THE LOANPRO API

CUSTOM QUERY

loanprosoftware.simnang.com

API SAMPLE

Fork Custom Query — https://plnkr.co/edit/4YVXR9

Fork Getting Context Variables — https://plnkr.co/edit/U4Sgwv

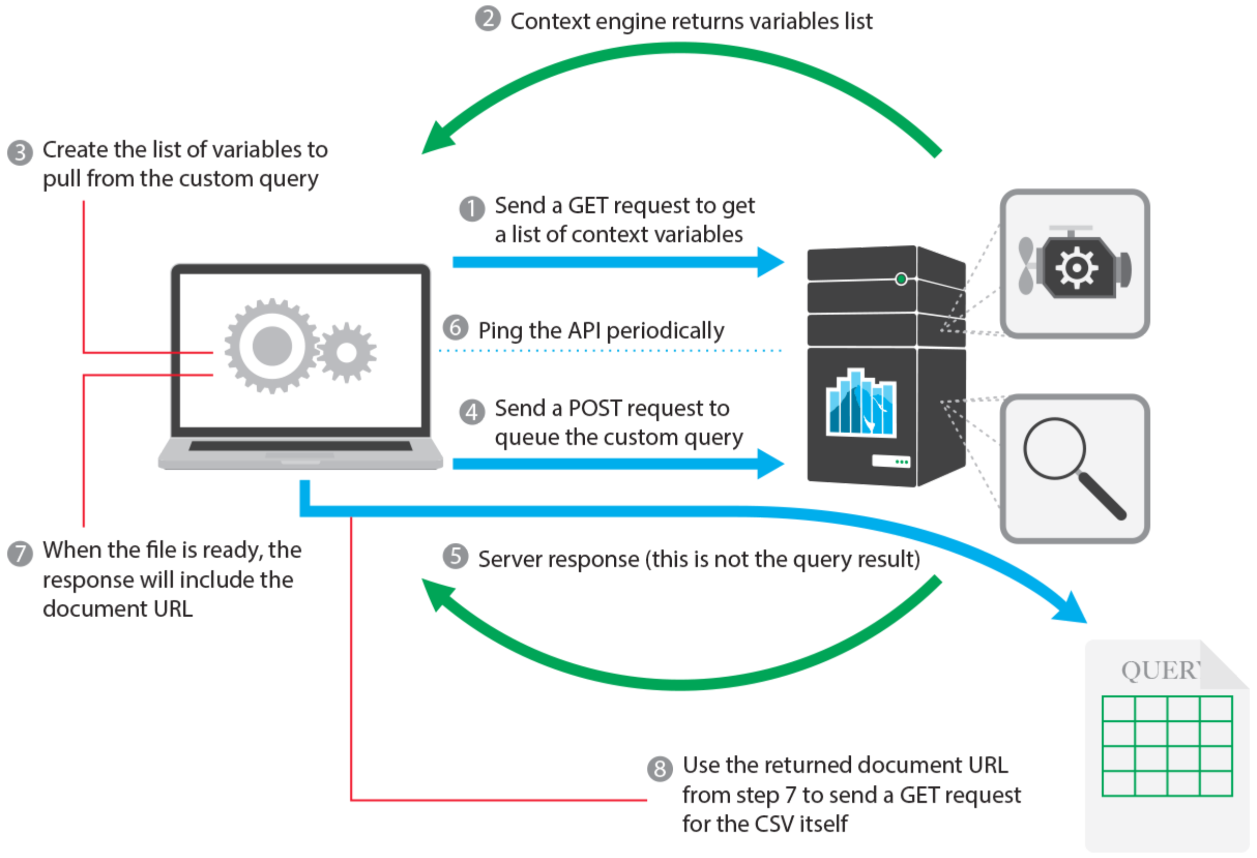
Fork Get Custom Query Status — https://plnkr.co/edit/BzgZLc

⚠ CAUTIONARY NOTE ⚠

The provided API samples store API credentials in the browser

Use the API samples to explore the API and not as your integration

THE PROCESS



2 Context engine returns variables list
3 Create the list of variables to pull from the custom query
1 Send a GET request to get a list of context variables
6 Ping the API periodically
4 Send a POST request to queue the custom query
7 When the file is ready, the response will include the document URL
5 Server response (this is not the query result)
8 Use the returned document URL from step 7 to send a GET request for the CSV itself
QUERY

GRABBING VARIABLES

Variables are the data fields to be exported

Variable lists differ between tenents

To get the list, send a GET request to the endpoint:

```
odata.svc/ContextVariables?nopaging
```

SAMPLE RESPONSE

Below is a sample response

```
{
 "d": {
 "results": [
  {
    "__metadata": {
     "uri": "http://loanpro.simnang.com/api/public/api/1/odata.svc/ContextVariables(id=69
     "type": "Entity.ContextVariable"
    },
    "id": 69,
    "parent": null,
    "name": "source-company-country",
    "friendlyName": "Country",
    "format": "context.format.selection",
    "section": "context.section.source",
    "computation": 0,
    "flags": 0,
    "stoplights": 0,
    "mailMerge": 1,
    "created": "/Date(1436818217)/"
   }
   ...
```

UNDERSTANDING THE RESPONSE

Each ContextVariable entity has the following key fields:

- name — the variable name of the field
- friendlyName — the human-readable name of the field
- format — the variable format
- computation — specifies if the variable value is calculated or static

CREATE A LIST OF VARIABLES TO USE

Create a JSON array containing JSON objects which represent the variables to use. Each JSON objects need to contain the following:

- name — The variable name of the context variable
- format — The format of the context variable
- columnName — The column header as you'd like it to appear in the custom query
    - Typically the **friendlyName** of the context variable

COMPUTED VARIABLES

To add a computed variable to the variable list for your query, you will need to give more data about the computed values

This is done by adding an **arcConf** object that has the following fields:

- set - how to obtain the value
  - **current** — recalculates the value
  - **archive** — grabs the value from the archive
  - **reverse** — reverse calculates as an archived value
- type - method of specifying the date for when a value will be calculated or pulled from an archive; "date" - specifies a date, "days" specifies a relative day offset
- val - the date value based on the type chosen
  - MM/DD/YYYY for dates
  - an int for day offsets

RECALCULATE EXAMPLE

A request for a computed field that is recalculated from a day ago would look like the following:

```
{
  "name": "status-next-payment-amount",
  "format": "context.format.text",
  "columnName": "Nxt Pmt Amt",
  "arcConf": {
    "set": "current",
    "type": "days",
    "val": 1
  }
}
```

ARCHIVE EXAMPLE

A request for a computed field that is pulled from the archive for the 25th of May, 2017 would look like the following:

```
{
  "name": "status-next-payment-amount",
  "format": "context.format.text",
  "columnName": "Nxt Pmt Amt",
  "arcConf": {
    "set": "current",
    "type": "date",
    "val": "05/25/2017"
  }
}
```

RECALCULATE EXAMPLE

A request for a computed field that is pulled form the reverse archive for five days ago would look like the following:

```
{
  "name": "status-next-payment-amount",
  "format": "context.format.text",
  "columnName": "Nxt Pmt Amt",
  "arcConf": {
    "set": "reverse",
    "type": "days",
    "val": 5
  }
}
```

CREATING THE REQUEST: THE SEARCH

Ideally, the custom query works on a subset of loans

- Allows it to finish in reasonable time
- Allows it to finish
- Only grabs data that you'll process

DEFINING THE SEARCH

- What values do you want in specific data fields?
- What values do you NOT want in specific data fields?
- What values are absolutely required?

GENERATE THE QUERY OBJECT

LoanPro uses the ElasticSearch query language

- Use **bool** to encapsulate the query
- Use **must** to encapsulate what's required
- Use **should** to encapsulate "at least one"
- Use **not** to encapsulate "I don't want this"

BOOL OBJECT

bool is used to represent that the child object will return*true* or *false*

```
"bool":{
  "must":[
    {
      ...
    }
  ]
}
```

Either *true* or *false* will be returned

MUST OBJECT

must states that all children objects must be true

- (think of it as an AND gate)

```
"must":[
    {
      "match":{
        ...
      }
    },
    {
      "nested":{
        "match":{
          ...
        }
      }
    }
  ]
]
```

Both the entity *and* the nested entity need to match

SHOULD OBJECT

should states that at least one child object must be true

- (think of it as an OR gate)

```
"should":[
    {
      "match":{
        ...
      }
    },
    {
      "nested":{
        "match":{
          ...
        }
      }
    }
  ]
]
```

Either the entity *or* the nested entity need to match

NOT OBJECT

not states that the child must be false in order to return true

- (think of it as an NOT gate)

```
"not":{
  "must":[
    {
      "match":{
        ...
      }
    }
  ]
}
```

The entity should *not* match

MORE RESOURCES

The query language is rather complex, allowing exact matches, query strings, regex, etc.

For a full list of options in LoanPro, see the article API Query objects

CREATING THE REQUEST: COMBINING EVERYTHING

First, create the following JSON:

```
{
   "search":{
      "query":{},
      "reportColumns":[],
      "savedSearchTitle": "Sample Query"
    }
}
```

The **query** object will hold our search query

The **reportColumns** array will hold our variable list

Set **savedSearchTitle** to be the human-readable name of the custom query

FILLING IN THE DATA

Replace the **query** object with the search query

Replace the **reportColumns** with the variable list

```
{
  "search": {
    "query": {
      "filtered": {
        "filter": {
          "bool": {
            "must": [
              {
                "range": {
                  "loanAge": {
                    "gte": 0,
                    "lte": 7
                  }
                }
              }
            ]
          }
        }
      }
    },
    "reportColumns": [      {
        "name": "status.next.payment.amount"
```

SENDING THE REQUEST

Send the created payload as a POST request to the endpoint:

```
CustomQueryReport/Autopal.SearchDataDump()/csv
```

INTERPRETING THE RESPONSE

Below is a sample response from the server

```
{
    "d": {
        "__metadata": {
            "uri": "https://loanpro.simnang.com/api/public/api/1/odata.svc/DataDumps(id=3
            "type": "Entity.DataDump"
        },
        "id": 3516,
        "entityType": "Reports.CustomQuery.Admin",
        "fileName": null,
        "url": null,
        "status": "dataDumpProcess.status.inProgress",
        "created": "/Date(1494255979)/",
        "createUser": "Simnang Demo",
        "info": "Sample Query"
    }
}
```

The field to note is the **id** as we will use that later, and **status**

The **status** tells us the query is in progress

CHECKING THE STATUS

We can't download the report until it is complete.

Queries can sometimes take hours

There is currently no notification sent when a query is done, so you will need to ping the server periodically to check the status

This is done by sending a GET request to the following endpoint:

```
odata.svc/DataDumps(<id>)
```

Replace **<id>** with the ID from the submission response

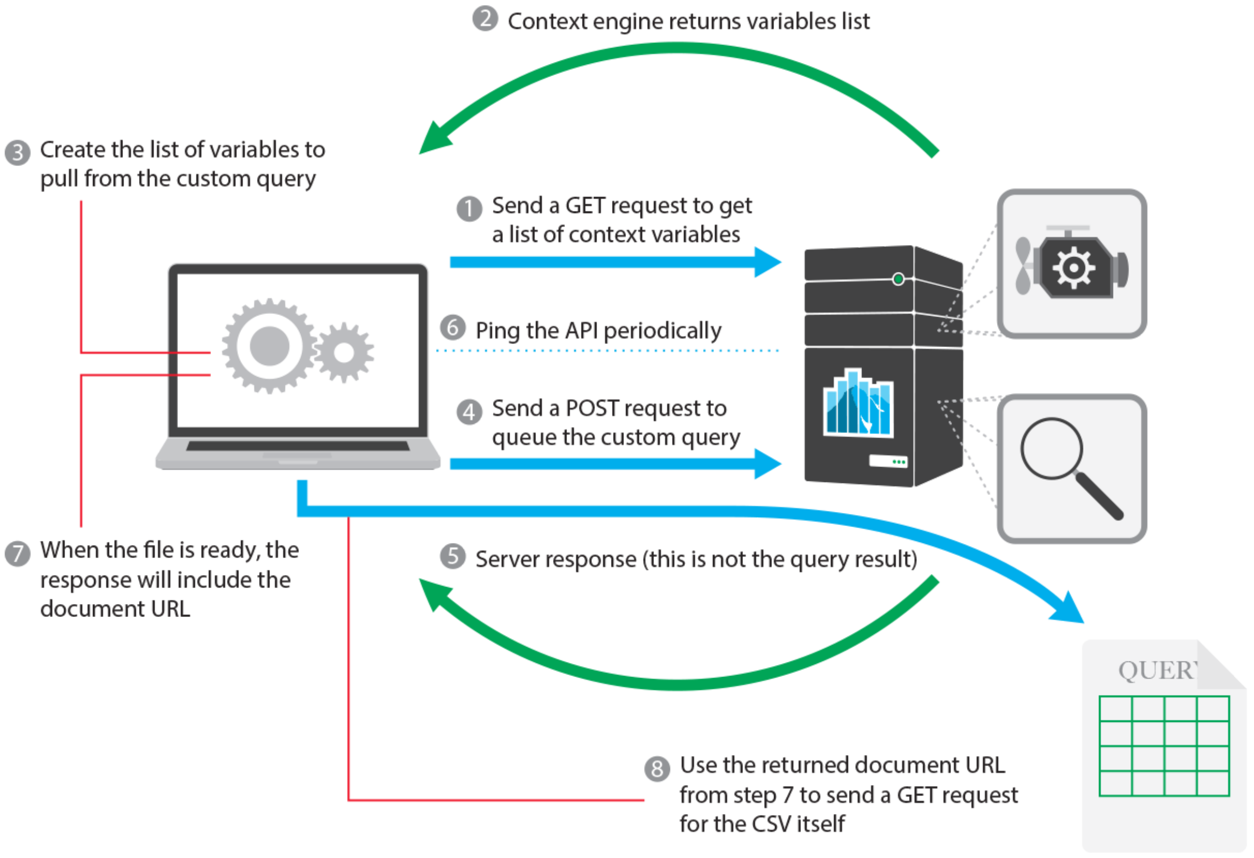
DOWNLOADING COMPLETE QUERIES

The status response value will be **dataDumpProcess.status.complete** when done

To download, send a GET request to the **url** field in the response

Contents of a CSV file will be returned

THE PROCESS


2 Context engine returns variables list
3 Create the list of variables to pull from the custom query
1 Send a GET request to get a list of context variables
6 Ping the API periodically
4 Send a POST request to queue the custom query
7 When the file is ready, the response will include the document URL
5 Server response (this is not the query result)
QUERY
8 Use the returned document URL from step 7 to send a GET request for the CSV itself

WHY SO LONG AND COMPLICATED?

All the steps are to make sure the query's data is good

- Variables can change between query requests
  - new custom fields, removal of old ones, etc.
- Queries are generated in the background and aren't ready immediately
- The generation process is long so mistakes are costly
  - Take precautions and perform tests

WHY DOESN'T MY QUERY GENERATE?

- The custom query was intended to only be used by the UI
    - Not very robust error reporting
- Malformed payloads will result in permanent a "In Progress" state
- If your queries don't generate, try breaking it down to find the issue
    - Send one variable at a time to see which ones stop generating
    - Send part of the search query to see which ones stop generating
- Sometimes, it can just take hours so be patient
- If needed, just use the UI

QUESTIONS?

Feel free to ask in our Forums

Or, email us at support@simnang.com