DYALOG

The tool of thought for expert programming

Dyalog™ for Windows

Workspace Transfer

Version 12.1.0

Dyalog Limited

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Tel: +44 (0)1256 830030
Fax: +44 (0)1256 830031
email: support@dyalog.com
http://www.dyalog.com

Version 12.1.0 produced on 2011-03-29

First Edition July 2009

Contents

Workspace Transfer

Introduction

It is often necessary to either transfer Dyalog APL workspaces from one machine to another, or to transfer workspaces created by another version of APL to Dyalog APL.

Since the internal structure of an APL workspace is dependent upon the architecture of the machine on which it was created, and the version of APL that was used to create it, it is not always possible to transfer a workspace directly. It must first be transformed into a format that is common to both source and target environments. This can then be transferred, and used to create a workspace on the target machine.

The following sections describe the steps that are involved in moving workspaces between machines and different versions of any APL. APL component files are transferred in a similar way to workspaces; each component is read into an APL variable which is then transferred.

The subsequent sections describe specific examples of this process. The basic technique is the same in most cases. Doubtless better techniques can be used, but the following examples are useful as a basis from which to work.

The problems of conversion from one version of APL to Dyalog APL are not discussed in this section. Nor are the procedures for transferring from Dyalog APL to other APLs. However, using the techniques discussed in this section, it is a simple (if tedious) matter to produce software that will transfer in the opposite sections.

Where possible, the tools required to transfer to Dyalog APL are supplied with the product; where this is impractical, listings of the appropriate software is given or the location where they can be found.

General Techniques

Create a Export File and Read it back

If we can produce a text file on the source machine that contains all of the statements necessary to recreate the workspace, we could execute each line of this file on the target machine using Dyalog APL, thereby recreating the workspace in the correct format. The process of creating the file is called EXPORT and the process of reading it is called IMPORT.

How do we transform a workspace into a text format? Functions are easy as we can just produce a listing of them. Variables are more difficult; we must produce lines of text that when executed, recreate the variables with the correct contents, type, shape and depth.

Any APL programmer should be capable of writing the simple APL system required to perform both of these tasks; however, Dyalog provides methods to do this on the Dyalog side.

Transfer to Target Machine

Once this export file has been created, it should be transferred from the source to the target machine by whatever means.

However the transfer is done, the file should arrive on the target machine EXACTLY as it left the source machine; i.e. the transfer mechanism should not perform any conversion or translation. Note that such files produced by are raw text file and may contain any characters in the range 0 to 255; some network transfer packages react badly to characters in the range 0 to 32 unless you ask that the file is transferred in transparent mode.

Methods used

Dyalog provides methods to *export* a workspace by creating a file from looking at it and other methods to *import* a workspace from reading the same kind of file. There are several ways to do this. One of these methods is built on an 80 column punched card format that APL2 came up with in the 1980s. Because one of the designs created a file with an ATF extension (for **A**PL **T**ransfer **F**ile) this format is sometimes known as the ATF format. There are in fact at least 2 variants to this format and one of them comes in a flavour that uses a DXF extension instead of ATF.

Other APL vendors have adopted these formats and it is common for those vendors to offer a method of exporting a workspace using that ATF format (although they may be more flexible and forego the ATF extension for example).

There are restrictions with these formats as it wasn't made for the new objects Dyalog APL now uses like Namespaces and Classes. Since the objects the ATF format supports is a close subset of what Dyalog supports it is usually not a major problem to import other vendors' workspaces. The problem is when one wants to export a Dyalog workspace containing newer objects and recreate it even on Dyalog APL itself (e.g. on a earlier version)

To circumvent this problem Dyalog also offers its own extended version which allows to export, still with restrictions (no GUI object can be exported for example), Dyalog workspaces and component files.

Technicalities

Dyalog supplies 2 workspaces to deal with the DXF files (1 in and 1 out), 1 to deal with the ATF files (in only) and 1 workspace to deal with the extended format.

It also supplies 2 sets of user commands (both in and out) for the DXF/ATF files and the extended format.

All will be discussed here.

Create an export file from a workspace or APL file

Using the DXF format

Each object's canonical representation is sent one by one, untranslated, to an export file. To do this you can use the DWSOUT workspace or the]OUT user command.

Using the ATF format

There is no way to produce an ATF file from Dyalog.

Using the extended Dyalog format

Each object's canonical representation is sent one by one, untranslated, to an export file. To do this you can use the XFRCODE workspace or the]OUTX user command.

Import a workspace or APL file by reading an export file

Using the DXF format

Each object's definition is read, one by one, and recreated in the workspace. To do this you can use the DWSIN workspace or the]IN user command.

Using the ATF format

Each object's definition is read, one by one, and recreated in the workspace. To do this you can use the ATFIN workspace or the]IN user command.

Using the extended Dyalog format

Each object's definition is read, one by one, and recreated in the workspace. To do this you can use the XFRCODE workspace or the]INX user command.

Exporting

There are essentially two ways to export code: using a workspace or using a user command. The user commands are basically a cover for the workspaces. In each case there are also two alternatives: one dealing with older (pre-namespaces) version of Dyalog APL and one dealing with newer version. All four are described here.

Other APL vendors have different methods of exporting workspaces. Some vendors of APL provide a workspace named DWSOUT or a system function to do the work or, as in APL+'s case, a user command.

The DWSOUT Workspace

A typical DWSOUT workspace contains a function ⍙⍙DWSOUT that is effectively a very simple workspace lister, which lists the contents of variables as well as functions. Sometimes, a function , ⍙⍙DCFOUT, is also supplied which lists the contents of component files by reading each component into a variable, and listing that variable using the same techniques as ⍙⍙DWSOUT.

Each version of ⍙⍙DWSOUT in these APLs is essentially the same. Hence, the description of the use of ⍙⍙DWSOUT given below is valid for all versions. The particular restrictions of each version are discussed in the relevant sections.

The DWSOUT workspace creates files with a DXF extension

Exporting a Workspace

The function ⍙⍙DWSOUT is used to transfer entire workspaces, or named objects from a workspace. The full syntax of the function call is:

```
{names} ⍙⍙DWSOUT wsname
```

`wsname` is a simple character vector containing the name under which the workspace is to be `)SAVE`d on the target system. `names`, if present, is a character matrix containing the names of the objects to be transferred. The default is the entire workspace.

A file called **wsname**, followed b **.DXF**[1] is created, and listings of the requested objects, system variables and constants, and some control statements are appended to the file.

[1] Dyalog will always add the DXF extension

Restrictions

The following restrictions apply to all versions of ⍙⍙DWSOUT:

- The state indicator must be empty.
- The names of objects to be transferred may not begin with ⍙⍙.
- Locked functions cannot be transferred.
- Class 9 objects like Namespaces are not transferred
- []NULLS and composed functions will not be transferred
- Characters not in []AV cannot be dealt with

Example

Create a text file called NEWWS.DXF containing a complete representation of a workspace called MYWS.

```
      )LOAD MYWS
saved ...

      )FNS
FOO GOO HOO

      )VARS
A B C

      )COPY DWSOUT
saved ...

      ⍙⍙DWSOUT 'NEWWS'
Functions ...
FOO, GOO, HOO,
Variables ...
A, B, C,
System Variables ...
Finished
```

Exporting a Component File

Where applicable, the function ⍙⍙DCFOUT is used to transfer entire component files. The full syntax of the function call is:

```
{options} ⍙⍙DCFOUT filename
```

filename is a simple character vector containing the name of the file to be created on the target system.

options, if present, is either a simple text vector containing the name of the APL file to be transferred, or a two element vector whose first element contains the name of the APL file, and whose second contains a passnumber. If **options** is omitted, the APL file name is assumed to be the same as **filename** with a passnumber of 0.

A file called `filename.DXF` is created. Each component in the file is read and assigned to a variable; this variable is then transferred using the same techniques as ⍙⍙DWSOUT. The access matrix is read and transferred in the same manner.

Example

Create a text file called `PJB.DXF` containing a complete listing of a component file called PJB.

```
      )LOAD DWSOUT
saved ...

      ⍙⍙DCFOUT 'PJB'

Components ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Finished
```

Using the user command]OUT

Under Dyalog and APL+ you can also use the user command OUT to export workspaces only. This may be simpler. The syntax is different:

```
]OUT  filename -obj=a,b,c…
```

The file **filename.dxw**[2] will contain a complete representation of the current workspace unless **–obj** is used (in Dyalog only) to specify the objects to transfer.

[2] filename.**atf** under APL+

The XFRCODE workspace

If the workspace to export contains more exotic objects like namespaces you can use another method to export the workspace.

This workspace only exists in Dyalog. Older versions of this workspace exist for other APLs and they are doing essentially the same thing: export all the objects in the workspace into a native file. Those workspaces are not supplied with Dyalog APL but can be found on the Internet[3]

Exporting a Workspace

Under Dyalog APL the function to call is

```
xfr.Δxfrto[3] '\tohost\filename  [switches]'
```

The argument is a string specifying the name of the file to produce followed optionally by some switches:

-q do not display banner
-obj= objects to transfer (default all)

The extension XDW will be added to the filename.

Example:

```
    )xload YOURWS
    )copy XFRCODE
    xfr.Δxfrto '\tmp\dbutil'
* XFR version 3.08
17 objects transferred
```

Exporting a component file

Component files can also be exported.

Use the same function with the same syntax and the following switches:

-file[=] use the APL file named

[3] Under other APLs the workspace name is XFRPC and the function **Δxfrto** resides in the workspace as there are no namespaces in those APLs. The syntax is the same but the switches start with a / instead of a – and the filename extension varies from one APL to another. See HTTP://www.milinta.com/xfrpc.htm for details.

-range=n1[,n2] transfer components n1 to n2
-lock= file lock to use for reading the file, if any

Example:

```
)copy XFRCODE
xfr.Δxfrto '\tohost\DOSfilename -file=MyFile'
```

Using the user command]OUTX

You can also use this command to export workspaces and component files. The syntax is:

```
]OUTX  \path\to\file  [switches]
```

where switches is the same as above:

-q do not display banner
-obj= workspace objects to transfer (default all)
-file[=] use the APL file named instead of the workspace
-range=n1[,n2] transfer components n1 to n2
-lock= file lock to use

If the export is done for a workspace the filename's extension will be **xdw**. If it is done for a file the extension will be **xdf**.

Example:

Export objects **A, BC** and **Def** in the current workspace to file **\tmp\myws.xdw**:

```
]outx  \tmp\myws  -obj=A BC Def
```

The command is smart enough to export :*included* namespaces and :*Based* classes first in order to be able to recreate them properly when reading back the file.
If you use **–obj** to specify the objects to export make sure you list all the objects needed in the proper order otherwise recreating them may not be possible.

Example:

Export quietly cpts 10 to 20 of APL file **MyFile** to native file **\temp\MF.xdf** :

```
]outx  \temp\MF  -q  -range=10 20  -file=MyFile
```

Importing

There are essentially two ways to import code: using a workspace or using a user command. The user commands are basically a cover for the workspaces. In each case there are also two alternatives: one dealing with older (pre-namespaces) version of Dyalog APL and one dealing with newer version. All four are described here.

The DWSIN Workspace

This workspace is used to import workspaces and files created by the DWSOUT workspace.

Importing a Workspace

DWSIN contains function ⍙⍙DWSIN that can process files produced by the DWSOUT workspaces of different APLs.

⍙⍙DWSIN is used to read the files produced by ⍙⍙DWSOUT and ⍙⍙DCFOUT. The file is read to recreate the workspace or component file. The full syntax of the function call is:

```
{source} ⍙⍙DWSIN filename
```

`filename` is a simple character vector containing the name of the file to be processed. This file must have been produced by a previous call of ⍙⍙DWSOUT. Note that the suffix "`.DXF`" is appended to the given `filename` so you should rename your file if the extension is different.

`source`, if present, is a simple text vector containing the name of the source APL, taken from the set `'DYALOG'`, `'APLPLUS'`, `'STSCMF'` or `'VSAPL'`. If `source` is omitted, the default is `'DYALOG'`.

⍙⍙DWSIN reads the file, applying the relevant translation from the source APL to Dyalog APL, to recreate the objects.

If an object cannot be recreated because of badly formed lines in the file, then that object is ignored and a warning message printed. The names of such objects are held in a variable `WontFix`.

Example

Create a Dyalog APL workspace from the file `NEWWS.DXF`, which was created by Dyalog APL possibly on another machine.

```
      )LOAD DWSIN
saved ...

      ΔΔDWSIN 'NEWWS'
Processing script ...
Functions & Operators ...
FOO, GOO, HOO,
Variables ...
System variables ...
****************************
**** Workspace name is NEWWS
**** REMEMBER TO )SAVE IT !!!
****************************
```

Importing a Component File

The same **ΔΔDWSIN** function is used to import APL files created by the **ΔΔDCFOUT** function.

Example

Create a Dyalog APL component file from the file `PJB.DXF`, which was created by APL*PLUS/PC.

```
      )LOAD DWSIN
saved ...

      'APLPLUS' ΔΔDWSIN 'PJB'
Processing script ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Finished.
```

The ATFIN Workspace

This workspace is used to import workspaces created by other APLs in ATF format. Other vendors have been using this format but have not always kept a consistent format. The ATFIN workspace supplied only accepts ATF files of a specific record length. The user command]IN is a bit more permissive and should detect the record length.

Importing a Workspace

ATFIN contains function **ΔΔatfin** that can process files produced by different APLs. The full syntax of the function call is:

```
      ⍙⍙atfin filename
```

`filename` is a simple character vector containing the name of the file to be processed. Note that the extension `ATF` must be appended to the given file name as it is not automatically supplied.

`⍙⍙atfin` reads the file, applying the relevant translation from the source APL to Dyalog APL, to recreate the objects.

If an object cannot be recreated because of badly formed lines in the file, then that object is ignored and a warning message printed. The names of such objects are held in variable `⍙⍙_problems`.

Example

Create a Dyalog APL workspace from the file `\tmp\dbutil`, which was created by APL+

```
      )LOAD ATFIN
saved ...

    ⍙⍙atfin '\tmp\dbutil.atf'
...Processing script
Variable : a
Fn/Op    : cut
Cannot form variable ⎕PR
Variable : ⎕LX
... Finished
To remove ⍙⍙APL2IN and associated objects, execute the
 following line
⎕EX ⍙⍙_names
↑⍙⍙quadthings   to see the systems variables reset
⍙⍙_problems contains unfixed objects
```

At this point you should save your workspace if satisfied.

Using the user command]IN

Under Dyalog you can also use the user command IN to import workspaces and files in ATF format. This may be simpler. The syntax is:

```
      ]IN  filename -fliplu  -apl=…
```

The file **filename.atf** must be a file containing a complete representation of objects created by some other APL system. The ATF extension will be added if not present.

You can specify which APL version the file came from. There are 8 acceptable cases: APL2PC, APLX, APLC, APL2MF, DYALOG, APLPLUS, VSAPL and STSCMF. If you do not specify which APL vendor this file came from the command will try to guess and will be right most of the time but there is no guarantee this will work.

Because some APL vendors use a different convention for upper and lower case alphabets you might have to use the **–fliblu** switch to reverse the cases. This is often the case for APLX and APL+ for example.

Example

Import a workspace from the file **\tmp\dbutil.atf**, which was created by APL+.

```
    )CLEAR
    ]IN  \tmp\dbutil
...Processing script
Variable : Á
Fn/Op    : ÃþÝ
Variable : ⎕PP
Variable : ⎕IO
Variable : ⎕CT
...
Cannot form variable ⎕PR
Variable : ⎕LX
... Finished
    )save \tmp\dbutil.atf
*** ΔΔPROBLEMS contains unfixed objects
```

Because APL+ uses a different convention for upper and lowercase the variable names are mangled (APL+ does not have variable names like **ÃþÝ**). If you see this happen use the –**fliplu** switch to solve it.

Example

Import a workspace from the file **\tmp\abc.def**, which was created by APLX. Flip the lower/upper alphabet cases.

```
    )CLEAR
    ]IN  \tmp\abc.def  -fliplu  -apl=APLX
...Processing script
Variable : bar
Fn/Op    : foo
Fn/Op    : xyz
…
Variable : ⎕PP
Variable : ⎕IO
Variable : ⎕CT
Variable : ⎕LX
... Finished
    )save \mypath\abc.def
```

The XFRCODE workspace

If the exported code was produced by **Δxfrto** you must use this method to import it. The function to use is **xfr.Δxfrfrom** which takes a filename as argument and accepts a series of switches to alter its behaviour:

```
xfr.Δxfrfrom 'tohost\filename  [switches]'
```

The switches are:

-q	do not display banner
-apl=	specify which APL vendor produced this file This can usually be deduced by the file extension. Valid values are SAM (Sharp APL MF), SAX (Sharp APL Unix), A2K (APL+ family), APX (APLX), AP2 (APL2), DYW (Dyalog)
-file[=]	filename of component file to recreate (for a component file)
-list	to not bring objects in, simply list the object names to be defined
-noam	do not bring the access matrix (for component files)
-obj=	only bring the objects listed
-range=	only define the range of components listed (for component files)
-replace	overwrite already defined objects
-trans[=] 1 2	translate (1) code to fit Dyalog code and (2) add code to simulate missing elements

Code translation (e.g. []ELX vs []SIGNAL) is possible but unlikely to work. See the variable **Describe** in the workspace for details.

Example

Import a workspace from the file **\tmp\dbutil.xsw**, which was created by APL+.

```
      )load xfrcode
Saved …
      xfr.Δxfrfrom'\tmp\util.xsw'
* XFR version 3.43
A2K3.08 20090824 224036; WS=C:\APLWIN50\XUTIL
* "#.⎕lx:C" not redefined
* "#.⎕pw:N" not redefined
* "#.⎕io:N" not redefined
* "#.⎕ct:N" not redefined
13 objects defined
```

The output shows this version is 3.43, the file was produced by APL+ (A2K) ver 3.08 on 2009/8/24 at 22:40:36H from workspace C:\aplwin50\xutil. Four objects were not

redefined because they already existed and that the switch –**replace** was not specified. In total 13 objects were defined in the workspace.

Using the user command]INX

Under Dyalog you can also use the user command INX to import workspaces and files in extended format. The syntax is:

```
]INX  filename  [switches]
```

It accepts the same switches as **xfr.Δxfrfrom** and performs the same function.

Example

Import a workspace from the file `\tmp\dbutil.xuw`, which was created by Sharp APL under Unix:

```
      )CLEAR
      ]inx \tmp\dbutil.xuw -replace
* XFR version 3.43
SAX3.08 20090714 123016; WS=/home/db/dbutil
17 objects defined
```

Problems Transferring Workspaces and files

Dyalog APL workspaces and component files are binary compatible across machines with similar architectures. For example, workspaces created under Dyalog APL for Windows® can be used immediately by Dyalog APL under Linux, as long as both environments are running the same version of Dyalog APL. However, if you are not running the same VERSION (e.g. ver 11 vs ver 12.1), it is best to transfer workspaces of the higher version using `XFRCODE` or the equivalent user commands **]OUTX** and then to use **]INX** to recreate them by the lower version[4].

The `XFRCODE` workspace for Dyalog APL copes with nested arrays, defined functions and operators, assigned functions, `⎕OR` of objects and component files. However, the following restrictions apply:

- Arrays containing the `⎕OR` of an assigned function cannot be transferred.

[4] however there is no guarantee the code will run as the new version may contain features that do not exist in the older one

- Locked functions cannot be transferred.
- Functions such as `SUM←+/` cannot be transferred.

Importing other APL Workspaces

Workspaces from other APL usually cannot be imported perfectly as there are a number of incompatibilities (most often system functions) that get in the way. Sometimes those will even prevent functions to fix in the workspace, for example if the offending code is in the header.

The XFRCODE workspace (and the]INX user command) allow to perform some translation of the code as it is brought in, in order to solve some of these problems.

If the code imported contains a lot of **[]AV[…]** you should use the **–translate=2** switch to translate the code and import a global variable **ΔAV** which will be used instead of **[]AV** all over the code. Read the **Describe** variable in the XFRCODE workspace or see **]??INX** for details.

Examples of transferring workspace from other vendors

Here are a few examples of how to export workspaces from other vendors to Dyalog APL.

Importing IBM APL2 Workspaces

IBM APL2 has a system function `)OUT` which produces a text file that represents the workspace. This file must be transferred and decoded line by line on the target machine. This decoding is performed by the user command]IN.

Create a script file

Load your workspace in APL2, and use `)OUT` to create a file.

Transfer to target machine

Move file to target machine; no translation should be performed.

Create Dyalog APL workspace

Use the user command]IN to process the script file. This command takes the name of the script file as argument. Once the file has been processed, you must save the resulting workspace.

Example

a) On APL2/PC:

```
    )LOAD MYWS
saved ...

    )OUT 'c:\tmp\apl2.atf'
```

b) Transfer file to target machine, with NO translation

c) On target machine where Dyalog resides:

```
    )CLEAR

    ]IN 'c:\tmp\apl2.atf'
```

```
)SAVE newname
```

NOTES:

1. APL2 may allow to use ANY filename with)OUT. Depending on the APL2 system you use (mainframe or PC) you may have to use quotes around the name and/or to specify the extension.]IN supposes the file has an ATF extension. To avoid ambiguities you should always specify the full pathname.

Importing APLX Workspaces

The procedure is similar to APL2 workspaces (see above).

Create a script file

Load your workspace in APLX, and use `)OUT` to create a file.

Transfer to target machine

Move file to target machine; no translation should be performed.

Create Dyalog APL workspace

Use the user command]IN to process the script file. This command takes the name of the script file as argument. Once the file has been processed, you must save the resulting workspace.

Example

a) In APLX:

```
    )LOAD MYWS
saved ...
    )OUT myaplxfile.atf
```

b) Transfer file to target machine, with NO translation

c) On target machine where Dyalog resides:

```
)CLEAR

]IN myaplxfile.atf
```

```
)SAVE newname
```

NOTES:

1. APLX allows to use ANY filename with)OUT.]IN supposes the file has an ATF extension. To avoid ambiguities you should always specify the full pathname

2. APLX alphabet mapping is the reverse of Dyalog. You may have to use the **–fliplu** switch to flip the alphabet casing if unusual names are being defined instead of proper names.

Importing APL+Win Workspaces

The procedure is similar to APL2 workspaces (see above). APL+Win uses User Command]OUT to do the work.

Create a script file

Load your workspace in APL+, and use `]OUT` to create a file.

Transfer to target machine

Move file to target machine; no translation should be performed.

Create Dyalog APL workspace

Use the user command]IN to process the script file. This command takes the name of the script file as argument. Once the file has been processed, you must save the resulting workspace.

Example

a) In APL+:

```
    )LOAD MYWS
saved ...
    ]OUT myapl+file.atf
```

b) Transfer file to target machine, with NO translation

c) On target machine where Dyalog resides:

```
)CLEAR

]IN myapl+file.atf

)SAVE newname
```

NOTES:

1. APL+ allows to use ANY filename with)OUT.]IN supposes the file has an ATF extension. To avoid ambiguities you should always specify the full pathname

2. APL+ alphabet mapping is the reverse of Dyalog. You may have to use the –**fliplu** switch to flip the alphabet casing if unusual names are being defined instead of proper names.