BRANCHES

REMOTES

STASHES

TAGS

WORKTREES

SEARCH & COMPARE

ComputeStats.aplf (Working Tree) M X

```

1 r←ComputeStats data;⍋CT;fn;keys;results
2 ⍋CT←0 A for key lookup
3
4 (keys results)←RESULTS
5 :If (#keys)≥i←keys⊆data A Cache Hit
6     r←i+results
7     ⍋←' (cached) '
8 :Else
9     r←0 2p0
10    :For fn :In STATFNS
11-   r;←fn(⊆fn,' data')
12    :EndFor
13    ⍋←' (computed) '
14    RESULTS←(keys,=data)(results,=r)
15 :EndIf
16

```

```

1 r←ComputeStats data;⍋CT;fn;keys;results
2 ⍋CT←0 A for key lookup
3
4 (keys results)←RESULTS
5 :If (#keys)≥i←keys⊆data A Cache Hit
6     r←i+results
7     ⍋←' (cached) '
8 :Else
9     r←0 2p0
10    :For fn :In STATFNS
11+   r;←fn((statfns⊆fn) data)
12    :EndFor
13    ⍋←' (computed) '
14    RESULTS←(keys,=data)(results,=r)
15 :EndIf
16

```

# Highlights of Link 4.0

- ◆ Configuration files (user or folder specific)
  - ◆ Also used to record Stop and Trace settings
- ◆ Default to current namespace as source or target (Create, Import, Export)
- ◆ Search Installed Library folders
- ◆ Multi-Line Character Data
- ◆ Transfer file timestamps to workspace
- ◆ Create Link from single Class or NS

# Configuration Files

```
]link.configure # watch:dir  
Was watch:  
  ]link.stop dup 1  
Was dup 0
```

```
]link.configure c:\tmp\dup  
Contents of "c:\tmp\dup/.linkconfig":  
Settings : watch:dir  
Stop/Trace:  
  dup[1/]
```

ns or dir

\* = User

```
]link.configure *  
No configuration options set in "C:\Users\mkrom\Documents\.linkconfig"
```



The screenshot shows a code editor window titled ".linkconfig". The editor displays the following JSON configuration:

```
{  
  LinkVersion: { ID: "4.0.19"},  
  Settings: {  
    watch: "dir",  
  },  
  SourceFlags: [  
    {  
      Name: "dup",  
      Stop: [  
        1,  
      ],  
    },  
  ],  
}
```

The editor interface includes a menu bar with "Fil", "Rediger", and "Vis". The status bar at the bottom shows "Ln 1, Col 1", "162 tegn", "20%", and "Macintosh (i".

# Default to Current NS

```
)cs dup  
]link.export c:\exports\dup
```

- ◆ Import, Export and Create default to current namespace if given a single argument
- ◆ In Link 3.0, you always had to write

```
]link.export dup c:\exports\dup
```

# Search Library Folders

Also: Linking single source file

```
]link.import HttpCommand  
Imported: #.HttpCommand ← C:\Program Files\Dyalog\  
Dyalog APL-64 20.0 Unicode\Library\Conga\HttpCommand.dyalog
```

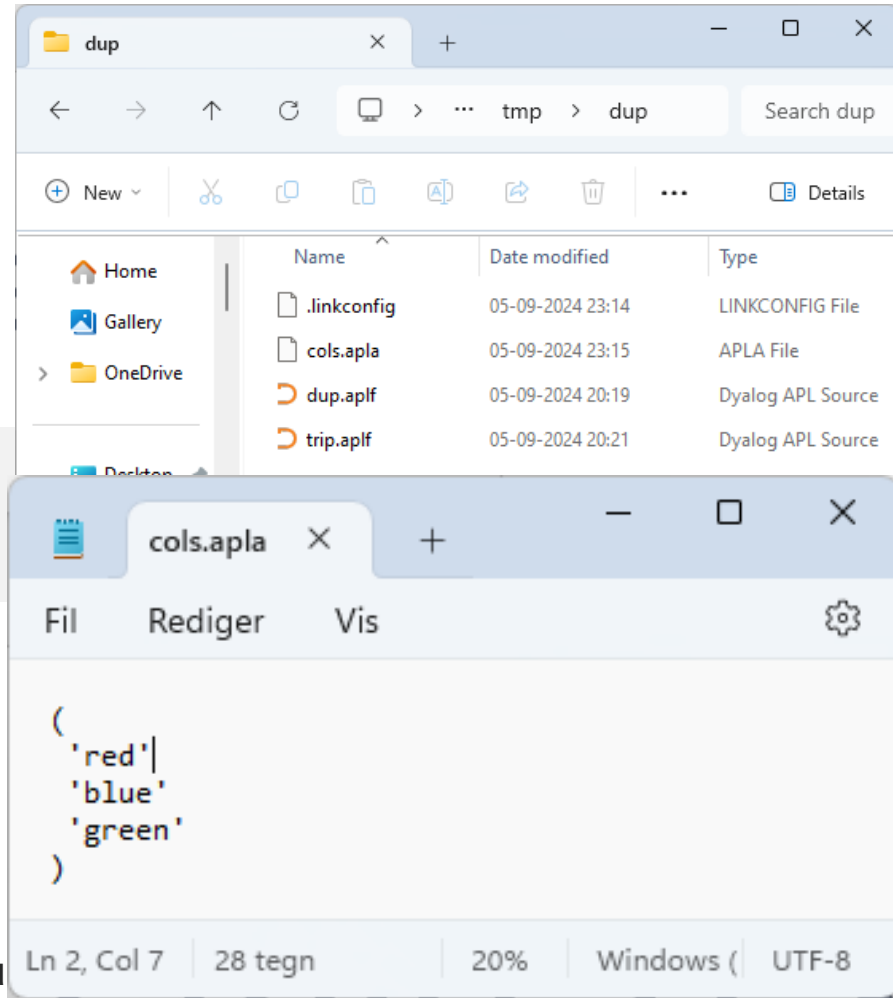
The screenshot shows two overlapping File Explorer windows. The top window is at the path `Dyalog APL-64 20.0 Unicode > Library > Conga` and displays a list of files and folders. The bottom window is at the path `Dyalog APL-64 20.0 Unicode > Library > Core` and displays a list of files and folders.

Name	Date modified	Type
Documentation	25-08-2024 12:18	File folder
FtpClient.dyalog	24-08-2024 20:07	Dyalog APL Source
HRUtils.dyalog	24-08-2024 20:07	Dyalog APL Source
HttpCommand.dyalog	24-08-2024 20:07	Dyalog APL Source
HttpUtils.dyalog	24-08-2024 20:07	Dyalog APL Source
InitConga.dyalog	24-08-2024 20:07	Dyalog APL Source
Documentation	25-08-2024 12:18	File folder
APLProcess.dyalog	24-08-2024 20:07	Dyalog APL Source

# Plain Text Formats

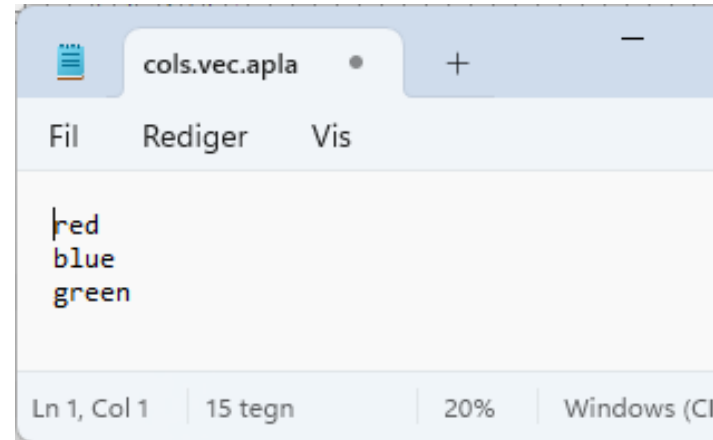
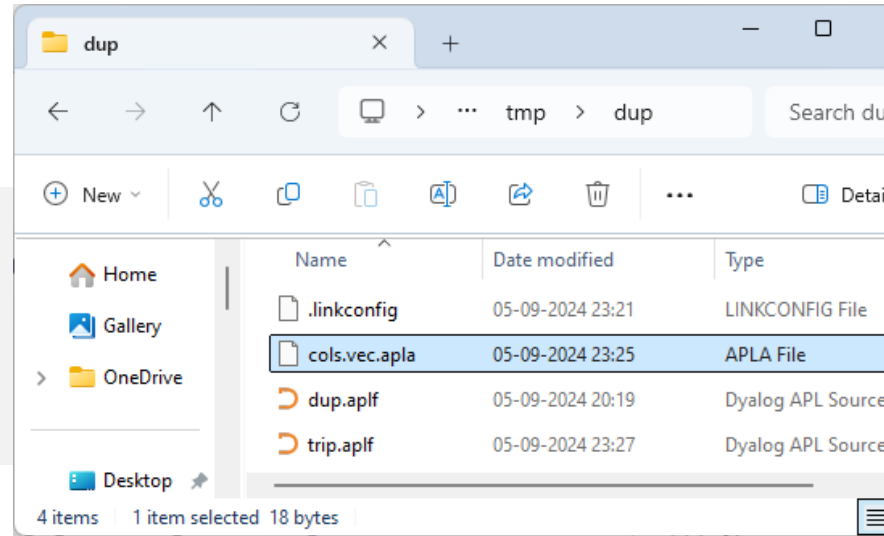
- Array Notation is not the best way to represent matrices or vectors or character vectors:

```
cols←'red' 'blue' 'green'  
]link.add cols  
Added: #.cols
```



# Plain Text Formats

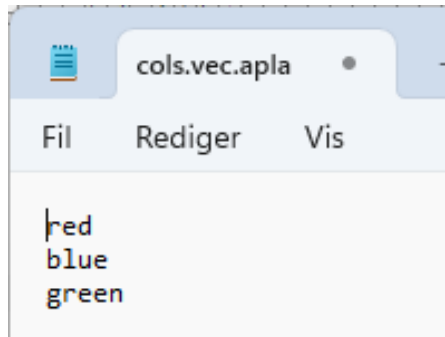
```
]link.configure # text:plain  
Was text:  
  cols←'red' 'blue' 'green'  
]link.add cols  
Added: #.cols
```





# Plain Text Formats

- Penultimate segment of name decides target representation
  - This feature is still considered experimental



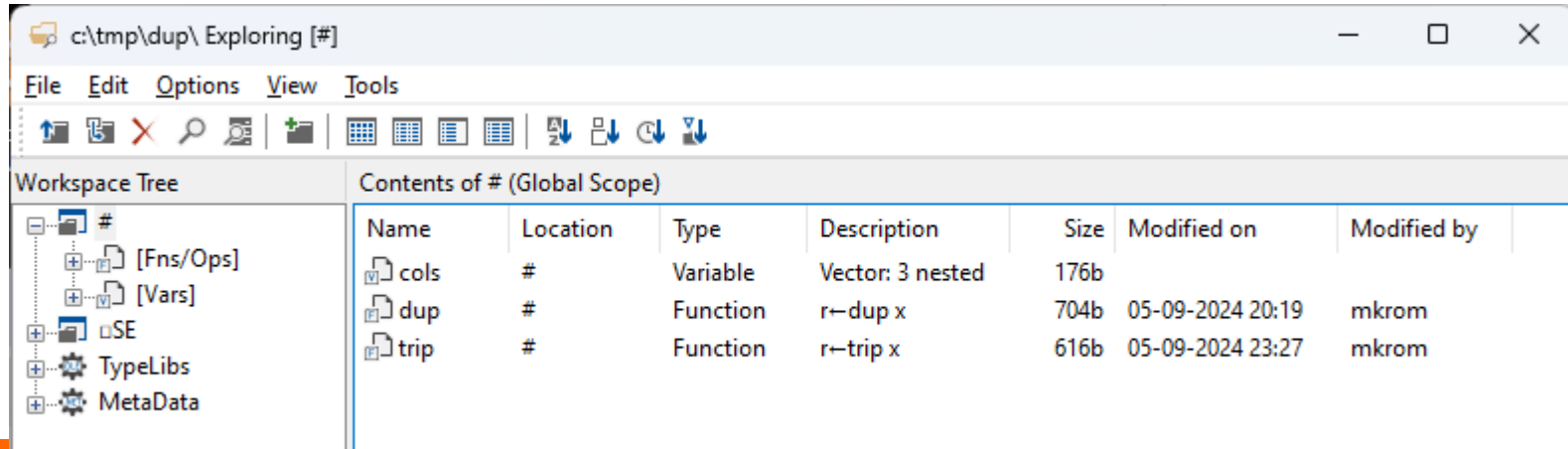
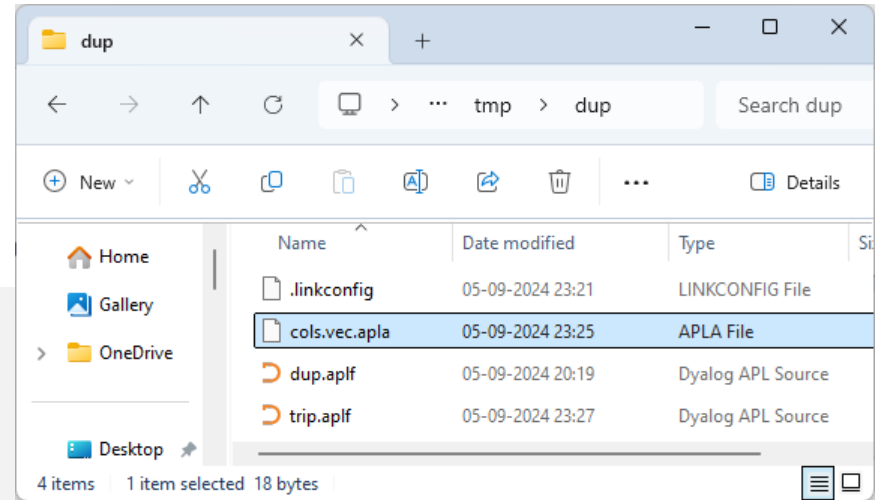
```
cols.vec.apla
File Rediger Vis
red
blue
green
```

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

# File TimeStamps

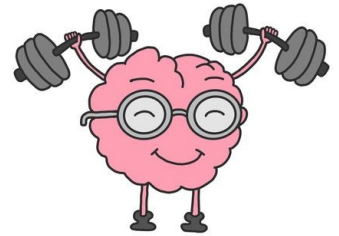
- Link.Create sets "AT" information using file timesamps

```
]link.create # c:\tmp\dup
mod←(2÷24)+21 □ATX fns←'dup' 'trip'
fns,;'DD-MM-YYYY hh:mm' (1200I) mod
dup    05-09-2024 20:19
trip   05-09-2024 23:27
```



# Exercise 5 – text:plain

- Set the configuration for your project to use text:plain
- )CLEAR and re-create the Link
- Notice that `st at f ns . ap l a` still exists and has been loaded
  - The setting only affects how NEW files are created
- Create a new variable which is plain text – either a matrix or a vector of vectors
- Use `]link . add` to save it and inspect the file



# Saved Workspaces with Links

- ◆ If you need to take a break...
- ◆ Or a batch job crashes and saves a workspace
- ◆ You can ) **SAVE** with active links
  - ◆ (Since Link 3.0)
- ◆ ... and pick up where you left off ...

# Saved Links

## Create a Source Folder & Do some work

```
□MKDIR 'c:\tmp\dup'  
(c'r←dup x' 'r←x x') □NPUT 'c:\tmp\dup\dup.aplf'  
]link.create # c:\tmp\dup  
Linked: # ↔ c:\tmp\dup  
    )save c:\tmp\dup  
c:\tmp\dup.dws A saved Thu Sep  5 20:19:10 2024  
    )off A Go to Lunch
```

## Someone Edited the File While we were at Lunch

```
(c'r←dup x' 'r←x x A duplicate x') □NPUT 'c:\tmp\dup\dup.aplf' 1
```

## We return from Lunch

```
)load c:\tmp\dup  
c:\tmp\dup.dws A saved Thu Sep  5 20:19:10 2024  
Link Warning: IMPORTANT: 1 namespaces linked in this workspace: #  
Link Warning: IMPORTANT: Link.Resync is required
```

## ”Resync is Required”

```
]link.resync
1 update required: use -proceed option to synchronise
Name  Direction  File                               Comments
#.dup ←          c:/tmp/dup/dup.aplf  File is dated Now, WS copy is dated 1 minute ago
```

## Instead, we mess things up a bit more...

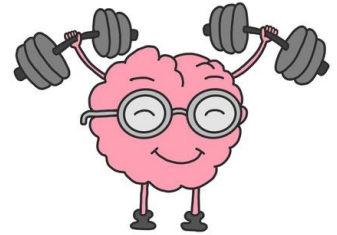
```
)copy dfns cmpx
]link.resync
2 updates required: use -proceed option to synchronise
Name  Direction  File                               Comments
#.cmpx →                               Item has no corresponding file
#.dup ←          c:/tmp/dup/dup.aplf  File is dated Now, WS copy is dated 1 minute ago
```

## Finally, clean up:

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

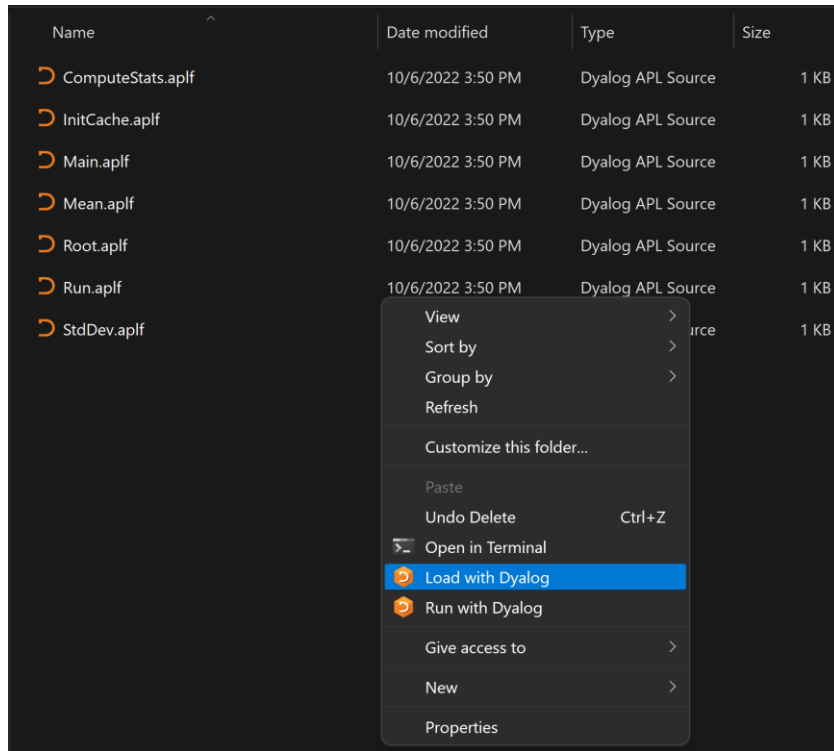
# Exercise 6 – )SAVE with Links

- )SAVE your workspace
- )CLEAR or )OFF
- Edit and change one of the source files using notepad or similar
- )LOAD the workspace and sort things out



# Launch from Source

- Windows only, v18.2 or later:  
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 the default APL version





# LOAD= Parameter

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

## Example .dcfg file

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



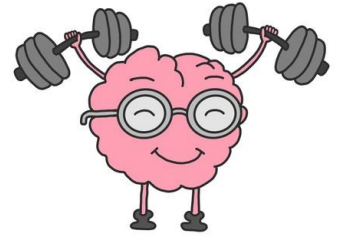
```
C:\> Command Prompt  
Microsoft Windows [Version 10.0.22000.978]  
(c) Microsoft Corporation. All rights reserved.  
C:\>dyalogrt.exe LOAD="C:/Git/ProjectA/MyFunction.aplf"
```

# Boot or Build?

- ◆ It is fine (even encouraged!) to dynamically load text source during development
- ◆ It is NOT recommended to dynamically load source from large numbers of text files in production environments
- ◆ Break links (or use `Import` rather than `Create`) and `)SAVE` to build workspace for production use

# Exercise 7

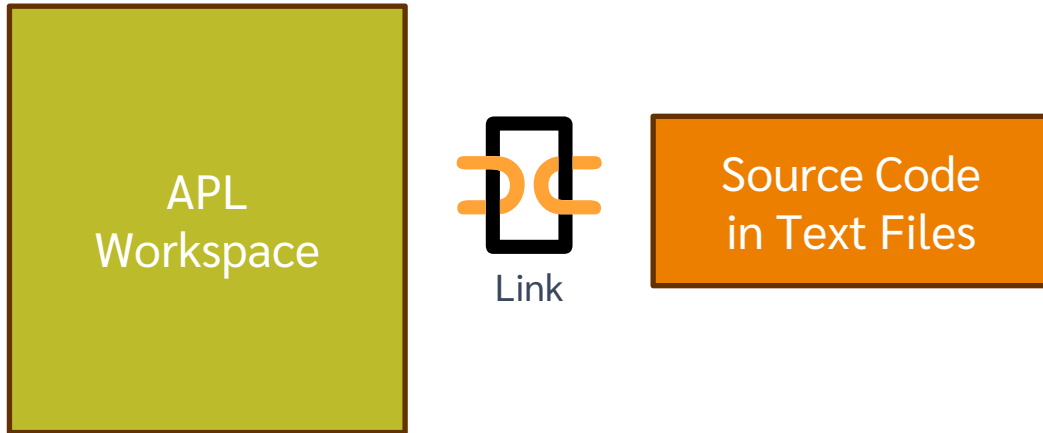
- Write a "Build" function



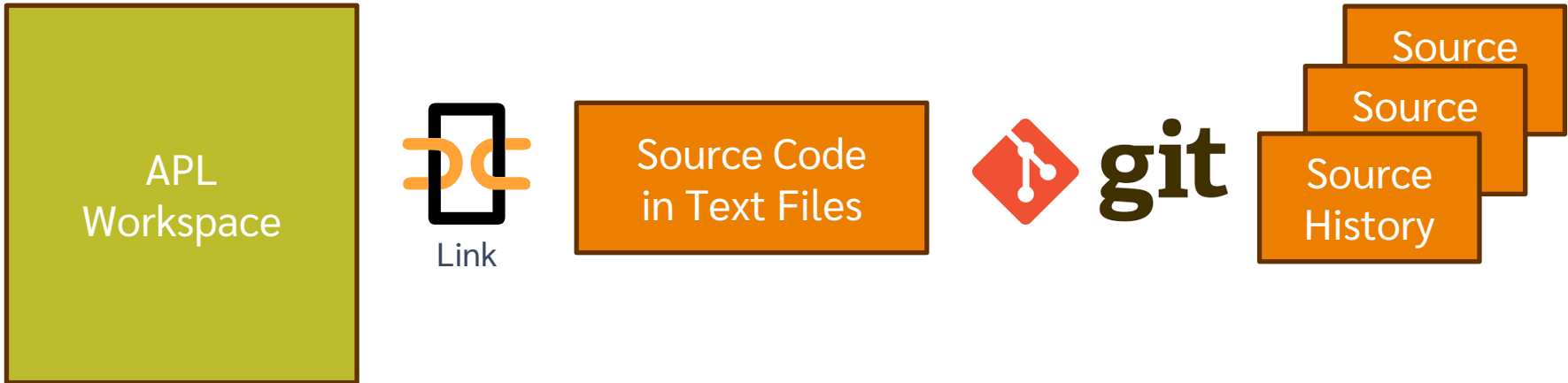
# First There Was The Workspace



# Then There was Link



# Then There was Link (and git/svn etc)



# Packages

- ◆ We have our own code under control
- ◆ The next step is to add tools to manage "other people's code"
  - ◆ A **PACKAGE** manager
- ◆ To integrate the packages into our application, we need
  - ◆ A **PROJECT** manager

# So... What is a Package?



(From Longman Dictionary of Contemporary English)



## A Project is...

Source Code +

- ◆ Dependencies (packages) loaded from a package manager
- ◆ Environment configuration
- ◆ Development tools and processes
- ◆ Can be opened and "set up" by a Project Manager
- ◆ We will use "Cider"

## A Package is...

A "build" of a project...

- ◆ In a standard format
- ◆ Can be **found, downloaded** and **installed** by a "Package Manager"
- ◆ Cider supports the development of Tatin Packages
- ◆ Cider can load Tatin + NuGet Packages



# Introducing Cider

- Load other code that we **depend** on
- Run some code on **opening** the project
- Run a **build** function
- Decide **where** to load the code
- Run **tests**
- Set **Link options** to be used when loading the source code
- Set **IO, ML**

```
cider.config
+
File Rediger Vis
{
  CIDER: {
    cider_version: "0.42.2",
    dependencies: {
      nuget: "nuget-dependencies",
      tatin: "tatin-dependencies",
    },
    dependencies_dev: {
      tatin: "",
    },
    distributionFolder: "Dist",
    init: "",
    make: "",
    parent: "#",
    projectSpace: "tp3cp",
    project_url: "",
    source: "APLSource",
    tests: "",
    version: "",
  },
  LINK: {
    forceFileNames: 1
  },
  SYSVARS: {
    io: 1,
    ml: 1,
  },
  USER: {
  },
}
```

# Creating a Project

```
]cider.createproject /tmp/tp3cp
```

When done, you should have:

```
)obs  
tp3cp
```

So now you can

```
)ed tp3cp.Foo
```

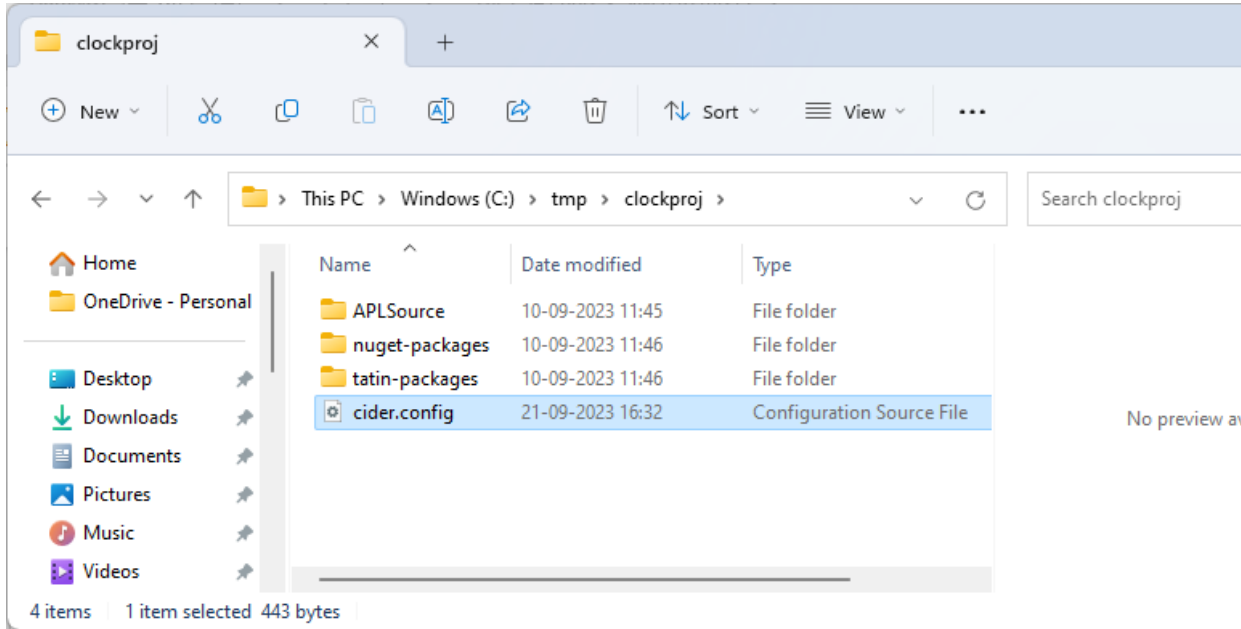


The screenshot shows a code editor window titled "SE.edit.cider\_config". The editor displays a JSON configuration for a CIDER project. The configuration includes fields for version, dependencies, development dependencies, distribution folder, initialization, make command, parent directory, project space, project URL, source, tests, and version. The project space is set to "tp3cp".

```
{  
  CIDER: {  
    cider_version: "0.42.3",  
    dependencies: {  
      nuget: "nuget-dependencies",  
      tatin: "tatin-dependencies",  
    },  
    dependencies_dev: {  
      tatin: "tatin-dependencies_dev",  
    },  
    distributionFolder: "Dist",  
    init: "",  
    make: "",  
    parent: "#",  
    projectSpace: "tp3cp",  
    project_url: "",  
    source: "APLSource",  
    tests: "",  
    version: "",  
  },  
  LINK: {  
  },  
  SYSVARS: {  
    io: 1,  
    ml: 1,  
  },  
  USER: {  
  },  
}
```

Modified Nested Array (coloured as JSON)

# A Cider Project Folder



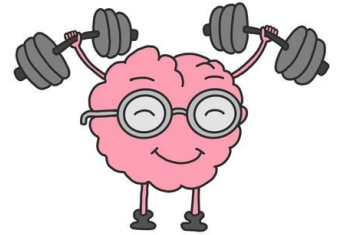
```
cider.config
File Rediger Vis

{
  CIDER: {
    cider_version: "0.42.3",
    dependencies: {
      nuget: "nuget-dependencies",
      tatin: "tatin-dependencies",
    },
    dependencies_dev: {
      tatin: "",
    },
    distributionFolder: "Dist",
    init: "",
    make: "",
    parent: "#",
    projectSpace: "tp3cp",
    project_url: "",
    source: "APLSource",
    tests: "",
    version: "",
  },
  LINK: {
  },
  SYSVARS: {
    io: 1,
    ml: 1,
  },
  USER: {
  },
}
```

Ln 9, Col 15 | 446 tegn | 30% | Windows (CRLF) | UTF-8

# Exercise 8

- ◆ If you have not already done it, activate Cider and Tatin
  - ◆ `]Tools.Activate all`
  - ◆ restart APL
  - ◆ `]UReset`
  - ◆ `]Cider.Version`
- ◆ Create a Cider project
  - ◆ `]Cider.CreateProject /folder/name`
  - ◆ Inspect the contents of the folder
  - ◆ Create a function in the project namespace
- ◆ `)CLEAR` and `]Cider.OpenProject`



# Tatin



Package manager for Dyalog APL

A tasty way to package APLs

48 Packages

2023

```
]z←tatin.listPackages
{α,≠ω}⊔{(-1+ωι'-')↑ω}¨3↑z[;1]
```

```
aplteam 42
davin 4
dyalog 2
```

~2↑z

```
dyalog-HttpCommand 1
dyalog-Jarvis 1
```

# NuGet



Package manager for .NET

Related to "Chocolatey"

~~361,905~~ 416,844 Packages



2024

```
]z←tatin.listPackages
{α,≠ω}⊔{(-1+ωι'-')↑ω}¨3↑z[;1]
```

```
aplteam 44
davin 4
dyalog 5 A 150% growth!
```

~5↑z

```
dyalog-APLProcess 1
dyalog-HttpCommand 1
dyalog-Jarvis 1
dyalog-NuGet 1
dyalog-OpenAI 1
```

# Finding Packages – www.tatin.dev



## List of packages

Package name	Description	Major Versions	Project URL	OS	UC	Tags
<a href="#">aplteam-ADOC</a>	Automated generation od documentation	1	<a href="#">github.com</a>	Lin, Mac, Win	Yes	documentation
<a href="#">aplteam-APLGit2</a>	Git interface from Dyalog APL via Git Bash	1	<a href="#">github.com</a>	Lin, Mac, Win	Yes	apl-git-interface
<a href="#">aplteam-APLGUI</a>	Collection of GUI utilities	1	<a href="#">github.com</a>	Win		gui-tools,gui
<a href="#">aplteam-APLProcess</a>	Start an APL process from within Dyalog APL	1	<a href="#">github.com</a>	Lin, Mac, Win		process
<a href="#">aplteam-APLTreeUtils2</a>	General utilities required by most members of the APLTree library	1	<a href="#">github.com</a>	Lin, Mac, Win		tools,utilities
<a href="#">aplteam-Cider</a>	A project manager for Dyalog APL that cooperates with Tatin	1	<a href="#">github.com</a>	Lin, Mac, Win	Yes	project-management
<a href="#">aplteam-CodeBrowser</a>	Tool useful for code reviews	1	<a href="#">github.com</a>	Lin, Mac, Win	Yes	code-browsing,code-reviews
<a href="#">aplteam-CodeCoverage</a>	Monitors which parts of an application got actually executed	1	<a href="#">github.com</a>	Lin, Mac, Win		code-coverage,test-framework,unit-tests
<a href="#">aplteam-CommTools</a>	Communication tools for interactions in the session: YesOrNo and Select	1	<a href="#">github.com</a>	Lin, Mac, Win		communication-tools,yes-or-no,select-tool
<a href="#">aplteam-Compare</a>	Allows comparing and merging objects in the workspace with a file or a file with another file	3	<a href="#">github.com</a>	Lin, Mac, Win		comparison-tool,merge-tool
<a href="#">aplteam-CompareFiles</a>	Cover for comparison utilities	1	<a href="#">github.com</a>	Lin, Mac, Win	Yes	file-comparison,comparison-utilities

# Finding Packages



## Tatin Registry

### List of packages

zip

Package name	Description	Major Versions	Project URL	OS	UC	Tags
<a href="#">aplteam-DotNetZip</a>	Zippping and unzipping with .NET Core on all major platforms	2	<a href="#">github.com</a>	Win		zip-tools
<a href="#">aplteam-SevenZip</a>	Zip files with the Open Source tool 7Zip	1	<a href="#">github.com</a>	Win		zip-tools
<a href="#">aplteam-ZipArchive</a>	Zippping and unzipping with .NET on Windows and zip/unzip on other platforms	2	<a href="#">github.com</a>	Lin, Mac, Win		zip-tools

Created by Tatin version 0.100.1+1627 from 2023-08-28 under Linux-64 18.2.45645.0 S Runtime — Bugs, questions, problems: [info@tatin.dev](mailto:info@tatin.dev)



# Finding Packages



## List of packages

Package name	Description	Major Versions	Project URL	OS	UC	Tags
<a href="#">aplteam-APLProcess</a>	Start an APL process from within Dyalog APL	1	<a href="#">github.com</a>	Lin, Mac, Win		process
<a href="#">aplteam-Execute</a>	Start a process from within APL	1	<a href="#">github.com</a>	Win		process,run-applications
<a href="#">dyaalog-APLProcess</a>	Utility to start APL processes	1	<a href="#">github.com</a>	Lin, Mac, Win		pi

53 packages is enough to (sometimes) make it difficult to decide which one to use



## Tatin Registry

Home page of group "dyalog"

### The dyalog group

This group contains packages published and supported by Dyalog, LTD.

Single email address:

support@dyalog.com

Edit

#### Packages owned by "dyalog"

Filter...

Package name	Description	OS
<a href="#">dyaLog-APLProcess</a>	Utility to start APL processes	Lin, Mac, Win
<a href="#">dyaLog-HttpCommand</a>	Utility to execute HTTP requests	Lin, Mac, Win
<a href="#">dyaLog-Jervis</a>	JSON and REST Web Service Framework	Lin, Mac, Win
<a href="#">dyaLog-NuGet</a>	Use NuGet packages from Dyalog APL	Lin, Mac, Win

\* UC means "User Command"

## Details of <aplteam-CodeCoverage-0.10.0>

```
{  
  api: "CodeCoverage",  
  assets: "",  
  date: 20230323.182755,  
  description: "Monitors which parts of an application got actually executed",  
  documentation: "",  
  files: "",  
  group: "aplteam"  
  io: 1,  
  license: "MIT",  
  lx: "",  
  maintainer: "kai@aplteam.com",  
  minimumApiVersion: "18.0",  
  ml: 1,  
  name: "CodeCoverage",  
  os_lin: 1,  
  os_mac: 1,  
  os_win: 1,  
  project_url: "https://github.com/aplteam/CodeCoverage",  
  source: "APLSource/CodeCoverage.aplc",  
  tags: "code-coverage,test-framework,unit-tests",  
  userCommandScript: "",  
  version: "0.10.0+55",  
}
```



## Tatin Registry

### Dependencies of "aplteam-CodeCoverage-0.10.0"

#### Dependencies

[aplteam-APLTreeUtils2-1.1.3](#)

[aplteam-Tester2-3.3.1](#)

[aplteam-CommTools-1.3.0](#)

# ]Tatin.ListPackages

```
]Tatin.ListPackages -group=dyalog
```

```
Registry: https://tatin.dev
```

Group & Name	# major versions
-----	-----
dyalogAPLProcess	1
dyalog-HttpCommand	1
dyalog-Jarvis	1
dyalog-NuGet	1
dyalog-OpenAI	1

```
]Tatin.ListPackages -tag=crypto
```

```
Registry: https://tatin.dev
```

Group & Name	# major versions
-----	-----
aplteam-HashPasswords	1

```
]tatin.listtags
tags from https://tatin.dev
-----
apl-git-interface
build
calculations
chm
code-browsing
code-coverage
code-reviews
command-generation
communication-tools
comparison-tool
comparison-utilities
components
config-files
converter
copy
cryptography
date
dates
...
...
utilities
validation
webservice
windows-event-log
windows-registry
wincsp-interface
write
yes-or-no
zip-tools
```

# Adding a Tatin Dependency

- Example: I use `HttpCommand` in just about every new project
- To add it to our Cider project:

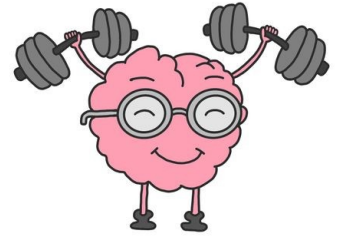
```
]Cider.AddTatinDependencies HttpCommand  
1 Tatin dependency added:  
  dyalog-HttpCommand-5.2.0
```

- Since we did not specify a version, we get the latest.
- A reference is created to the loaded package within our project space:

```
tp3cp.HttpCommand.Get 'www.dyalog.com'  
[rc: 0 | msg: | HTTP Status: 200 "OK" | #Data: 22580]
```

# Exercise 9

- ◆ Add `HttpCommand` (or some other Tatin Package) to your project
- ◆ Inspect your project folder
- ◆ Close and reopen the project to convince yourself that it all works
- ◆ Update your function to use your new package



# NuGet

- ◆ NuGet is the .NET package manager
- ◆ You can use NuGet packages from Dyalog APL, with .NET 6.0 or later

The screenshot shows the NuGet Gallery homepage. The header includes the NuGet logo and navigation links: Packages, Upload, Statistics, Documentation, Downloads, and Blog. A search bar is prominently displayed with the text "search for packages...". Below the search bar, a large graphic illustrates the scale of the repository with three callout boxes: "6,261,988 package versions", "380,701,196,969 package downloads", and "371,905 unique packages". The graphic itself consists of a central circuit board with several 3D cube-like packages connected to it by lines. Below this, the heading "What is NuGet?" is followed by a paragraph: "NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers." At the bottom, there are three circular icons: the first shows a person at a desk with a lightbulb idea, the second shows a person with a telescope, and the third shows a person with tools and a box.

# Adding a NuGet Package

- You must have .NET 6.0 or later, and Dyalog APL configured to use it (DYALOG\_NETCORE=1).

NuGet support currently requires .NET 6.0, 7.0 or 8.0

Support for "Framework" packages MAY follow

```
□ USING← ' '
```

```
System.Environment.Version
```

```
8.0.8
```



# NuGet Tests

## Files

main

Go to file

APLSource

NuGet

Tests

parquet

find\_nuget\_api.aplf

test.aplf

test\_clock.aplf

test\_htmlsanitizer.aplf

test\_humanizer.aplf

test\_kafka\_wip.aplf

test\_mailkit.aplf

nuget / APLSource / Tests /



..



parquet

Transform into a Cider project



find\_nuget\_api.aplf

Transform into a Cider project



test.aplf

Transform into a Cider project



test\_clock.aplf

Transform into a Cider project



test\_htmlsanitizer.aplf

Add test\_htmlsanitizer



test\_humanizer.aplf

Use () rather than {} in APLAN, in test\_humanizer



test\_kafka\_wip.aplf

Transform into a Cider project



test\_mailkit.aplf

Transform into a Cider project



test\_parquet.aplf

Transform into a Cider project



test\_selenium.aplf

Transform into a Cider project

## Files

main

Go to file

- APLSource
  - NuGet
  - Tests
    - parquet
      - find\_nuget\_api.aplf
      - test.aplf
      - test\_clock.aplf

## nuget / APLSource / Tests / test\_clock.aplf

mkromberg Transform into a Cider project

Code Blame 4 lines (4 loc) · 135 Bytes

```
1 test_clock project_dir;⎕USING
2 NuGet.Add project_dir 'Clock/1.0.3'
3 ⎕USING←NuGet.Using project_dir
4 'The time is: ',⎕Clock.UtcNow
```

# Adding a NuGet Package

- We can add Clock to our Cider project (by default, adds the latest version):

```
]Cider.AddNuGetDependencies Clock  
Clock 1.0.3
```

- A reference to a namespace hosting the .NET package is created:

```
tp3cp.Clock.UtcNow.(Hour Minute)  
14 43
```

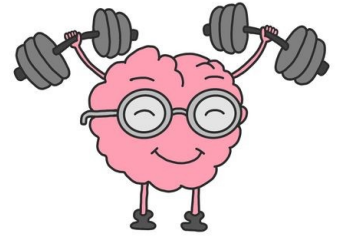
- In fact, the namespace is empty except for `USING`:

```
tp3cp.Clock.[]USING  
,c:/tmp/tp3cp/nuget-packages/published/Clock.dll
```

# Exercise 10

- ◆ Add Clock (or another NuGet package of your choice) to your project
- ◆ Possible inspiration at

<https://github.com/Dyalog/nuget/tree/main/APLSource/Tests>



# Morten's Solution

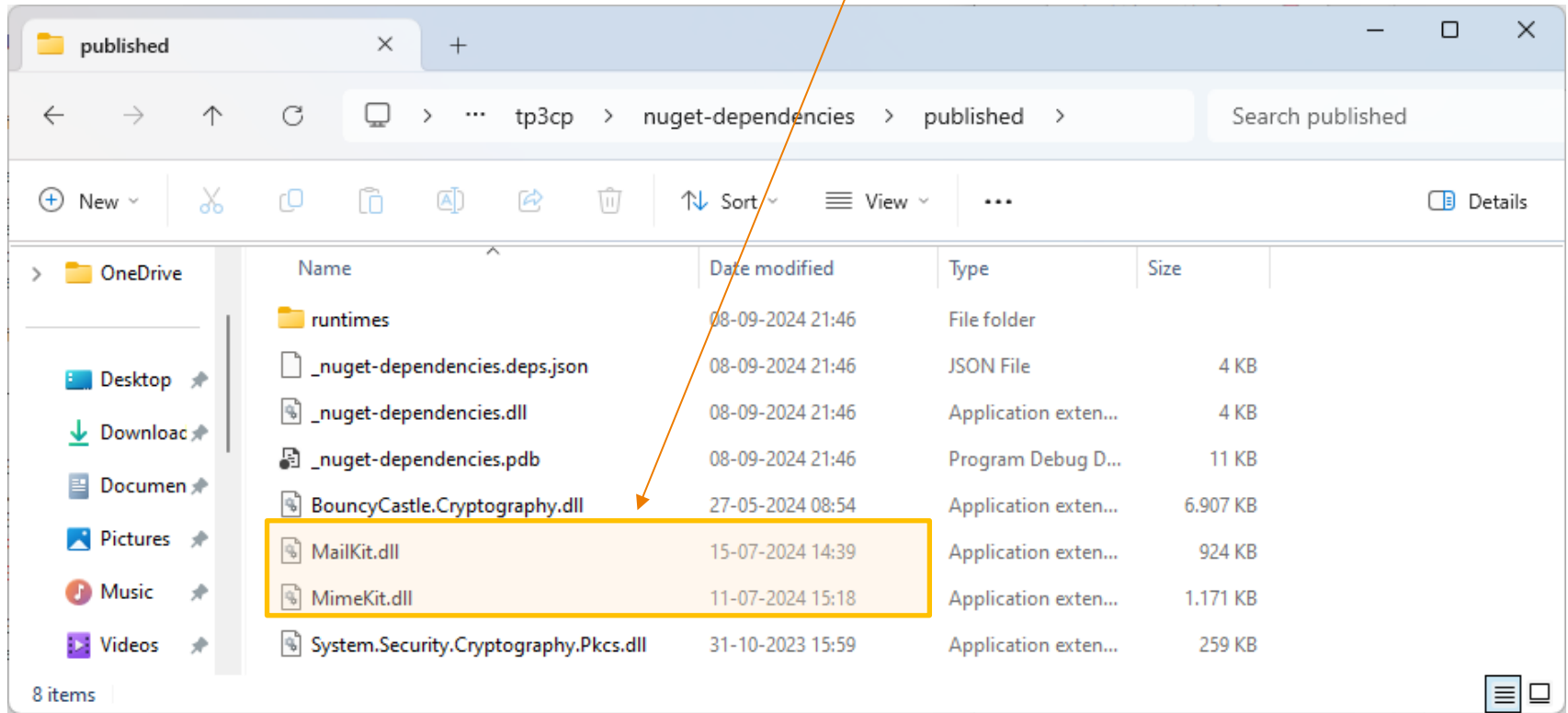
- ◆ We can add Clock to our Cider project (by default, adds the latest version).
- ◆ But more interesting:

```
]cider.addNuGetDependencies MailKit,MimeKit  
Would you like to (re-)load all NuGet dependencies? (Y/n) y  
MailKit MimeKit
```

- ◆ See the "ReadMail" Cider project in the TP3 folder:

```
]cider.openProject c:\devt\2024-TP3\ReadMail
```

We asked for these – what are the others?



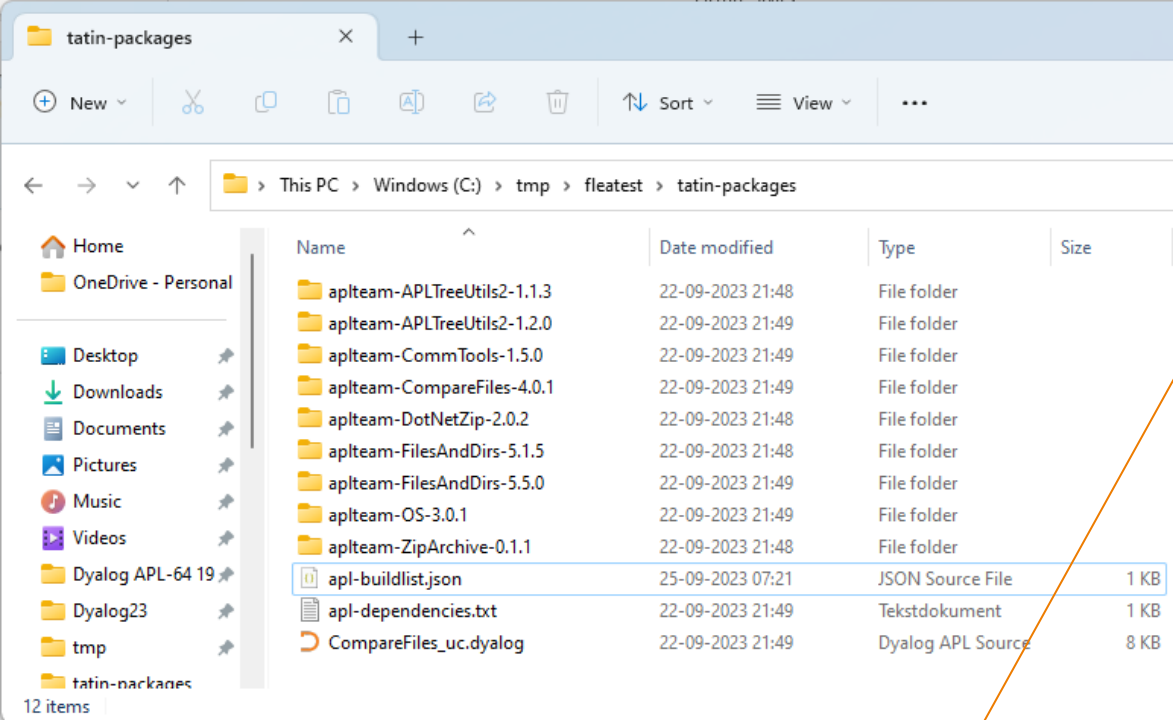
# Dependencies of Dependencies

Great fleas have little fleas upon their backs to bite 'em,  
And little fleas have lesser fleas, and so *ad infinitum*.

Both Tatin and NuGet will automatically load such dependencies

**Augustus De Morgan** was a British mathematician and logician. He formulated De Morgan's laws and introduced the term mathematical induction, making its idea rigorous.



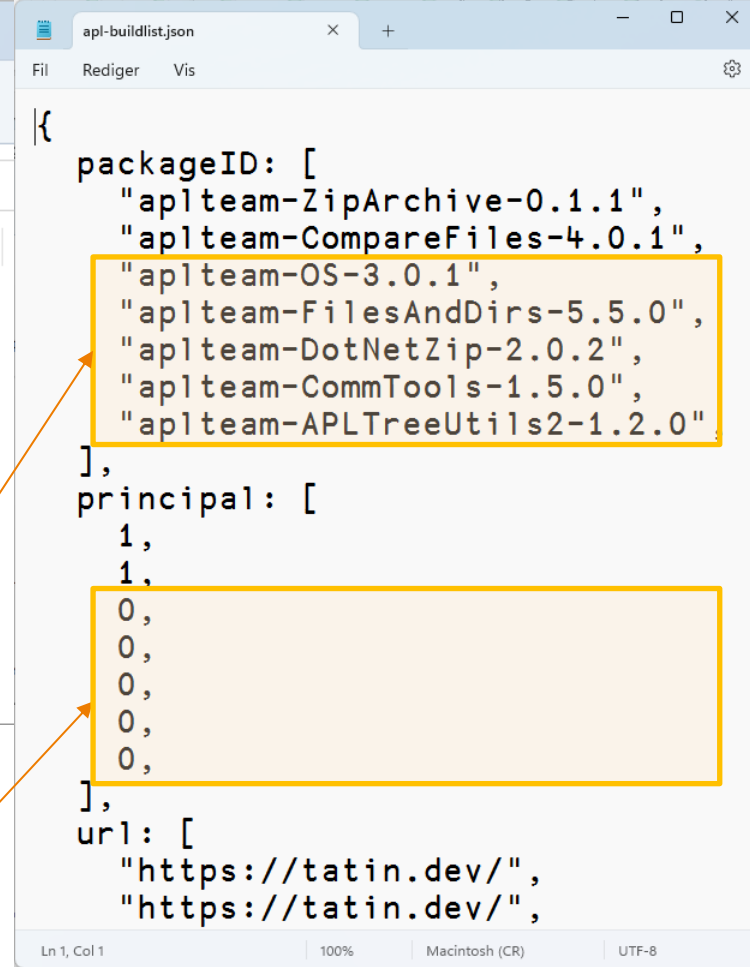
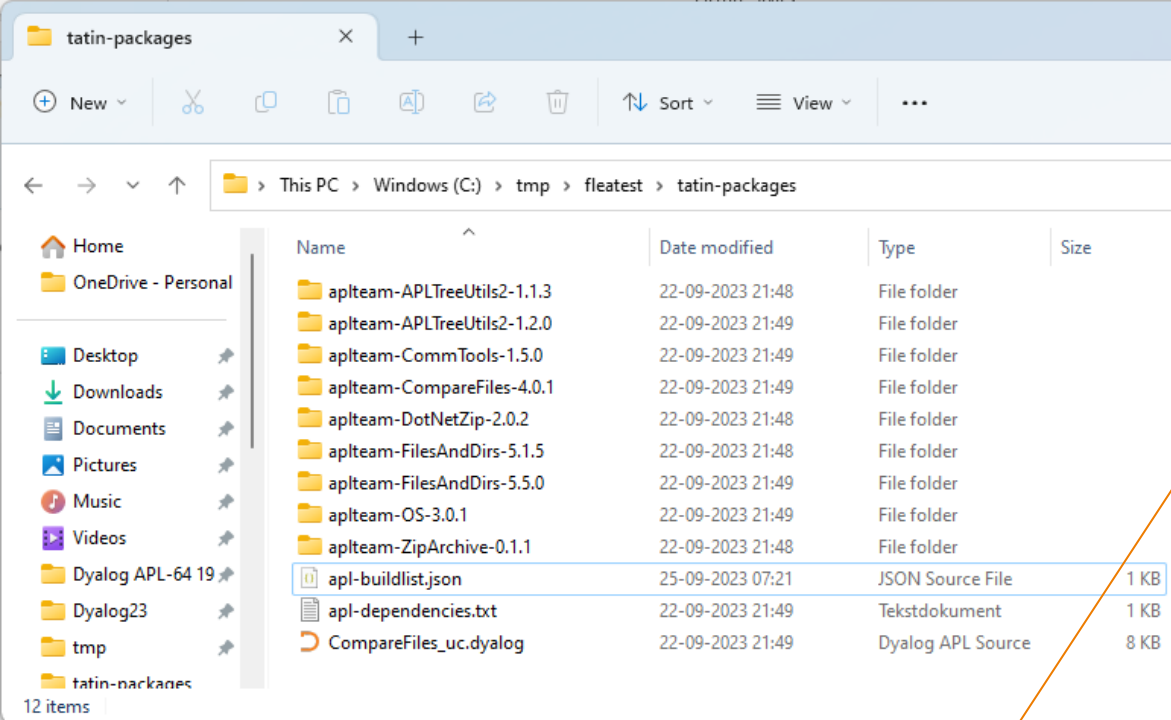


```
apl-buildlist.json
File Rediger Vis

{
  packageID: [
    "aplteam-ZipArchive-0.1.1",
    "aplteam-CompareFiles-4.0.1",
    "aplteam-OS-3.0.1",
    "aplteam-FilesAndDirs-5.5.0",
    "aplteam-DotNetZip-2.0.2",
    "aplteam-CommTools-1.5.0",
    "aplteam-APLTreeUtils2-1.2.0",
  ],
  principal: [
    1,
    1,
    0,
    0,
    0,
    0,
    0,
  ],
  url: [
    "https://tatin.dev/",
    "https://tatin.dev/",
  ]
}
```

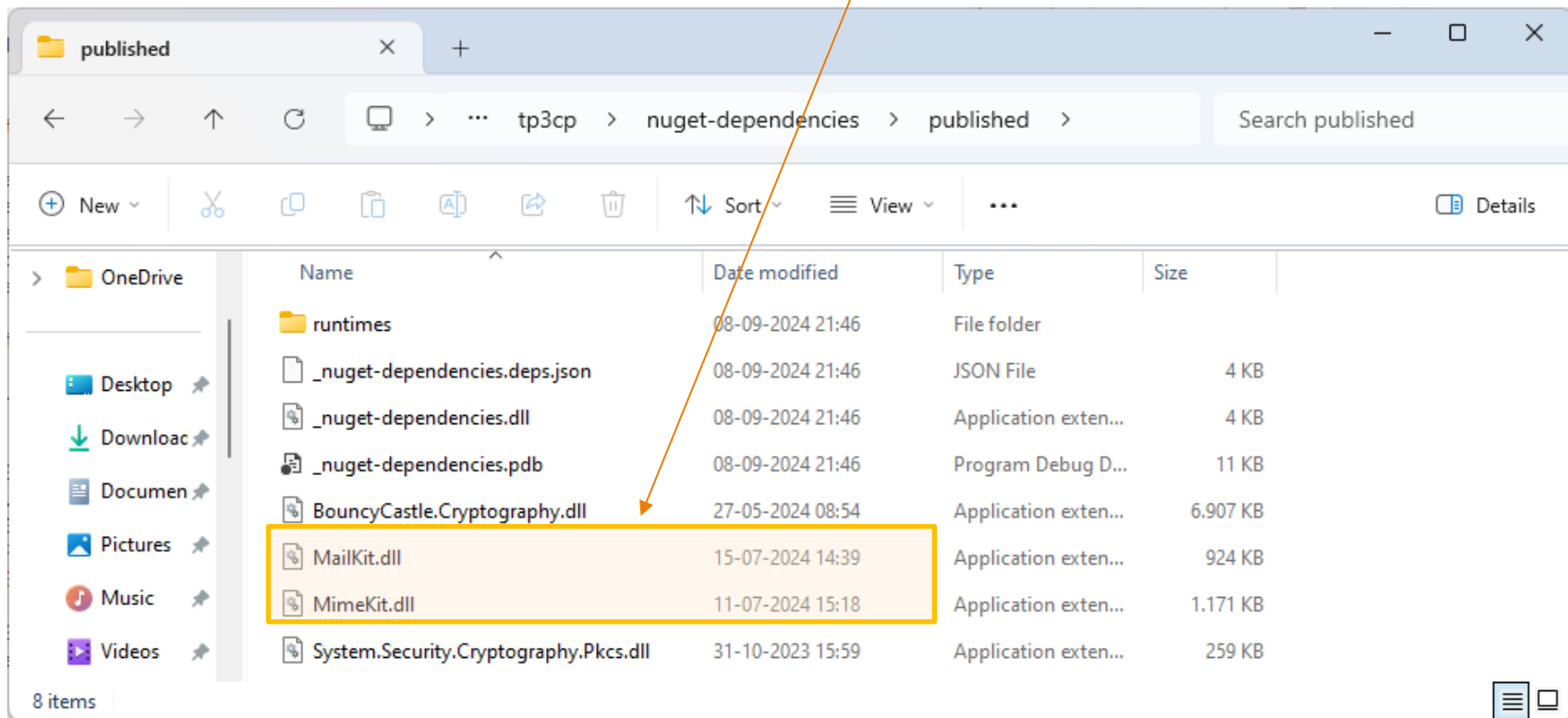
"Principal" dependencies (that we added)





"Lesser" fleas

"Principal" dependencies (that we added)



# Where Do Dependencies Go?

```
]Cider.OpenProject C:\tmp\fleatest  
Project successfully loaded and established in "#.fleatest"
```

```
)cs fleatest  
#.fleatest
```

```
  [NL -9  
CiderConfig CompareFiles ZipArchive
```

Our Dependencies

## CompareFiles

```
#.tatin.aplteam_CompareFiles_4_0_1.API
```

```
  ;#.tatin.[NL -9  
aplteam_APLTreeUtils2_1_2_0  
aplteam_CommTools_1_5_0  
aplteam_CompareFiles_4_0_1  
aplteam_DotNetZip_2_0_2  
aplteam_FilesAndDirs_5_5_0  
aplteam_OS_3_0_1  
aplteam_ZipArchive_0_1_1
```

Lesser Fleas

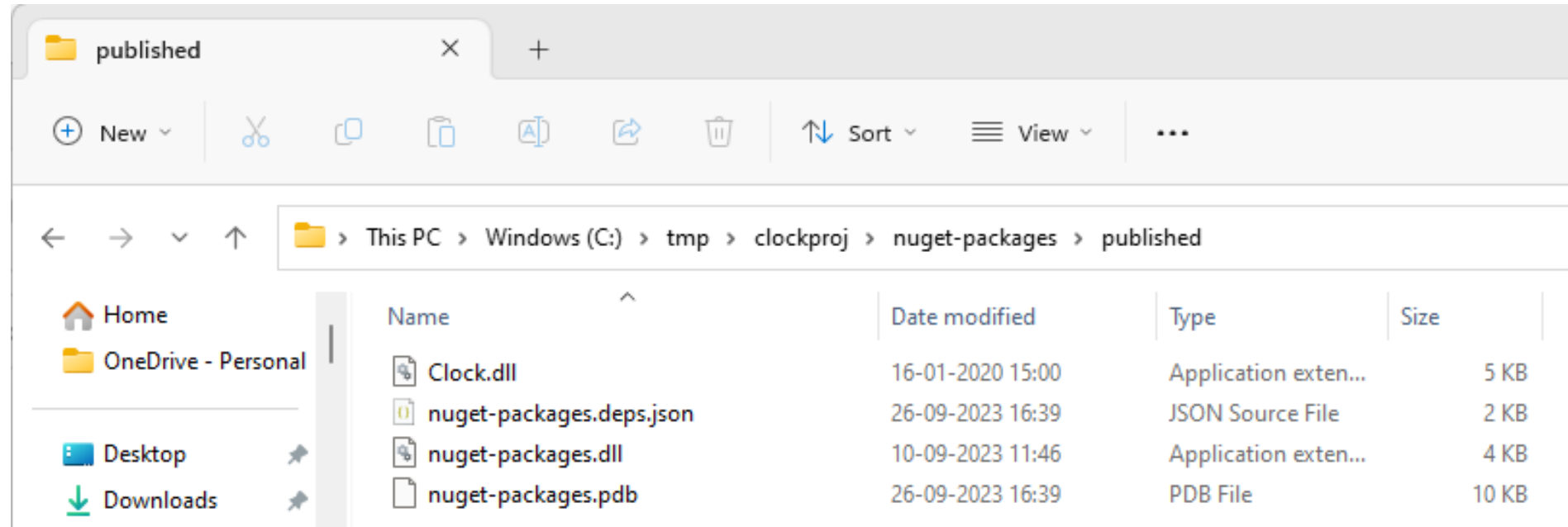
```
  #.tatin.aplteam_CompareFiles_4_0_1.[NL -9  
API APLTreeUtils2 Admin CommTools ComparisonTools FilesAndDirs TatinVars
```

# dotnet command-line tool

- ◆ Under Windows, Linux and macOS, .NET provides a "dotnet" command which:
  - ◆ Creates .NET projects that we use to define and manage dependencies (complete with a C# class that we never use)
  - ◆ Adds Dependencies
  - ◆ "Publishes" collections of DLLs that implement packages
- ◆ Dyalog's NuGet support depends heavily on this
  - ◆ We just set `□USING` to point to the published DLLs
  - ◆ The alternative is to try to replicate poorly documented .NET behaviours

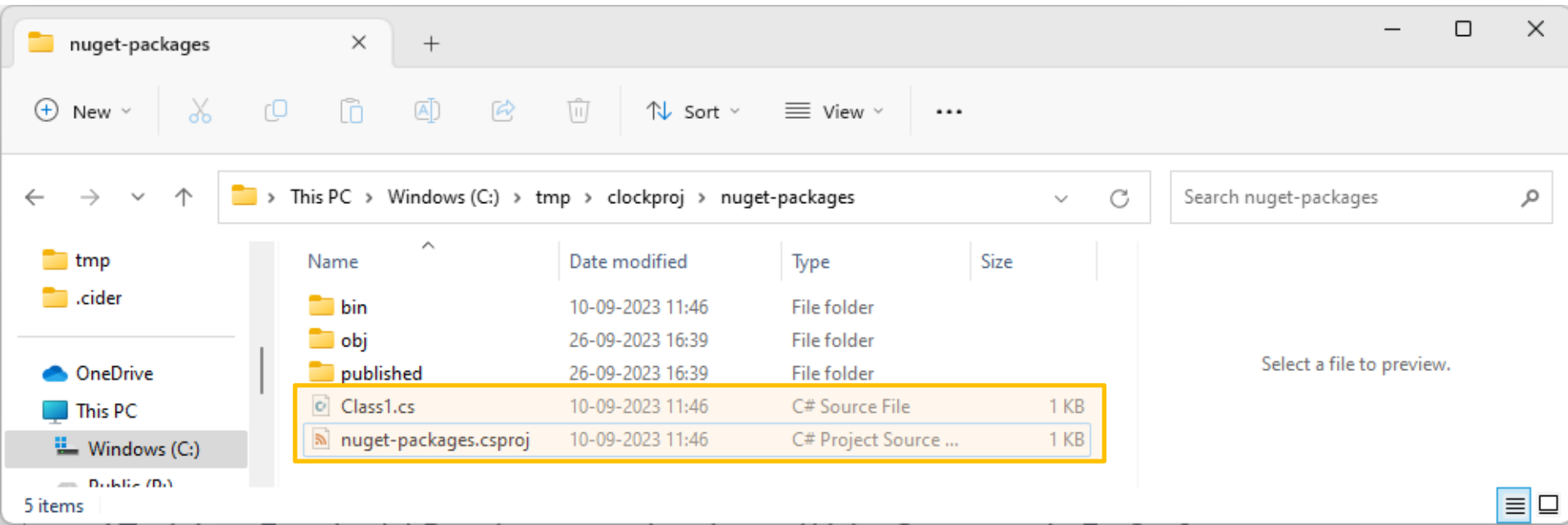
# NuGet Packages – Under the Covers

- ◆ NuGet DLL's go in a folder called "published"



# NuGet Packages – Under the Covers

- The dotnet command line tool has created some C# code which "pretends" to use the NuGet packages



# NuGet Packages – Under the Covers

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <RootNamespace>_nuget_dependencies</RootNamespace>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <LangVersion>Latest</LangVersion>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="MailKit" Version="4.7.1.1" />
    <PackageReference Include="MimeKit" Version="4.7.1" />
  </ItemGroup>
</Project>
```

```
namespace _nuget_dependencies;

public class Class1
{
}
```

OneDrive  
This PC  
Windows (C:)  
Public (D:)

5 items

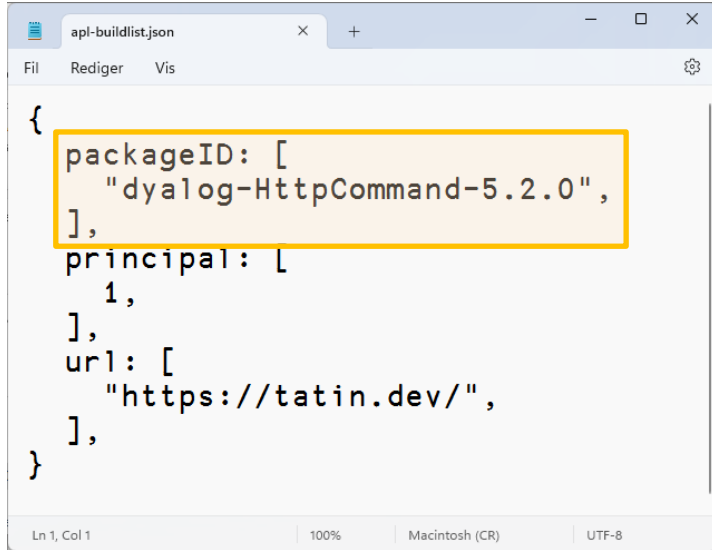
obj	26-09-2023 16:39	File folder	
published	26-09-2023 16:39	File folder	
Class1.cs	10-09-2023 11:46	C# Source File	1 KB
nuget-packages.csproj	10-09-2023 11:46	C# Project Source ...	1 KB

Select a file to preview.



# Same Same but Different

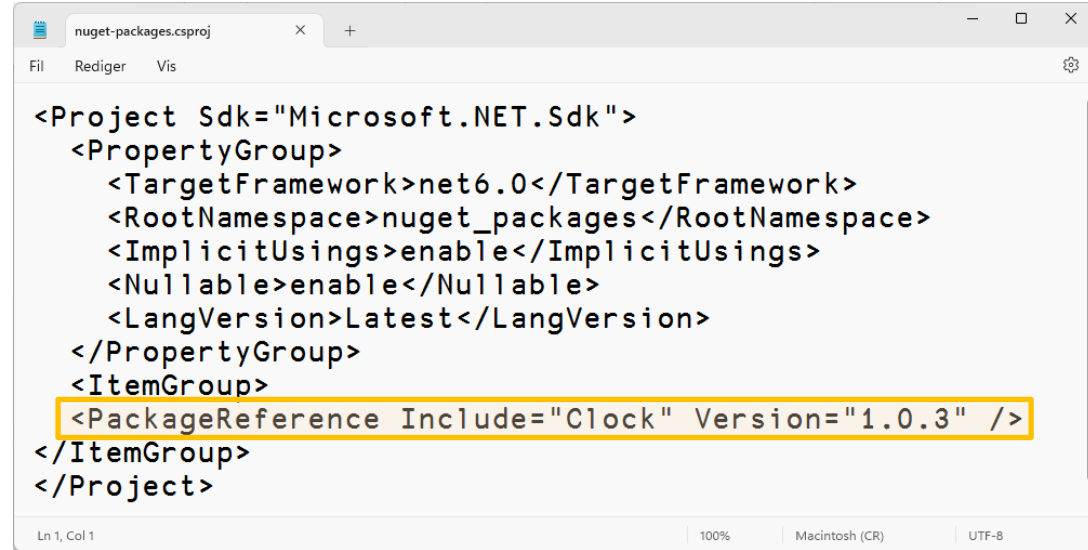
## Tatin



```
{
  packageID: [
    "dyalog-HttpCommand-5.2.0",
  ],
  principal: [
    1,
  ],
  url: [
    "https://tatin.dev/",
  ],
}
```

#.projectSpace.HttpCommand

## NuGet



```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <RootNamespace>nuget_packages</RootNamespace>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <LangVersion>Latest</LangVersion>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Clock" Version="1.0.3" />
  </ItemGroup>
</Project>
```

#.projectSpace.Clock



# The Cast, in order of appearance



**Link Synchronises** Source Files and Workspace  
The workspace and source files are "Linked"



**Cider is a Project Manager**  
A Project is a linked source folder,  
a config file, plus optional dependencies



**Tatin is the APL Package Manager**  
A Package is a project wrapped up for consumption by others



**NuGet is the .NET Package Manager**  
The Dyalog.NET Bridge allows APL to use .NET libraries

