Basics

add_field(my.name, patrick)
my:
name: patrick

move_field(my.name, your.name)

your:
name: nicolas
name2: nicolas

remove_field(your.name)

your:
name: nicolas

rename(your."ae", "X")

Your:
name: nicolas
name2: nicolas

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, '')
title:2

title:2

title:2

reverse_all(title, 'au', 'X')
title: cXtmXndX

reverse(title)
title: udnamtac

Substring(title, 0, 3)
title: cat

trim(title)
title: catmandu

upcase(title)
title: CATMANDU

Data manipulation

given:
numbers: [41, 42, 6, 6]
person:
name: François
age: 12
date: 1918-11-11
animals: ['Lion', 'Cat', 'Tiger']
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, 6]

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'

sort_field(numbers)
numbers: [41, 42, 6, 6]

sort_field(numbers, numeric: 1)
numbers: [6, 6, 42, 41]

split_field(date, '-')
date: [1918, '11', '11']

reverse(numbers)
numbers: [6, 6, 42, 41]

sort_field(numbers, numeric: 1, reverse: 1)
numbers: [42, 41, 6, 6]

sum(numbers)
numbers: 95

to_json(person)
person:
"name": "Fran%C3%A7ois"

uniq(numbers)
numbers: [41, 42, 6]

url_decode(person.name)
inverse of uri_encode(…)

uri_encode(person.name)
person:
name: Fran%3C%7Aois

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]

Array <> Hash

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

hash(foo)
foo:
a: A
b: B

array(foo)
foo: [ a, A, b, B ]

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:

<table>
<thead>
<tr>
<th>JSON Path</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers.0</td>
<td>41</td>
</tr>
<tr>
<td>numbers.$end</td>
<td>6</td>
</tr>
<tr>
<td>numbers.$start</td>
<td>41</td>
</tr>
<tr>
<td>numbers.$prepend</td>
<td>numbers.$start - 1</td>
</tr>
<tr>
<td>numbers.$append</td>
<td>numbers.$end + 1</td>
</tr>
<tr>
<td>person.age</td>
<td>12</td>
</tr>
<tr>
<td>deep.1.0.0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-&gt; select the whole record</td>
</tr>
</tbody>
</table>

Examples:

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

reverse(numbers)
numbers: [6, 42, 41]

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%3C%7Aois

vacuum()
delete all empty/undef fields in the record
Catmandu Fixes: CHEAT SHEET

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

```plaintext
if Condition(params,...) fix(...) end
if Condition(params,...) fix(...) end
else fix(...) end
unless Condition(params,...) fix(...) end
reject Condition(params,...) select Condition(params,...)
```

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.
  ```plaintext
  x: 1
  nums: [3,2,1]
  if in(x,nums)
    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.

```plaintext
given:
  color1: red
color2: brown
lookup(color1,"lookup.csv",sep_char:",")
end
```

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

```plaintext
_author: "_id: 1234
  name:
    first: Albert
    last: Einstein
dateBirth:
  "
```

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

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

```plaintext
_author: 
_id: 1234
name:
  first: Albert
  last: Einstein
dateBirth: 1879
```

```plaintext
lookup(color2, "lookup.csv", sep_char:",")
```

```plaintext
lookup_in_store(color2, MongoDB, database: authors)
```

```plaintext
lookup(color1, MongoDB, database: authors)
```

```plaintext
all similar examples
```

CSV Data

**File: lookup.csv**

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

CC BY Patrick Hochstenbach • patrick.hochstenbach@ugent.be • librecat.org • package version 1.2002 • Updated: 2019-06
**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(..))
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()
  do identity()
    bar()
    bar()
  end
end
```

**hashmap()**

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’.

**maybe()**

Skip fixes when one returns undef or throws an error:

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

**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
```

**with(path:JSONPath)**

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

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

**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 mycommand.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!
```

**Select / Reject**

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

```
select 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
```

**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

**Popular packages:**

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

**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()
    bar()
  end
end
```

**Maybe()**

Skip fixes when one returns undef or throws an error:

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

**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
```

**With(path:JSONPath)**

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

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

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

#### MARC PATH

MARC paths are used to point to zero or more MARC (sub)fields in your record.

### Conditions

A condition can be used in if/else/end statements to have conditional execution of fixes. See “Conditions” on page 2.

Most MARC Conditions are best executed in a surrounding “marc_each” block:

```perl
do marc_each()
  if marc_hash(245)
    # execute for each 245 in MARC
  end
end
```

### Bind

Binds are wrappers for one or more fixes. They give extra control functionality for fixes such as loops. See “Bind” on page 3.

```perl
marc_map(999, has.999, value: "yes ok")
  has.999 = "yes ok"
```

```perl
marc_add(MARCField, subfield, value...)  
  Add a new MARC field to the record
```

```perl
marc_add(900, a, test, b, test2)
  creates: 900 $a$test$b$test2
```

### Marc_map(650/0-1, test)

```perl
test = "Alpha"
marc_map(650/0-1, test)
mymap = [ 'Alpha', 'Beta', 'Gamma' ]
marc_map(650, mymap.$append)
marc_map(245, my.title, join: "@@")
mymap = [ "Title / ", "Name""]
marc_map(245, my.title)
mymap = [ "Title / Name" ]
```