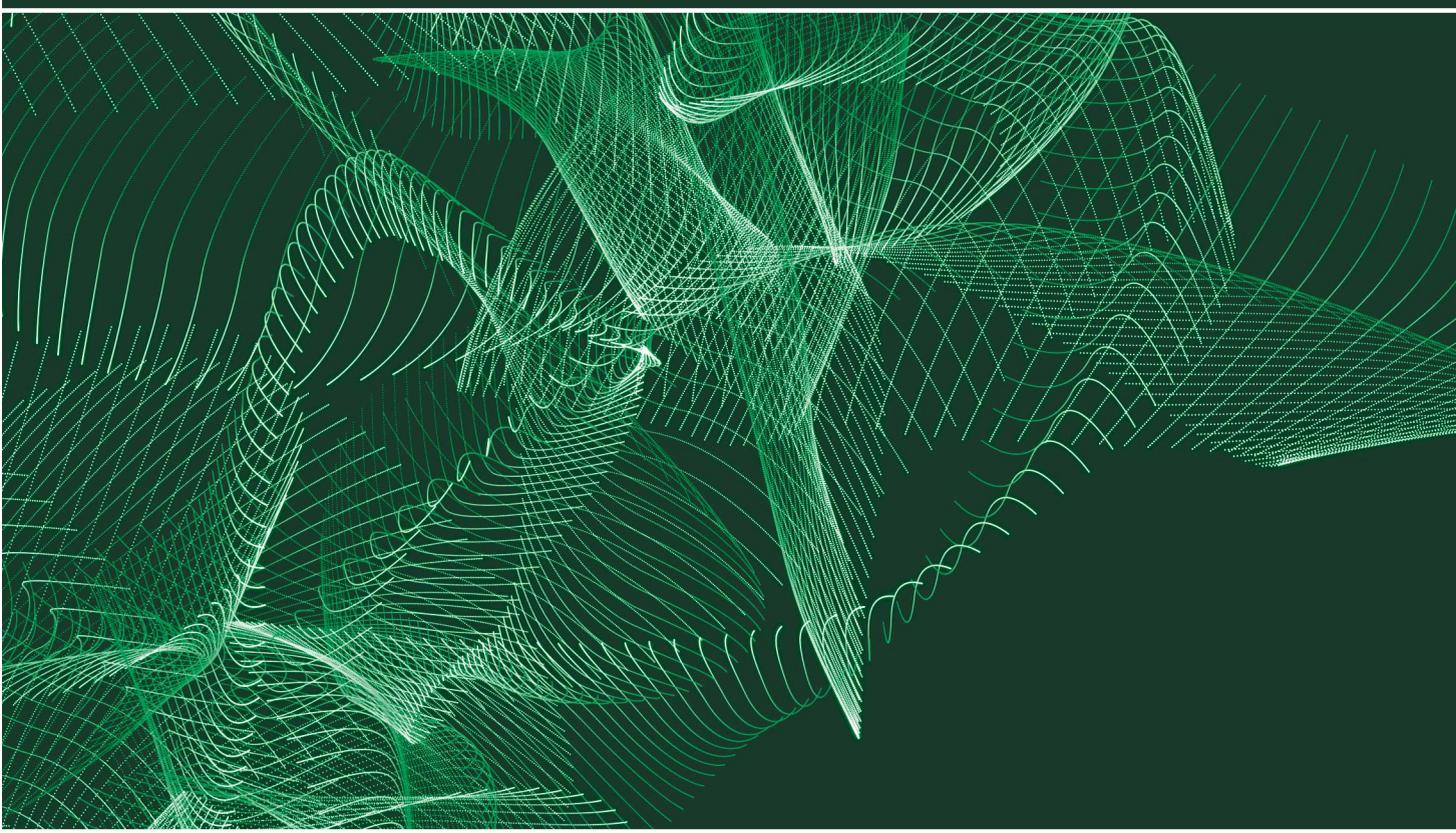


# External Workspaces (EXPERIMENTAL FUNCTIONALITY)

Dyalog version **16.0**



**DYALOG**

The tool of thought for software solutions

*Dyalog is a trademark of Dyalog Limited  
Copyright © 1982-2017 by Dyalog Limited  
All rights reserved.*

External Workspaces (EXPERIMENTAL FUNCTIONALITY)

Dyalog version 16.0  
Document Revision: 20170310\_160

*No part of this publication may be reproduced in any form by any means without the prior written permission of Dyalog Limited.*

*Dyalog Limited makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Dyalog Limited reserves the right to revise this publication without notification.*

*email: [support@dyalog.com](mailto:support@dyalog.com)  
<http://www.dyalog.com>*

#### **TRADEMARKS:**

*SQAPL is copyright of Insight Systems ApS.*

*Array Editor is copyright of davidliebtag.com*

*Raspberry Pi is a trademark of the Raspberry Pi Foundation.*

*Oracle®, Javascript™ and Java™ are registered trademarks of Oracle and/or its affiliates.*

*UNIX® is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.*

*Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.*

*Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.*

*Mac OS® and OS X® (operating system software) are trademarks of Apple Inc., registered in the U.S. and other countries.*

*All other trademarks and copyrights are acknowledged.*

# Contents

<b>1</b>	<b>About This Document</b>	<b>1</b>
1.1	Audience	1
1.2	Conventions	1
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Benefits Offered by External Workspaces	3
2.2	Fundamental Limitations	4
2.3	Temporary Limitations	5
2.4	Summary of Limitations	6
<b>3</b>	<b>Performance</b>	<b>7</b>
3.1	Loading	7
3.2	Code Execution	7
<b>4</b>	<b>Technical Reference</b>	<b>9</b>
4.1	Save External Workspace	9
4.2	Attach External Workspace	10
4.3	Assimilate External Workspaces	10
4.4	Detach External Workspaces	11
4.5	List External Workspaces	12
4.6	List Attached Names	12
<b>5</b>	<b>Technical Details</b>	<b>13</b>
5.1	External Workspaces are Read Only	13
5.2	Attaching, Assimilating and Detaching External Workspaces	14
<b>A</b>	<b>Worked Example</b>	<b>19</b>

# 1 About This Document

This document is intended as an introduction to external workspaces, introduced for the purpose of improving the performance of large applications while reducing their memory consumption and initialisation time.



The functionality and behaviour of external workspaces has not yet been finalised and is subject to change by Dyalog Ltd. at any time. External workspaces should be considered experimental; users are encouraged to experiment with them and provide feedback to Dyalog Ltd. regarding the benefits and shortcomings of the current features to help prioritise further work. Users should be aware that the features of external workspaces may change significantly based on feedback.

## 1.1 Audience

It is assumed that the reader has a reasonable understanding of Dyalog.

For information on the resources available to help develop your Dyalog knowledge, see <http://www.dyalog.com/introduction.htm>.

## 1.2 Conventions

Unless explicitly stated otherwise, all examples in Dyalog documentation assume that `□IO` and `□ML` are both 1.

Various icons are used in this document to emphasise specific material.

General note icons, and the type of material that they are used to emphasise, include:



Hints, tips, best practice and recommendations from Dyalog Ltd.



Information note highlighting material of particular significance or relevance.

-  Legacy information pertaining to behaviour in earlier releases of Dyalog or to functionality that still exists but has been superseded and is no longer recommended.
-  Warnings about actions that can impact the behaviour of Dyalog or have unforeseen consequences.

Although the Dyalog programming language is identical on all platforms, differences do exist in the way some functionality is implemented and in the tools and interfaces that are available. A full list of the platforms on which Dyalog version 16.0 is supported is available at [www.dyalog.com/dyalog/current-platforms.htm](http://www.dyalog.com/dyalog/current-platforms.htm). Within this document, differences in behaviour between operating systems are identified with the following icons (representing Mac OS, Linux, UNIX and Microsoft Windows respectively):



## 2 Introduction

While a standard Dyalog workspace (a **.dws** file) needs to be read by the interpreter and loaded in its entirety, an *external workspace* (a **.dwx** file) has a structure that allows it to be attached to the active workspace with a minimum of file operations. This significantly decreases the start time of application processes, especially when several processes run on the same machine.

When an external workspace is attached to the active workspace, a list of all the *names* of the functions, operators and variables that it contains is loaded into the active workspace. However, the definitions and values of these names are only paged into virtual memory the first time that the names are referenced, and they are not loaded into the active workspace unless their content is modified. In addition, because external workspaces are memory-mapped by the operating system, the definitions and values are shared by concurrent processes. This means that, if an external workspace is already in use by one Dyalog application, then the name list can be loaded from shared memory by another application; the same is true for any of the names that are already in use (unless the system is low on memory, in which situation memory mapped pages are flushed from memory).



The use of external workspaces is only supported on 64-bit Unicode interpreters and there are no current plans to extend this support. External workspaces are memory mapped, and they can only be attached by interpreters that use exactly the same memory layout as the system that generated them.

### 2.1 Benefits Offered by External Workspaces

Many large applications are currently forced to load more code than is necessary because it is difficult to predict precisely what code will be used. The main benefit of the external workspace technology is that applications only load code and data on demand.

**BENEFIT 1: *Significantly reduced start-up times for applications***

Similarly, computational sub-processes (such as isolates) can be launched in a fraction of the time that would otherwise be required.

**BENEFIT 2: *Reduced workspace usage***

Workspace size can be reduced or more space can be used to execute code more efficiently. An external workspace is materialised one page at a time, as memory is referenced. This means that the contents of comments, which are typically in a separate part of the file from the actual code lines, are usually only read from file if the code is edited. Similarly, running a compiled function is unlikely to require loading the source of the function into virtual memory.

**BENEFIT 3: *Reduced file I/O and memory consumption***

This is most apparent on machines that run several processes using the same code and is due to the sharing of memory-mapped files.

**BENEFIT 4: *More efficient application execution***

Objects residing in an external workspace remain outside the dynamic portion of the workspace unless they are modified. In some applications, this means that the complexity of the active workspace is significantly reduced. As a result, memory allocations are generally cheaper and less memory needs to be inspected and moved around when compactions occur.

## **2.2 Fundamental Limitations**

Despite the benefits offered by external workspaces, they will not replace the standard Dyalog workspace due to some fundamental limitations.

**RESTRICTION 1: *External workspaces are read only***

Multiple processes can memory-map external workspaces simultaneously; each process that uses an external workspace is using the contents of that external workspace directly as memory. This means that an external workspace cannot be updated while it is in use. Instead, when an application modifies an object that resides in an external workspace, a copy of the relevant part of the external workspace is made in memory and the original file is not modified.

**RESTRICTION 2:** *External workspaces each have a fixed virtual memory address*

An external workspace contains pointers to absolute memory locations contained within it. This means that the virtual memory address to which it is memory mapped must be fixed when the memory-mapping occurs. When an external workspace is created, a parameter specifies the virtual memory address at which it will be loaded and all pointers contained within it are adjusted to fit this address. If an application uses more than one external workspace, each external workspace must have a different address.

**RESTRICTION 3:** *External workspaces cannot be shared across architectures*

An external workspace cannot be converted in any way when it is used. This means that, unlike workspaces, component files and arrays transmitted using TCP objects or CONGA, external workspaces cannot be shared between different platforms or versions of Dyalog.

**RESTRICTION 4:** *64-bit Unicode only*

The benefits of memory mapping are only realisable in 64-bit address spaces. This, combined with the other fundamental limitations, means that external workspaces are only supported for 64-bit Unicode systems; there are no current plans to extend this support.

## 2.3 Temporary Limitations

There are several restrictions when using external workspaces that could be removed in future Dyalog versions.

**RESTRICTION 1:** *All objects saved must be visible from the root of the current workspace*

The following cannot be saved in an external workspace:

- GUI namespaces and their derivatives
- Functions created by starting an auxiliary processor
- External functions created using name association (□NA)
- □SM

**RESTRICTION 2:** *It is not possible to have more than 8 external workspaces simultaneously attached*

A maximum of 8 virtual memory addresses are available for external workspaces; these *slots* have identifiers 1 to 8. In a future release this limit could be significantly increased, but the fundamental issue of needing a separate slot for each external workspace

simultaneously will remain.

**RESTRICTION 3:** *Cannot )SAVE a workspace that has external workspaces attached*

It is not possible to )SAVE the current workspace if any external workspaces are attached (they must be assimilated or detached first – see *Section 4.3* and *Section 4.4* respectively).

**RESTRICTION 4:** *Attaching an external workspace containing namespaces copies all the namespaces*

Attaching external workspaces results in data being copied from the external workspace as needed (see *Section 4.4*). However, namespaces are always copied.

**RESTRICTION 5:** *Only certain content can be saved in an external workspace*

The content of an external workspace is limited to namespaces, nested arrays, simple arrays, tradfns, tradops, dfns, dops and derived functions (futures and external variables are instantiated and become arrays). If other content (for example, .NET objects, shared variables and COM objects) is present in a workspace then that workspace cannot be saved as an external workspace (see *Section 4.1*).

## 2.4 Summary of Limitations

Fundamental limitations:

- External workspaces are read only
- External workspaces have a fixed virtual memory address
- External workspaces cannot be shared across architectures
- 64-bit Unicode only

Temporary limitations:

- All objects saved must be visible from the root of the current workspace
- It is not possible to have more than 8 external workspaces simultaneously attached
- Cannot )SAVE a workspace that has external workspaces attached
- Attaching an external workspace containing namespaces copies all the namespaces
- Only certain content can be saved in an external workspace

## 3 Performance

The main purpose of external workspaces is to reduce the execution time and memory consumption of APL applications.

### 3.1 Loading

Unless an application uses a large proportion of its constituent code soon after start-up, it is significantly faster to start that application in a nearly empty workspace and attach external workspaces containing the rest of the code. This also reduces the memory footprint of the application.

The performance improvement is most noticeable in an environment where several processes run the same application on a single machine, for example, applications using isolates and/or running on Citrix servers or other terminal servers. This is because:

- external workspaces are memory-mapped; once one process has caused a part of the application to be paged in, subsequent processes have very fast access to the same part of that application.
- as the memory-mapped files are shared, only a small part of a function needs to be copied into each active workspace that shares them; this reduces the overall memory usage across all processes.

### 3.2 Code Execution

Code or data that is located in an external workspace is paged into virtual memory the first time that it is used. This incurs a performance overhead; however, subsequent calls to that code or data (or anything else on the same page) by the same or any other process do not experience the same performance impact.

Similarly, the first time that the content of a name (function, operator or variable) in an external workspace is amended also involves a performance overhead (the content of an

external workspace is read-only; modifying the content of a name causes it to be copied into the active workspace). However, subsequent writes to that the content of that name by the same process do not experience the same performance impact.

Not only do subsequent calls/writes not experience the same performance impact, their performance is often improved when compared with performing the same operations without external workspaces. This is due to the workspace memory manager running more efficiently when it has a smaller set of data in the main workspace than if everything was in the main workspace. Specifically:

- more workspace is available for application data, making it easier for memory manager algorithms to allocate memory.
- the contents of the external workspaces are ignored by compaction and garbage collection algorithms.

## 4 Technical Reference

The operations that comprise the external workspace mechanism are implemented using three I-Beams – 8659I, 8666I and 8667I. Specifically:

- `[names](8667I){slot}{file}`  
save external workspace – see *Section 4.1*
- `[nameclasses](8666I){file}`  
attach external workspaces – see *Section 4.2*
- `(8666I)[]NULL`  
assimilate external workspaces – see *Section 4.3*
- `(8666I)Op<' '`  
detach external workspaces – see *Section 4.4*
- `R←(8659I)⊘`  
list external workspaces – see *Section 4.5*
- `R←{slot}(8659I){ncs}`  
list attached names – see *Section 4.6*

### 4.1 Save External Workspace

**Purpose:** Saves an external workspace.

**Syntax:** `[names] (8667I) {slot} {file}`

where:

- `names` is a vector of character vectors or a matrix specifying the names to save; this list of names of functions, operators and variables restricts the names in the external workspace that are saved.
- `slot` is the slot identifier (an integer in the range 1-8) for the unique fixed virtual memory address at which to load the external workspace.
- `file` is a character vector of the filename for the external workspace. If a filename extension is not provided, then `.dwx` is used. If a file of this name already exists, or the file cannot be created for any reason, then the operation will fail.



A multi-user development team might need a strategy for creating (and attaching) cycles of external workspaces as external workspaces could remain in use for some time by members of the development team. This should not be an issue with distributed applications.

This creates an external workspace, optionally based on a list of names of functions, operators or variables. Restrictions apply to the location and structure of objects that can be placed into an external workspace; most importantly, the names must all be visible in the root (#) of the active workspace. For a complete list of restrictions, see *Section 2.4*.

## 4.2 Attach External Workspace

**Purpose:** Attaches one or more external workspaces to the active workspace.

**Syntax:** `[nameclasses] (8666I) {file}`

where:

- `nameclasses` is a list of nameclass identifiers to be brought over (cannot include sub-classes). The default is 2 3 4 9 (variables, functions, operators and namespaces respectively).
- `file` is a vector of character vectors of external workspaces to load, or a single character vector (a character scalar is not acceptable). If filename extensions are not provided, then `.dwx` is used.

The effect of attaching external workspaces is analogous to performing a )PCOPY (protected copy) from the external workspace files, that is:

- names that already have a definition are preserved unaltered; if the same name appears in more than one external workspace, then the files are searched in the specified order and the first occurrence of the name is used.
- names in attached files immediately affect the results of system functions that provide metadata, such as `□NL` or `□NC`.

If any external workspaces are already attached, then they are detached from the active workspace before new external workspaces are attached (see *Section 4.4*). Multiple external workspaces cannot be attached using separate calls to `8666I`.

## 4.3 Assimilate External Workspaces

**Purpose:** Copies referenced objects in the external workspaces into the active workspace.

**Syntax:** `(8666I)□NULL`

When the right argument to `(8666I)` is `NULL`, all referenced objects in the external workspaces are copied into the active workspace. The active workspace then contains all the code and data that was visible to it when the external workspaces were attached; it can then be saved and used independently of the external workspaces. The external workspaces that are attached to the active workspace are then disconnected from the active workspace.

 Significant space might be required to assimilate all the code in the external workspaces into the active workspace. If a `WSFULL` error occurs then the operation will fail; it cannot be rolled back and leaves the workspace in an indeterminate (but consistent) state. In this situation, the external workspaces are not disconnected from the active workspace as doing so could result in further errors.

## 4.4 Detach External Workspaces

**Purpose:** Detaches all external workspaces from the active workspace.

**Syntax:** `(8666I)0ρ<' '`

When the right argument to `(8666I)` is `0ρ<' '` (that is, a zero-length list of names), any existing attached external workspaces are detached.

 Detaching external workspaces results in data being copied from the external workspace as needed. However, namespaces are always copied when the external workspace is first attached.

Before an external workspace is disconnected from the active workspace:

- if a name that was brought into the active workspace when the external workspace was attached has not had its associated code/data changed, then the name is expunged from the active workspace.
- if a name in the active workspace embeds references to objects residing in an external workspace, then the entire definitions of the referenced objects are copied (assimilated) into the active workspace. This includes (for example), tacit functions that are derived from functions in an external workspace and arrays that contain references to data in an external workspace. These objects must still be functional following the disconnect.

 As external workspaces are read-only, they cannot be updated while they are in use. Instead, if an external workspace needs to be updated, it must be rebuilt. When a new version of an external workspace becomes available, anyone using the old version should detach it and attach the new one instead as soon as is practical.

## 4.5 List External Workspaces

**Purpose:** Lists the external workspaces that are attached to the current workspace.

**Syntax:** `R←(8659I)⊖`

where:

- R is a 2-column matrix listing the external workspaces that are attached to the current workspace:
  - [ ; 1 ] is the slot identifier for the fixed virtual memory address of the external workspace.
  - [ ; 2 ] is the full name of the external workspace file that was loaded.

The rows of the matrix (one row for each external workspace) are ordered to correspond to the order in which the external workspaces were specified when they were originally attached, that is, in the right argument to `8666I` (see *Section 4.2*).

## 4.6 List Attached Names

**Purpose:** Lists the names in the external workspaces identified by the specified memory address.

**Syntax:** `R←{slot}(8659I){ncs}`

where:

- `slot` is the slot identifier (an integer in the range 1-8) for the unique fixed virtual memory address of the external workspace.
- `ncs` is an integer vector that would be a valid right argument to `□NL`; it identifies the nameclasses and subclasses for which the names should be listed.

R lists the names in the external workspace identified by `slot`. If any element of `ncs` is negative, then positive values in `ncs` are treated as if they were negative and R is a vector of character vectors. Otherwise, R is a simple character matrix.

## 5 Technical Details

This section contains discussions intended to clarify the functionality of external workspace support.

### 5.1 External Workspaces are Read Only

An external workspace is a read-only repository. Items within it can be modified, but doing so can result in data being copied into the main workspace.

Consider these cases where item A is modified:

1. A is a function
  - $B \leftarrow A$  will introduce a new name B into the main workspace but no new data.
  - When A is edited or otherwise re-fixed, the new version will be stored in the main workspace.
2. A is a simple array such as 1 2 3 4.
  - $B \leftarrow A$  will introduce a new name B into the main workspace but no new data.
  - $C \leftarrow A, 1$  will introduce a new name C and new data into the main workspace.
  - $A, \leftarrow 1$  will create new data in the main workspace.
3. A is a nested array such as 'AB' 'CD'.
  - $B \leftarrow A$  will introduce a new name B into the main workspace but no new data.
  - $A[1] \leftarrow 'XY'$  will introduce some new data into the main workspace.

In each of these cases, the content of the attached external workspace remains unaltered. This means that, if names of items in an external workspace are expunged using `□EX` and the external workspace(s) are detached and reattached, then the items in the external workspace will be restored to their original values. The only way to change the values in an external workspace is to recreate the entire file.

Although an external workspace can contain data, these values should either be constants or initial values for structures that will be copied into the workspace as soon as the application modifies them.

## 5.2 Attaching, Assimilating and Detaching External Workspaces

When one or more external workspaces is attached, the following rules apply:

- When items with the same name exist in multiple workspaces, the one that is used in the active workspace is the first one found when going through the workspaces in the following order:
  1. the active workspace
  2. the external workspace specified first when attaching (see *Section 4.2*)
  3. the external workspace specified second when attaching, etc.
- When the external workspaces are assimilated:
  - all references to the external workspace are resolved by copying data from the external workspace to the active workspace as required.
- When the external workspaces are detached:
  - names in the active workspace that reference data in the external workspace are deleted (namespace references are not deleted).
  - all remaining references to the external workspace are resolved by copying data from the external workspace to the active workspace as required.

In a future version, namespaces might follow the same rules as other names in the workspace. A namespace would be analogous to an array – if any item within it is amended, then the entire namespace would be considered to have been amended. This would mean that if any item in a namespace in an external workspace is updated or removed, or another item is added, then the entire namespace would be retained when the external workspace is detached, otherwise none of the namespace would be retained when the external workspace is detached.

EXAMPLE

The active workspace MAIN is populated using the following assignments:

```
FN1 ← {ω × 1}
FN2 ← {ω × 2}
NS1 ← □NS ''
NS1.A ← 1
```

Name	Parent	Value
FN1	#	{ω × 1}
FN2	#	{ω × 2}
NS1	#	Namespace ref
A	NS1	1

External workspace DWS1 is populated using the following assignments:

```

FN1 ← {ω × 1.1}
FN3 ← {ω × 3}
V ← 'AB' 'CD'
NS1 ← □NS ''
NS1.A ← 2
NS1.B ← 3

```

Name	Parent	Value
FN1	#	{ω × 1.1}
FN3	#	{ω × 3}
V	#	'AB' 'CD'
NS1	#	Namespace ref
A	NS1	2
B	NS1	3

External workspace DWS2 is populated using the following assignments:

```

FN3 ← {ω × 3.1}
FN4 ← {ω × 4}
NS2 ← □NS ''
NS2.A ← 4
NS3 ← □NS ''
NS3.A ← 5

```

Name	Parent	Value
FN3	#	{ω × 3.1}
FN4	#	{ω × 4}
NS2	#	Namespace ref
A	NS2	4
NS3	#	Namespace ref
A	NS3	5

After attaching DWX1 and DWX2 (in that order) to MAIN the following will be accessible:

Name	Parent	Value	Location of Value	Notes
FN1	#	{ $\omega \times 1$ }	WS	FN1 in DWX1 is inaccessible
FN2	#	{ $\omega \times 2$ }	WS	
FN3	#	{ $\omega \times 3$ }	DWX1	FN3 in DWX2 is inaccessible
FN4	#	{ $\omega \times 4$ }	DWX2	
V	#	'AB' 'CD'	DWX1	
NS1	#	Namespace ref		
A	NS1	1	WS	NS1 . A and NS1 . B in DWX1 are inaccessible
NS2	#	Namespace ref		
A	NS2	4	DWX2	
NS3	#	Namespace ref		
A	NS3	5	DWX2	

Following these assignments:

FN3 ← { $\omega \times 3.2$ }  
 FN5 ← FN4  
 V[1] ← c'XY'  
 NS2.B ← 6

The following are now accessible:

Name	Parent	Value	Location of Value	Notes
FN1	#	{ $\omega \times 1$ }	WS	
FN2	#	{ $\omega \times 2$ }	WS	
FN3	#	{ $\omega \times 3.2$ }	WS	Updated value
FN4	#	{ $\omega \times 4$ }	DWX2	
FN5	#	{ $\omega \times 4$ }	DWX2	

Name	Parent	Value	Location of Value	Notes
V	#	'XY' 'CD'	Split between WS and DWX1	
NS1	#	Namespace ref		
A	NS1	1	WS	
NS2	#	Namespace ref		
A	NS2	4	DWX2	
B	NS2	6	WS	New value
NS3	#	Namespace ref		
A	NS3	5	DWX2	

The external workspaces are now disconnected. This is achieved either by assimilating them into the active workspace or by detaching them; the result of each of these operations is shown below.

Following assimilation of the external workspaces, the main workspace will contain:

Name	Parent	Value	Notes
FN1	#	{ $\omega \times 1$ }	
FN2	#	{ $\omega \times 2$ }	
FN3	#	{ $\omega \times 3.2$ }	
FN4	#	{ $\omega \times 4$ }	Copied into WS
FN5	#	{ $\omega \times 4$ }	Copied into WS
V	#	'XY' 'CD'	Partially copied into WS
NS1	#	Namespace ref	
A	NS1	1	
NS2	#	Namespace ref	
A	NS2	4	Copied into WS

Name	Parent	Value	Notes
B	NS2	6	
NS3	#	Namespace ref	
A	NS3	5	Copied into WS

Alternatively, following detachment of the external workspaces, the main workspace will contain the following values:

Name	Parent	Value	Notes
FN1	#	{ $\omega \times 1$ }	
FN2	#	{ $\omega \times 2$ }	
FN3	#	{ $\omega \times 3.2$ }	
FN5	#	{ $\omega \times 4$ }	Copied into WS
V	#	'XY' 'CD'	Partially copied into WS
NS1	#	Namespace ref	
A	NS1	1	
NS2	#	Namespace ref	
A	NS2	4	Copied into WS, namespace has changed
B	NS2	5	
NS3	#	Namespace ref	All namespaces in external workspaces are retained (see <i>Section 2.3</i> )
A	NS3	5	Copied into WS

## A Worked Example

This appendix comprises an annotated example that demonstrates the use of some of the cases of the I-Beam functions described in *Chapter 4* (examples of *assimilate* and *detach* are not included).

First, load the dfns workspace:

```
)LOAD dfns
C:\...\ws\dfns.dws saved Sun Apr 12 17:18:38 2015
```

An assortment of D Functions and Operators.

```
tree #                A Workspace map.
↑~10↑↓attrib □nl 3 4  A What's new?
⌘notes find 'Word'   A Apropos "Word".
□ed'notes.contents'  A Workspace overview.
```

Now compute the size of all the functions, variables and namespaces in the workspace (approximately 6 MB):

```
+/□SIZE □NL 110
5947936
```

Define a helper function called saveDWX to create an external workspace:

```
saveDWX←8667I
```

Create an external workspace containing everything in the dfns workspace, mapped at virtual memory address 1:

```
saveDWX 1 'dfns.dwx'
```

If this fails due to the file already existing, then erase it and try again:

```

    saveDWX 1 'dfns.dwx'
DOMAIN ERROR: External workspace already exists
    saveDWX 1 'dfns.dwx'
^
    □DELETE 'dfns.dwx'
    saveDWX 1 'dfns.dwx'

```

Clear the workspace and define two new helper functions, attachDWX and listDWX:

```

    )CLEAR
clear ws

    attachDWX←8666I
    listDWX←8659I

```

Attach the external workspace to the active workspace and compute how much workspace was consumed in doing so:

```

    wa←□WA
    attachDWX 'dfns.dwx'

    □WA-wa
~64320

```

(rather than consuming space, 64 KB was released due to workspace reorganisation)

Check how many names are now visible in the workspace and call the `easter` function to find the date for Easter Sunday in 2015 (to prove that functions in the attached workspace can be run successfully):

```

    ≠□NL ι10
273

    easter 2015
20150405

```

List the external workspaces that are attached to the active workspace (the first column shows the slot identifier). Next, display the first 5 names made available by the external workspace in slot identifier 1:

```

    listDWX θ
1 dfns.dwx

    5↑1 listDWX ι10
Cholesky

```

```

NormRand
UndoRedo
X
_fk

```

Finally, verify that the result of `ρNL` and the names exposed by the external workspace are identical (the only difference should be the three names defined since the `)CLEAR` operation):

```

      (ρNL i10)≡1 listDWX i10
0
      ρ1 listDWX i10
270 12
      ρρNL i10
273 12
      (↓ρNL i10)~↓1 listDWX i10
attachDWX      listDWX      wa

```