

Development Environment

Menu Commands

Component Files

Keyed Files

Find in Files

Printing

Menu Commands

Several menu commands invoke J sentences, which typically execute verbs in the j locale. These are as follows:

Menu	Shortcut	Verb
File New Class		filenewform ''
File Print	Ctrl+P	fileprint ''
Edit Input Log	Ctrl+D	editinputlog ''
Edit Find	Ctrl+F	editfind ''
Edit Find in Files	Ctrl+Shift+F	fif ''
Edit Project Manager	Ctrl+B	projectmanager ''
Edit Configure		config ''
Studio Labs		lab ''
Studio Advance		lab 0
Studio Chapters		lab 1
Studio Author		lab 2
Studio Demos		demos
{none}	Ctrl+F1	help ''

Component Files (jfiles)

Jfiles are component files for J data. A jfile can be thought of as a boxed list which is stored on file. An element of the boxed list is referred to as a component, and can store a noun of any type, shape or size. File components are numbered sequentially from 0 upwards.

To access the system (assuming you are using the standard profile):

```
load 'jfiles'
```

This populates a locale `files` with utility functions, and defines the following verbs in the `z` locale:

```
jcreate  
jerase  
jappend  
jread  
jreplace  
jdup  
jsize
```

You create a file using `jcreate`.

`jappend` adds new items to the file. Each item added to the file is stored in a new component. Several items can be added to the file at a time.

`jread` reads items from file, and `jreplace` replaces items on file.

`jdup` duplicates a file, and `jsize` returns its size.

You can refer to a file either by its file name, or by its file handle. The default file extension is `.ijf` - this is used if no file extension is given.

You cannot delete components once created. If you need to reclaim space no longer required, either replace the components with empties, or else duplicate the file, copying only the components required.

Create file

`jcreate`

create a file. The right argument is the filename. Any existing file is overwritten. The result is 1 if successful, else 0. For example:

```
jcreate 'mydata'  
1
```

Note that since the file extension was not specified, this actually creates a file named `mydata.ijf`.

Read and write file

jappend

append to file. The left argument is a boxed list, with each item in the list stored in a new file component. An open noun is treated as a single boxed item. The right argument is the filename. The result is the new component numbers created. For example:

```
'header' jappend 'mydata'
0

('rec1';'rec2') jappend 'mydata'
1 2

(< <\1 2 3) jappend 'mydata'
3

(2 2$10 20 30 40) jappend 'mydata'
4
```

jread

read file. The right argument is the filename, linked with one or more component numbers. The result has the same shape as the component numbers. For example:

```
> jread 'mydata';4
10 20
30 40

jread 'mydata';i.5
+-----+-----+-----+-----+
|header|rec1|rec2|+-----+|10 20| | | | |
|      |    |    ||1|1 2|1 2 3||30 40|
|      |    |    |+-----+|
+-----+-----+-----+-----+

jread 'mydata';i.2 2
+-----+-----+
|header|rec1      |
+-----+-----+
|rec2  |+-----+| | | | |
|      ||1|1 2|1 2 3||
|      |+-----+|
+-----+-----+
```

jreplace

replace in file. The result is the components replaced. For example:

```
(1000;'abcde') jreplace 'mydata';1 2
1 2

jread 'mydata';i.3
+-----+-----+
|header|1000|abcde|
+-----+-----+
```

The left argument is reshaped if necessary to match the components in the right argument. For example, the following replaces components 1-3, each with the word 'reserved':

```
'reserved' jreplace 'mydata';1 2 3
1 2 3
```

Utilities

jdup

duplicate file. The left argument is the new filename; if elided, the file is duplicated in place. The right argument is the source filename, optionally linked with one or more component numbers to be copied to the new file, in the order given. By default, the entire file is duplicated. The result is the number of components written. For example:

```
'newdata' jdup 'mydata'
5

'newdata' jdup 'mydata';2 0 1
3
```

jsize

size of file, as 4 numbers:

- starting component number (0)
- number of components
- length of file in bytes (same as result of 1! : 4)
- amount of free space that could be recovered by duplicating the file

For example:

```
jsize 'newdata'
0 5 1312 0
```

jerase

erase file, for example:

```
jerase 'newdata'
1
```

File handles

You can refer to a file either by its filename, or by its file handle if you have already opened the file. For most purposes we recommend using the filename. However, if you have a great deal of file activity, you may find it faster to open the file first, then use the file handle; this means the system does not have to open and close the file each time it is accessed. The utilities `jopen` and `fclose` in the `jfiles` locale can be used for this purpose. For example:

```
h=. jopen_files_ 'mydata'
```

... process file using handle h...

```
fclose_files_ h
1
```

File structure

Each file is structured as a header record, followed by data. The header record is as follows:

- [0] version
- [1] starting component
- [2] number of components
- [3] file length
- [4] directory pointer
- [5] freelist pointer
- [6] sequence number

Freelist

Each component is stored in its 3!:1 representation in a space that is a power of 2. This means that on average, the representation fills 75% of the space allocated, and allows some space for growing replaces. If the space required on replacement is less than half the space allocated, then the balance is freed up.

If you replace a component with a noun of smaller size, then this may result in some unused space. The system keeps track of this in the freelist, and attempts to reuse it where possible. The total free space available is given in the fourth element of the result of `jsize`, and this is the space that would be freed by duplicating the file using `jdup`.

Keyed Files (kfiles)

A keyed file is a J component file in which the components are accessed using keywords.

A keyword may be any character string.

Load the keyed file system with:

```
load 'kfiles'
```

This defines the main functions:

kcreate	create file
kdir	keyword directory
kerase	erase file
kread	read data for keyword
kwrite	write data for keyword

read: if keyword not found, return " else return value

write: if data=" then keyword is deleted

Examples:

```
kcreate 'mydata'
1
'Peter Rabbit' kwrite 'mydata';'name'
'Lake District' kwrite 'mydata';'loc'

kdir 'mydata'
loc name

kread 'mydata';'loc'
Lake District
```

Find in Files

Ctrl+Shift+F or menu item Edit|Find in Files starts the Find in Files utility

You can search for simple text, or more complicated patterns with a Regular expression. The Regex button selects the regular expression search; this is automatically enabled when you use the Insert button to insert components that match certain characters, strings, or J-related strings. You can also search in specific contexts, such as assignment or use of names.

If you have projects created by the Project Manager, Find in Files loads with the current Project files in the search path. You can select other projects to search.

If Find in Files is open and you change the current Project in Project Manager, the change is not reflected immediately in Find in Files. To change to the new Project, select menu Options|Refresh Project.

Otherwise, you can search through folders. In this case, you can specify the file types to search, and the folders to search. Customize these by pressing the buttons next to the selection boxes.

Run a search by pressing Enter or Find.

Open a file by putting the caret on a line and pressing Open.

Printing

Printing from J is handled by the print package. Load with:

```
load 'print'
```

This defines utilities in the jprint locale, plus the following user definitions:

print	print text
printfile	print file
print2	print text in 2-up mode
printfile2	print file in 2-up mode

verb: print

form: opt print data

where opt is an optional list separated by semicolons:

ascii [1 0]	set ascii box-drawing characters on off
font fontspec	set font
fontsize points	set fontsize
filename	set filename text
fit	fit text to page width
header text	set header text
footer text	set footer text
land landscape	set landscape mode (default is portrait)
ruler	add ruler to top of each page (use with fixed pitch font)

verb: printfile

form: opt printfile files

Options are as for print, except the filename is set automatically

verbs : print2 printfile2

These verbs are the same as print and printfile, except that the page is printing in landscape mode, 2 pages per sheet. For example, this is good for printing script files.

Options are as for print, except that fit, land and landscape are ignored, and there is a new option:

cols	set columns, default 80
------	-------------------------

Default fonts and options can be set in menu Edit|Configure|Print.

The font for 2-up printing should be fixed pitch and around 7-7.5 point. "Lucida Console" 7.25 bold is good if available - use oem if your printer supports it, else ansi.

The ruler option uses the current session box-drawing characters. Choose a box-drawing font, or include the 'ascii' option.

Examples:

```
'font arial 12;land;footer just testing' print i.3 4 5  
'ascii;font "courier new" 14' print ;/i.3 4 5  
'fontsize 7.5;land;ruler' printfile 'system\examples\data\orders.prn'  
printfile2 'system\main\pack.ijs'
```