Configuring a Sync Data to Transactions Action

The **Sync Data to Transactions** Action is a powerful tool that can be used to quickly migrate a large chunk of data into Onit, or, with a bit more configuration, support a data integration between an external system and Onit.

The Action is responsible for creating, updating or upserting Records from a flat file. The Action can also be configured to delete Records when their corresponding data row is missing from the imported file.

In this tutorial, we'll cover the basics of setting up the Action for a simple migration of data over to Onit. Configuring this Action for a data integration takes a bit more work and is covered in our Setting Up a Data Integration tutorial.

Note: This Action is also covered in brief on our Onit Actions Reference page here.

Before We Start ...

This tutorial will assume you understand the following concepts:

- Building Your First App
- Creating an Action
- Creating a Business Rule

Flat File Format Requirements

Before we jump into configuring our Action, let's take a quick look at the formatting requirements for flat files.

- The End of Line (EOL) format must be Carriage Return Line Feed (CRLF). Line Feed (LF) format will not parse correctly. (This can be a problem with CSV files generated from Excel on a Mac.)
- If data includes the delimiter, place quotes around it. (E.g., "Jacobs, Mike", Male, 57)
- Double quotes must be used to include quotes in data. (E.g., Bob ""The Hammer"" Johnson, Male, 23)
- This Action only supports UTF-8 encoded files

Let's Get Started!

In this example, we'll walk through setting up the **Sync Data to Transactions** Action to make a one-time migration of data into an app.

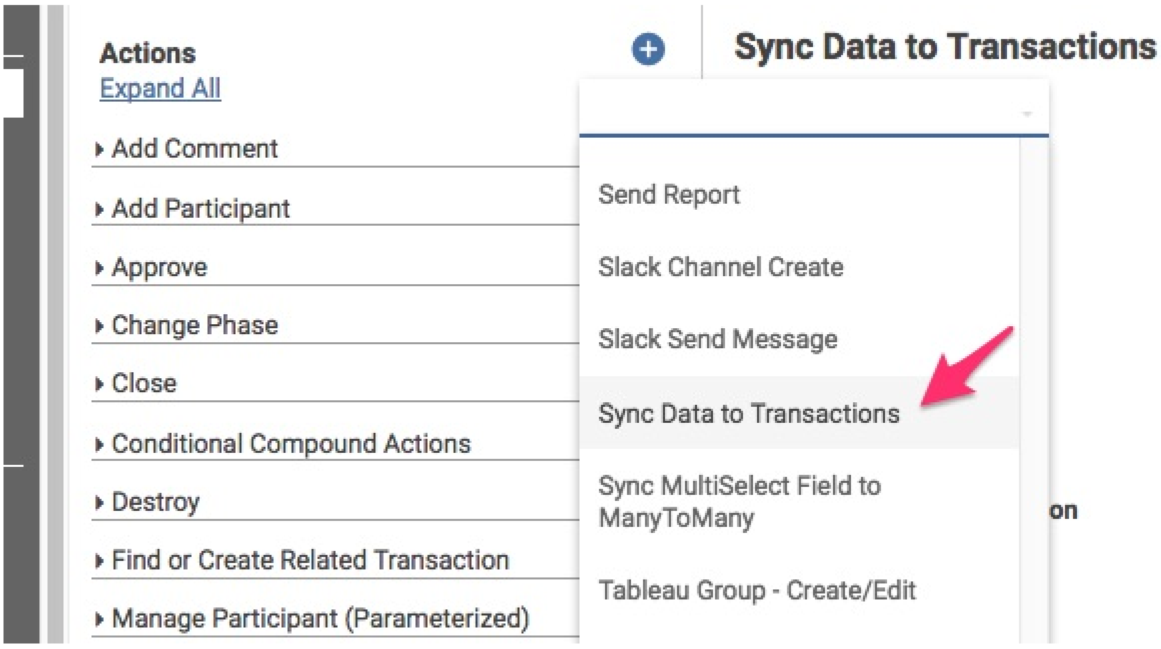
Add an Attachment Field for your flat file

Before we create our Action, let's add an Attachment Field to our app where we can upload our flat file. We'll name our Attachment Field **File** and plan to reference this in our **Sync Data to Transactions** Action later.

Configure the Sync Data to Transactions Action

Now we're ready to configure our **Sync Data to Transactions** Action in our **File Importer** App.

From the Advanced Designer page, select the **Actions** node from the sidebar on the left then choose **Sync Data to Transaction** Action.


Actions
Expand All
Add Comment
Add Participant
Approve
Change Phase
Close
Conditional Compound Actions
Destroy
Find or Create Related Transaction
Manage Participant (Parameterized)
Sync Data to Transactions
Send Report
Slack Channel Create
Slack Send Message
Sync Data to Transactions
Sync MultiSelect Field to ManyToMany
Tableau Group - Create/Edit

The **Sync Data to Transactions** Action configuration screen will appear. Below are some tips on how to navigate these options:

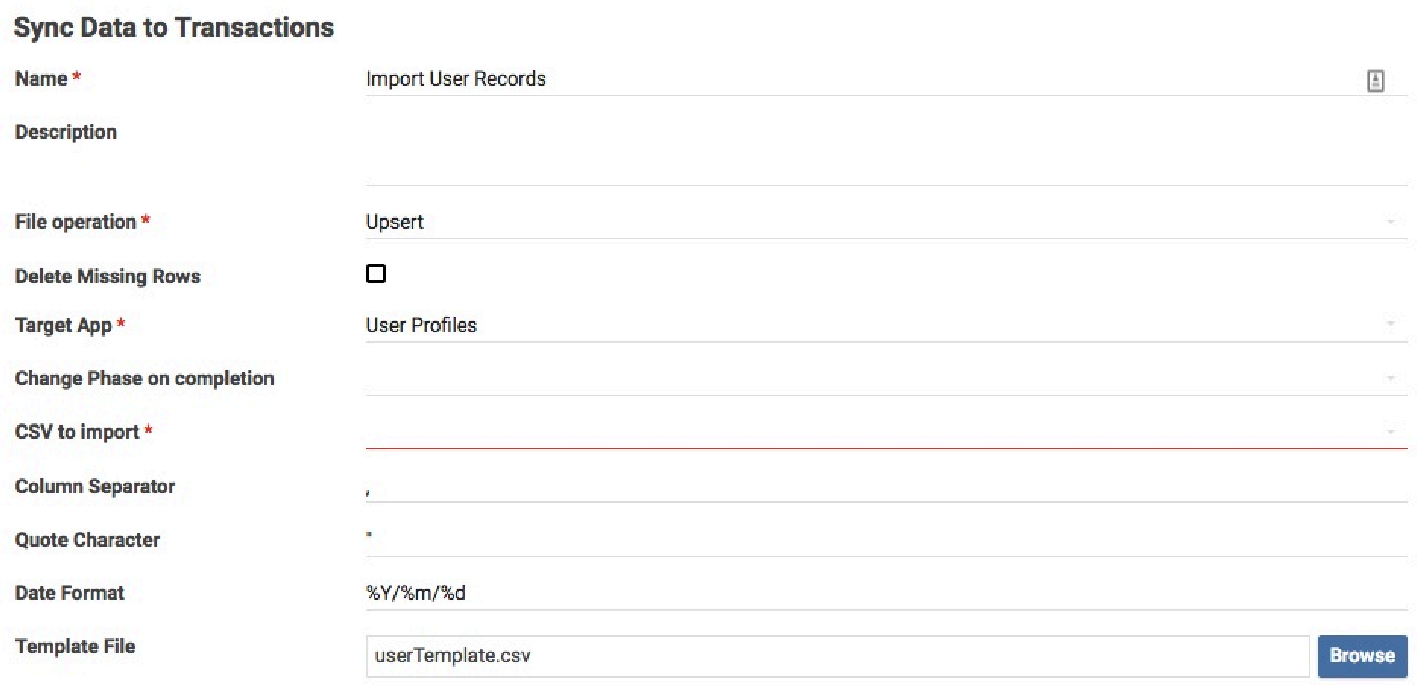
- **File Operation:** This property determines the Action's overall scope:
  - **Insert:** Only brand new records will be created. No updates are allowed. For each row in the flat file whose key matches the key of the Onit record, the Action will ignore the record.
  - **Update:** The Action will only update existing records. No creations are allowed. For each row in the flat file whose key does not match they key of an existing Onit record, the Action will ignore that row in the flat file.
  - **Upsert:** This will likely be the option that you want, as it is the most robust. The Action is allowed to both create new records and update existing records. The Action determines which operation to perform based on keys.
- **Delete Missing Rows:** If a row exists in the Onit App but does not exist in the flat file, the Record will be deleted in Onit. As a best practice, we don't recommend selecting this option, as deleting records may break other Apps/Records in your Onit environment. Instead, we recommend changing the Record's Phase to **Closed** or **Inactive** (or something similar) to indicate that it is no longer in use (more on phase changes below).
- **Target App:** The App containing the Records that you want to create/update/delete. (In our migration example, this is the same App that this Action is configured in.)
- **Change Phase on Completion:** Setting this property will automatically change the Phase of the Record when the ingestion process is completed. Note that using this property to change phases is the only way to trigger other Actions, (by tying them to a **Phase Change** Business Rule), either within your orchestrating App or on the Records your file is updating.

Advanced Tip: When this Action finds a row whose values completely matches an existing Record, it will not trigger an update on that Record. Therefore, if you need to run an Action only on Records that have either had their data updated or are newly created after your flat file is processed, you can optimize your secondary Action by pairing it with a Condition to check the time stamp of these Records' **last_activity** property.

- **CSV to import:** The name of the Attachment Field containing the flat file.
- **Column Separator:** The delimiter value used in the flat file (e.g., a comma (,), or a pipe (|).) The default is set to a comma.
- **Quote separator:** The escape character for quotes in the flat file. The default is a double quote (").
- **Date Format:** Liquid date format (e.g., %Y/%m/%d)
- **Template File:** In order to map each flat file column to a Field in your target App, you'll need to create and upload a sample flat file which includes a column header to be used as a template. Below is a sample flat file that contains an employee's ID, email address, name, and department:

```
EMPLOYEE_ID|EMAIL_ADDRESS|USERFIRSTNAME|USERLASTNAME|DEPARTMENT
123|minnie_mouse@onit.com|Minnie|Mouse|Sales
456|mickey_mouse@onit.com|Mickey|Mouse|HR
789|donald_duck@onit.com|Donald|Duck|IT
```

At this point, your Action should look something like the following:


Sync Data to Transactions

Name *	Import User Records
Description	
File operation *	Upsert
Delete Missing Rows	☐
Target App *	User Profiles
Change Phase on completion	
CSV to import *	
Column Separator	,
Quote Character	"
Date Format	%Y/%m/%d
Template File	userTemplate.csv

Creating Field Mappings

The last thing we need to do before this Action is complete is create Field Mappings for it.

After adding your Template File, it will be digested and you'll notice a series of Field Mappings will appear below the **Template File** property.

Field Mappings map each of the column names of the flat file we want to pull data from to the target Field that defines where the column's values should be saved. You'll need to fill out the following properties for each mapping you make:

- **Key Field:** Indicates which Field the Action should use as a key. As mentioned earlier, a key allows the **Sync Data to Transactions** Action to differentiate between existing records that it need to be updated and brand new records that it needs to create. We recommend setting exactly one key. Note that Key Fields cannot be calculated.
- **Calculated:** If the **Source** property (below) will be configured to simply extract data from the flat-file as is (without transforming it), do not select the **Calculated** checkbox. If, on the other hand, the **Source** property will contain a Liquid expression (to transform the data before it is imported into Onit), select the **Calculated** checkbox. For example, if data appears in lowercase in the flat-file, but you want to store this data in Onit as uppercase, you would select the **Calculated** checkbox and then insert a Liquid expression in the **Source** column that modified the data accordingly. Note any Field designated as a Key Field cannot be calculated.
- **Source:** This is where you do the actual mapping. There are two ways to map a flat file column to an App Field:
  - **Simple Mapping:** To simply extract the exact value from the column and ingest it into Onit, use the following syntax: `{{ row.column_name }}`. Where "column_name" is replaced with the exact name of the column header used in your flat file.
  - **Advanced Mapping:** The mappings that you define can transform the data from the flat file before it inserts it into an App Field using Liquid. This can be very helpful if you want to manipulate the flat file data before moving it into Onit.

    For example, let's say your flat file contains one column named **first_name** and another separate column named **last_name**. However, let's also say that in Onit, you just have one Field named **full_name**. In this situation, you could transform the data before ingesting it into Onit. To achieve this you'd use the following syntax: `{% capture full_name %}{{row.USER_FIRST_NAME}} {{row.USER_LAST_NAME}}{% endcapture %}{{full_name}}`. This Liquid concatenates the first name and the last name (with a blank character in between) into a single string that will be inserted into your target App.

Warning: Flat files with spaces and/or dashes cannot be directly included in the {{ row.COLUMN }} syntax when doing Liquid in the configuration.

For example, the flat file has a header named "COLUMN - NAME" and another named "COLUMN - EMAIL". In order to run a Liquid concatenation into a third field in the Target App, you would need to use the following: {{ row.COLUMN___NAME }} {{ row.COLUMN___EMAIL }} with three underscores (_ _ _).

- **Property:** This is the App Field that you want to ingest data into. Fields are listed alphabetically using the following pattern: **Field Label (field_name)**.

Note that your Action **must have**:

- A Field Mapping for the **name** Field.

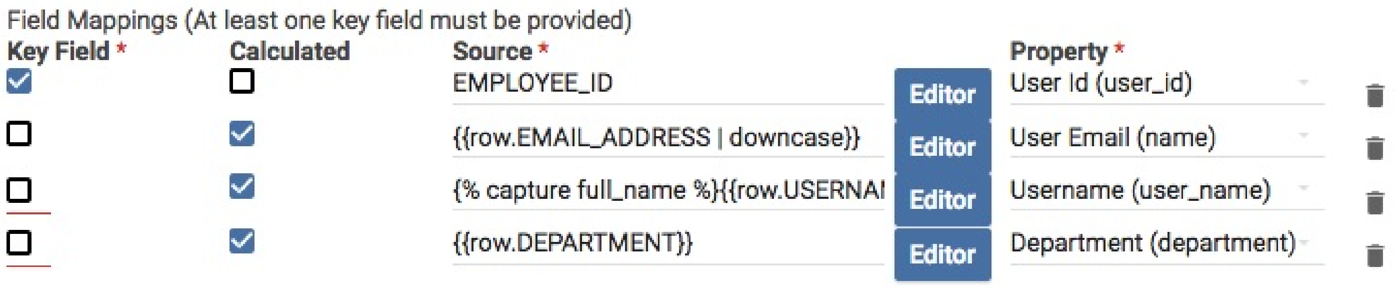
- At least one **Key Field** selected.

Field Mappings are not supported for the following Field Types:

- **MultiCurrency**
- **MultiSelect**
- **DateTime**

Below are Field Mappings using the columns contained in the template file we referenced above:

Note: Key Fields cannot be calculated.


Field Mappings (At least one key field must be provided)

Key Field *	Calculated	Source *		Property *
✓	☐	EMPLOYEE_ID	Editor	User Id (user_id)
☐	✓	{{row.EMAIL_ADDRESS | downcase}}	Editor	User Email (name)
☐	✓	{% capture full_name %}{{row.USERNAI	Editor	Username (user_name)
☐	✓	{{row.DEPARTMENT}}	Editor	Department (department)

After you've configured your Action, make sure to hit the **Save** button.

Note: When creating 100 or more Records using the **Sync Data to Transactions** Action each batch of 100 Records will have the ***same*** atom number.

Tip: UI Actions can also set the content of a MultiSelect Field. You can use a comma-separated list of values, or if your value has a comma in it we also support JSON: ["Acme, Corp.","Onit, Inc."].

To generate JSON for your MultiSelect Field use the following Liquid guide:

```
{% assign list  = "Coyote|Acme, Corp.|Roadrunner" %}
{% assign list_split = list | split: "|" %}
{% assign json = list_split | to_json %}
{{json}}
```

This Liquid splits the original list on the delimiter (in this case a pipe, '|'), then converts the split list to JSON using the **to_json** Liquid filter. Plug in your pipe delineated list in the **list** variable, and the Liquid will do the rest.

Assign the Action to a Business Rule or Button

Remember to assign your new Action to either a Business Rule or a Button, depending on your use case.

Tips

To access the status of a Sync Data Action with Liquid, use the following configuration:

```
{% for b in batch_process_statuses %}

{% for d in b.details %}

{{d}}

{% endfor %}

{% endfor %}
```

This Liquid will provide you with the status, total number of atoms processed, error count, insert count, update count, untouched count, deleted count, and missing count. If there are any errors use `.details` to print error descriptions.

```
{"sidekiq_batch_id"=>"rcu5nubQ-ya1ow", "status"=>"COMPLETE", "total"=>1011,
"processed"=>1011, "error_count"=>0, "insert_count"=>0, "update_count"=>0,
"untouched_count"=>1011, "deleted_count"=>0, "missing_count"=>0,
"id"=>"5c360681ed404f6f44000005"}
```

Checking for Errors

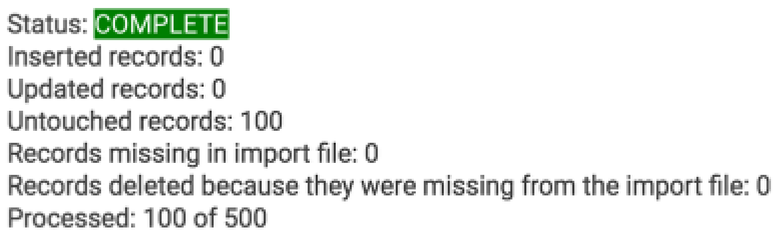
If the **Sync Data to Transactions** Action runs into an error when processing a file, it will log an error for the row containing the issue and continue to process.

You can view a summary of the batch processes completed with this Action by navigating to the new React UI admin page. Remove "/Apps" from your URL and replace it with "/react/admin" as so:

Original URL: https://acme.onit.com/apps

Modified URL: https://acme.onit.com/react/admin

Then select the **Batch Processes** option from the left-hand menu. Use the **Details** link of each row to navigate to that process' details.



```
Status: COMPLETE
Inserted records: 0
Updated records: 0
Untouched records: 100
Records missing in import file: 0
Records deleted because they were missing from the import file: 0
Processed: 100 of 500
```

Managing Manually Created Records

Any Record created manually (i.e., by launching a new Record form) *after* a **Sync Data to Transactions** Action has been run **will not** automatically update when **Sync Data to Transactions** is run again. This gotcha involves the way Onit creates Records with **Sync Data to Transactions** behind the scenes.

When configuring your **Sync Data to Transactions** Action you specified a Key Field. As previously discussed, the Action uses this key to decide whether to create or update Records. When a Record is created with this Action it contains a Field not visible in the App Builder called the **key_hash.** The value of the **key_hash** Field is the key value specified when you configured the Action (e.g., the Record ID). When a Record is created manually it does not contain the **key_hash** Field, therefore the **Sync Data to Transactions** Action is unaware of the Record.

Luckily, solving this gotcha is a simple fix. Create a new **Update Transaction** Action that targets the **key_hash** Field. Then, using the **Update Transaction** Action, populate the value of the **key_hash** Field with the same key specified in the **Sync Data to Transactions** Action. For example, a **Sync Data to Transactions** Action uses the Record's ID as its key. So, using the **Update Transaction** Action, the **key_hash** Field will populated with the Record's ID.

Gotchas

- Although the **Sync Data to Transactions** Action can be configured to update transactions based on file data, the update of these transactions *does not* trigger an **Update Transactions** Business Rule.

  However, setting this Action's **Phase Change on Completion** property will allow you to use the **Phase Change** Business Rule as a trigger for other Actions.