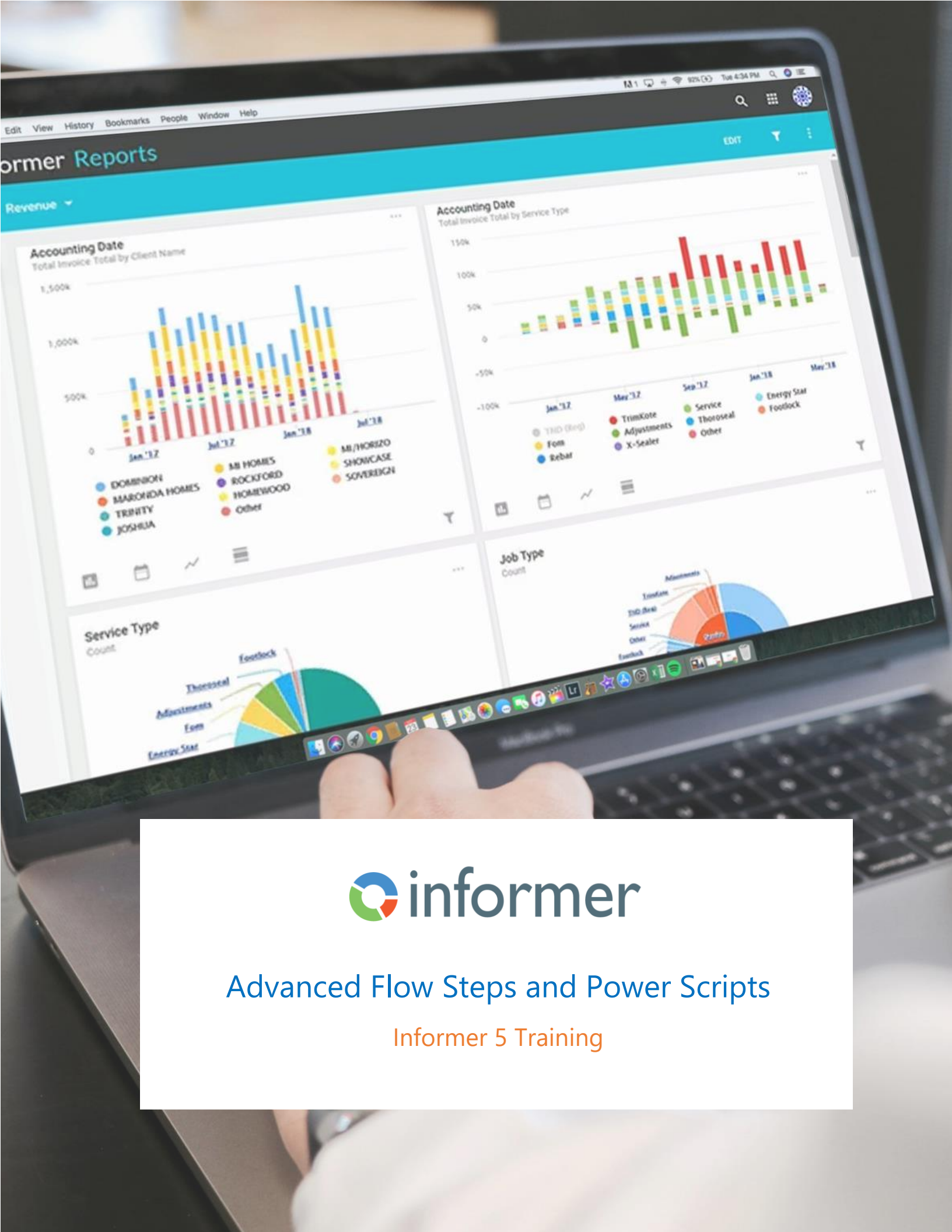

informer

Advanced Flow Steps and Power Scripts

Informer 5 Training

Table of Contents

Types of Flows

Informer has Flow Steps that are used to manipulate the data from Queries.

- **"Add Field" Flow Steps** add new Fields through JavaScript Calculated Fields or a few pre-built features that are easy to use.
- **"Transform" Flow Steps** change the format or values of the data with features that are also easy to use.
- **"Remove" Flow Steps** remove Fields that do not need to be displayed in the data but were needed for some earlier Flow Step.
- **"Advanced" Flow Steps** fall into any of the other categories, depending on how they are used.

"Flush" Flow Steps influence the timing of other Flow Steps, allowing earlier Flow Steps to finish before later Flow Steps begin. Flush Flow Steps are described in more detail later in this guide. Java Calculated Fields should be avoided. They accomplish the same thing as Calculated Fields (via JavaScript) but are much slower. They exist only to accommodate imported Informer 4 content. Power Script Flow Steps, the focus of this guide, use JavaScript to accomplish a variety of tasks including removing rows of data, adding new Fields, and overwriting existing data.

Power Scripts vs. Calculated Fields

Informer's Power Script Flow Steps (Power Scripts) use JavaScript like the Calculated Field Flow Step, so there is some overlap of functionality between the two. Through a few reserved variables and a slightly broader scope, however, Power Scripts are capable of more than Calculated Fields. This extra functionality also makes them more complex.

Displaying Values

The first difference between Calculated Fields and Power Scripts is how to display a value in the Data Grid. Each Calculated Field creates a single new Field which displays whatever value is "seen" by the JavaScript interpreter last. Power Scripts have a reserved variable, `$record`, that can create multiple new Fields as well as overwrite preexisting ones. For example:

```
$record.myNewField = "some new field value";
$record.oldFieldAlias = "Overwriting data";
```

The "Insert Field" button on the right inserts a correct reference to the chosen Field, including the '$record.' portion. The rest of Power Script's reserved variables are discussed in depth later in this guide.

JavaScript Objects

Almost all of Informer's reserved variables, including $record, are JavaScript Objects. Most of the calculations that can *only* be written in Power Script heavily rely on the use of Objects. An Object in JavaScript is a collection of key:value pairs, where the value can be any data type (including Strings, Arrays, or more Objects). These

Objects are written in a specific format called JavaScript Object Notation or JSON. For example:

```
var person = {
      "firstName" : "John",
      "lastName": "Doe",
      "age": 25,
      "job": {
            "company": "Awesome Company inc.",
            "title": "Hard Worker",
            "responsibilities": [
                  "Answer phone",
                  "Data Analysis"
            ]
      }
}
```

Refer to each of the values with dot notation:

```
person.firstName
person.job.title
```

Or refer to a value with bracket notation, which can accept a variable:

```
person['lastName']
var personalDetail = 'age';
person[personDetail]
```

Power Script Built-in Variables

Object	Description
$record	Refers to other fields on the report.
$local	Can create a variable that persists across rows.
$inputs	Refers to input values.
$index	Row counter that starts at 0. Not in any order.
$omit()	Function that removes a row.
_ (aka lodash)	Public utility library for manipulating collections.
moment	Public utility library for manipulating dates.
$fields, $query, $datasource	Refers to metadata. Rarely used.

$record

$record is a JavaScript object that represents the current row of data. Each of the properties on $record is a Field alias key paired with the data in that Field for this row. For example, $record.firstName might hold the data "John".

$local

$local is an object that is not reset between rows. It is discussed in detail later in this guide.

$inputs

$inputs is an object that holds the values entered into the Inputs at the Query's runtime. It is keyed by the alias of the Input. For example, consider an Input with the alias myInput:

```
$record.userInput = $inputs.myInput;
if($inputs.myInput == "some default"){
  $record.defaultInput = true;
} else $record.defaultInput = false;
```

When there is only one value in the Input, it is the data type of the Input (text, numeric, etc.). If the User enters multiple values in the Input, $inputs.myInput becomes an array.

$index

$index is an integer that increments with each row. Since the rows are streamed in no particular order, $index will not necessarily be in order. $index starts at 0 and is not typically used outside of arbitrarily numbering rows.

$omit()

$omit() is a function that removes the current row of data from the Query. It can be used to more permanently filter on Calculated Fields and should always be combined with an if-statement.

```
if($record.someCalculation == "some unwanted value"){
  $omit();
}
```

Lodash_

Lodash is an object from a public utility that provides tools for manipulating collections. It accomplishes many common tasks, and all its functions automatically check for null. For example, to see if there is already an entry called 'runningTotal' in $local:

```
if(_.hasIn($local, 'runningTotal'){
  // do something
```

}

See https://lodash.com/docs/ for more information.

Moment

Moment is an object from a public utility that provides additional tools for manipulating dates. It accepts a JavaScript date (as well as several other date formats) as a parameter and creates a moment version of the date, supplying access to all its functions. For example, to see the number of days between startDate and endDate ignoring the time of day:

var mStart = moment($record.startDate);

mStart = mStart.startOf('day');

var mEnd = moment($record.endDate);

mEnd = mEnd.startOf('day');

$record.diff = mStart.diff(mEnd, 'days');

Like many objects, moment functions can be chained together:

$record.diff = (moment($record.startDate).startOf('day')).diff((moment($record.endDate).startOf('days)),'days');

See https://momentjs.com/docs/ for more information.

$fields

$fields is a rarely used object that houses the metadata about each Field, including `dataType`, `position`, `label`, and `name`. Note that this information should not be changed here. Use the drop-down arrow next to the Field's header instead.

$query

$query is a rarely used object that has `params`, `dataTypes`, `fields`, `options`, `language`, `user`, and `result`. Note that this information cannot be changed.

$datasource

$datasource is a rarely used object that contains `defaultLinkType`, `family`, `languages`, `naturalId`, `id`, `ownerId`, `name`, `description`, `type`, `scannedAt`, `schemaUpdatedAt`, `schemas`, `source` ,`sourceId`, `settings`, `createdAt`, `updatedAt`, `fileId`, and `features`. Note that this information cannot be changed.

Using $local to aggregate

One of the most common use cases for Power Scripts is calculating values across all the rows of the Query. For instance, the Percent Of Total Flow Step can be recreated in a Power Script using the $local variable.

$local is an Object that keeps its values from one row to the next. No other variable does this, so any aggregate or value that requires details from multiple rows of data must be stored in $local. For example, to create a running total:

```
$local.runningTotal = $local.runningTotal || 0;
$local.runningTotal+= $record.someNumericField;
$record.runningTotal = $local.runningTotal;
```

Without using $local above, the running total would reset to zero on every new row. Note the assignment in the first line: because $local.runningTotal must be initialized but should not be initialized for each row, this line checks if $local.runningTotal has a value and assigns that value if it exists. If it does not exist, it assigns zero.

Flush Flow Steps

Consider expanding the earlier example to creating a Percent of Total in Power Script. It requires the complete total, not just a running total of all the values seen so far. To accomplish this, a Flush Flow Step is needed. The Flush Flow Step collects rows of data while waiting for all previous Flow Steps to finish. In other words, it finishes all previous Flow Steps before continuing to the later Flow Steps. This allows one Power Script to create an aggregate using $local and another Power Script to use the finished aggregate.

```
// First Power Script
$local.total = $local.total || 0;
$local.total+= $record.someNumericField;


// Flush
// Second Power Script
$record.percentOfTotal = $record.someNumericField / $local.total;
```