

# Catmandu Fixes: : CHEAT SHEET



## Basics

```
add_field(my.name,patrick)
my:
  name: patrick
move_field(my.name,your.name)
your:
  name: nicolas
copy_field(your.name,your.name2)
your:
  name: nicolas
  name2: nicolas
remove_field(your.name2)
your:
  name: nicolas
rename(your,['ae'],'X')
your:
  nXmX: nicolas
```

## Set

```
set_field(my.name,patrick)
my:
  name: patrick
set_array(my.array)
my:
  array: []
set_array(my.array,1,2,3,4)
my:
  array: [1,2,3,4]
set_hash(my.object)
my:
  object: {}
set_hash(my.object, a: A, b: B)
my:
  object:
    a: A
    b: B
```

## Array ⇔ Hash

```
given:
  foo: [ a, A, b, B ]

hash(foo)
foo:
  a: A
  b: B
array(foo)
foo: [ a, A, b, B ] reverse of hash
```

## Strings

```
given:
  title: catmandu

append(title,'?!')
  title: catmandu ?!
capitalize(title)
  title: Catmandu
downcase(title)
  title: catmandu
prepend(title,'I love ')
  title: I love catmandu
index(title,'t')
  title:2
replace_all(title,['au'],'X')
  title: cXtmXndX
reverse(title)
  title: udnamtac
substring(title,0,3)
  title: cat
trim(title)
  title: catmandu (spaces removed)
upcase(title)
  title: CATMANDU
```

## Hint

Most fixes work in this cheat sheet work on **strings**, **numbers** and **lists**.

E.g., given as data input:

```
string: test
list:
  - test1
  - test2
```

the fix **upcase(string)** would change the **string** field:

```
string: TEST
list:
  - test1
  - test2
```

And, **upcase(list.\*)** would change all the entries in the **list** field:

```
string: test
list:
  - TEST1
  - TEST2
```

## Data manipulation

```
given:
  numbers: [ 41, 42, 6, 6 ]
  person:
    name: François
    age: 12
    date: 1918-11-11
    animals: ['Lion','Cat','Tiger']
    deep: [ 1, [2, [3, 4]] ]
    pairs:
      - key: name
        val: Albert
      - key: age
        val: 12

assoc(result,pairs.*.key, pairs.*.val)
  result: { name: Albert, age: 12 }
count(numbers)
  numbers: 4
compact(numbers)
  numbers: [ 41, 42, 6, 5 ] (removes null values)
filter(animals,['Cc]at')
  animals: [ 'Cat' ]
flatten(deep)
  deep: [ 1, 2, 3, 4 ]
format(numbers,'%10.10d %-5.5d')
  numbers: 0000000041 00042
format(name,'%10s: %s')
  person: "name      : François"
from_json(field)
  inverse of to_json(field)
join_field(numbers, '/')
  numbers: '41/42/6/6'
parse_text(date,'(\d{4})-(\d{2})-(\d{2})')
  date: [ '1918', '11', '11' ]
parse_text(date,'(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})')
  date:
    year: '1918'
    month: '11'
    day: '11'
paste(result,person.name,person.age)
  result: "François 12"
paste(result,person.name,person.age,
  join_char:',')
  result: "François,12"
paste(result,person.name,~is,person.age)
  result: "François is 12"
random(test,100)
  test: 13 (adds a random number)
retain(numbers,person)
  delete all fields except numbers and person
```

## JSON Path

**JSON paths** are used to point to zero, one or more fields in your record. Given the data in the **yellow** box on the left:

JSON Path	Value
numbers.0	41
numbers.\$end	6
numbers.\$start	41
numbers.*	[41,42,6,6]
numbers.\$prepend	-> <i>numbers.\$start - 1</i>
numbers.\$append	-> <i>numbers.\$end + 1</i>
person.age	12
deep.1.1.0	3
.	-> <i>select the whole record</i>

Examples:

```
copy_field(person.age,list.$append)
  list: [ 12 ]
copy_field(person.age,list.5)
  list: [ ~, ~, ~, ~, ~, 12 ]
```

```
reverse(numbers)
  numbers: [6,6,42,41]
sort_field(numbers)
  numbers: [41,42,6,6]
sort_field(numbers,numeric:1)
  numbers: [6,41,42]
sort_field(numbers,numeric:1,reverse:1)
  numbers: [42,41,6,6]
split_field(date,'-')
  date: ['1918','11','11']
sum(numbers)
  numbers: 95
to_json(person)
  person: '{"name": "Albert", "age": "12"}'
uniq(numbers)
  numbers: [41,42,6]
url_decode(person.name)
  inverse of uri_encode(...)
uri_encode(person.name)
  person:
    name: Fran%C3%A7ois
vacuum()
  delete all empty/undef fields in the record
```

# Catmandu Fixes: : CHEAT SHEET



## Conditions

A condition can be used in an **if-else** statement to have conditional execution of fixes. They can also be used as **guards** for **reject** or **select** statements. All conditions have the syntax:

```
if Condition(params,...)
```

```
  fix(..)  
  fix(..)
```

```
end
```

```
if Condition(params,...)
```

```
  fix(..)  
  fix(..)
```

```
else
```

```
  fix(..)
```

```
end
```

```
unless Condition(params,...)
```

```
  fix(..)  
  fix(..)
```

```
end
```

```
reject Condition(params,...)
```

```
select Condition(params,...)
```

```
Condition(params,...) and fix(..)
```

```
Condition(params,...) or fix(..)
```

Here is a list of all conditions implemented in Catmandu:

```
all_match(JSONPath, REGEX)
```

Execute the fix(es) when **all** values in the JSONPath matches the REGEX

```
any_match(JSONPath, REGEX)
```

Execute the fix(es) when **at least one** value in the JSONPath matches the REGEX

```
exists(JSONPath)
```

Execute the fix(es) when a JSONPath contains a value (a string, number, list or hash)

```
all_equal(JSONPath, String)
```

Execute the fix(es) when **all** values in the JSONPath are equal to a String

```
any_equal(JSONPath, String)
```

Execute the fix(es) when **at least one** value in the JSONPath is equal to a String

```
greater_than(JSONPath, Value)
```

Execute the fix(es) when **all** values in the JSONPath are greater than Value

```
less_than(JSONPath, Value)
```

Execute the fix(es) when **all** values in the JSONPath are less than Value

```
in(JSONPath1, JSONPath2)
```

Execute the fix(es) when all values in the JSONPath1 can be found in JSONPath2. E.g.

```
foo: 1  
bar: [3,2,1]
```

```
if in(foo,bar)
```

```
  add_field(test,ok)
```

```
end
```

```
is_true(JSONPath)
```

Execute the fix(es) when **all** the values in the JSONPath are boolean true, 1 or 'true'

```
is_false(JSONPath)
```

Execute the fix(es) when **all** the values in the JSONPath are boolean false, 0 or 'false'

```
is_array(JSONPath)
```

Execute the fix(es) when the JSONPath points to an array

```
is_object(JSONPath)
```

Execute the fix(es) when the JSONPath points to a hash

```
is_number(JSONPath)
```

Execute the fix(es) when the JSONPath contains a number

```
is_string(JSONPath)
```

Execute the fix(es) when the JSONPath contains a string

```
is_null(JSONPath)
```

Execute the fix(es) when the JSONPath contains a null value

```
is_valid(data, JSONSchema, schema:file)
```

Execute the fix(es) when the data is valid against a JSONSchema defined in file

### CSV Data

```
File: lookup.csv
```

```
en,nl  
blue,blauw  
red,rood  
green, groen  
yellow,geel  
purple,paars
```

## Import / Export

Import and export fixes can be used to import values from external files into the record. Or, to export data from the record to external files and databases.

```
given:
```

```
  color1: red  
  color2: brown
```

```
lookup(color1,"lookup.csv",sep_char:",")
```

```
  color1: "rood"
```

```
lookup(color2,"lookup.csv",default:NA)
```

```
  color2: NA
```

```
lookup(color2,"lookup.csv",delete:1)
```

```
>> color2 is deleted, because 'brown' is  
not available in the lookup.csv
```

In the following examples we assume a MongoDB database is available which contains the record:

```
_id: red  
color_eng: red  
color_dut: rood  
color_ger: rot
```

```
lookup_in_store(color1,MongoDB,database_name:colors)
```

```
  color1:
```

```
    _id: red  
    color_eng: red  
    color_dut: rood  
    color_ger: rot
```

```
lookup_in_store(color2,MongoDB,database_name:colors,default:NA)
```

```
  color2: NA
```

```
lookup_in_store(color2,MongoDB,database_name:colors,delete:1)
```

```
>> color2 is deleted, because 'brown' is  
not available in the database
```

In the following example we assume the data contains this record:

```
author:  
  _id: 1234  
  name:  
    first: Albert  
    last: Einstein  
  dateBirth:1879
```

```
add_to_store(author,MongoDB,database_name:authors)
```

The values in 'author' will be added to the MongoDB store

```
in general:
```

```
  add_to_store(field,Store,options..)
```

```
add_to_exporter(author,CSV,header:1,file:/tmp/data.csv)
```

The values in 'author' will be added to the CSV file.

```
in general:
```

```
  add_to_exporter(field,Exporter,options..)
```

```
export_to_string(author,YAML)
```

```
author: "_id: 1234\nname:\nfirst: Albert\nlast: Einstein\ndateBirth:"
```

```
in general:
```

```
  export_to_string(field,Exporter,options..)
```

```
import_from_string(author,YAML)
```

```
>> the inverse of export_to_string
```

```
search_in_store(query,'Solr',url:"http://localhost:8983/solr",limit:10)
```

```
>> execute the string in query and
```

```
replace the field with the search results
```

```
import(foo,JSON,file:data.json,data_path:data.*)
```

```
>> replace foo with the content found  
in the JSON file at path data
```

```
include('/tmp/myfixes.txt')
```

```
>> include the fixes from a file in this  
Fix script
```

## Hint

Execute these fixes on the Unix command line:

```
$ catmandu  
  convert JSON to  
  YAML —fix test.fix < data.json > data.yml
```

where test.fix contains all your fix commands.

Read more about the Catmandu **convert** command:

```
$ catmandu help convert
```

# Catmandu Fixes: : CHEAT SHEET



## Select / Reject

**Select** and **reject** fixes are used to filter records out of a stream based on a **condition**.

### **reject exists(my.badfield)**

*reject the record if it contain my.badfield*

### **select all\_match(title,'Catmandu')**

*select only the records that have Catmandu in the title field*

## External Commands

### **cmd("java MyClass")**

*>> send the record as JSON to the STDIN of the external command and replace it with the JSON from the STDOUT*

### **perlcode("mycommand.pl")**

*>> run the my command.pl on the data in the record*

### **sleep(1,SECOND)**

*do nothing for one second*

## Logging

### **log("test1234",level:DEBUG)**

*>> send a message to the logs*

### **error("eek!")**

*>> abort processing and say 'eek!'*

## Hint

Add more **Catmandu** fixes and commands by installing more packages:

```
# cpanm install PACKAGE
```

Popular packages:

- Catmandu::Identifier
- Catmandu::MARC
- Catmandu::RDF
- Catmandu::Stat
- Catmandu::VIAF
- Catmandu::XML

## Bind

**Binds** are wrappers for one or more fixes. They give extra control functionality for fixes such as loops.

All binds have the syntax:

### **do Bind(params,...)**

```
fix(..)
fix(..)
end
```

The most easy Bind is probably **iterate** which iterates fixes in a loop:

### **do iterate(start:1, end:10, step:1 var:i)**

```
copy_field(i,numbers.$append)
end
```

This bind will create the array *numbers*::

```
numbers: [1,2,3,4,5,6,7,8,9,10]
```

Here is an overview of Bind provided by the main Catmandu package:

### **benchmark(output:FILE)**

This fix calculates the execution time of Fix functions:

```
do benchmark(output:/dev/stderr)
  foo()
  bar()
end
```

### **hashmap(**

```
exporter:EXPORTER, [opt:vallue,...]
store:STORE, [opt:vallue,...]
uniq:0|1
join:CHAR
count:0|1)
```

Add fields 'key' and 'value' to an internal hash map and print the content to a JSON exporter when all records have been processed

```
do hashmap()
  copy_field(isbn,key)
  copy_field(id,value)
end
```

This will create a JSON output with isbn values as '\_id' and an array of id values as 'value'

### **identity()**

This Bind does nothing special and is mostly used to group fixes as a single operation for other binds.

```
do benchmark(output:/dev/stderr)
  foo()
  do identity()
    bar()
  end
end
```

### **importer(IMPORTER, [opt:value,...])**

Used in standalone catmandu Fix scripts to set the importer to read data from.

```
#!/usr/bin/env catmandu run
do importer(OAI,url:http://somewhere.org)
  retain(_id)
  add_to_exporter(.,YAML)
end
```

### **iterate(**

```
start:NUM,
end:NUM,
step:NUM,
var:NAME)
```

Iterate numbers from start to end with the provided step. Set the field NAME to the number and execute the fixes.

```
do iterate(start:1, end:10, step:1 var:i)
  copy_field(i,numbers.$append)
end
```

### **list(path:JSONPath[,var:NAME])**

Execute all the fixes in the context of every element in the JSONPath array

```
do list(path:demo)
  if all_equal(.,'green')
    upcase(.)
  end
end
```

or when you need to have access to the root element

```
do list(path:demo,var:c)
  copy_field(c,mylist,$append)
end
```

### **maybe()**

Skip fixes when one returns undef or throws an error

```
do maybe()
  foo()
  error("Help") # bar will be ignored
  bar()
end
rest() # rest will be executed
```

### **timeout(**

```
time:NUM,
units:seconds|minutes|hours)
```

Ignore the effect of the fixes on the data after some timeout

```
do timeout(time:5,unit:seconds)
  add_field(foo,ok) # will be ignored
  sleep(10,seconds)
  set_field(foo,error) # will be ignored
end
```

### **visitor([path:JSONPath])**

Execute all fixes in the context of every element in the data. This fix will set special context variables:

*scalar* - for every scalar value found  
*array* - for every array value found  
*hash* - for every hash value found  
*key* - the field name on which the scalar array or hash is found

```
# upcast every 'name' field in the record
do visitor()
  if all_equal(key,name)
    upcase(scalar)
  end
end
```

### **with(path:JSONPath)**

Execute all the fixes in the context of the JSONPath

```
do with(path:my.deep.path)
  # Treat path as root
  # create: my.deep.path.name = Patrick
  add_field(name,Patrick)
end
```