How to use the SDK

This document encompasses the installation and basic uses of the **iGrafx P360 Live Mining SDK**. It also contains SQL and Pandas examples.

The **iGrafx P360 Live Mining SDK** is an open source application that can be used to manage your processes. It is a python implementation of the iGrafx P360 Live Mining API.

With this SDK, you will be able to create and manipulate workgroups, projects, datasources and graphs (and graph instances). You will also be able to create and add a column mapping.

Please note that you must have an iGrafx account in order to be able to use the SDK. Please contact us to create an account.

Table of Contents

Installing

With pip:

To install the current release of the iGrafx P360 Live Mining SDK with pip, simply navigate to the console and type the following command:

```
pip install igrafx_mining_sdk
```

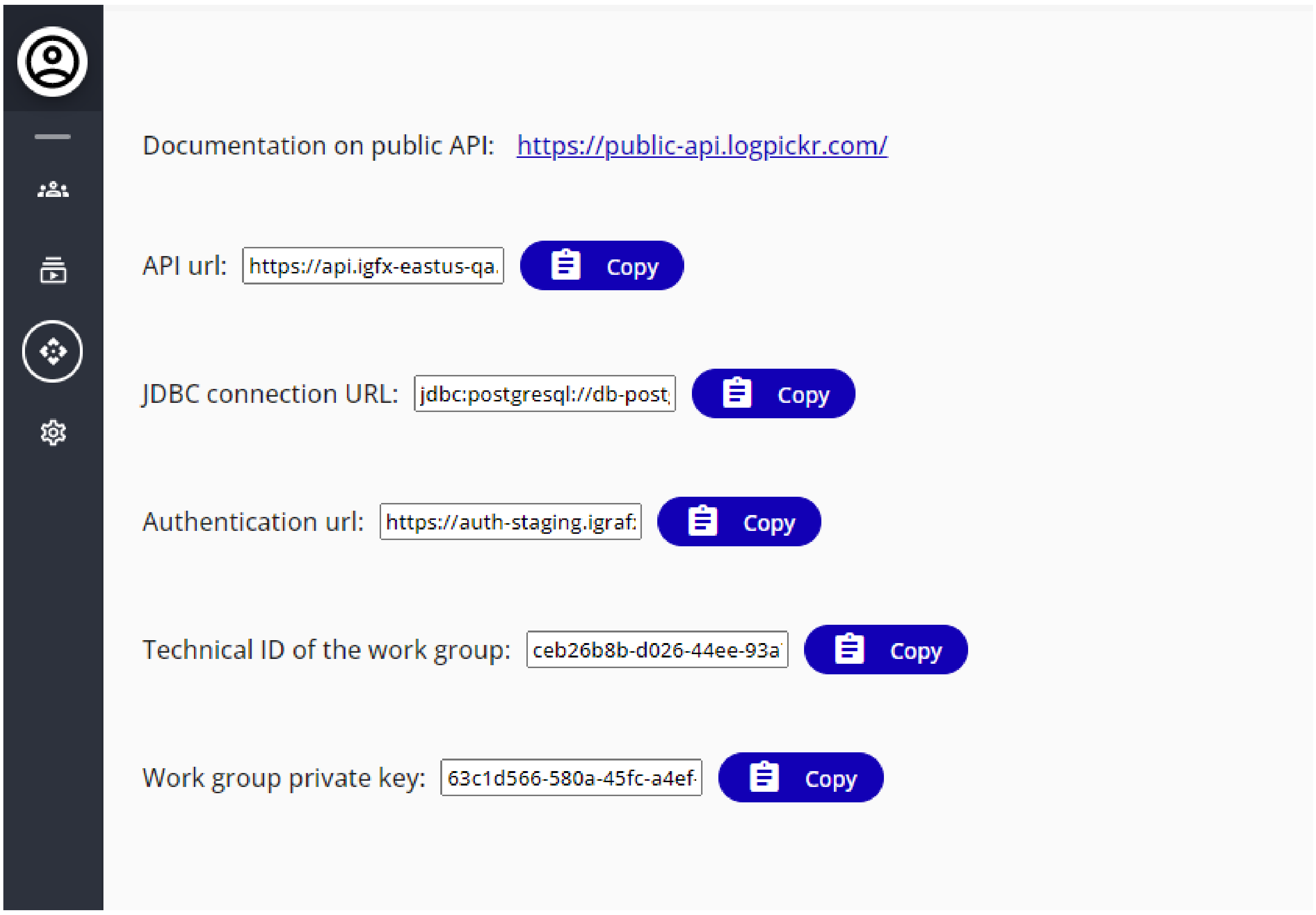
From Wheel:

Download the latest version of the wheel here. Then, navigate to your download folder and run:

```
pip install igrafx_mining_sdk.whl
```

Getting Started

First, open up **Process Explorer 360**, and go to your workgroup settings. In the settings page, go to the **Public API** tab. There, you should see your **workgroup's ID** and **secret key**. These are the values that will be used by the SDK to log in to the iGrafx P360 Live Mining API.



Documentation on public API: https://public-api.logpickr.com/

API url: https://api.igfx-eastus-qa. Copy

JDBC connection URL: jdbc:postgresql://db-post Copy

Authentication url: https://auth-staging.igraf: Copy

Technical ID of the work group: ceb26b8b-d026-44ee-93a Copy

Work group private key: 63c1d566-580a-45fc-a4ef- Copy

To begin:

Go ahead and **import** the package:

```
import igrafx_mining_sdk as igx   # the 'as igx' is entirely optional, but it will
make the rest of our code much more readable
```

Workgroups and Projects

The first step of using the iGrafx P360 Live Mining SDK will be to **create a workgroup**, using the credentials you copied from **Process Explorer 360**:

```
w_id = "<Your Workgroup ID>"
w_key = "<Your Workgroup KEY>"
api_url = "https://dev-api.logpickr.com"
auth_url = "https://dev-auth.logpickr.com"
w_realm = "<Your Realm>"
wg = igx.Workgroup(w_id, w_key, api_url, auth_url, w_realm)
```

Once the workgroup is created, you can access the list of projects associated with the workgroup through the `projects` property:

```
project_list = wg.projects
```

The list of project IDs associated with the workgroup can be accessed with:

```
project_id_list = [p.id for p in project_list]
```

A specific project ID can be accessed from the list by specifying its index:

```
project_id_list[0]
```

The project ID can also be found in the URL:

```
https://igfx-eastus-qa.logpickr.com/workgroups/<Workgroup ID>/projects/<Project
ID>/data
```

On top of that, if you already know the ID of the project you want to work with you can use:

```
my_project = wg.project_from_id("<Your Project ID>")
```

Once you have the project you want to use, several actions can be taken.

You can check if the project `exists`:

```
my_project.exists
```

Moreover, you can `reset` the project if needed:

```
my_project.reset
```

Sending Data

To be able to add data, you must create a file structure and add a column mapping. A column mapping is a list of columns describing a document(.CSV, .XLSX, .XLS).

To add a column mapping, you must first define the file structure:

```
filestructure = FileStructure(
    file_type=FileType.xlsx,
    charset="UTF-8",
    delimiter=",",
    quote_char='"',
    escape_char='\\',
    eol_char="\\r\\n",
    comment_char="#",
    header=True,
    sheet_name="Sheet1"
)
```

It is important to note that aside from the `file_type`(which can be set to CSV, XLSX or XLS) and the `sheet_name`(which is optional), the attributes above are set by default. That means that unless the value of your attribute is different, you do not need to set it.

Now, a column mapping can be created:

```
column_list = [
            Column('<Column name>', <Column index>, <Column Type>),
            Column('<Column name>', <Column index>, <Column Type>),
            Column('<Column name>', <Column index>, <Column Type>,
time_format='<Your time format>')
        ]
        column_mapping = ColumnMapping(column_list)
```

- `Column name` is the name of the column.
- `Column index` is the index of the column of your file. Note that the **column index** starts at **0**.
- `Colulmn Type` is the type of the column. It can be `CASE_ID`, `TASK_NAME`, `TIME`, `METRIC` (a numeric value) or `DIMENSION`(can be a string).

It is also possible to check whether a column mapping exists or not:

```
my_project.column_mapping_exists
```

Afterwards, the column mapping can be added:

```
my_project.add_column_mapping(filestructure, column_mapping)
```

Finally, you can add CSV, XLSX or XLS files:

```
wg = Workgroup(w_id, w_key, api_url, auth_url, w_realm)
p = Project("<Your Project ID>", wg.api_connector)

filestructure = FileStructure(
    file_type=FileType.xlsx,
    sheet_name="Sheet1"
)
column_list = [
    Column('Case ID', 0, ColumnType.CASE_ID),
    Column('Start Timestamp', 1, ColumnType.TIME, time_format='%Y/%m/%d
%H:%M:%S.%f'),
    Column('Complete Timestamp', 2, ColumnType.TIME, time_format='%Y/%m/%d
%H:%M:%S.%f'),
    Column('Activity', 3, ColumnType.TASK_NAME),
    Column('Ressource', 4, ColumnType.DIMENSION),
]
column_mapping = ColumnMapping(column_list)
p.add_column_mapping(filestructure, column_mapping)
p.add_file("ExcelExample.xlsx")
```

Graphs

Additionally, you can access the project components: the model Graph, the Graph Instances, and the datasources.

The model graph is accessed through the `.graph()` function:

```
my_project = wg.project_from_id("<Your Project ID>")
g = my_project.graph()
```

The Graph class inherits from the NetworkX library. This means that it is possible to create a display method according to your liking in the Graph class:

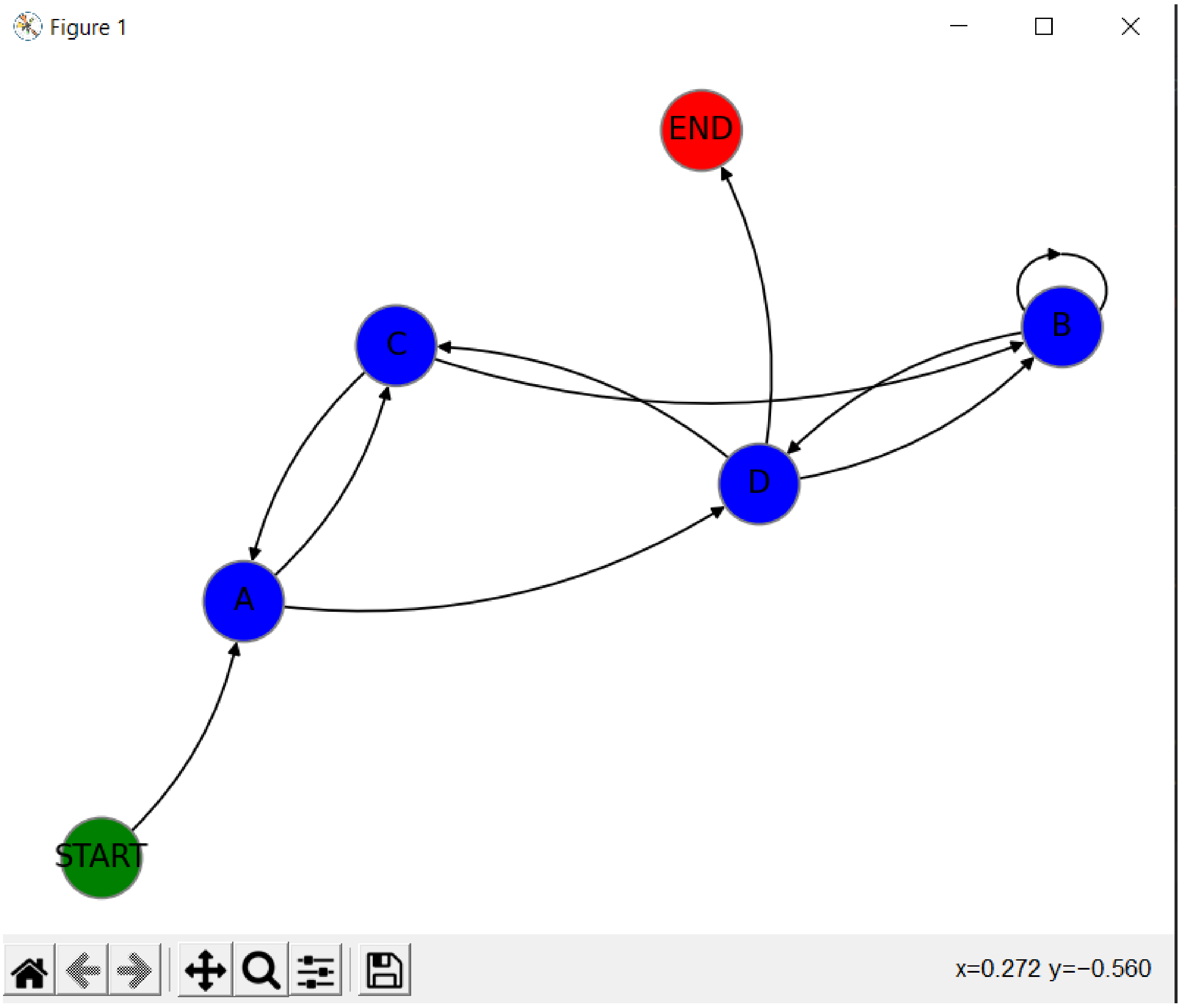
```
import matplotlib.pyplot as plt

"""Displays the graph using NetworkX library"""
# Using the kamada kawai layout algorithm
pos = nx.kamada_kawai_layout(g)
# Set the START node to green and the END node to red
color_mapper = {"START": "green", "END": "red"}
labels = {n: g.nodes[n]["name"] for n in g.nodes} # Use name as label
```

```
colors = [color_mapper.get(g.nodes[n]["name"], 'blue') for n in g.nodes] # Set
nodes colors (default to blue)

# Draw and show the graph
nx.draw(g, pos=pos, labels=labels, node_color=colors, node_size=1000,
connectionstyle='arc3,rad=0.2')
plt.show()
```

It will give us a graph similar to the subsequent one:



The graph can then be saved as GEXF:

```
nx.write_gexf(g, 'graph_name.gexf')
```

It can also be saved as GML:

```
nx.write_gml(g, 'graph_name.gml')
```

Graph Instances

Moreover, the graph instances can be accessed as a list with:

```
my_project = wg.project_from_id("<Your Project ID>")
graph_instance_list = my_project.get_graph_instances()
```

The process keys can also be accessed as a list with:

```
my_project = wg.project_from_id("<Your Project ID>")
process_key_list = my_project.process_keys
```

A graph instance can directly be requested by using one of the project's process keys:

```
pk = process_key_list[0]
gi = my_project.graph_instance_from_key(pk)
```

Datasources

Each project is linked to **datasources**. Those datasources have 3 types: `vertex`, `simplifiedEdge` and `cases`. To access those we can do:

```
db1 = my_project.nodes_datasource # For vertex
db2 = my_project.edges_datasource # For simplifiedEdge
db3 = my_project.cases_datasource # For cases
```

Those datasources are Python objects that can be used to facilitate access to corresponding data. Once created, they are empty and need to be requested so that data can be fetched.

The easiest way to do this is to use the `.load_dataframe()` method, which is equivalent to a `SELECT * FROM [datasource]` request. Optionally, the `load_limit` parameter can be used to fetch a subset of the dataframe. This method returns a **Pandas Dataframe** object.

```
df = db1.load_dataframe(load_limit=10) # load 10 rows of nodes_datasource
```

You can also return a list of all datasources associated with the workgroup. Note that in that case, all types of datasources are returned in the same list:

```
datasources_list = wg.datasources
```

Using Pandas methods

The **Pandas** methods is the simplest option when it comes to handling your dataset (compared to SQL queries) but may be less performant. Pandas can also be used to easily plot graphs.

If you want to see the structure of the datasource, you can use the `.columns` method:

```
df = my_project.edges_datasource.load_dataframe(load_limit=1000)
print(df.columns)
```

Operations can be done on dataframe. For example, we can calculate the mean value of the `duration` column:

```
duration_mean = df['duration'].to_numpy().mean()
```

We can also get the maximum or minimum of the `enddate` column:

```
enddate_max = df['enddate'].max()
enddate_min = df['enddate'].min()
```

Moreover, in the next example, we group the dataframe by `case ID`:

```
by_caseid = df.groupby('caseid')
```

Furthermore, the Pandas `.describe()` method can be applied to our dataframe:

```
stats_summary = df.describe()
```

This method returns a statistics summary of the dataframe provided. It does the following operations for each column:

- count the number of not-empty values
- calculate the mean (average) value
- calculate the standard deviation
- get the minimum value
- calculate the 25% percentile
- calculate the 50% percentile
- calculate the 75% percentile

- get the maximum value

It then stores the result of all previous operations in a new dataframe (here, `stats_summary`).

If need be, you can directly use the datasource's `connection` and `cursor` methods, which can be used as specified in the Python Database API:

```
ds = my_project.edges_datasource
ds.connection
ds.cursor
```

Using SQL Queries

In the last section, it was shown how to load the entire dataframe and do operations with **Pandas**. With **SQL** you can load the dataframe and do the operations in the same query, hence why it is faster. Those requests are often more powerful and performant than the **Pandas** methods presented above. However, the Pandas methods are simpler to use and understand.

In those requests, the name of the table is accessible through the `name` attribute and must be given in between double quotes. The use of f-strings is highly recommended.

A description of the tables is available here. It contains a glossary and descriptions of the different elements in a table.

Therefore, the following request returns all the columns of the given datasource for the specified process key.

```
ds = my_project.edges_datasource
edges_filtered = ds.request(f'SELECT * FROM \"{ds.name}\" WHERE "processkey" =
\'<Your Process Key>\'')
```

We can also calculate the mean value of the `duration` column. For example:

```
edge_duration_mean = ds.request(f'SELECT AVG(duration) FROM \"{ds.name}\"')
```

In a similar manner, the subsequent requests return respectively the minimum and the maximum value of the `enddate` column:

```
edge_enddate_min = ds.request(f'SELECT MIN(enddate) FROM \"{ds.name}\"')
edge_enddate_max = ds.request(f'SELECT MAX(enddate) FROM \"{ds.name}\"')
```

Finally, the following SQL statement lists the number of `cases` by `detail destination`, sorted high to low:

```
count = ds.request(f'SELECT COUNT(caseid), detail_7_lpkr_destination FROM \"
{ds.name}\" GROUP BY detail_7_lpkr_destination ORDER BY COUNT(caseid) DESC')
```

Using the public API

Workgroups and projects can also be accessed with the **public API**. The documentation for the *iGrafx P360 Live Mining API* can be found here.

To do so, we use `curl`.

For instance, we can use the `GET` HTTP method to access the list of available projects:

```
curl -X GET "http://localhost:8080/pub/projects" -H "accept: application/json"
```

With the same method, we can **access** a project graph model:

```
curl -X GET "http://localhost:8080/pub/project/<Your Project ID>/graph" -H
"accept: application/json"
```

In a similar manner, the projects' **datasources** are returned by:

```
curl -X GET "http://localhost:8080/pub/datasources?id=<Your Project ID>" -H
"accept: application/json"
```

Furthermore, with the HTTP method `POST`, several actions can be done.

The project's **column mapping** cant be posted:

```
curl -X POST "http://localhost:8080/pub/project/<Your Project ID>/column-mapping"
-H "accept: */*" -H "Content-Type: application/json" -d "{\"fileStructure\":
{\"charset\":\"UTF-
8\",\"delimiter\":\";\",\"quoteChar\":\"\\\"\",\"escapeChar\":\"\\\\\",\"eolChar\"
:\"\\\\",\"header\":true,\"commentChar\":\"#\",\"fileType\":\"csv\",\"sheetName\":
\"string\"},\"columnMapping\":{\"caseIdMapping\":
{\"columnIndex\":0},\"activityMapping\":{\"columnIndex\":0},\"timeMappings\":
[{\"columnIndex\":0,\"format\":\"string\"}],\"dimensionsMappings\":
[{\"name\":\"Activity\",\"columnIndex\":0,\"isCaseScope\":true,\"aggregation\":\"F
IRST\"}],\"metricsMappings\":
[{\"name\":\"Activity\",\"columnIndex\":0,\"unit\":\"string\",\"isCaseScope\":true
,\"aggregation\":\"FIRST\"}]}}"
```

Additionally, to **reset** all project data except it's name, description and user rights, we can use the subsequent `curl` command:

```
curl -X POST "http://localhost:8080/pub/project/<Your Project ID>/reset" -H
"accept: */*"
```

Finally, `DELETE` methods can be used.

For instance, we can use that method to **stop** the train task for a project:

```
curl -X DELETE "http://localhost:8080/pub/train/<Your Project ID>" -H "accept:
*/*"
```

Predictions

Once you have your project, you can train it and run predictions on it. For the training, the project has the `.launch_train()` and `.stop_train()` methods, as well as the `.train_status property`, which can be used like so:

```
my_project = wg.project_from_id("<Your Project ID>")

my_project.train_status
False

my_project.lauch_train()
my_project.train_status
True

my_project.stop_train()
my_project.train_status
False
```

Once the train is complete, you can run predictions on the case IDs you want in your project:

```
my_project = wg.project_from_id("<Your Project ID>")
my_project.train_status # we can make sure the training is finished
False

case_list = ["<Case ID 1>", "<Case ID 2>", "<Case ID 3>"]
prediction_data = my_project.prediction(case_list)
```

Access Druid database via JDBC

The Druid database can be accessed via JDBC. To do so, it is recommended to use Avatica JDBC driver. We then use the connect string `jdbc:avatica:remote:url=http://BROKER:8082/druid/v2/sql/avatica/`.

Please note that **JDBC uses Java**. Thus, we can use the subsequent code to connect to Druid:

```
// Connect to /druid/v2/sql/avatica/ on your Broker.
String url =
"jdbc:avatica:remote:url=http://localhost:8082/druid/v2/sql/avatica/";

// Set any connection context parameters you need here
// Or leave empty for default behavior.
Properties connectionProperties = new Properties();

try (Connection connection = DriverManager.getConnection(url,
connectionProperties)) {
  try (
      final Statement statement = connection.createStatement();
      final ResultSet resultSet = statement.executeQuery(query)
  ) {
    while (resultSet.next()) {
      // process result set
    }
  }
}
```

Access database via Druid Rest SQL queries

Sending a query:

The database can also be accessed with Druid Rest SQL queries.

To do so, first of all, using a `POST` method, send your query to the Router. `Curl` can be used to send SQL queries from the command-line:

```
curl  -X POST -H "Content-Type: application/json" -u druid_system:igfx-eastus-
qa__<Your Process ID> http://api.igfx-eastus-demo.logpickr.com:8008/druid/v2/sql/
--data-raw '{"query": "SELECT * FROM <ds.name> "}'
```

The query must be specified after `"query"`, in `--data-raw`.

Alternatively, SQL queries can also be sent as follows:

```
 cat query.json
{"query": "SELECT * FROM <ds.name> "}
```

```
curl  -X POST -H "Content-Type: application/json" -u druid_system:igfx-eastus-qa__<Your Process ID>  http://api.igfx-eastus-demo.logpickr.com:8008/druid/v2/sql/ --data-raw @query.json
```

Responses:

The result format of the query can be specified with `"resultFormat"`:

```
{
  "query": "SELECT * FROM <ds.name>",
  "resultFormat": "array"
}
```

More information can be found in the section Further documentation.

Further documentation

In this section, documentation can be found for further reading.

Support is available at the following address: support@logpickr.com

- Workgroups, Projects and Mapping
- iGrafx Help
- Druid SQL API
- iGrafx P360 Live Mining API