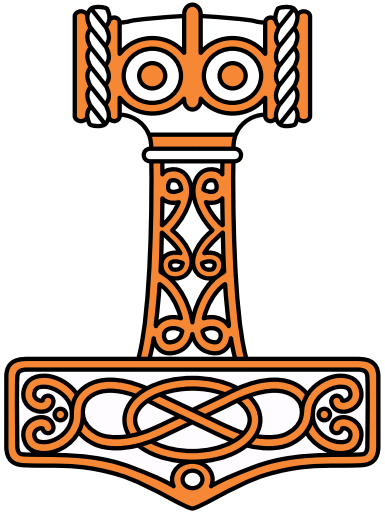
DYALOG

FinnAPL @ Suomenlinna, April 18th 2023

Highlights of v19.0



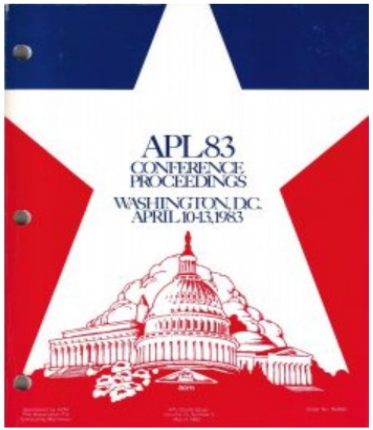
Morten Kromberg

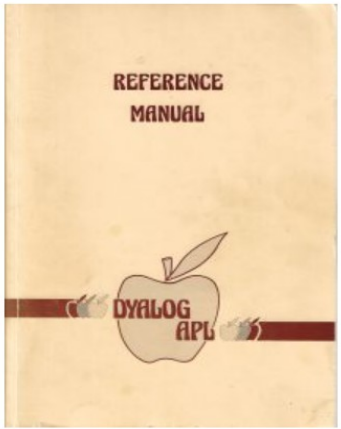
11 April 2023 – A Day to Celebrate!

Today we reach two very significant milestones.

40 Years of Dyalog APL

On this day, we have cause for celebration: it is 40 years since the release of Dyalog version 1.0! Geoff Streeter would say that from his perspective we are already in the 42nd year, as he and John Scholes started work on the new interpreter in 1981. On the other hand, Pete Donnelly might argue that the interpreter wasn't really ready for serious use until a few years after that date. The fact remains that the APL '83 conference in Washington DC saw the first official release of the product, and is considered to be the "birth" of Dyalog APL.

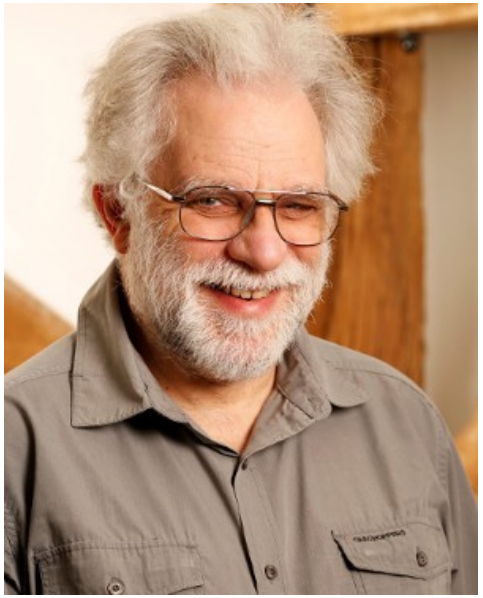




Farewell Geoff Streeter

Geoff has Retired (!)

- With John Scholes, Geoff Streeter implemented Dyalog APL v1.0 in 1981-1983
- We hope to welcome him back for a retrospective talk at Dyalog'24



Geoff has Retired (!)

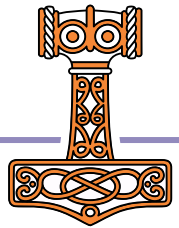
- With John Scholes, Geoff Streeter implemented Dyalog APL v1.0 in 1981-1983
- We hope to welcome him back for a retrospective talk at Dyalog'24

Recuiting in 2023

- Admin person (UK)
  - Started April 1st
- Developer / Tool Builder / Evangelist (UK)
  - Started May 1st
- Full Time Tester (UK or India)
  - We have a candidate who is being evaluated
- 1 for the IT Department (UK)
  - "Before end of 2023"

Current Head Count ~25 FTE

(from 5 in 2004)



Dyalog … The Next Generation


Aaron (ACE)


Adam (AE)


Rich (AE)

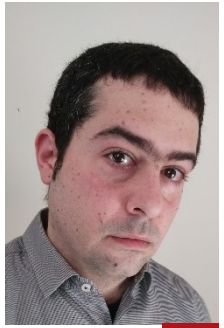

Josh (A)


Stine (D)


Karta (C)


Silas (C)


Peter (C)

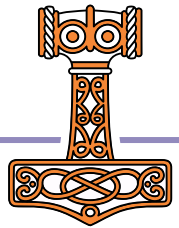

Jesus (E)


Jada (D)


Stefan (AE)

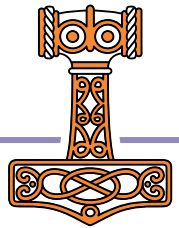
Legend:

A = APLer
C = C developer
D = Admin
E = Doc/Evangelism



Highlights of Version 19.0

- Structural Operations on Scalars



Extending Structural Functions to Scalars

Traditionally, the set of monadic reversing or reflecting primitives, *Reverse-First* (⊖), *Transpose* (⍉), and *Reverse* (⌽) apply to entire arrays and are defined as identity functions on scalar arguments. Dyalog v19.0 extends the definitions to provide equivalent reflections on scalars.

Character Data

We expect that the new transformations will be most useful on characters. For example:

```
      (⍉'A')(⌽'P')(⊖'L')
⍋⍀⍒
```

Note that you can apply the same transformation to all the items of an array using *Each* (¨) or *Rank* (⍤0):

```
      ⌽¨ 'New APL?'
ИǝW A⅂⅃ʔ
      (⊖⍤0) 'CAN HAPPEN SOON!'
⊂∀N H∀ddƎN ƧOOИ¡
```

Composition allows the combination of reflections to perform rotation:

```
      ⊖∘⍉¨ '→A/common'  ⍝ 90° clockwise rotation
↓⊃⍀\∩o≡≡o⊐
      ⌽∘⊖¨ '→A/common'  ⍝ 180° rotation
←∀/⊃oɯɯou
```

Character Data

We expect that the new transformations will be most useful on characters. For example:

```
      (⍉'A')(⌽'P')(⊖'L')
∢ꟼΓ
```

Note that you can apply the same transformation to all the items of an array using *Each* (¨) or *Rank* (⍤0):

```
      ⌽¨ 'New APL?'
Иɘw Aꟼ⅃⸮
      (⊖⍤0) 'CAN HAPPEN SOON!'
C∀N H∀bbEN ƧOOИ¡
```

Posted on April 1, 2023 by Morten Kromberg

Extending Structural Functions to Scalars

Traditionally, the set of monadic reversing or reflecting primitives, *Reverse-First* (⊖), *Transpose* (⍉), and *Reverse* (⌽) apply to entire arrays and are defined as identity functions on scalar arguments. Dyalog v19.0 extends the definitions to provide equivalent reflections on scalars.

Character Data

We expect that the new transformations will be most useful on characters. For example:

```
      (⍉'A')(⌽'P')(⊖'L')
ᐊꟼΓ
```

Note that you can apply the same transformation to all the items of an array using *Each* (¨) or *Rank* (⍤0):

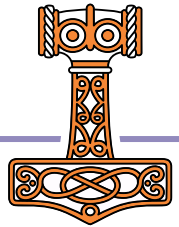
```
      ⌽¨ 'New APL?'
Иɘw Aꟼ⅃⸮
      (⊖⍤0) 'CAN HAPPEN SOON!'
C∀N H∀bbEN ƧOOИ¡
```

Composition allows the combination of reflections to perform rotation:

```
      ⊖∘⍉¨ '→A/common'  ⍝ 90° clockwise rotation
↓ᐅ\ᑎoƎƎoᑐ
      ⌽∘⊖¨ '→A/common'  ⍝ 180° rotation
←∀/ɔoɯɯou
```

Highlights of Version 19.0

- Installing & Managing Your System
  - Keyboarding
  - Multiple session files
  - Health Monitor [demo]
- Building Production Systems
  - Timeouts
  - Token range reservation [demo]
  - WS FULL handling [demo]
  - NCOPY/NMOVE callbacks
- Developer Productivity / IDE
  - Source "as typed" by default [demo]
  - Multi-line input on by default [demo]
  - HTMLRenderer updates
  - Link 4.0: Crawler, Support for text data
  - Namespace Improvements
- Platform Support / Distribution
  - 64-bit ARM support
  - .NET 6/7/8 (6 by default)
  - Bound executables on all platforms



Keyboarding (Mostly for New Users)

Issues:

- Dyalog IME does not work with Windows Universal Windows Platform applications
- New users report that "ctrl" is undesirable as the APL key

Immediate Solutions (v19.0):

- AutoHotkey / Downloadable keyboards for Windows which offer alternative "APL" keys (Alt, AltGr, etc). Thanks to Kimmo Linna!
- Backtick-style keyboards for all platforms

Longer Term:

- A new IME which offers a similar experience across supported platforms and works in and out of the IDE (this will take a bit longer)

Multiple session log files

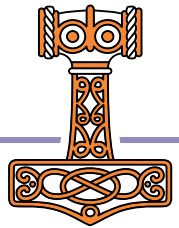
Earlier versions (default behaviour):

- All APL sessions read the same log file at startup, and update the same file at shutdown
- The last session to be closed overwrites the log file
- In rare circumstances, multiple starting or stopping APL interpreters crash due to conflicts reading or writing the log file

Version 19.0 default:

- Each APL session locks its log file
- Subsequent sessions will generate a new file name (e.g. default-2.dlf)
- If you regularly open several simultaneous sessions, they will have separate logs
- NB: In all versions, you can set the file name using LOGFILE=

Version 19.0 log files are JSON log files



Health Monitor

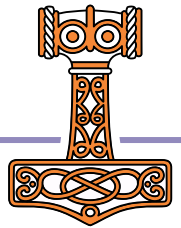
Experimental TCP-based monitor:

- Regular updates on (for example) :
  - CPU consumption
  - Memory statistics
  - Are any threads suspended?
  -)SI stack and Error information
- Notification on
  - untrapped error
  - ws compaction
- Exact execution location if "breadcrumbs" enabled
- Information about whether a RIDE connection is possible

Health Monitor Example

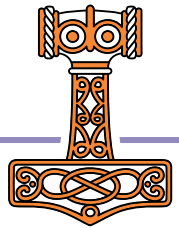

```
["PollFacts",{"Facts":["AccountInformation","Workspace","ThreadCount"],"Interval":5000,"UID":"1 1"}]
```

```
["Facts",
{"Facts": [ {
  "ID": 2, "Name": "AccountInformation",
  "Value": {
  "ComputeTime": 438,
  "ConnectTime": 46973,
  "KeyingTime": 0,
  "UserIdentification": 0
}},{
  "ID": 3, "Name": "Workspace",
  "Value": {
  "Allocation": 33882064,
  "AllocationHWM": 33882064,
  "Available": 2144207480,
  "Compactions": 2,
  "FreePockets": 186682,
  "GarbageCollections": 0,
  "GarbagePockets": 0,
  "Sediment": 2120,
  "Used": 3276168,
  "UsedPockets": 23209,
  "WSID": "CLEAR WS"
}},{
  "ID": 6, "Name": "ThreadCount",
  "Value": {
  "Suspended": 1,
  "Total": 2
}}
],
"Interval": 5000,
"UID": "1 1"
}]
```

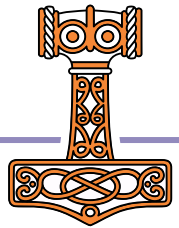
Timeouts and Interrupts

- `⎕FHOLD` now takes a left argument which is a timeout in milliseconds.
  - 1006 TIMEOUT is signalled if the lock cannot be acquired
- `⎕SIGNAL` allows signalling of 1006 TIMEOUT, which was mis-classified as an interrupt (which it is not)



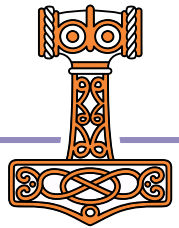
Token Range Reservation

- Independent components which use ⎕TGET/⎕TPUT for synchonisation can interfere with each other if they use the same token ranges.
- A new system function ⎕TALLOC allocates token ranges, allowing applications to avoid interference.
- ⎕TALLOC returns a single integer, granting the right to use floating-point token ids in the range **< n , n+1 >**
  - NB NOT INCLUSIVE – the integers can continue to be used by old style non-collaborating components
- Demo will hopefully clarify the design...



WS FULL Handling

- If a WS FULL leaves VERY little free space, the interpreter and IDE can malfunction
  - For example, a runaway recursion can leave only a few kilobytes of free workspace
  - **Error trapping may not be possible (system might just stop)**
- Version 19.0 allocates 1% of MAXWS as a buffer which is released on WS FULL
  - Allows WS FULL traps to be safely processed
    - (the reservation size is configurable)
    - After successful trap handling, space is re-acquired



⎕NCOPY / ⎕NMOVE Callbacks

- A 'ProgressCallback' variant allows you give the user a progress update

```
dest (⎕NCOPY ⎕ 'ProgressCallback' ('callbackfn' [larg])) src
```

- callbackfn will be called with a right argument of (**Function Event Info**)
- **Function** is '⎕NCOPY' or '⎕NMOVE'.
- **Event** is one of Start | Scan | Progress | Done
- **Info** is a namespace, containing
  - **Progress**: A number between 0 and Limit
  - **Limit**: The maximum value of Progress (nb could change)
  - **Last**: A vector of file names processed since the last call.
  - **Options**: See next Page

⎕NCOPY / ⎕NMOVE Callbacks

The callback function can set the following Options:

- **ScanFirst** (default: 1): Should code do a "scan pass" before moving/copying any files (gives "correct" Limit value).
- **Delay** (default: 0): ms to wait before the callback will be called again.
- **Skip** (default: 0): Specifies a number of files to process before the next call.
- **LastFileCount** (default: 1): How many of the latest filenames will be stored in the Last field.

The result of callbackfn should be 1 if processing should continue, else 0 (signals 1003 interrupt).

Source "as typed" by default

- "as typed": Preserve white space, numerical constants (well, everything) exactly as typed by the user.
- For several releases, Dyalog APL has preserved source "as typed" if a function or operator was created using ⎕FIX
  - Typically by Link, with source kept in a file outside the workspace

```
2 ⎕FIX 'file://myapp/foo.aplf'
```

- From version 19.0, the default is to preserve source "as typed" within the workspace for all fns and ops

Source "as typed" by default

From version 19.0, the default is to preserve source "as typed" **within the workspace** for all fns and ops

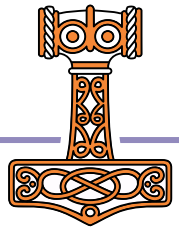
- **Also** applies to fns/ops with source in an external file
  - Allows recovery of source from saved wss with active Links
- AutoFormat is ignored in this mode
- Configurable: can be turned off if workspace is precious and you have a LOT of code
- An I-Beam will discard all source held in the workspace, for example when distributing workspaces

Source "as typed"

- ⎕ATX provides access to both canonical and verbatim sources:

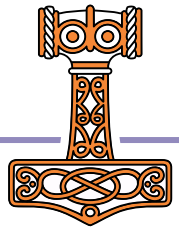
```
      2 ⎕FIX'R←DUP  X' 'R← X   X'

      ↑60 ⎕ATX 'DUP'
R←DUP  X
R← X   X

      ⎕CR 'DUP' ⍝ "Canonical Representation"
R←DUP X
R←X X

      (↑61 ⎕ATX 'DUP')≡⎕CR 'DUP'
1
```

Definition	60	Verbatim source (as typed)
	61	Normalised source (with AUTOFORMAT=1 and TAB_STOPS=4)
	62	Most precise available source (verbatim with fallback to normalised)



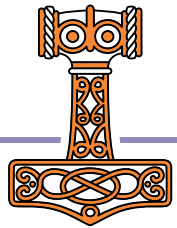
Multi-line input on by Default

- Multi-line input, which has been an experimental feature for a couple of releases, will be enabled by default
- Allows entering multi-line dfns and control structures directly in the session
-]demo c:\demos\2023\multiline



HTMLRenderer updates

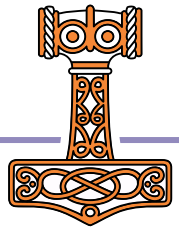
- Most important: Find a way to easily upgrade the Chromium Embedded Framework
  - In the medium term, turn the HTMLRenderer into an Open Source project to allow community participation
- Enhancements in v19.0
  - Support Multiple windows that take turns being modal
  - Hide title bar, add Handle property, a few more fixes ...



Source Code Management

Link 4.0 will be available with v19.0. Highlights include:

- A "Crawler" which will regularly compare the workspace and source files and detect differences
  - Will detect changes made using)COPY, assignment, □FX, etc
  - Alternative to the DotNet based "File System Watcher"
- Support for simple text vectors, vectors of text vectors, and character matrices in simple text files (not ".apla")
- The Cider project manager and the Tatin package manager will be bundled with v19.0
- More Dyalog-produced packages *will* appear



Source Code Management

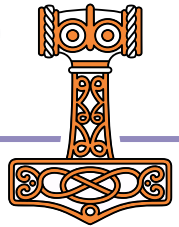
Link 4.0 will be available with v19.0. Highlights include:

- A "Crawler" which will regularly compare the workspace and source files and detect differences
  - Will detect changes made using)COPY, assignment, □FX, etc
  - Alternative to the DotNet based "File System Watcher"
- Support for simple text vectors, vectors of text vectors, and character matrices in simple text files (not ".apla")
- The Cider project manager and the Tatin package manager will be bundled with v19.0
- More Dyalog-produced packages *will* appear

Link (source)

Cider (projects)

Tatin (packages)



Tatin

tatin.dev/v1/packages

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

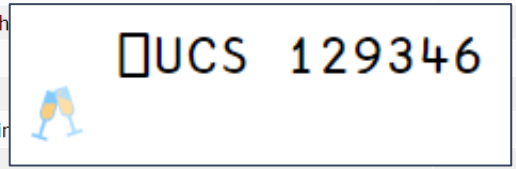
Tatin Registry

List of all packages (aggregated)

Package name	Description	Major versions	Link to project
aplteam-APLGit	Git interface from Dyalog APL via Git Bash	1	github.com
aplteam-APLProcess	Start an APL process from within Dyalog APL	1	github.com
aplteam-APLTreeUtils2	General utilities required by most members of the APLTree library	1	github.com
aplteam-CodeCoverage	Monitors which parts of an application got actually executed	1	github.com
aplteam-Compare	Allows comparing and merging objects in the workspace with a file or a file with another file	2	github.com
aplteam-CompareSimple	Allows comparing objects in the workspace with a file or a file with another file	2	github.com
aplteam-DateAndTime	Utilities related to Date and Time, including doing math	1	github.com
aplteam-DotNetZip	Zipping and unzipping with.NET Core on all major platforms	2	github.com
aplteam-EventCodes	Constants with meaningful names for Dyalog error codes	1	github.com
aplteam-Execute	Start a process from within APL	1	github.com
aplteam-FilesAndDirs	Utilities for doing gymnastics with files and directories	1	github.com

(... many more of Kai's packages skipped ...)

aplteam-WindowsEventLog	Tools to read from and write to the Windows Event Log	1	github.com
aplteam-ZipArchive	Zipping and unzipping with .NET on Windows and zip/unzip on oth	2	github.com
davin-DateTime	Easy calculations with dates	1	github.com
davin-FilePlus	Extend component files to use named components	1	github.com
davin-SQLFns	Easily create text SQL commands for use with any SQL program ir	1	github.com
davin-Tester	Simplified function-level testing of programs	1	github.com
dyalog-HttpCommand	Utility to execute HTTP requests	1	dyalog.github.io
dyalog-Jarvis	JSON and REST Web Service Framework	1	dyalog.github.io



Namespace Improvements

- Be more tolerant of errors when fixing namespace scripts
- Do not inject references to all sibling namespaces in a nested namespace script
  - (continue to do this for classes)
- When JSON creates namespaces, provide an option to **not** create a namespace hierarchy:

```
data←(¯1 ⎕JSON ⍠⎕NGET 'somefile.json').Data.Records
```

- Drawback: ## does not work within a namespace structure w/no hierarchy

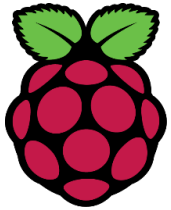
Arm64

64-bit ARM chips are appearing in places that Dyalog should support:

- M1 & M2 Macs
- Raspberry Pi – 64 Bit
- Amazon Web Services "Graviton"

Best price performance for compute-intensive workloads

aws



Differences between Arm and x86

The primary difference between Arm and x86 processors is the way that they execute their instructions. x86 systems are based on complex instruction set computer (CISC) designs, while Arm is based on RISC designs. In short, x86 processors can execute multi-step operations on each instruction, while Arm processors use a limited, but highly optimized, set of instructions.

This execution difference is not the only reason that Arm is beginning to outperform x86 in important markets. Arguably, that change is due equally to Arm's attractive price-to-performance ratio.

For example, consider the AWS Graviton processor, Amazon's first AWS Arm offering. Arm can save AWS users as much as 40 percent in costs for performance equal to that of Amazon's previous x86 cloud processors.

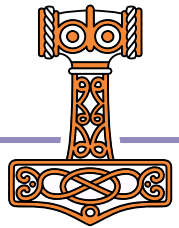
Arm's flexible licensing process comes into play as a critical difference between Arm and x86. Intel's x86 processors are still a proprietary chip, meaning that Intel is the sole creator of the physical hardware and

[Microsoft].NET

As .NET celebrates 20 years of existence, Microsoft is pushing everyone to move from proprietary Microsoft.Net Framework to the new open source, cross-platform .NET.

Name	Platforms	Version Numbers
Microsoft.NET Framework	Windows	1 2 4
.NET (previously ".NET Core")	Windows Linux macOS	3 5 6 7 8

Dyalog v18.0 added a bridge to .NET 3, to complement the 20 year old bridge to the .NET framework.



v19.0 .NET Bridge

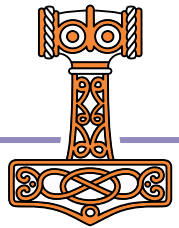
- Add support for .NET 6, 7, 8 ...
  - Tested with 6 (and 4 – aka ".NET Framework")
  - We will test with and support 8 when it is officially released late 2023
- Export APL code as .NET assemblies
  - (v18 .NET bridge can only *USE* .NET classes)
  - Will allow embedding APL code in .NET frameworks like ASP.NET Core, etc
- Support for named arguments to .NET methods
- Various other tweaks not yet finalised

.NET 6 is the current Long Term Support version of .NET

Bound Executables

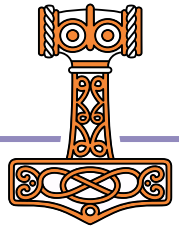
A bound executable is a file which combines an interpreter and a workspace into a single .exe file

- "Always" been available under Windows
- In v19.0, definitely also available for Linux
  - Maybe also MacOS (we just hired a Mac expert)
- In the longer term, we will look at encrypting and signing application code



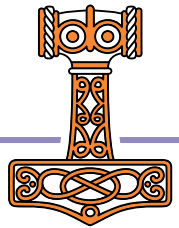
Highlights of Version 19.0

- Installing & Managing Your System
  - Keyboarding
  - Multiple session files
  - Health Monitor [demo]
- Building Production Systems
  - Timeouts
  - Token range reservation [demo]
  - WS FULL handling [demo]
  - NCOPY/NMOVE callbacks
- Developer Productivity / IDE
  - Source "as typed" by default [demo]
  - Multi-line input on by default [demo]
  - HTMLRenderer updates
  - Link 4.0: Crawler, Support for text data
  - Namespace Improvements
- Platform Support / Distribution
  - 64-bit ARM support
  - .NET 6/7/8 (6 by default)
  - Bound executables on all platforms



Demos

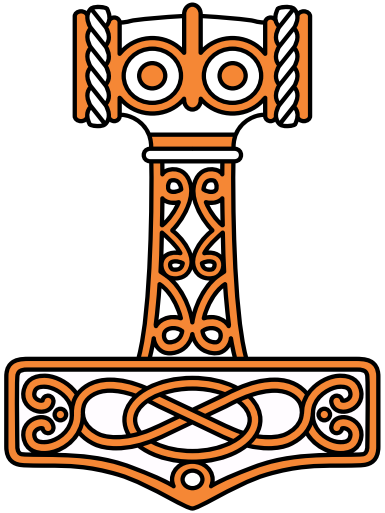
1. WS FULL handling
2. Token Allocation
3. Health Monitor
4. Multiline Input & JSON Log Files
5. Source as Typed by Default



DYALOG

FinnAPL @ Suomenlinna, April 18th 2023

Highlights of v19.0



Morten Kromberg