

Metadatenworkflows mit Metafacture erstellen und verwalten



Pascal Christoph & Petra Maier & Katinka Tauber
Offene Infrastruktur, Hochschulbibliothekszentrum NRW (hbz)

Hands-on Lab
Studiengang Bibliotheks- und Informationswissenschaft an der HTWK Leipzig
im Modul „Metadaten und Metadatenmanagement“
bei Prof. Dr. Manuel Bärwald
Leipzig, 11. Januar 2024

<https://slides.lobid.org/2024-01-metafacture-workshop/> : (PDF)



Agenda

1. Einführung & Kontext
2. Metafactory-Workflows
10 min Pause 
3. Transformieren
10 min Pause 
4. Exkurs: Anreicherung / Exkurs: Conditionals
5. Ausblick, Fragen & Diskussion

Code of Conduct

Falls alle einverstanden sind, duzen wir uns.

Wir möchten für alle gleichermaßen ein einladendes und unterstützendes Umfeld schaffen. Dazu gehört ein höflicher und respektvoller Umgangston.

Bei Fragen, Schwierigkeiten oder sonstigen Wortmeldungen nutzt den Chat oder die Handhebefunktion.

Es ist ein Workshop: Wortmeldungen und aktive Beteiligung sind willkommen.

Falls etwas nicht funktioniert: bitte melden - meist lernen dadurch alle etwas!

Ansonsten: am besten das Mikro stumm schalten, um überflüssige Geräusche zu vermeiden.

1. Einführung & Kontext

Workflow: Extract, Transform, Load (ETL)



Quelle: <https://github.com/culturegraph/culturegraph-workflow/blob/master/docs/src/docs/asciidoc/manual.adoc>

Transformationsprozesse in der Praxis

Transformationen von Metadaten gehören zu Routineaufgaben wissenschaftlicher Bibliotheken.

Es gibt viele unterschiedliche Methoden, die meist Programmierkenntnisse voraussetzen.

Datentransformationen werden oft im Zusammenspiel von Fachabteilungen & IT umgesetzt, verbunden mit größerem Kommunikationsaufwand.

Existierende Transformationsprozesse sind meist nicht öffentlich und somit nicht nachnutzbar.

Das heißt:

Es gibt großes Potenzial, wiederkehrende Aufgaben kollaborativer und effizienter zu gestalten.

ETL: Anwendungsbeispiele

Datenaggregation z.B. aus unterschiedlichen Quellen

Systemmigration, z.B. nach Alma oder Folio

Datenaufbereitung, z.B. zur Visualisierung mit **Kibana**

Datenanreicherung, z.B. Ergänzung von Daten aus **Wikidata**

Datenanalyse, z.B. Feldabdeckung im Katalog

Metadatenanalyse/-transformation

Tools (Auswahl)

MarcEdit **OpenRefine**

Regular Expressions **Python** XSLT

Perl **Catmandu** **Metafacture**

→ je nach Ausgangslage (Datenformat, Quelle, Datenmenge etc.), Verfügbarkeit von Software, Vorhandensein etablierter/existierender Workflows, Kenntnis und persönlicher Vorliebe

Wo wir Metafactory nutzen

Transformation der Alma MARC-XML Daten des Verbundkatalogs zu JSON-LD (lobid.org)

Metadaten aus verschiedenen Quellen im OER Suchindex (OERSI) aggregieren (oersi.de)

Transformation der Daten der Rheinland-Pfälzischen Bibliographie (rpb.lobid.org)

Transformation der ZDB (PICA), DBS (CSV) und Wikidata (CSV) Daten ([lobid-organisations](https://lobid-organisations.org))

Dokumentation

Beispiele: produktiv in **OERSI** (diverse Web-Quellen → JSON),
für ALMA **hbz-Verbundkatalog** (ALMA MARC → JSON) und
Rheinland-Pfälzische Bibliographie (Allegro → JSON)

Integration-Tests: Input, Flux, Fix, Output als Dateien wie bei
Real-World-Setup

Metafactory / Playground

Was ist Metafacture?

Ein vielseitiges ETL-Werkzeug zur Verarbeitung von semi-strukturierten Daten mit dem Fokus auf Bibliotheksdaten

nutzbar als Kommandozeilentool, als Java/JVM library,
für Batch-Verarbeitung oder on-the-fly

offenes Framework: Weiterentwicklung, Wiederverwendung
und Austausch (von einzelnen Modulen und ganzen
Workflows)

Warum Metafacture?

Metafacture ist performant 🚀

Beispiel ETL bei 50 GB MARC-XML Input nach lobid JSON-LD
(28 M Dokumente):

- 14h mit Fix (7h mit Morph)
- auf einem 10 Jahre alten Server
- u.a. dank Multithreading

Metafacture-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

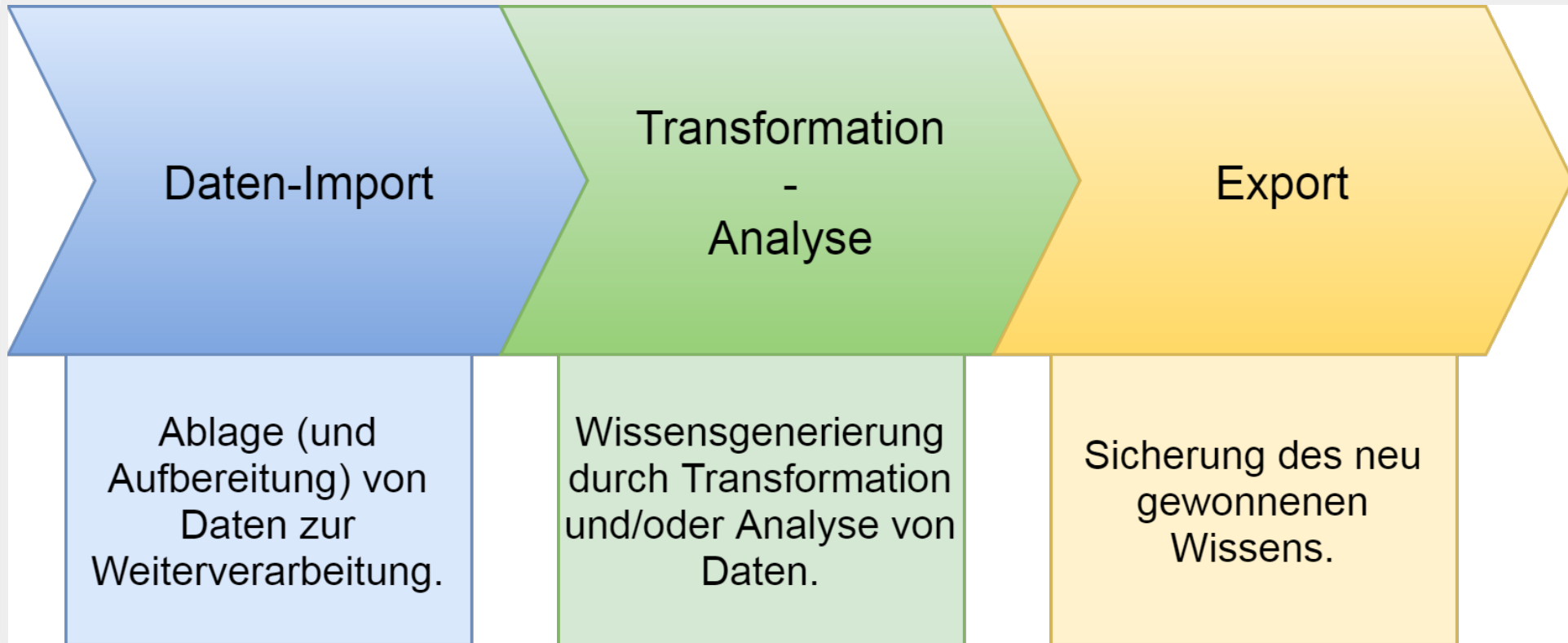
2013: Umzug auf GitHub, Open-Source-Projekt geworden

2019: Mit der Zeit immer weniger DNB-Ressourcen für Metafacture, hbz wird Maintainer

2019: Start von Metafacture Fix

2021: Start von Metafacture Playground

ETL



-

Wie Metafactory funktioniert

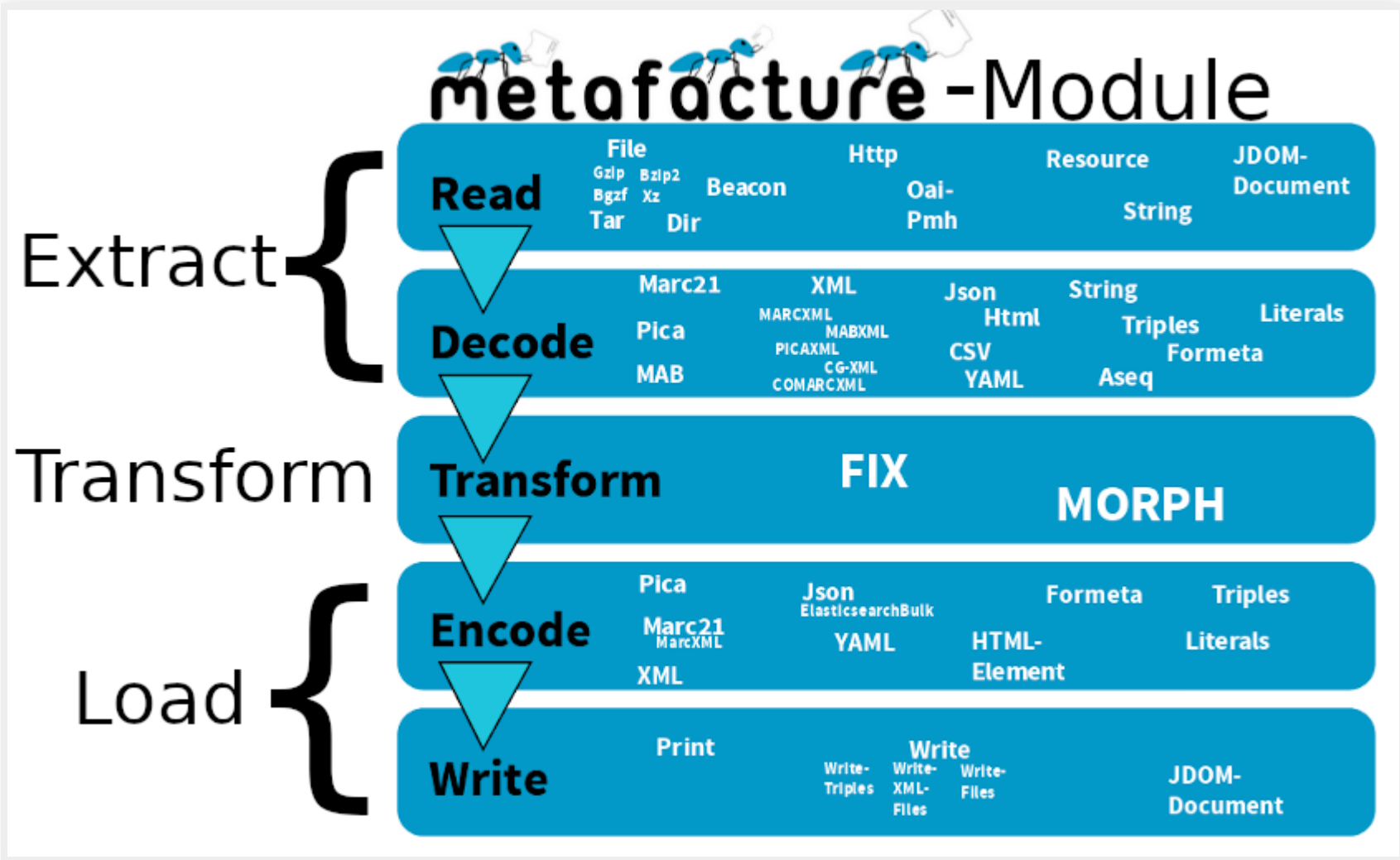
Grundidee: Daten fließen durch mehrere Module:
→ read → decode → transform → encode → write →

Jedes Modul erwartet Input eines bestimmten Typs und erzeugt Output eines bestimmten Typs

Verschiedene Formate werden unterstützt (z.B. METS/MODS, MARC), erweiterbares Framework für eigene Formate

Mittels Kombination einzelner Module bauen wir einen Workflow

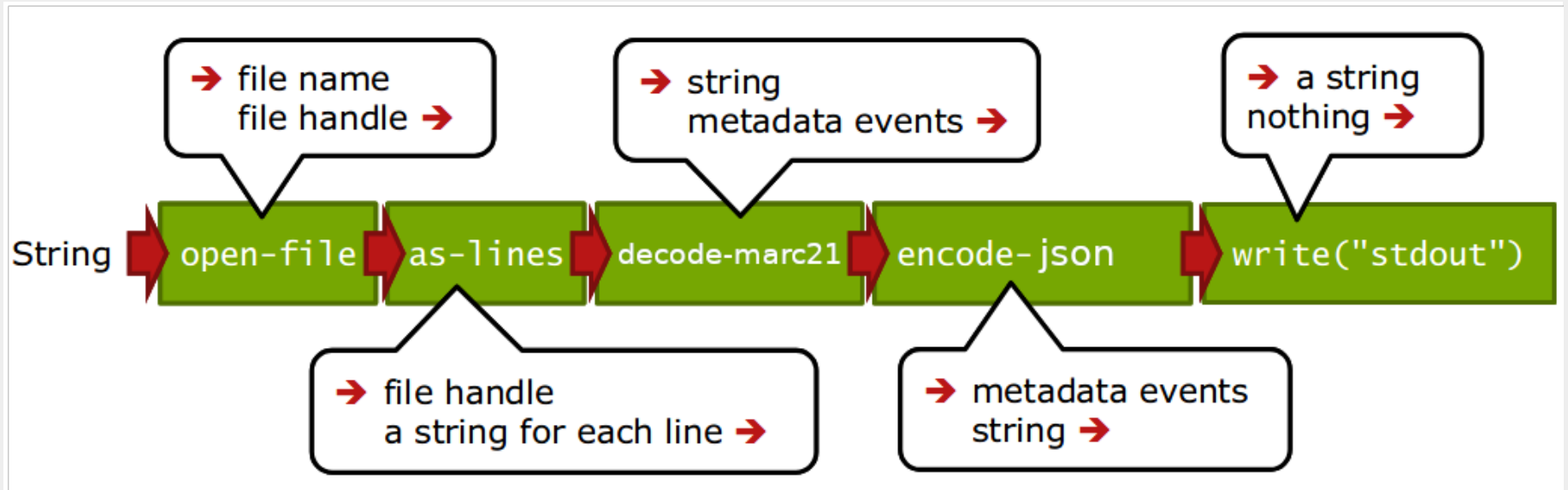
Metafactory-Module



Autoren: Tobias Bülte & Pascal Christoph (2021-2023)

2. Metafactory-Workflows

Ein Workflow



Aus: Christoph Böhme, nach http://swib.org/swib13/slides/boehme_swib13_131.pdf

```
"dateiName" | open-file | as-lines | decode-marc21 | encode-json | write("stdout");
```

(Beispiel-Inhalt einer "flux" Datei)

Workflows konfigurieren und ausführen

Workflows können in **Flux** (einer speziellen Konfigurationssprache) oder mit Java (typischer über Java Generics) bearbeitet werden

Flux-Workflows können in einem Texteditor editiert und auf der Kommandozeile ausgeführt werden; Java-Workflows funktionieren wie andere Java-Komponenten

Der Workshop führt in die Nutzung der **Flux**-Workflows ein, zum Ausführen verwenden wir den Metafactory Playground

Metafactory Playground

Webbasierte Oberfläche zum Ausprobieren und
Austauschen von Metafactory-Workflows

Ziel: Einstiegshürde für Metafactory senken, keine
Installation o.Ä. notwendig

Für Entwicklung, Dokumentation, Tutorials, Workshops

Mitarbeit erwünscht: [Metafactory Playground auf Github](#)

Funktionen des Playgrounds

<https://metafactory.org/playground>

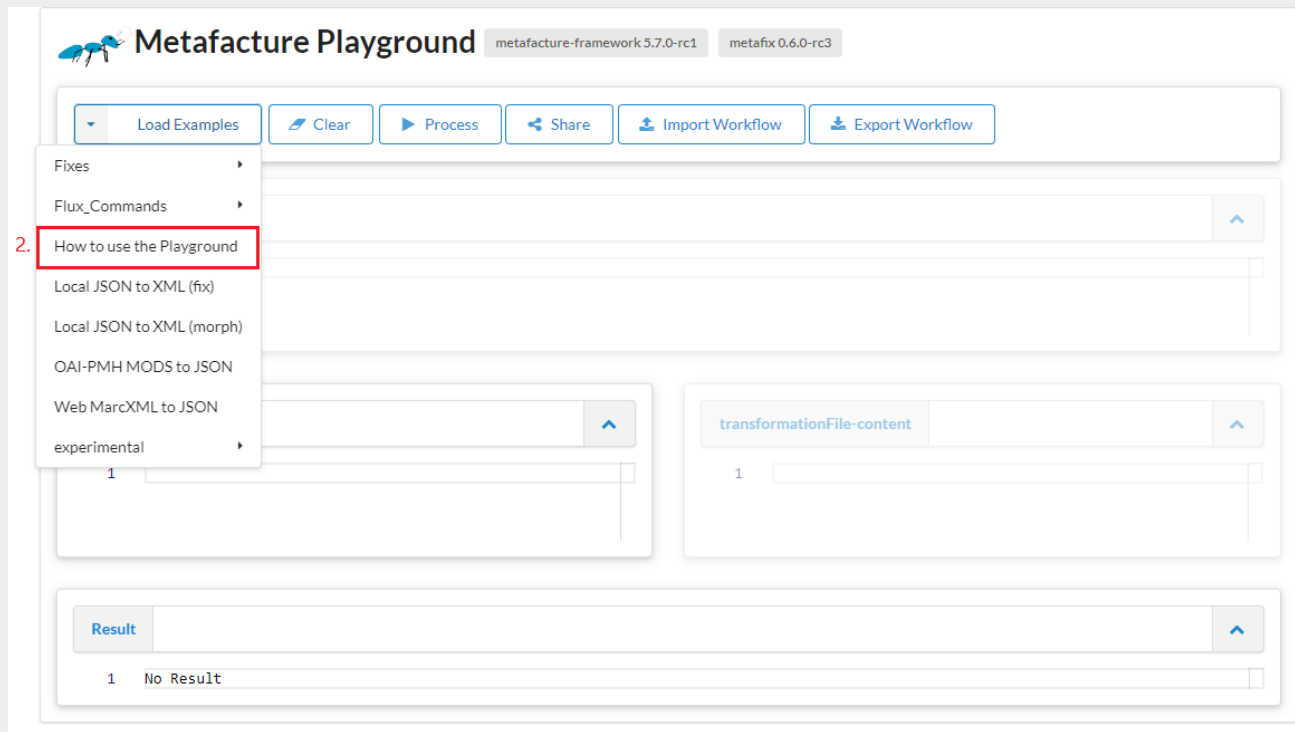
Funktionen des Playgrounds I

Load Examples

The screenshot displays the Metafacture Playground interface. At the top left, the logo features a blue ant and the text "Metafacture Playground". To the right of the logo are two version tags: "metafacture-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below the header is a toolbar with several buttons: "Load Examples" (highlighted with a red box), "Clear", "Process", "Share", "Import Workflow", and "Export Workflow". The main workspace contains three panels, each with a title bar and a text area. The top panel is titled "inputFile-content" and has a text area with the number "1" and a scroll bar. The bottom-left panel is titled "Flux File" and also has a text area with "1" and a scroll bar. The bottom-right panel is titled "transformationFile-content" and has a text area with "1" and a scroll bar. At the bottom of the interface is a "Result" panel with a text area containing the text "No Result" and a scroll bar.

Funktionen des Playgrounds I

Load Examples



The screenshot displays the Metafacture Playground interface. At the top, the logo and title 'Metafacture Playground' are visible, along with version information: 'metafacture-framework 5.7.0-rc1' and 'metafix 0.6.0-rc3'. Below this, a toolbar contains buttons for 'Load Examples', 'Clear', 'Process', 'Share', 'Import Workflow', and 'Export Workflow'. The 'Load Examples' dropdown menu is open, showing a list of example categories. The item 'How to use the Playground' is highlighted with a red box and a red '2.' next to it. Other items in the list include 'Fixes', 'Flux_Commands', 'Local JSON to XML (fix)', 'Local JSON to XML (morph)', 'OAI-PMH MODS to JSON', 'Web MarcXML to JSON', and 'experimental'. Below the menu, the main workspace shows a 'transformationFile-content' panel with a scrollable area containing the number '1'. At the bottom, a 'Result' panel shows '1 No Result'.

Metafacture Playground metafacture-framework 5.7.0-rc1 metafix 0.6.0-rc3

Load Examples Clear Process Share Import Workflow Export Workflow

Fixes

Flux_Commands

2. How to use the Playground

Local JSON to XML (fix)

Local JSON to XML (morph)

OAI-PMH MODS to JSON

Web MarcXML to JSON

experimental

1

transformationFile-content

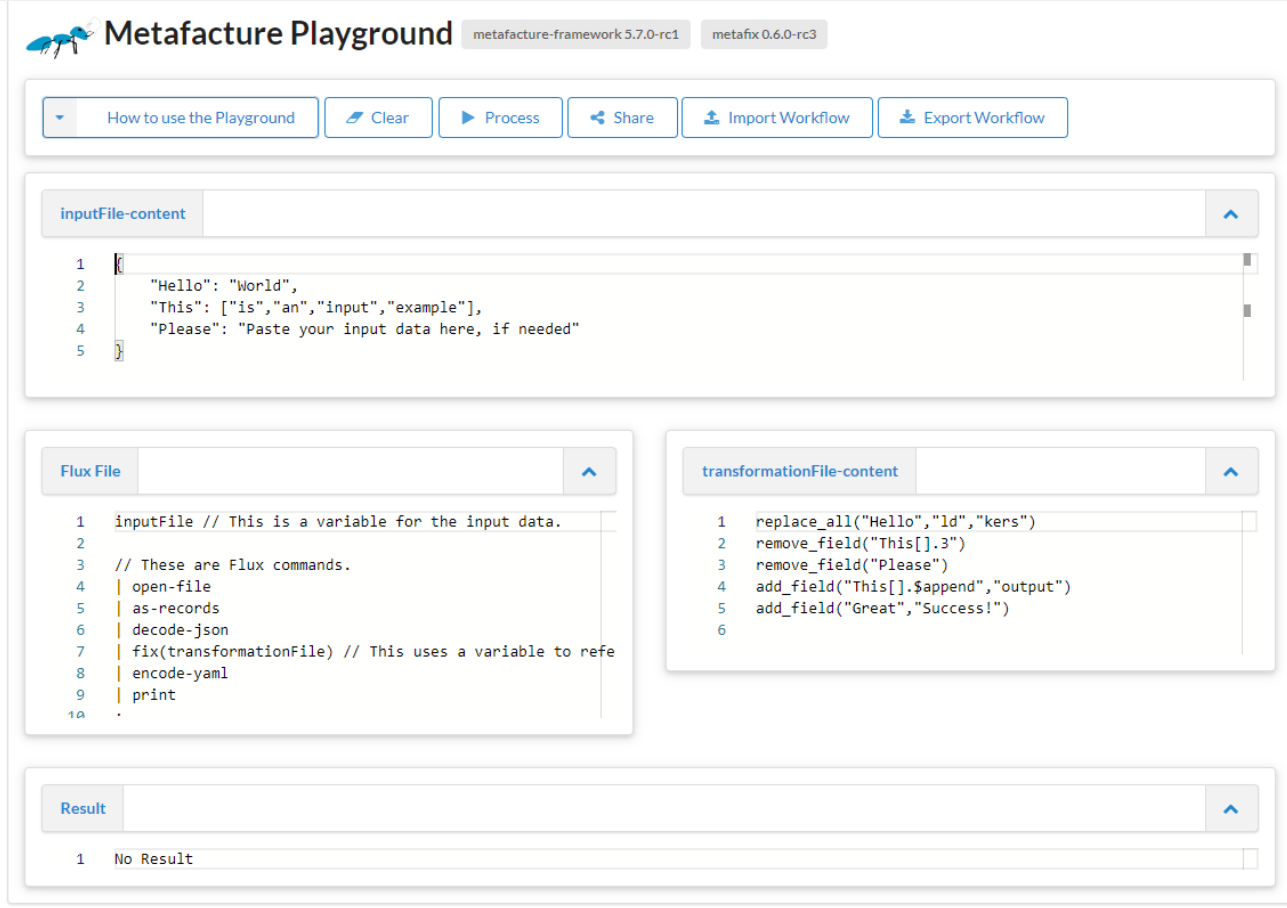
1

Result

1 No Result

Funktionen des Playgrounds I

Load Examples



The screenshot displays the Metafactory Playground interface. At the top, the logo features a blue ant and the text "Metafactory Playground" with version information: "metafactory-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below this is a navigation bar with buttons for "How to use the Playground", "Clear", "Process", "Share", "Import Workflow", and "Export Workflow".

The main workspace is divided into three panels:

- inputFile-content:** A code editor containing a JSON object:

```
1 {  
2   "Hello": "World",  
3   "This": ["is", "an", "input", "example"],  
4   "Please": "Paste your input data here, if needed"  
5 }
```
- Flux File:** A code editor containing Flux DSL commands:

```
1 inputFile // This is a variable for the input data.  
2  
3 // These are Flux commands.  
4 | open-file  
5 | as-records  
6 | decode-json  
7 | fix(transformationFile) // This uses a variable to refe  
8 | encode-yaml  
9 | print  
10 .
```
- transformationFile-content:** A code editor containing transformation rules:

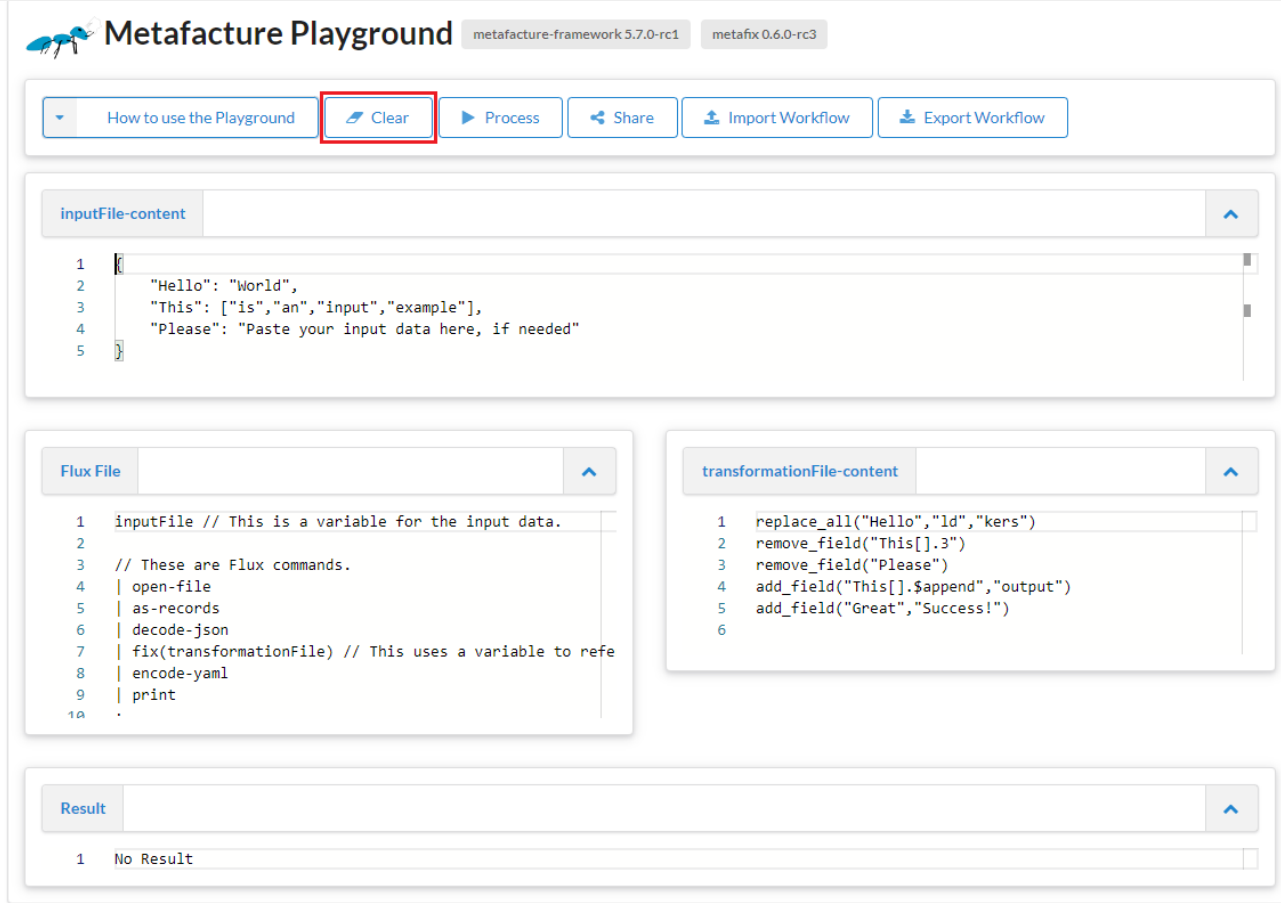
```
1 replace_all("Hello", "ld", "kers")  
2 remove_field("This[.3]")  
3 remove_field("Please")  
4 add_field("This[.$.append", "output")  
5 add_field("Great", "Success!")  
6
```

At the bottom, the **Result** panel shows the output:

```
1 No Result
```

Funktionen des Playgrounds II

Clear



The screenshot displays the Metafactory Playground interface. At the top, the logo and version information are shown: "Metafactory Playground" with versions "metafactory-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below this is a navigation bar with several buttons: "How to use the Playground", "Clear" (highlighted with a red box), "Process", "Share", "Import Workflow", and "Export Workflow".

The main workspace is divided into three sections:

- inputFile-content:** A code editor containing a JSON object:

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```
- Flux File:** A code editor containing Flux commands:

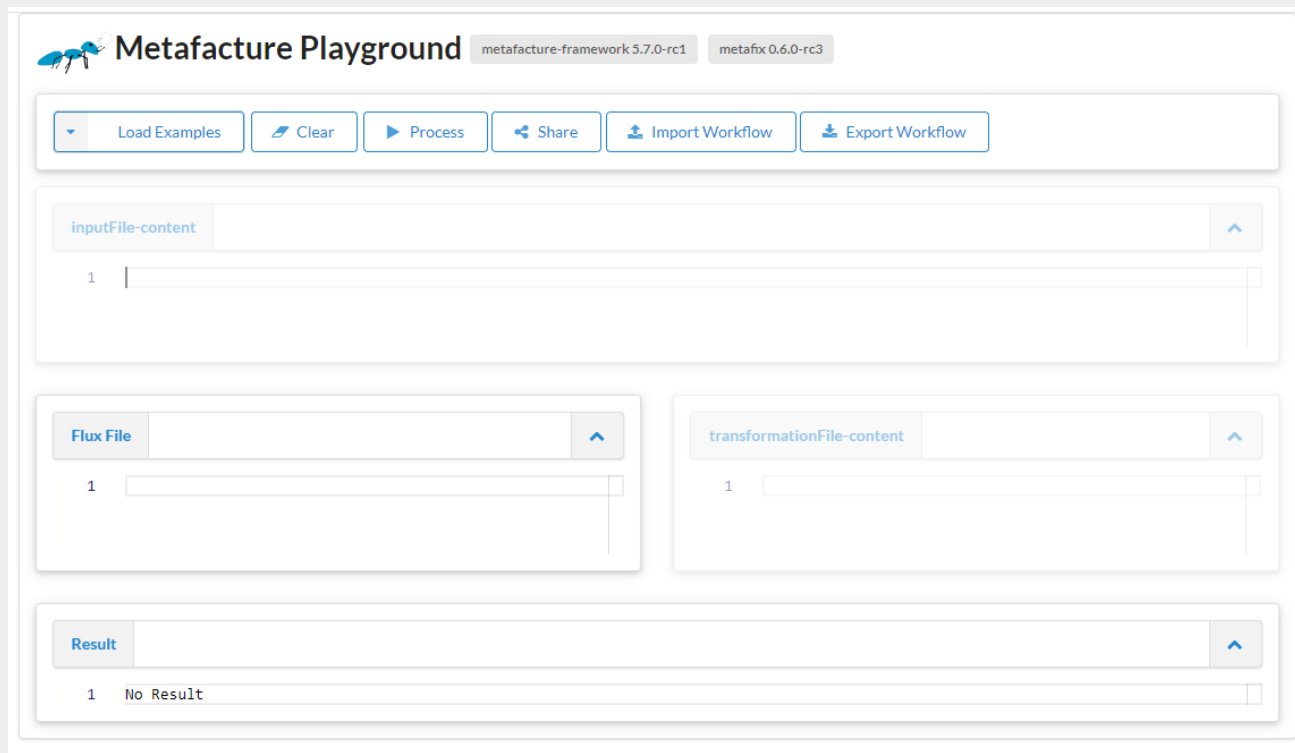
```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refe
8 | encode-yaml
9 | print
10 .
```
- transformationFile-content:** A code editor containing transformation rules:

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.$append", "output")
5 add_field("Great", "Success!")
6
```

At the bottom, the **Result** section shows "1 No Result".

Funktionen des Playgrounds II

Clear



The screenshot displays the Metafacture Playground interface. At the top left, the logo features a blue ant and the text "Metafacture Playground". To the right of the logo, two version tags are visible: "metafacture-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below the header is a toolbar with six buttons: "Load Examples" (with a dropdown arrow), "Clear" (with a trash icon), "Process" (with a play icon), "Share" (with a share icon), "Import Workflow" (with a download icon), and "Export Workflow" (with a download icon). The main workspace contains four text input fields, each with a tab and a scrollable area. The "inputFile-content" tab is active, showing a single line with the number "1" and a vertical cursor. Below it are two smaller input fields: "Flux File" and "transformationFile-content", both also showing "1" and a cursor. At the bottom, the "Result" tab is active, displaying "1 No Result" in the scrollable area.

Funktionen des Playgrounds III

Process

Metafacture Playground metafacture-framework 5.7.0-rc1 metafix 0.6.0-rc3

How to use the Playground Clear Process Share Import Workflow Export Workflow

inputFile-content

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

Flux File

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refe
8 | encode-yaml
9 | print
10 .
```

transformationFile-content

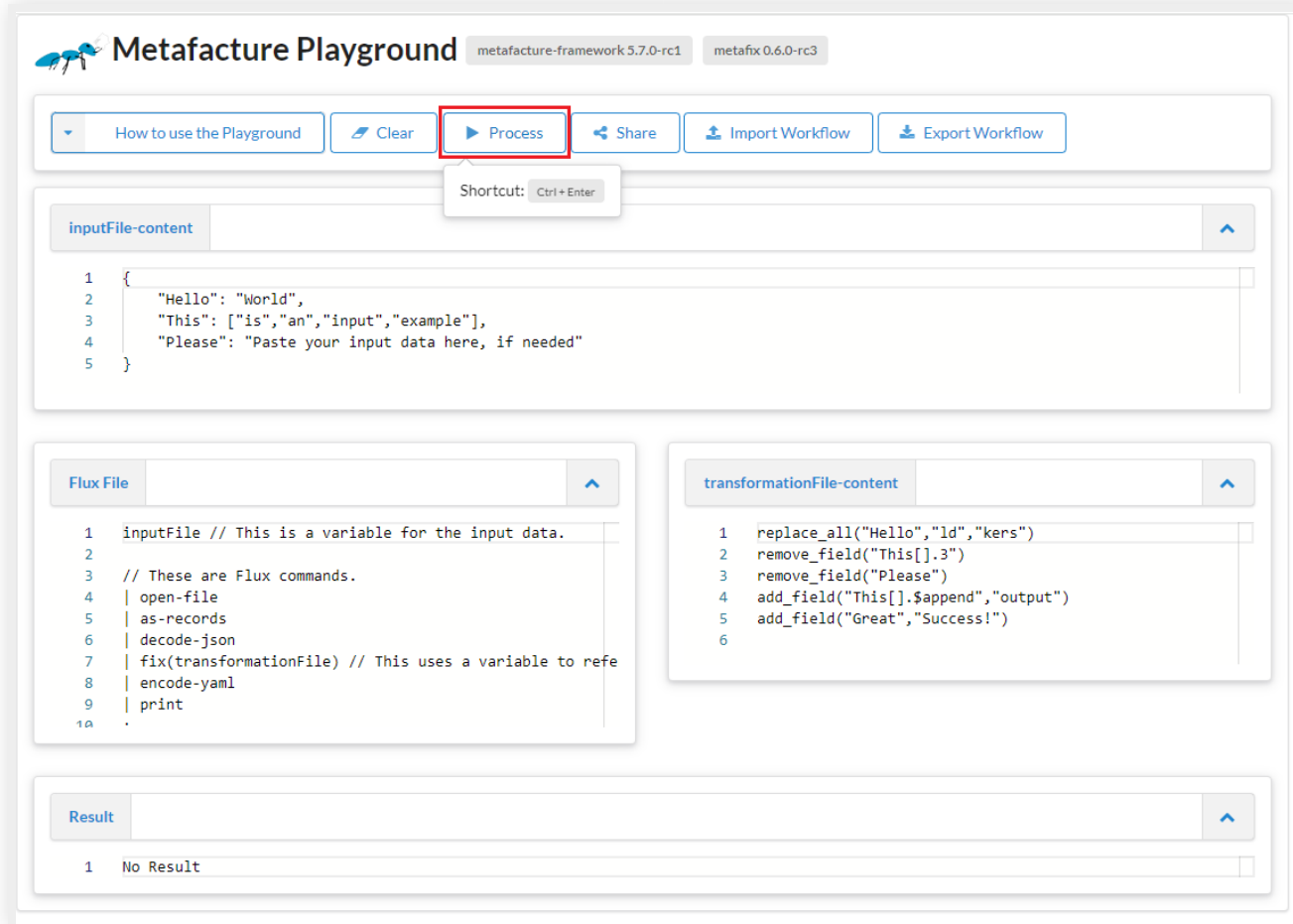
```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.$.append", "output")
5 add_field("Great", "Success!")
6
```

Result

```
1 No Result
```

Funktionen des Playgrounds III

Process



The screenshot displays the Metafacture Playground interface. At the top, the title "Metafacture Playground" is followed by version information: "metafacture-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below this is a navigation bar with buttons for "How to use the Playground", "Clear", "Process", "Share", "Import Workflow", and "Export Workflow". The "Process" button is highlighted with a red box, and a tooltip above it shows the shortcut "Shortcut: Ctrl+Enter".

The main workspace is divided into three sections:

- inputFile-content:** A code editor containing a JSON object:

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```
- Flux File:** A code editor containing Flux commands:

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refe
8 | encode-yaml
9 | print
10 .
```
- transformationFile-content:** A code editor containing transformation rules:

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[].$append", "output")
5 add_field("Great", "Success!")
6
```

At the bottom, the **Result** section shows the output:

```
1 No Result
```

Funktionen des Playgrounds III

Process

[How to use the Playground](#)[Clear](#)[Process](#)[Share](#)[Import Workflow](#)[Export Workflow](#)

inputFile-content

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

Flux File

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refe
8 | encode-yaml
9 | print
10 .
```

transformationFile-content

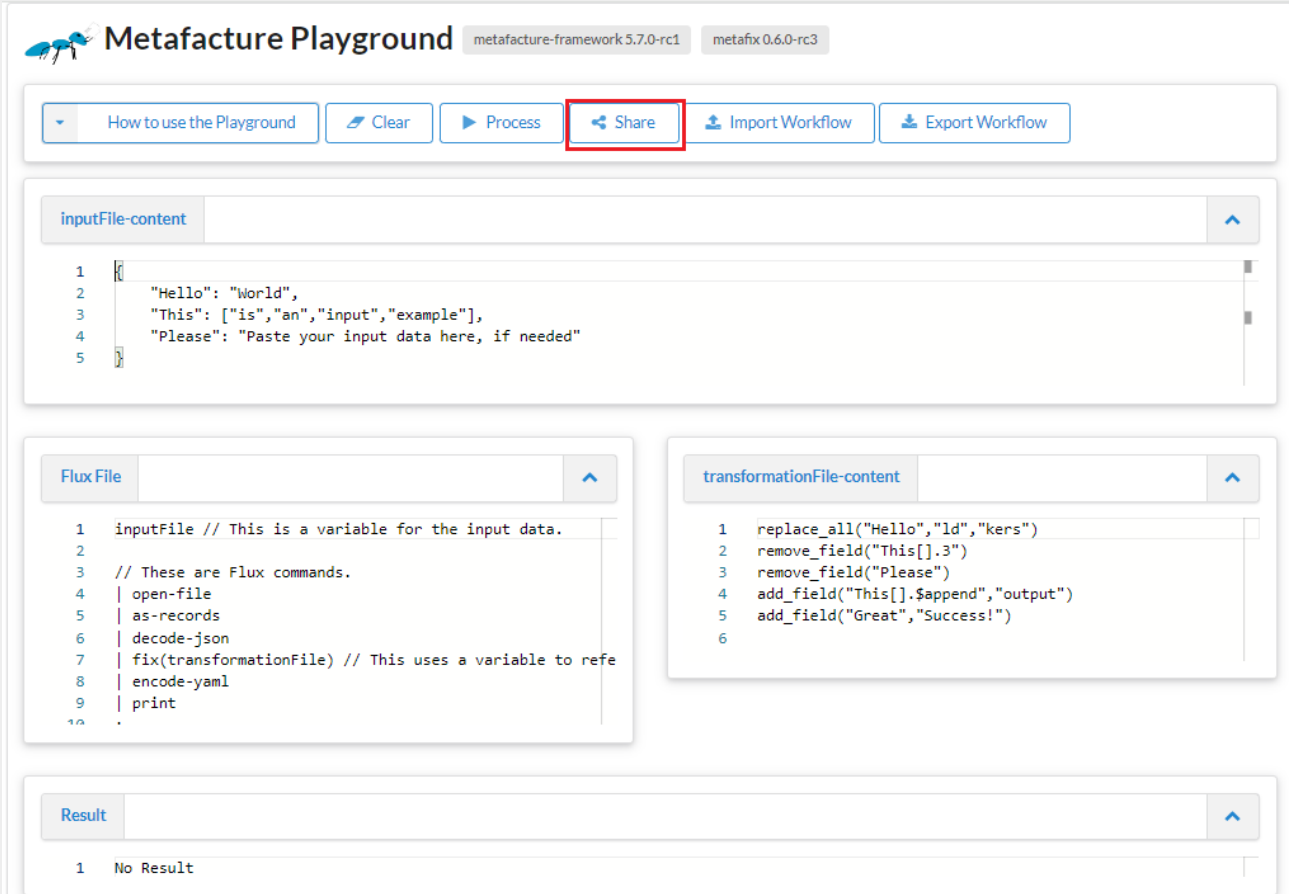
```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.$.append", "output")
5 add_field("Great", "Success!")
6
```

Result

```
1 ---
2 Hello: "Workers"
3 This:
4 - "is"
5 - "an"
6 - "example"
7 - "output"
8 Great: "Success!"
9
```

Funktionen des Playgrounds IV

Share



The screenshot displays the Metafactory Playground interface. At the top, the logo and title "Metafactory Playground" are visible, along with version information: "metafactory-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below this is a navigation bar with several buttons: "How to use the Playground", "Clear", "Process", "Share" (highlighted with a red box), "Import Workflow", and "Export Workflow".

The main workspace is divided into three sections:

- inputFile-content:** A text editor containing the following JSON data:

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```
- Flux File:** A text editor containing the following Flux script:

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refer to the transformation file.
8 | encode-yaml
9 | print
10 .
```
- transformationFile-content:** A text editor containing the following Flux transformation script:

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.$append", "output")
5 add_field("Great", "Success!")
6
```

At the bottom, the **Result** section shows "1 No Result".

Funktionen des Playgrounds IV

Share

The screenshot displays the Metafacture Playground interface. At the top, the logo and version information (metafacture-framework 5.7.0-rc1 and metafix 0.6.0-rc3) are visible. Below this is a navigation bar with buttons for 'How to use the Playground', 'Clear', 'Process', 'Share', 'Import Workflow', and 'Export Workflow'. The 'Share' button is highlighted with a red box, and its dropdown menu is open, showing two options: 'Workflow' and 'Result call'. Both options include a URL starting with 'https://metafacture.org/playground/?' and a share icon. The main workspace is divided into several panels: 'inputFile-content' containing a JSON object, 'Flux File' containing a Flux script, 'transformationFile-content' containing a list of Flux commands, and 'Result' showing 'No Result'.

Metafacture Playground metafacture-framework 5.7.0-rc1 metafix 0.6.0-rc3

How to use the Playground Clear Process Share Import Workflow Export Workflow

Workflow
https://metafacture.org/playground/?

Result call
https://metafacture.org/playground/?

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

Flux File

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refer to the transformation file.
8 | encode-yaml
9 | print
10 .
```

transformationFile-content

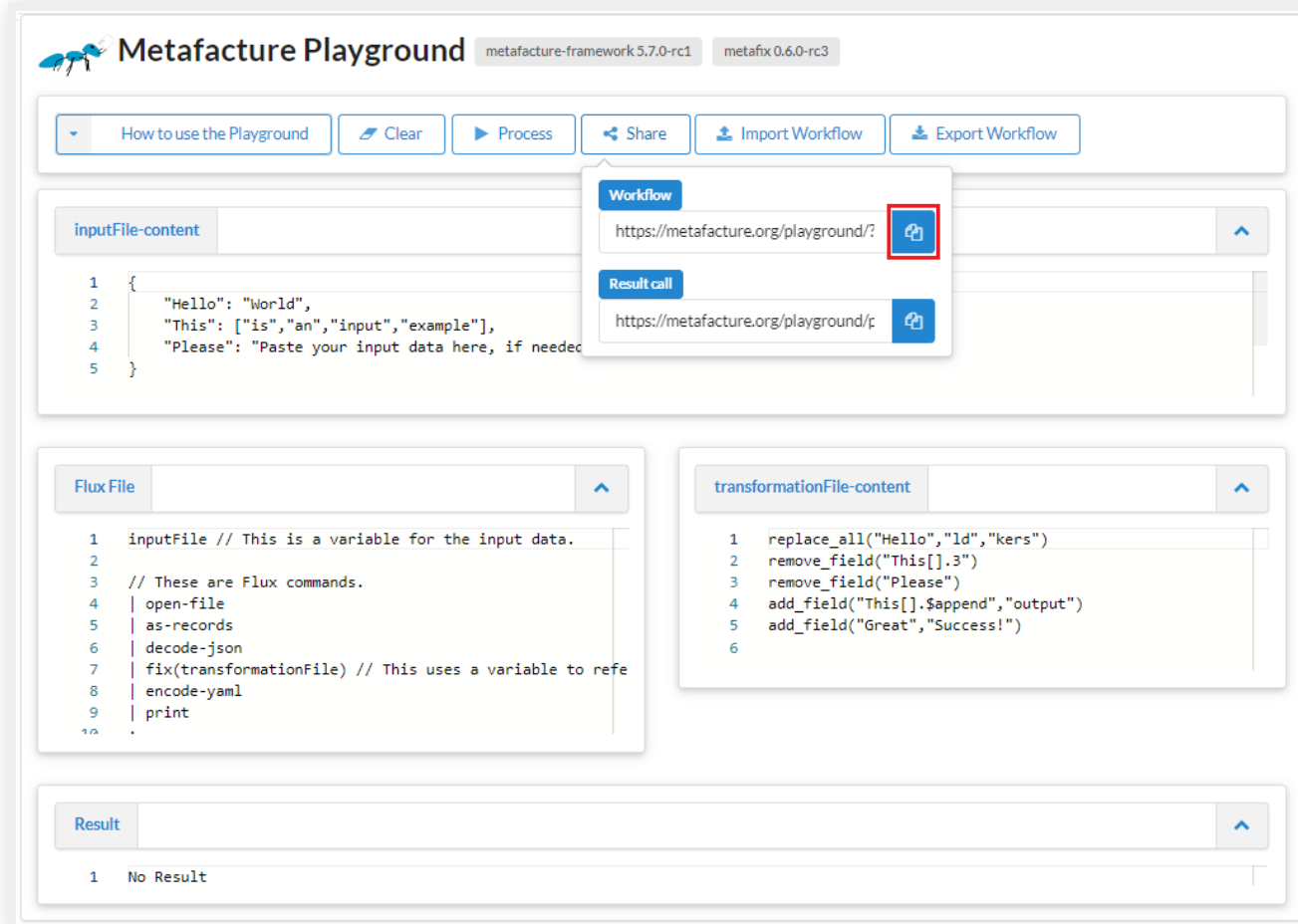
```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.].$append", "output")
5 add_field("Great", "Success!")
6
```

Result

```
1 No Result
```

Funktionen des Playgrounds IV

Share



The screenshot displays the Metafacture Playground interface. At the top, the title "Metafacture Playground" is accompanied by version information: "metafacture-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below the title is a navigation bar with buttons for "How to use the Playground", "Clear", "Process", "Share", "Import Workflow", and "Export Workflow". The "Share" button is highlighted with a red box, and a dropdown menu is open, showing two options: "Workflow" and "Result call", each with a share icon and a URL starting with "https://metafacture.org/playground/?".

The main content area is divided into several sections:

- inputFile-content**: A text area containing a JSON object:

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```
- Flux File**: A text area containing Flux commands:

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refer to the transformation file.
8 | encode-yaml
9 | print
10 .
```
- transformationFile-content**: A text area containing Flux transformation commands:

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.].$append", "output")
5 add_field("Great", "Success!")
6
```
- Result**: A text area showing "1 No Result".

Funktionen des Playgrounds IV

Share

The screenshot displays the Metafacture Playground interface. At the top, the URL `metafacture.org/playground/?example=How+to+use+the+Playground` is highlighted in a red box. Below the URL bar, a navigation bar contains buttons for "How to use the Playground", "Clear", "Process", "Share", "Import Workflow", and "Export Workflow". The "Share" button is active, and its dropdown menu is open, showing two options: "Workflow" and "Result call", each with a corresponding URL and a share icon. The main workspace is divided into several panels: "inputFile-content" with a JSON object, "Flux File" with a Flux script, "transformationFile-content" with a transformation script, and "Result" with "No Result".

```
metafacture.org/playground/?example=How+to+use+the+Playground
```

Workflow
<https://metafacture.org/playground/?>

Result call
<https://metafacture.org/playground/?>

```
inputFile-content
```

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

```
Flux File
```

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refer to the transformation file.
8 | encode-yaml
9 | print
10 .
```

```
transformationFile-content
```

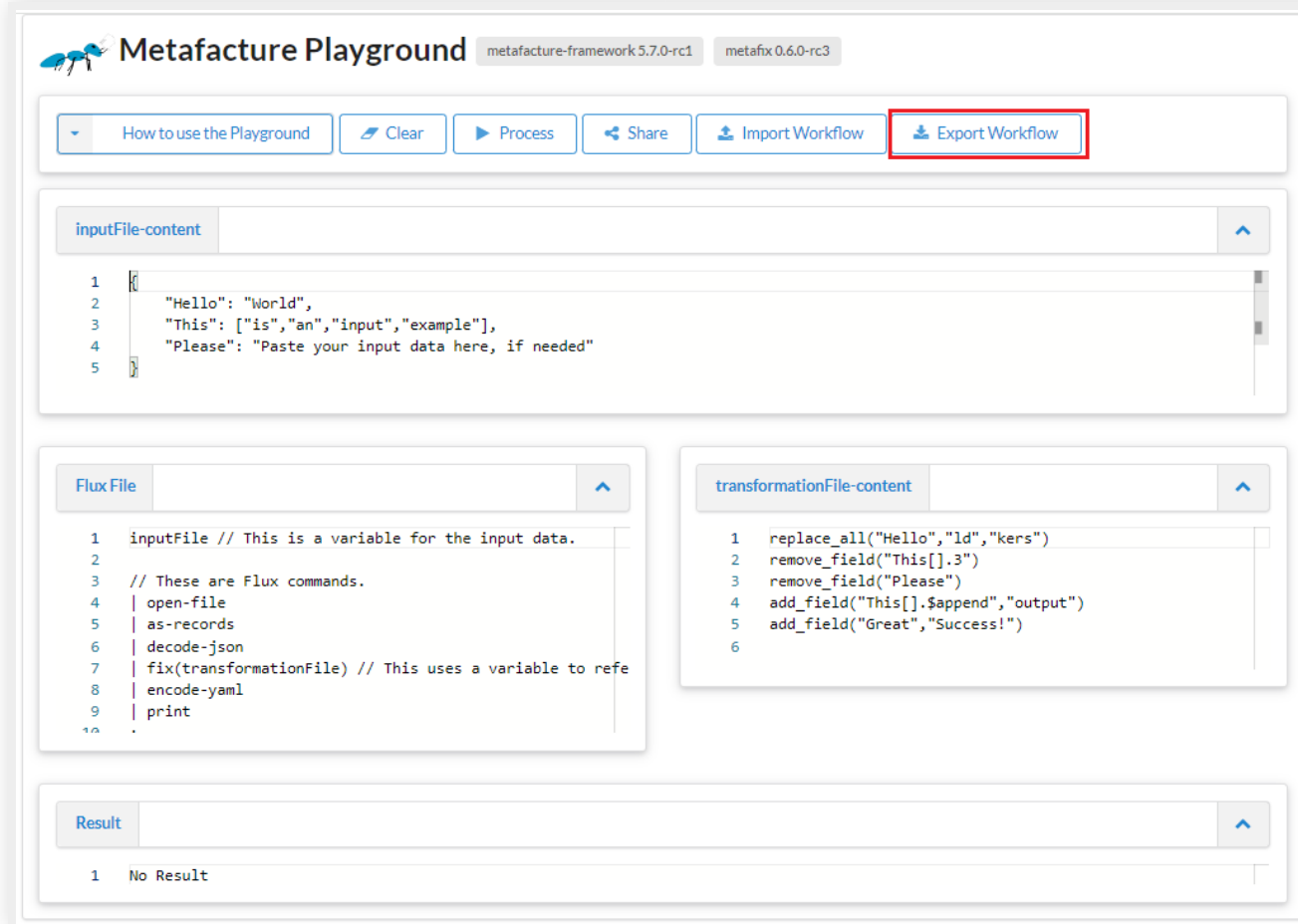
```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.].$append", "output")
5 add_field("Great", "Success!")
6
```

```
Result
```

```
1 No Result
```

Funktionen des Playgrounds V

Export Workflow



The screenshot displays the Metafacture Playground interface. At the top, the logo and version information are shown: "Metafacture Playground" with versions "metafacture-framework 5.7.0-rc1" and "metafix 0.6.0-rc3". Below this is a navigation bar with buttons for "How to use the Playground", "Clear", "Process", "Share", "Import Workflow", and "Export Workflow". The "Export Workflow" button is highlighted with a red border. The main workspace is divided into three sections: "inputFile-content", "Flux File", and "transformationFile-content". The "inputFile-content" section contains a JSON object. The "Flux File" section contains a Flux script. The "transformationFile-content" section contains a list of Flux transformation commands. The "Result" section at the bottom shows "No Result".

Metafacture Playground metafacture-framework 5.7.0-rc1 metafix 0.6.0-rc3

How to use the Playground Clear Process Share Import Workflow **Export Workflow**

inputFile-content

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

Flux File

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to refe
8 | encode-yaml
9 | print
10 .
```

transformationFile-content

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[.3]")
3 remove_field("Please")
4 add_field("This[.$append", "output")
5 add_field("Great", "Success!")
6
```

Result

```
1 No Result
```

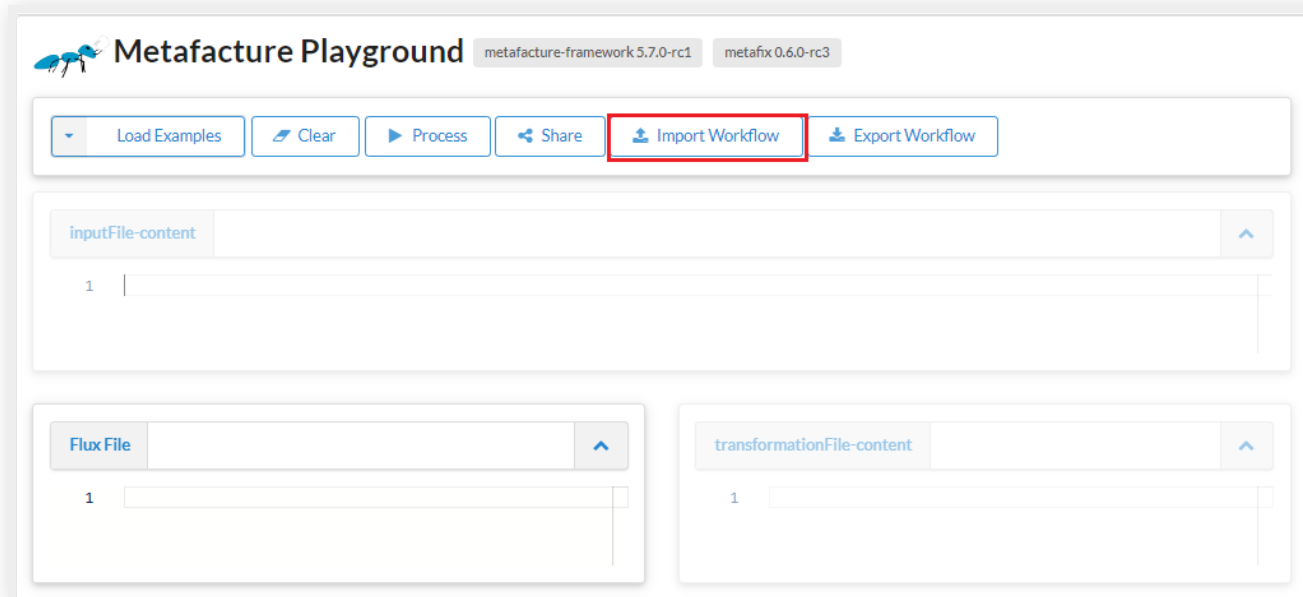
Funktionen des Playgrounds V

Export Worflow

```
1 default inputFile = FLUX_DIR + "playground.data";
2 default transformationFile = FLUX_DIR + "playground.fix";
3 inputFile // This is a variable for the input data.
4
5 // These are Flux commands.
6 | open-file
7 | as-records
8 | decode-json
9 | fix(transformationFile) // This uses a variable to refer
10 | encode-yaml
11 | print
12 ;
```

Funktionen des Playgrounds V

Import Workflow



Funktionen des Playgrounds V

Import Workflow



Info

- The flux content has been adapted to work in the playground. Additional adjustments could be necessary.
- Imported workflow with files:
 - playground.data
 - playground.fix
 - playground.flux

▾ Load Examples🗑 Clear▶ Process↔ Share📄 Import Workflow📄 Export Workflow

inputFile-content

```
1 {
2   "Hello": "World",
3   "This": ["is", "an", "input", "example"],
4   "Please": "Paste your input data here, if needed"
5 }
```

Flux File

```
1 inputFile // This is a variable for the input data.
2
3 // These are Flux commands.
4 | open-file
5 | as-records
6 | decode-json
7 | fix(transformationFile) // This uses a variable to reference the content of the box transformationFile.
8 | encode-yaml
9 | print
10 .
```

transformationFile-content

```
1 replace_all("Hello", "ld", "kers")
2 remove_field("This[].3")
3 remove_field("Please")
```


Übung 1

Erste Schritte im Playground

Bevor wir mit der ersten Übung starten...
... ein paar Hinweise/Tipps/Hilfestellungen:
die wichtigsten **Flux-commands** für heute:

open-file / open-http

as-lines / as-records

decode-xml / decode-json

handle-marcxml

fix

encode-marcxml / encode-json

print

Übung 1 - Daten einlesen & ausgeben

Beispieldatensatz: <https://d-nb.info/1251888070>

wir brauchen den Link zum MARC-XML dieses Records:
<https://d-nb.info/1251888070/about/marcxml>

→ dieser URL wird in der Flux-File als Pfad zu den Daten angegeben

```
"https://d-nb.info/1251888070/about/marcxml"
```

Übung 1 - Daten einlesen & ausgeben

wir möchten Daten direkt von einer http Ressource öffnen und benötigen daher...?

parsing der xml-Daten und lesen der Daten erfolgt in 2 Schritten ...

ein Blick in die Doku: "Parsing" der XML-Datei und "Lesen" des MARC-XML

probiert im Playground folgenden Ablauf selbst aus

```
"https://d-nb.info/1251888070/about/marcxml"  
| open-http(accept="application/xml")  
| decode-xml  
| handle-marcxml  
| encode-marcxml  
| print  
;
```

Übung 1 - Daten einlesen & ausgeben

ändert die Ausgabe der Daten in JSON ab
wie kann man das JSON leichter lesbar ("schöner")
ausgeben?

💡 Blick in die Doku der **Flux-commands** hilft

Wer fertig ist: bitte im Status "Daumen hoch" setzen

Pause 

3. Transformieren

ETL: Transformation

Daten-Import

Transformation
-
Analyse

Export

Ablage (und
Aufbereitung) von
Daten zur
Weiterverarbeitung.

Wissensgenerierung
durch Transformation
und/oder Analyse von
Daten.

Sicherung des neu
gewonnenen
Wissens.

Transformieren

Manipulation von Feldnamen und -werten; filtern, kombinieren, trennen, normalisieren, anreichern etc.

Änderung des Datenformats, der Hierarchie

"Vorarbeiten" der Datentransformation: Analyse

Überblick über vorhandene Felder verschaffen (hierfür ist bspw. das Flux-Command "**list-fix-paths**" hilfreich)

```
"https://d-nb.info/1251888070/about/marcxml"  
| open-http(accept="application/xml")  
| decode-xml  
| handle-marcxml  
| list-fix-paths(count="false")  
| print  
;
```

Ggf. Metadatenstandard vorab anschauen (Pflichtfelder,
Wiederholbarkeit, Struktur)

Feldinhalte (in Stichproben) anschauen

"Vorarbeiten" der Datentransformation: Analyse

Überblick über vorhandene **Feldinhalte** verschaffen

→ schaut in der **Flux-Commands-Übersicht**, womit und wie man die Inhalte analysieren kann. (2 min)

Ausgeben eines bestimmten Feldinhalts:

```
"https://d-nb.info/1251888070/about/marcxml"  
| open-http(accept="application/xml")  
| decode-xml  
| handle-marcxml  
| list-fix-values("650 7.*.a")  
| print  
;
```

oder auch die Inhalte aller Felder mit

```
list-fix-values("*")
```

Transformationsmodul Fix

eigene Entwicklung

Catmandu-Fix-artig

Record-basiert

Metafactory Fix: Ziele

Erleichterung der Transformationskonfiguration

Anknüpfung an existierende Konfigurationssprache aus
Catmandu

Vergrößerung der Zielgruppe um Bibliothekar:innen und
andere Metadatenfachleute

Metafactory Fix: Dokumentation

Eine Einführung in Metafactory Fix bietet der [Fix User Guide](#)

Darüber hinaus lohnt ein Blick ins ["Kochbuch"](#)

Bevor wir mit der zweiten Übung starten...
die wichtigsten Fix-Funktionen für heute:

`copy_field / move_field`

`retain`

Einbinden der Fix in die Flux im Playground: entweder mit `fix(transformationFile)` → Fix in das entsprechende Feld ODER direkt in das Flux-Command `fix("...")`

Path wildcards:

* → 1 & mehr Zeichen ; ? → genau 1 Zeichen

Kurzes Recap MARC21-Felder

Gesamtübersicht bei der LoC

LEADER 06 - Type of record (NR)	a - Language material m - Computer file ...
020 - ISBN (R)	Ind1; Ind2 \$a ISBN
245 - Title Statement (NR)	Ind1 (NE); Ind2 (Nichtsortierzeichen); \$a Titel ; \$b Zusatz ; \$c Verantwortlichkeitsangabe
264 - Production, Publication, Distribution, Manufacture, and Copyright Notice (R)	Ind1; Ind2 \$a Entstehungsort ; \$b Verlagsname ; \$c Entstehungsjahr
650 - Subject Added Entry - Topical Term (R)	Ind1; Ind2 \$a Sachbegriff
700 - Added Entry - Personal Name (R)	Ind1; Ind2 \$a Personennamen; \$d Datumsangaben; \$e Funktionsbezeichnung; \$4 Code der Funktionsbezeichnung

Übung 2 - Metafactory Fix: Pfade

Um die verschiedenen Elemente und Felder für die Transformation anzusteuern, benötigt man die Pfade

Oberste Ebene: z.B. *id*

Unterfelder: z.B. *title.subtitle*

Wiederholte Felder werden als Listen (arrays) mit Index-Nummer angegeben: z.B. *creator.1.name.firstName*

Übung 2 - Metafactory Fix: Pfade

💡 Wiederholte Felder werden als "arrays/Listen" gelesen - was heißt das in der Praxis?

Feldname einfach

```
inputFile
---
name1: Zeng, Marcia Lei
name2: Coyle, Karen
name3: Avram, Henriette

Flux
inputFile
|open-file
|as-records
|decode-yaml
|fix("copy_field('name1', 'creator')
retain('creator')")
|encode-json(prettyPrinting="true")
|print
;
```

Feldname wiederholt

```
inputFile
---
name: Zeng, Marcia Lei
name: Coyle, Karen
name: Avram, Henriette


Flux
inputFile
|open-file
|as-records
|decode-yaml
|fix("copy_field('name.1',
'creator') retain('creator')")
|encode-json(prettyPrinting="true")
|print
;
```

Übung 2 - Metafactory Fix: Pfade

Basis ist der **Beispieldatensatz** aus Übung 1 (im **Playground**)

Vorgehen "Titel auslesen":

- Pfad bestimmen
- Feldinhalt in ein neues Feld "title" **kopieren**
- mit **retain** nur diesen Feldinhalt ausgeben lassen

 Hilfestellung: Feld 245 hat 2 Indikatoren

Wer fertig ist: bitte im Status "Daumen hoch" setzen

Übung 2 - Metafactory Fix: Pfade

Ausgeben des gesamten Title statement

Lösung für die Übung: Nur Titel ausgeben

Sieht es bei jemandem anders aus?

weitere Fix-Funktionen

set_array

do list

lookup

put_map

do list

Iteriert über eine Liste (zur Erinnerung: wiederholte Felder werden als Listen interpretiert!)

Durch das Setzen eines Variablennamens für das Wurzelement können die zugehörigen untergeordneten Elemente angesprochen werden.

wird mit **end** geschlossen

do list

im Playground

inputFile-Content

```
{ "author": [
  { "name": "British Library",
    "@type": "Organization" },
  { "name": "Ada Lovelace",
    "@type": "Person" },
  { "name": "Alan Turing",
    "@type": "Person" }
]
```

Flux File

```
inputFile
|open-file
|as-records
|decode-json
|fix(transformationFile)
|encode-yaml
|print
;
```

TransformationFile-Content

```
do list(path:"author[]", "var":"$i")
  move_field("$i.@type", "$i.type")
end
```

Pause 

Übung 3 - MARC-XML to DC

Vorarbeiten: wie sollen die Felder gemapped werden?

MARC21 und DC sind verbreitete Standards; es gibt bereits sog. Crosswalks (z.B. **LoC**), die zur Orientierung herangezogen werden können

Übung 3 - MARC-XML to DC

wir mappen

MARC21	DC
020 \$a	dc:identifier
245 \$a \$b	dc:title
700 \$a	dc:creator
264 \$c	dc:date
264 \$b	dc:publisher
LDR Pos. 7	dc:type [*]
650 \$a	dc:subject

[*] empfohlen wird die Verwendung v. kontrolliertem Vokabular wie bspw. [DCMI TYPE Vocabulary](#)

Übung 3 - MARC-XML to DC

Nehmen wir wieder den **Beispieldatensatz** aus Übung 1

Wie in Übung 2 könnt Ihr das Entstehungsjahr und den Verlag in ein Feld "date" bzw. "publisher" kopieren

Wer fertig ist: bitte im Status "Daumen hoch" setzen

Übung 3 - MARC-XML to DC

wir sollten nun alle **diesen Stand haben**

die Felder 700\$a, 020\$a und 650\$a müssen etwas anders abgearbeitet werden

die ISBN(s) und auch die wiederholten Felder werden in einem Array ausgegeben - auch wenn es nur 1 gibt

benutzt **set-array** und array-Wildcard **\$append** - probiert es aus (und schaut ggf. in die Doku)

Wer fertig ist: bitte im Status "Daumen hoch" setzen

Lösung bis hierhin im Playground

Übung 3 - MARC-XML to DC

für das Mappen des Leader Pos. 7 auf Kontrolliertes Vokabular → **lookup**

Zuerst: map erstellen entweder in der Fix oder in einer separaten Datei.

Key-value-pairs für die Werte: a - Language material → in DC "text"; m - Computer file → in DC "dataset"

```
put_map("ldr_type", key1:"value1",...)
```

Wer fertig ist: bitte im Status "Daumen hoch" setzen

Es sollte so aussehen:

```
put_map("ldr_type", "a":"text", "m":"dataset")
```

Übung 3 - MARC-XML to DC

Wie kommt man aber an die Pos. 7 im Leader?

Hierfür kann man Reguläre Ausdrücke in Kombination mit der Fix-Funktion `replace_all` nutzen

```
replace_all("type", "^.{6}(.).+$", "$1")
```

Übung 3 - MARC-XML to DC

Lösung im Playground

Von JSON zu JSON-LD

Ergänzung @context

4. Exkurs: Anreichern

Lookups auf Tabellen

Einfacher Lookup auf eine zweispaltige tsv Tabelle, die im Web liegt

RDF web lookups

In Alma Marc XML haben wir nur die IDs der "DNB Sachgruppen":

```
<datafield tag="084" ind1=" " ind2=" ">  
  <subfield code="a">333.7</subfield>  
  <subfield code="a">621.3</subfield>  
  <subfield code="q">DE-600</subfield>  
  <subfield code="2">sdnb</subfield>  
</datafield>
```

Lookup auf "DNB Sachgruppen" um mit skos:notation und skos:prefLabel anzureichern

(Anreicherungsbeispiel im lobid-Frontend)

4. Exkurs: Conditionals

Bedingte Anweisungen werden häufig benötigt bei der Transformation

in Fix eingeleitet mit **if** oder **unless** und geschlossen mit einem **end**

Das Abarbeiten mehrerer Bedingungen wird durch ggf. **elsif** / **else** realisiert

bspw. "if **exists**" / "if **any_match**"

```
...
if any_match("$auth", "^http.+$")
  copy_field("$auth", "contribution[].$last.agent.id")
elsif any_match("$auth", "^\\(DE-588\\).+$")
  copy_field("$auth", "contribution[].$last.agent.gndIdentifier")
  replace_all("contribution[].$last.agent.gndIdentifier", "\\(DE-588\\)", "")
else
  add_field("contribution[].$last.agent.gndIdentifier", "keine GND-ID")
end...
```

5. Ausblick, Fragen & Diskussion

Ausblick

Fix & Playground weiterentwickeln

Fix-Funktionalität erweitern, Fehler beheben, Catmandu-Kompatibilität erhöhen

Playground weiter verbessern, z.B. mehr Hinweise im Editor, integrierte Dokumentation (was gibt es für Module, wie kann ich sie kombinieren)

Standards nutzen und aufbauen

SKOS Lookups (zum Andocken an [SkoHub Vocabs](#)) ✓

[Entity Reconciliation](#) mit [OpenRefine-kompatiblen Diensten](#)

Fix-Standardisierung, s. <https://github.com/elag/FIG>

ETL Hub

ETL: Extract, Transform, Load

mehr Kollaboration, Teilen & Auffinden von Workflows ermöglichen (nicht nur für Metafactory)

Entwicklung von Best Practices zur Paketierung und Beschreibung von ETL-Konfigurationen

Aufbau eines ETL Hubs zum Entdecken existierender ETL-Prozesse für die einfache Nachnutzung und Anpassung

Großes Interesse bekundet auf der SWIB 23

Fragen und Diskussion

Auf der Discourse-Plattform unter metadaten.community



The screenshot shows the Discourse forum interface for 'metadaten.community'. The top navigation bar includes links for 'Über uns', 'Forumsregeln', 'Nutzungsbedingungen', 'Datenschutz', and 'Impressum'. The left sidebar contains navigation options like 'Themen', 'Meine Beiträge', 'Alle Veranstaltungen', and 'Kategorien'. The main content area features a post titled 'metafactory' with a logo showing three ants. The post text describes 'metafactory' as a specialized toolkit for data processing, developed within the CultureGraph framework and maintained by hbz since 2019. Below the text are filter buttons for 'Software & Tools', 'Metafactory', and 'alle Schlagwörter'. The post is categorized under 'Aktuell' and has 0 answers, 20 views, and was posted in December 2023. The post content reads: 'Das waren alle Themen in der Kategorie „Metafactory“. Bereit, eine neue Unterhaltung zu beginnen?'.

metadaten.
community

Über uns Forumsregeln Nutzungsbedingungen Datenschutz Impressum

Themen ●
Meine Beiträge
Alle Veranstaltungen
Mehr

Kategorien

Abschlussarbeiten ●
Allgemein
Arbeitsgruppen
Meta
Software & Tools ●
Veranstaltungen
Alle Kategorien

Schlagwörter

kos ●
lis ●

metafactory

metafactory ist ein auf bibliographische Metadaten spezialisiertes Toolkit für Datenprozesse. Ursprünglich wurde es von der DNB im Rahmen von CultureGraph entwickelt, 2019 hat das hbz die Maintainerschaft übernommen.

Software & Tools ▶ Metafactory ▶ alle Schlagwörter ▶

Aktuell Neu Ungelesen Angesagt + Neues Thema

Thema Antworten Aufrufe Aktivität

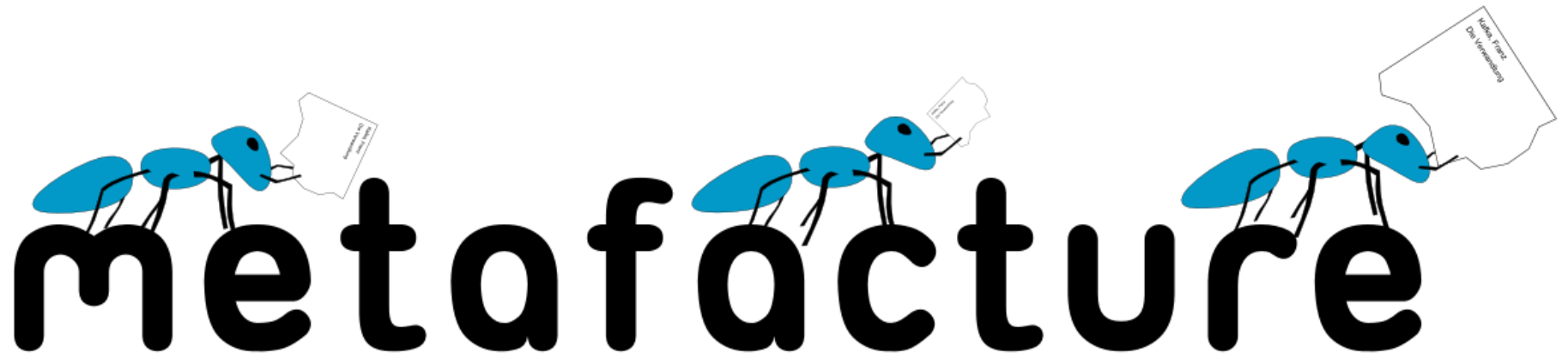
★ Über die Kategorie Metafactory 0 20 Dez. '23

Das waren alle Themen in der Kategorie „Metafactory“. Bereit, eine neue Unterhaltung zu beginnen?

Feedbackrunde

Immer im Wechsel: etwas Positives, etwas Negatives

Wir starten mit...



<https://metafacture.org>