**Basics**

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_field(my.name,patrick)</td>
<td>add_field(my.name,patrick)</td>
</tr>
<tr>
<td>my:name: patrick</td>
<td>my:name: patrick</td>
</tr>
<tr>
<td>move_field(my.name,your.name)</td>
<td>move_field(my.name,your.name)</td>
</tr>
<tr>
<td>your:name: Nicolas</td>
<td>your:name: Nicolas</td>
</tr>
<tr>
<td>copy_field(my.name,my.name2)</td>
<td>copy_field(my.name,my.name2)</td>
</tr>
<tr>
<td>your:name: Nicolas</td>
<td>your:name: Nicolas</td>
</tr>
<tr>
<td>remove_field(your.name2)</td>
<td>remove_field(your.name2)</td>
</tr>
<tr>
<td>your:name: Nicolas</td>
<td>your:name: Nicolas</td>
</tr>
<tr>
<td>rename(your,{'ae'},'X')</td>
<td>rename(your,{'ae'},'X')</td>
</tr>
<tr>
<td>your:nXM:name: Nicolas</td>
<td>your:nXM:name: Nicolas</td>
</tr>
</tbody>
</table>

**Strings**

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>append(title,'?!')</td>
<td>append(title,'?!')</td>
</tr>
<tr>
<td>capitalize(title)</td>
<td>capitalize(title)</td>
</tr>
<tr>
<td>downcase(title)</td>
<td>downcase(title)</td>
</tr>
<tr>
<td>prepend(title,'I love ')</td>
<td>prepend(title,'I love ')</td>
</tr>
<tr>
<td>title: I love catmandu</td>
<td>title: I love catmandu</td>
</tr>
<tr>
<td>index(title,'t')</td>
<td>index(title,'t')</td>
</tr>
<tr>
<td>title:2</td>
<td>title:2</td>
</tr>
<tr>
<td>replace_all(title,'[a]','X')</td>
<td>replace_all(title,'[a]','X')</td>
</tr>
<tr>
<td>upcase(title)</td>
<td>upcase(title)</td>
</tr>
<tr>
<td>trim(title)</td>
<td>trim(title)</td>
</tr>
<tr>
<td>upcase(hash)</td>
<td>upcase(hash)</td>
</tr>
</tbody>
</table>

**Data manipulation**

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>assoc(result,pairs::*key,.*val)</td>
<td>assoc(result,pairs::*key,.*val)</td>
</tr>
<tr>
<td>diassoc(result,pairs,.*val)</td>
<td>diassoc(result,pairs,.*val)</td>
</tr>
<tr>
<td>count(numbers)</td>
<td>count(numbers)</td>
</tr>
<tr>
<td>compact(numbers)</td>
<td>compact(numbers)</td>
</tr>
<tr>
<td>filter(animals,['Cat'])</td>
<td>filter(animals,['Cat'])</td>
</tr>
<tr>
<td>flatten(deep)</td>
<td>flatten(deep)</td>
</tr>
<tr>
<td>format(numbers,%.10f)</td>
<td>format(numbers,%.10f)</td>
</tr>
<tr>
<td>from_json(field)</td>
<td>from_json(field)</td>
</tr>
<tr>
<td>join_fields(numbers,:)</td>
<td>join_fields(numbers,:)</td>
</tr>
<tr>
<td>parse_date(date,'day')</td>
<td>parse_date(date,'day')</td>
</tr>
<tr>
<td>parse_text(date,'&lt;year&gt;')</td>
<td>parse_text(date,'&lt;year&gt;')</td>
</tr>
<tr>
<td>parse_text(date,'&lt;day&gt;')</td>
<td>parse_text(date,'&lt;day&gt;')</td>
</tr>
<tr>
<td>reverse(numbers)</td>
<td>reverse(numbers)</td>
</tr>
<tr>
<td>retain(numbers,person)</td>
<td>retain(numbers,person)</td>
</tr>
<tr>
<td>reverse(numbers)</td>
<td>reverse(numbers)</td>
</tr>
<tr>
<td>sort_field(numbers)</td>
<td>sort_field(numbers)</td>
</tr>
<tr>
<td>sort_field(numbers,numeric)</td>
<td>sort_field(numbers,numeric)</td>
</tr>
<tr>
<td>split_field(date,,-)</td>
<td>split_field(date,,-)</td>
</tr>
<tr>
<td>sum(numbers)</td>
<td>sum(numbers)</td>
</tr>
<tr>
<td>to_json(person)</td>
<td>to_json(person)</td>
</tr>
<tr>
<td>uniq(numbers)</td>
<td>uniq(numbers)</td>
</tr>
<tr>
<td>uri_decode(person.name)</td>
<td>uri_decode(person.name)</td>
</tr>
<tr>
<td>vacuum(person)</td>
<td>vacuum(person)</td>
</tr>
</tbody>
</table>

**JSON Path**

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>json_path</td>
<td>json_path</td>
</tr>
<tr>
<td>name: Francois</td>
<td>name: Francois</td>
</tr>
<tr>
<td>age: 12</td>
<td>age: 12</td>
</tr>
<tr>
<td>date: '1918-11-11'</td>
<td>date: '1918-11-11'</td>
</tr>
<tr>
<td>animals: ['Lion','Cat','Tiger']</td>
<td>animals: ['Lion','Cat','Tiger']</td>
</tr>
<tr>
<td>deep: [1,2,[3,4]]</td>
<td>deep: [1,2,[3,4]]</td>
</tr>
<tr>
<td>pairs:</td>
<td>pairs:</td>
</tr>
<tr>
<td>- key: name</td>
<td>- key: name</td>
</tr>
<tr>
<td>- val: Albert</td>
<td>- val: Albert</td>
</tr>
<tr>
<td>- key: age</td>
<td>- key: age</td>
</tr>
<tr>
<td>- val: 12</td>
<td>- val: 12</td>
</tr>
</tbody>
</table>

**Hint**

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

E.g., given as data input:

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

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

```json
string: TEST
list:
  - TEST1
  - TEST2
```

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

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

---

**Array <> Hash**

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo: [ a, b, B ]</td>
<td>foo: [ a, b, B ]</td>
</tr>
<tr>
<td>hash(foo)</td>
<td>hash(foo)</td>
</tr>
<tr>
<td>a: A</td>
<td>a: A</td>
</tr>
<tr>
<td>b: B</td>
<td>b: B</td>
</tr>
<tr>
<td>array(foo)</td>
<td>array(foo)</td>
</tr>
</tbody>
</table>
| foo: [ a, b, B ]          | foo: [ a, b, B ]                          | reverse of hash

---

**Examples**

```python
result: 'François is 12'
```
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:

if Condition(params,...) fix(...) fi
if Condition(params,...) fix(...) end
unless Condition(params,...) fix(...) fi
reject Condition(params,...) select Condition(params,...) fi
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.

x: 1
ums: [3,2,1]
if in(x,nums) add_field(test,ok) fi

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
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 int 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 int the database

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

author:
_id: 1234
name: 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
name: Albert
last: Einstein
dateBirth:"

in general:
export_to_string(field, Exporter, options..)
import_from_string(author, YAML)
>> the inverse of export_to_string
>> execute the string in query and replace the field with the search results
import(foo, JSON, file: data.json, data_path: data.csv)
>> 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
percode("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’!

Hint

Add more Catmandu fixes and commands by installing more packages:

  # cpanm install PACKAGE

Popular packages:

  - Catmandu::Identity
  - Catmandu::Marc
  - Catmandu::Rdf
  - Catmandu::Sta
  - 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:value,…]
  store:STORE, [opt:value,…]
  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

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,uri: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

each(path:JSONPath, var: NAME)
  Execute all the fixes in the context of every element in the JSONPath hash.
  do each(path: demo, var: t)
    copy_field(t.value, titles.$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:secondsinminutes|hours)
  Ignore the effect of the fixes on the data after some timeout
  do timeout(time:5,unit:seconds)
    add_field(fooo) # will be ignored
    sleep(10,seconds)
    set_field(fooo) # 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
### Catmandu::MARC

#### MARC PATH

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

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>= “1234”</td>
<td>result: 001 1234</td>
</tr>
<tr>
<td>245</td>
<td>= “Title / Name”</td>
<td>result: 245 $aTitle $cName</td>
</tr>
<tr>
<td>245a</td>
<td>= “Title /”</td>
<td>result: 245 $aTitle</td>
</tr>
<tr>
<td>245x</td>
<td>= “ABCD”</td>
<td>result: 245x $cAbcd</td>
</tr>
<tr>
<td>500</td>
<td>= “points to all 500 except $x”</td>
<td>result: 500 $a points to all 500 except $x</td>
</tr>
<tr>
<td>2..</td>
<td>= “points to all 200-299 fields”</td>
<td>result: 2.. $a points to all 200-299 fields</td>
</tr>
<tr>
<td>245[1]</td>
<td>= “points to 245 if ind1=1”</td>
<td>result: 245[1] $a points to 245 if ind1=1</td>
</tr>
<tr>
<td>245[1,0]</td>
<td>= “points to 245 if ind1=1 ind2=0”</td>
<td>result: 245[1,0] $a points to 245 if ind1=1 ind2=0</td>
</tr>
<tr>
<td>008/35-37</td>
<td>= “points to chars 35-37 in 008”</td>
<td>result: 008/35-37 $a points to chars 35-37 in 008</td>
</tr>
</tbody>
</table>

#### MARC Map

Copy the value(s) found at MARCPath to a JSONPath.

<table>
<thead>
<tr>
<th>MARC Path</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>marc_map(MARCPath, JSONPath, opts)</td>
<td>Copy the value(s) found at MARCPath to a JSONPath.</td>
<td>marc_map(999,has.f999.value:”yes ok”) has.f999 = “yes ok”</td>
</tr>
<tr>
<td>marc_map(245,my.title)</td>
<td>my.title = “Title / Name”</td>
<td>marc_map(245,my.title,split:1) my.title = [“Alpha”, “Beta”, “Gamma”]</td>
</tr>
<tr>
<td>marc_map(245ca,my.title)</td>
<td>my.title = “Title / Name”</td>
<td>marc_map(245ca,my.title,pluck:1) my.title = “NameTitle /”</td>
</tr>
<tr>
<td>marc_map(245,y.my.field)</td>
<td>y.my.field = “Value”</td>
<td>marc_replace_all(245a,Title,”Hello!”) result: 245 $aHello $cName</td>
</tr>
<tr>
<td>marc_map(245/0-1,test)</td>
<td>test = “Al”</td>
<td>marc_map(245/0-1,split:1)</td>
</tr>
</tbody>
</table>

#### MARC Append

Add a value at the end of a MARC (sub)field.

<table>
<thead>
<tr>
<th>MARC Path</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>marc_append(MARCPath, value)</td>
<td>Add a value at the end of a MARC (sub)field</td>
<td>marc_append(245a, “Hello”)</td>
</tr>
<tr>
<td>marc_append(245,“.”)</td>
<td>Add a period “.” at the end of the 245 field</td>
<td>marc_append(245, “.”)</td>
</tr>
</tbody>
</table>

#### MARC Replace All

Replace all occurrences of the regular expression Search with Replace at MARCPath.

<table>
<thead>
<tr>
<th>MARC Path</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>marc_replace_all(MARCPath, Search, Replace)</td>
<td>Replace all occurrences of the regular expression Search with Replace at MARCPath</td>
<td>marc_replace_all(245a,“Hello”,”Hello”) result: 245 $aHello $cName</td>
</tr>
<tr>
<td>marc_replace_all(245a,“Hello”,”Hello”)</td>
<td>result: 245 $aHello $cName</td>
<td></td>
</tr>
</tbody>
</table>

#### 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:

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

#### Marc Has

Execute the fix(es) when the MARC file contains a MARCPath value.

```erb
  marc_has(MARCPath)
```

#### Marc Match

Execute the fix(es) when the value at MARCPath matches the Regex.

```erb
  marc_match(MARCPath, Regex)
```

#### Bind

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

```erb
  marc_each()
  if marc_match(720, “promotor”)
    marc_map(720ab, authors.$append)
  end
```

#### Marc Each

Execute all the fix(es) in the Bind context on individual MARC fields (loop over all the fields).

```erb
  do marc_each()
    if marc_match(720e, “promotor”)
      marc_map(720ab, authors.$append)
    end
  end
```

#### Marc Each

Like marc_each, but now an implicit marc_copy of the MARC field in context has been stored in the “this” variable

```erb
  do marc_each(var: this)
    if all_match(this.tag,300)
      # rename tag to 301
      set_field(this.tag,301)
      marc_paste(this)
    end
  end
```