— ConMas Gateway —

マニュアル

2024.04

Rev.2.3.1

改定履歴

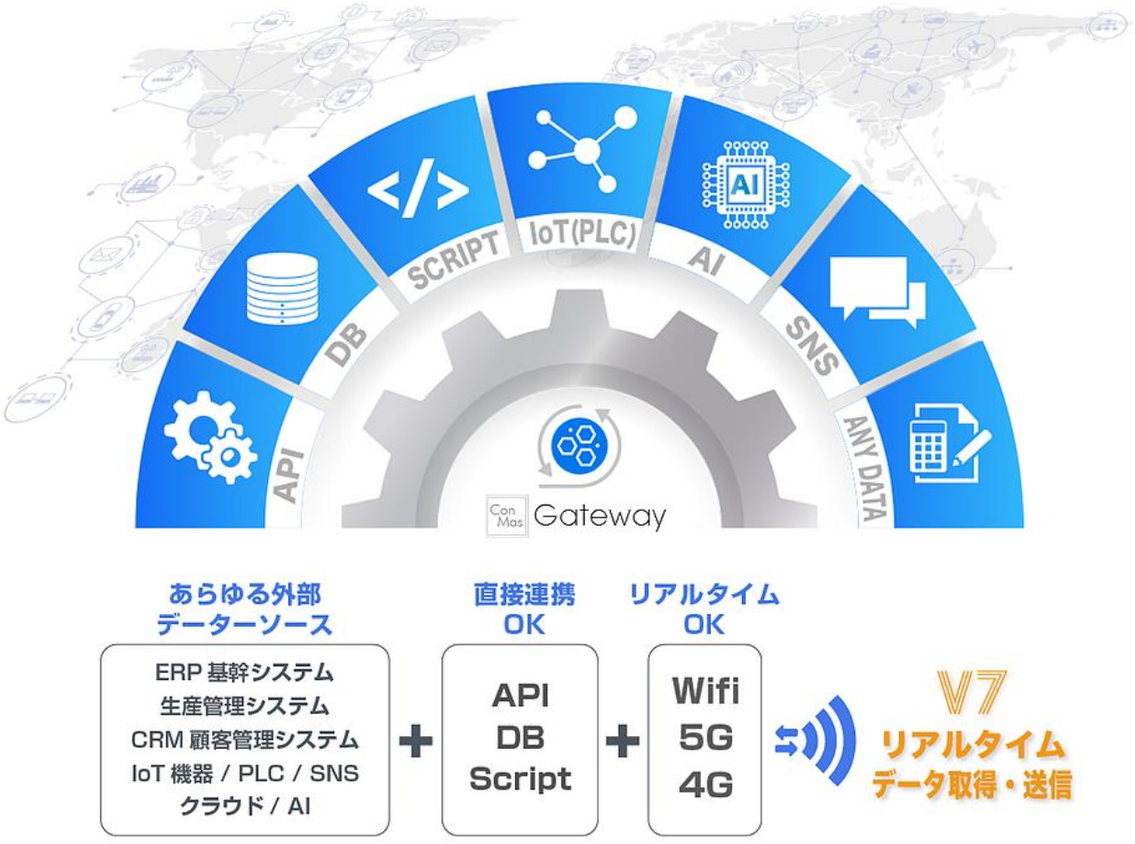
日付	Version	記載ページ	改定内容
2020.02.20	Rev 1.0.0	--	初版を発行しました。
2020.04.20	Rev 1.1.0	全ページ	1.1.200300 バージョンアップに伴い全改訂しました。
2020.06.16	Rev 1.2.0	P6	「ConMas Gateway で出来ること」に項目を追加しました。
		P10	Python スクリプト連携について Anaconda3 ではなく Python3 での動作検証としました。
		P15	SQL Server 接続においてインスタンス名が無い場合の記述を追加しました。
		P21	Oracle 用テーブル作成スクリプトの記述を追加しました。
		P24	3. タイマー起動を利用した自動取得設定を追加しました。
		P26	4. タイマー起動のアプリでの動作確認を追加しました。
		P32	Oracle 連携を追加しました。
		P57	Python スクリプト連携（Excel）を追加しました。
		P61	Python スクリプト連携（Kintone）を追加しました。
2020.06.24	Rev 1.2.1	P129	「使用上の注意事項」に Oracle 接続に関する記述を追加しました。
2020.09.28	Rev 1.3.0	P70	Python スクリプト連携（PostgreSQL からのデータ取得・計算・書き込み）を追加しました。
		P75	Python スクリプト連携（MS SQL Server からのデータ取得・計算・書き込み）を追加しました。
		P79	Python スクリプト連携（Excel からのデータ取得・計算・書き込み）を追加しました。
		P130	使用上の注意事項を追加しました。
		P29,P31	アクションファイル設定内の temp プロパティの記述を削除
		P98	Python スクリプト連携（Excel 関数を Python スクリプトで作成）を追加しました。
		P64	Python スクリプト連携（POST リクエスト用サンプル）を追加しました。
		P14	1. フォルダ構成についての内容を修正しました。
		P21	1. サンプルの準備の内容を修正しました。
2020.10.01	Rev 1.3.1	P65	アクションクラスター設定の URL を修正しました。
2021.01.27	Rev 1.3.2	P31	日付時刻列を条件列に含める場合の記述を追加しました。
2021.02.17	Rev 1.4.1	P31	SQL Server 接続方法において port 番号指定の記述を追加しました。

		P33	Oracle テーブルの BLOB 型の列に格納した画像を表示するための記述を追加しました。
2021.03.30	Rev 1.5.0	P10,P130	Python サンプルスクリプトの動作確認環境を明記しました。
		P15	pythonShell オプションについての記述を追加しました。
		P15	アップロード可能なファイルタイプの制限についての記述を追加しました。
		P16	データソースに MySQL 連携を追加しました。
		P16	データソースに IoT Data Share アイテム値の一括設定を追加しました。
		P33	Oracle で条件列が CHAR 型の場合の指定を追記しました。
		P34	MySQL 連携を追加しました。
		P130	MySQL 動作確認バージョンの記述を追加しました。
2021.04.12	Rev 1.5.1	P130	「使用上の注意事項」に 関する記述を修正しました。
2021.05.13	Rev 1.6.0	P15,P130	HTTPS 接続に関する説明を追加しました。
		P29,P31	PostgreSQL, Microsoft SQL Server のアクションファイル内 params 検索条件の型説明を修正しました。
		P29,P31	PostgerSQL, Microsoft SQL Server で画像を使用する場合の mappings プロパティの書式について修正しました。
		P118	「第 7 章 バインド変数の利用法」を追加しました。
		P122	「第 8 章 単一選択クラスターの選択肢取得」を追加しました。
2021.05.27	Rev 1.6.1	P65	POST 設定で利用可能なクラスター種別を変更しました。
2021.07.20	Rev 1.7.0	P10 P84 P107 P119 P130	使用する Node.js のバージョンについて更新しました。 Python スクリプト連携（Oracler からのデータ取得・計算・書き込み）を追加しました。 Python スクリプト連携（クラスターをフォーカスする機能）を追加しました。 マルチバイト識別子使用時の注意を追加しました。 ConMas Gateway のバージョン記載場所を追加しました。
2021.08.24	Rev 1.7.1	P39,42,46 P50,53,56, P66	画像クラスターを利用したサンプルを "type" : "string"　から　"type" : "image"に変更しました。
2021.11.1	Rev 1.7.2	P89 P110	Python スクリプト連携（CSV ファイルからのデータ取得・計算・書き込み）を追加しました。 Python スクリプト連携（複数行の検索結果を複数ページの帳票に展開するサンプル）を追加しました。
2021.12.23	Rev 1.7.3	P15 P16	default.json に関する説明を追加しました。
2022.02.09	Rev 1.7.4	P130	使用上の注意事項を追加しました。
2022.04.20	Rev 1.8.0	P130 P17	使用上の注意事項を追加しました。 SSL 通信に関するコメントを追加しました。
2022.06.22	Rev 1.8.1	P10 P21 P130	対応する Windows, Node.js, npm のバージョンを変更しました。 サンプルのサブフォルダを追加しました。 使用上の注意事項を追加しました。
2022.07.27	Rev 1.8.2	P10	Python 動作検証バージョンを更新しました。

		P67 P119 P120 P124 P130 P130	アプリからの POST データ例を追加しました。 第 7 章の構成を変更しました。 チェッククラスターの利用法を追加しました。 単一選択クラスター使用時の SELECT 列名について注記を追加しました。 JSON 書式について Python での真偽値扱いを注記しました。 対応クラスターにチェッククラスターを追加しました。 Python 動作検証バージョンを更新しました。 システム上 BOM 無しの UTF-8 エンコーディングを使用することを明記しました。
2022.11.16	Rev 2.0.0	P15 P21 P93 P125 P130	2.　設定ファイルについて　を修正しました。 第 5 章　サンプルの利用方法を追加しました。 19. Python スクリプト連携（MySQL からのデータ取得・計算・書き込み）を追加しました。 第 9 章　マスター選択クラスターの連携先設定を追加しました。 第 10 章　使用上の注意事項を修正しました。
2023.02.21	Rev 2.1.0	P14 P17 P115 P130	1. フォルダ構成についての内容を修正しました。 データソースの分離設定（config¥datasources フォルダ）について説明を追加しました。 Python スクリプト連携（外部連携 API（自動帳票作成）を呼び出す）を追加しました。 MySQL の動作確認環境を修正しました。
2023.05.10	Rev 2.1.1	P21 P131	アクションファイル内でシート番号を指定しない場合の仕様について説明を追加しました。
2023.05.26	Rev 2.1.2	P130	使用上の注意事項を追加しました。
2023.06.21	Rev 2.1.3	P21 P130	Python スクリプト連携（マスター選択クラスターの連携先設定）を追加しました。 使用上の注意事項を修正しました。
2023.10.25	Rev 2.2.0	P10 P13 P17 P67 P130	システム動作環境を修正しました。 Fig.3-4-1　ログの確認を修正しました。 【default.json】の内容の注釈を追加しました。 【アプリからの POST データ例】を修正しました。 対応クラスターを追加・修正しました。
2024.02.21	Rev 2.3.0	P10 P130	システム動作環境を修正しました。 使用上の注意事項を修正しました。
2024.04.17		P15,P18 P33,P122 P130	"instanceName"の注釈を変更しました。 注釈を修正しました。 使用上の注意事項を追加・修正しました。

第1章　ConMas Gateway の概要

ConMas Gateway により、あらゆる外部データソースの情報をリアルタイムに i-Reporter 帳票へ取得することが出来ます。また、外部データソースへ帳票入力データを送信することが出来ます。



1. ConMas Gateway で出来ること

ConMas Gateway を使用することで、以下のような例を実現することが出来ます。

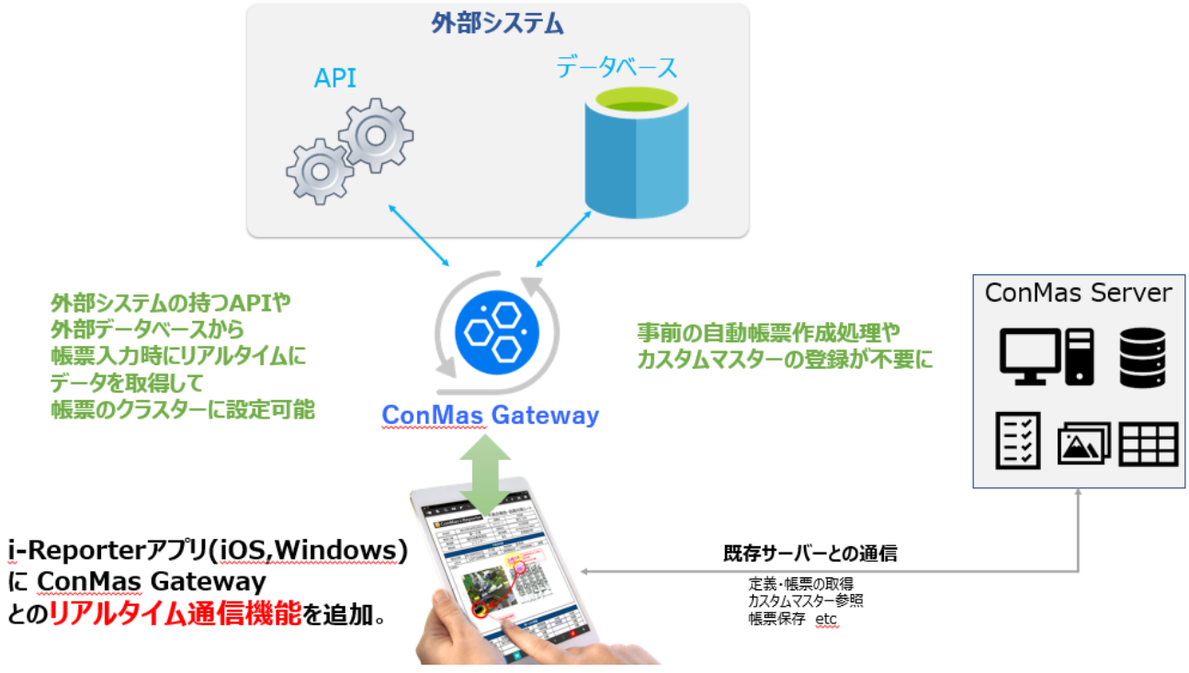
① 自動帳票作成をしないで他システムの今のリアルな情報を使用する。
② カスタムマスターを使用せず、業務システムの各種マスターDB から
マスターデータを直接、リアルタイムに取得する。
③ 基幹システム、生産管理システム、クラウドなど外部システムから
業務データを直接、リアルタイムに取得する。
④ IoT 機器のデータを機器から直接、リアルタイムに取得する。
⑤ 過去の他の帳票の入力内容（画像含む）を取得する。
⑥ 帳票に入力した複数のクラスター値を使用して帳票上に
- グラフを生成する。
- QR コードを生成する。
- 一次元バーコードを生成する。

⑦ 異常値が入力されたら
- BI ツールにリアルタイムに連携し、あんどんなどに直接表示する。
- パトライト等を赤く点灯させる。

⑧ 帳票に入力した複数のクラスター値を使用して、複雑な計算処理を行いその計算結果を取得する。
（未対応 EXCEL 関数への対応が可能に。）
⑨ 帳票に入力した複数のクラスター値を使用して
- Excel からデータを検索し表示する。
- Kintone からデータを検索し表示する。

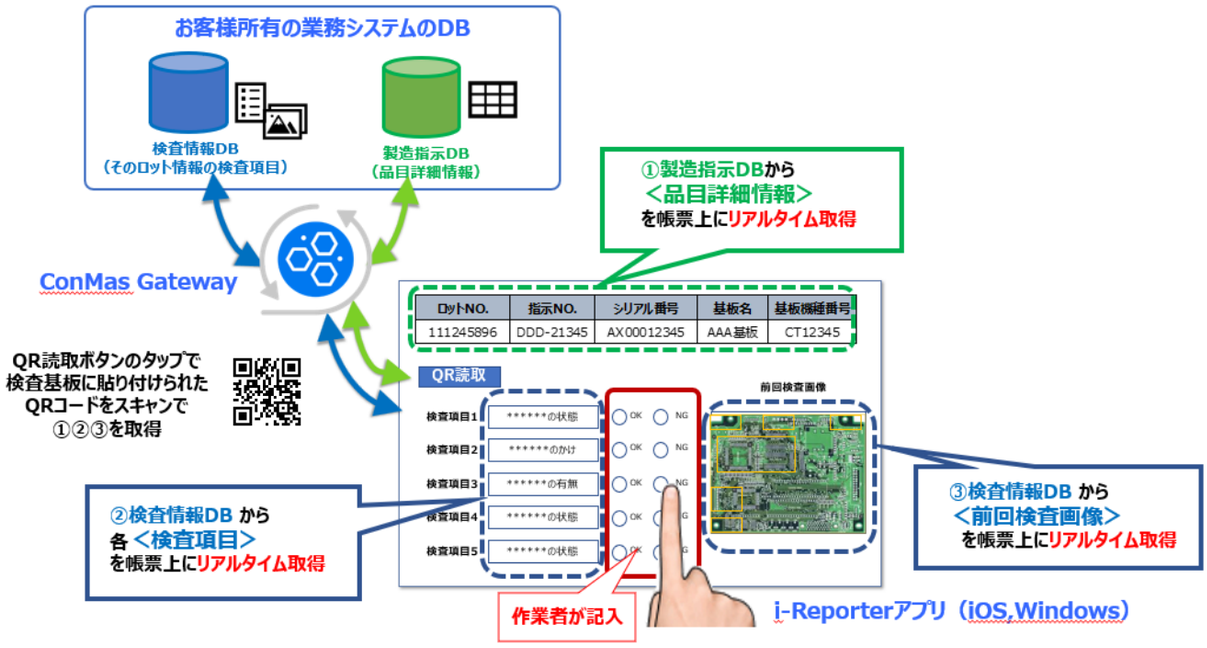
2. ConMas Gateway 機能概念図

既存の ConMas Server を介さず、帳票入力時にタブレットから直接外部データソースから
リアルタイムにデータ取得が可能になる仕組みが、ConMas Gateway です。



3. ConMas Gateway の業務適用イメージ

ConMas Gateway を使用して、入力中の検査帳票へ2つの業務システムのデータをリアルタイムに取得して検査業務を行う例です。



第2章　インストール及び動作環境について

1.　新規インストール

ConMas Gateway のインストール手順については、別紙【ConMas Gateway インストール手順書】を参照してください。

https://cimtops-support.com/i-Reporter/ir_manuals/jp/gateway_iot/ConMasGateway_Install_Procedure_Manual.pdf

2.　アップグレード

最新版のダウンロードについては、サポート WEB の「ソフトウェアのダウンロード」へアクセスしてください。

https://cimtops-support.com/i-Reporter/ja/

アップデート方法については、最新版パッケージ内の GatewayServer_Readme ドキュメントを参照ください。

3. システム動作環境

・ConMas Gateway

ハードウェア

プロセッサ:	Intel Corei5 以上
メモリ:	4GB 以上
ハードディスク:	10GB 以上の空き容量
ネットワークインターフェース:	必須

ソフトウェア

OS:	Windows 10 / 11　64 ビット　Pro　以上
	Windows Server 2016 / 2019 / 2022

その他

- Node.js 20.10.0　以上が動作する環境が必要です。
- サーバーOS を推奨します。
- Linux、Windows、MacOS で Node.js 20.10.0　以上が動作する環境でも利用可です。
- Python スクリプト連携については Python3.10 (Windows)での動作検証を行っております。
- Microsoft SQL Server 連携については、Microsoft SQL Server 2022 環境で動作検証を行っております。

第3章　ConMas Gateway の起動・終了・ログ出力

1.　コマンドプロンプトからの起動について

ここでの説明は標準的な設置環境を前提に書かれています。
ネットワーク環境や特別なカスタマイズ、また PC の機種の違いなどにより、お客様の環境が標準的な環境と異なる場合には、以下の内容についても異なる場合がございます。

1. 電源を入れ PC を起動します。
2. Windows に**ログオンします**
3. ConMas Gateway をインストールしたフォルダへ移動します。

インストールしたフォルダが[¥ConMas¥gateway]の場合、以下のコマンドでフォルダ移動します。

＞CD ¥ConMas¥gateway

4.以下のコマンドで実行します。

＞node index.js

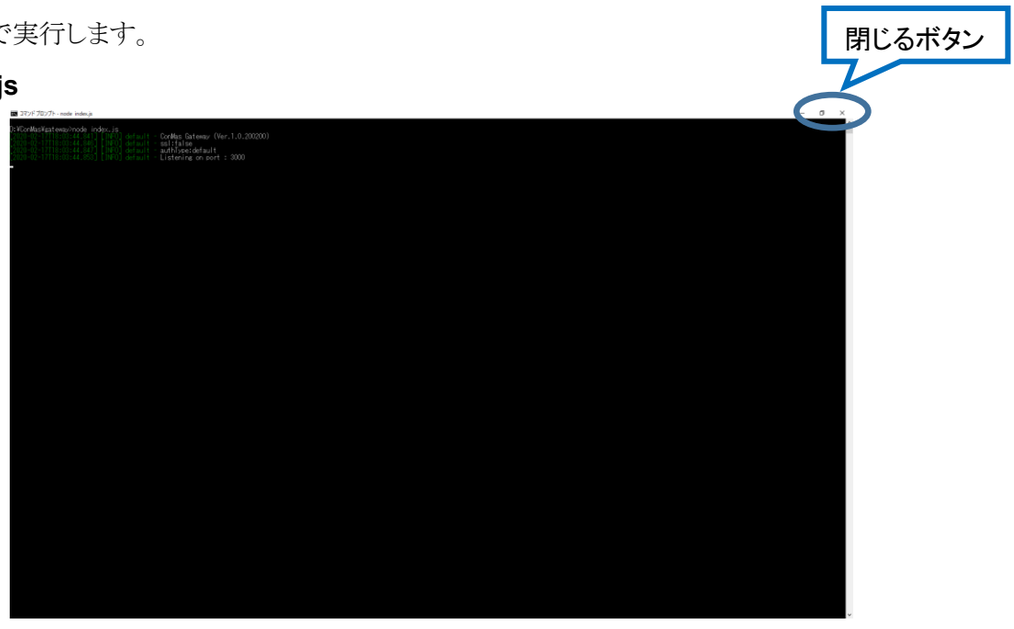


Fig.3-1-1 ConMas Gateway 起動画面

以上で、ConMas Gateway が起動します。

ConMas Gateway をサービス登録している場合はコマンドプロンプトからの起動は必要ありません。

2.　コマンドプロンプトの終了について

1. ConMas Gateway が起動しているコマンドプロンプトを終了します。

タイトルバーの[X]閉じるボタンでコマンドプロンプトを終了します。

プログラムのみ終了するには【Ctrl+C】を入力します。

3.　サービス登録／解除（※Windows）の設定

ConMas Gateway をサービス登録します。

（※　プロンプトで ConMasGateway が起動中の場合は「Ctrl+C」で事前に停止しておきます）

コマンドプロンプトを管理者権限で起動します。

コマンドプロンプトで[¥ConMas¥gateway]フォルダに移動し、以下のコマンドを実行します。

＞npm run install-service

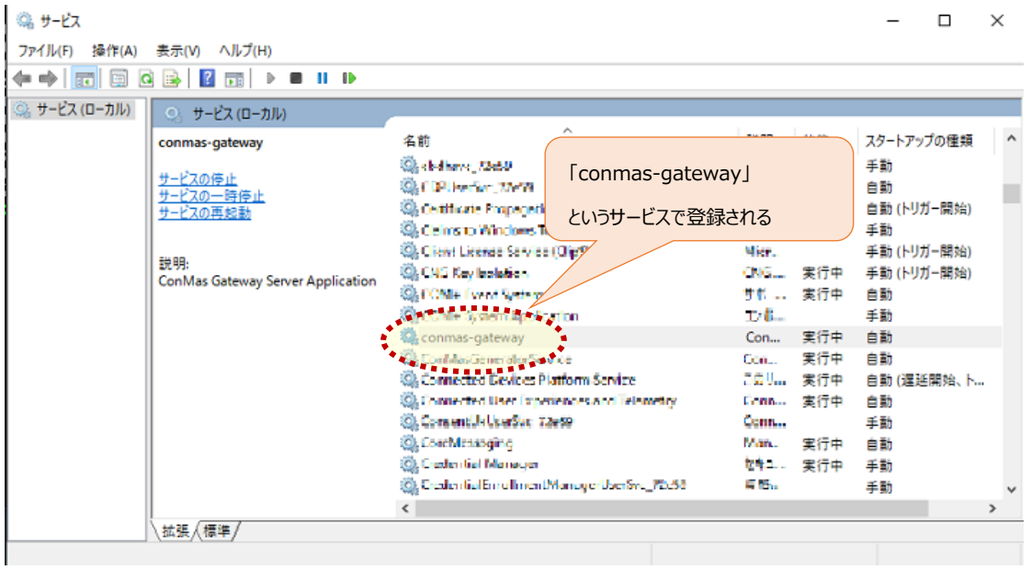


Fig.3-3-1　サービス登録

サービス登録を解除する際は、以下のコマンドを実行します。

＞npm run uninstall-service

4. ログの確認

ログ出力には「log4js」を使用しています。

https://github.com/log4js-node/log4js-node

《フォルダー構成例》

```
D:¥
└─ConMas
    ├─gateway
    |   ├─actions          …… アクションファイル格納フォルダー
    |   ├─config           …… 設定ファイル格納フォルダー
    |   |   └─log4js.json  …… log4js設定ファイル
    |   ├─ex               …… Pythonスクリプト連携サンプル利用フォルダー
    |   ├─logs             …… ログ格納フォルダー
    |   |   ├─access.log   …… アクセスログ
    |   |   └─system.log   …… システムログ
    |   ├─modules          …… 共通モジュール格納フォルダー
    |   ├─node_modules     …… ライブラリ
    |   ├─scripts          …… Python サンプルスクリプト
    |   ├─sql              …… SQL格納フォルダ
    |   ├─uploads          …… POSTリクエストされた画像ファイル格納フォルダ
    |   └─stock_ex         …… Pythonスクリプト連携サンプル利用フォルダ
    ├─samples              …… サンプル
    ├─Tools                …… ツール
    └─updater              …… バージョンアップ用ツール
```

Fig.3-4-1 ログの確認

「システムログ」にコンソール出力と同様のログ情報が出力されます。
サービス化後はログファイルで動作状況の確認を行います。

第4章　ConMas Gateway の設定

1.　フォルダ構成について

ConMas Gateway をインストールしたフォルダには下記サブフォルダ及びファイルが存在します。
設定ファイルの構造について説明します。

D:¥に配置した場合で説明します。

<<フォルダ構成例>>

```
D:¥
  |
  ---------ConMas
              |
              gateway
                 |---------actions          ……アクションファイル格納フォルダ
                 |---------config           ……設定ファイル格納フォルダ
                           |-----datasources ……データソース設定ファイル格納フォルダ
                 |---------logs             ……ログ格納フォルダ
                 |---------node_modules     ……ライブラリ
                 |---------scripts          ……Python スクリプト格納フォルダ
                 |---------sql              ……SQL フォルダ
                 |---------uploads          ……POST リクエストされた画像ファイル格納フォルダ
                 |---------ex               ……Python スクリプト連携サンプル利用フォルダ
                 |---------stock_ex         ……Python スクリプト連携サンプル利用フォルダ
```

Fig.4-1-1 サブフォルダ及びファイル一覧

2. 設定ファイルについて

config フォルダ

ConMas Gateway の config フォルダ内の基本設定ファイル及び actions フォルダ内のアクションファイルで指定する接続先等の情報を記載します。

【default.json】の内容

```
{
  "port": 3000,                    <---------- iOS アプリ、Win アプリとの通信用ポート番号（変更可能）
  "ssl": false,                    <---------- Gateway に HTTPS 接続する場合 true (i-Reporter と別サーバーの場合など)
  "sslkey": "./ssl/cert.key",      <---------- HTTPS 接続の秘密鍵ファイル
  "sslcert": "./ssl/cert.pem",     <---------- HTTPS 接続の証明書ファイル
  "authType": "default",           <---------- default で固定
  "token": "xxxxxxxxxxxxxxxxxx",   <---------- トークン（任意の文字列で設定）
  "pythonShell": {
    "mode": "text",                <---------- 標準入出力を扱う方式(固定)
    "encoding": "utf8",            <---------- 子プロセスのエンコーディング(固定)
    "pythonOptions": ["-X", "utf8"] <---------- ランチャー(Windows なら py)に渡すオプション
  },
  "uploadFileTypes": {             <---------- アップロード可能なファイルタイプを制限
    "binary": " bmp;gif;jpg;tif;png;wav",<---------- バイナリの場合の拡張子(無指定であれば無制限)
    "text": ""                     <---------- テキストの場合の拡張子(無指定であれば無制限)
  },
  "customMaster": {                <---------- マスター選択クラスター連携の設定
    "limit": 500                   <---------- アプリに戻す上限レコード件数（SQL で検索した結果がこの上限を超えた場合はエラー）
  },
  "datasource": [
    { "name": "conmasgwdb",        <---------- データソース名（任意の名前）:PostgreSQL 連携
      "type": "postgreSQL",        <---------- データソースタイプ（"postgreSQL"を設定）
      "user": "postgres",          <---------- ログインユーザー
      "host": "localhost",         <---------- 接続先
      "database": "gwdb",          <---------- 接続 DB
      "password": "cimtops",       <---------- ログインパスワード
      "port": 5432                 <---------- 接続ポート
    },
    { "name": "gwdb",              <---------- データソース名（任意の名前）:Microsoft SQL 連携
      "type": "mssql",             <---------- データソースタイプ（"mssql"を設定）
      "host": " localhost ",       <---------- 接続先
      "user": "sa",                <---------- ログインユーザー
      "password": "cimtops",       <---------- ログインパスワード
      "trustServerCertificate": true, <---------- Microsoft SQL 接続用 trustServerCertificate プロパティ
      "encrypt": true,             <---------- Microsoft SQL 接続 encrypt プロパティ
      "instanceName": "MSSQLSERVER", <---------- インスタンス名（インスタンスが無い場合は行を削除してください）
      "database": "gwdb"           <---------- 接続 DB
    },
```

```
{ "name": "oragwdb",              <---------- データソース名（任意の名前）：Oracle 連携
  "type": "oracle",               <---------- データソースタイプ（”oracle”を設定）
  "user": "system",               <---------- ログインユーザー
  "password": "cimtops",                  <---------- ログインパスワード
  "connectString": "localhost:1521/orcl"            <---------- 接続文字列
},
{ "name": "irepodb",              <---------- データソース名（任意の名前）：i-Reporter DB 連携
  "type": "postgreSQL",           <---------- データソースタイプ（”postgreSQL”を設定）
  "user": "postgres",             <---------- ログインユーザー
  "host": "localhost",            <---------- 接続先
  "database": "irepodb",          <---------- 接続 DB
  "password": "cimtops",          <---------- ログインパスワード
  "port": 5432                    <---------- 接続ポート
},
{ "name": "mysqlgwdb",            <---------- データソース名（任意の名前）：MySQL 連携
  "type": "mysql",                <---------- データソースタイプ（”mysql”を設定）
  "user": "root",                 <---------- ログインユーザー
  "password": "cimtops",          <---------- ログインパスワード
  "host": "localhost",            <---------- 接続先
  "port": "3306",                 <---------- 接続ポート
  "database": "gwdb"              <---------- 接続 DB
},
{ "name": "plcdata",              <----------データソース名（任意の名前）：IoT Data Share 連携
  "type": "iotdswebapi",          <----------データソースタイプ（” iotdswebapi”を設定）
  "url": "http://localhost:5225/api/v1/getvalues", <----------IoT Data Share 接続先
  "self_signed_certificate": true, <----------HTTPS 通信時の自己署名証明書可否
  "method": "POST",               <----------固定設定値
  "headers": {
    "Content-Type": "application/json",
    "Authorization": "Bearer XXXXXXXXXXXXXXXXXXXXXXXXXX"
  }                               <----------セキュリティートークン（IoT Data Share 側で設定時に有効）
},
{ "name": "plcput",               <----------データソース名（任意の名前）：IoT Data Share アイテム値の一括設定連携
  "type": "iotdswebapi",          <----------データソースタイプ（” iotdswebapi”を設定）
  "url": "http://localhost:5225/api/v1/putvalues", <---------- PLC 書込み用 IoT Data Share 接続先
  "self_signed_certificate": true, <----------HTTPS 通信時の自己署名証明書可否
  "method": "POST",               <----------固定設定値
  "headers": {
    "Content-Type": "application/json",
    "Authorization": "Bearer XXXXXXXXXXXXXXXXXXXXXXXXXX"
  }                               <----------セキュリティートークン（IoT Data Share 側で設定時に有効）
},
{ "name": "test1",                <----------データソース名（任意の名前）：接続テスト用
  "type": "static"                <----------データソースタイプ（” static ”を設定）
},
```

```
    { "name": "script",              <----------データソース名（任意の名前）:Python スクリプト連携
      "type": "python"               <----------データソースタイプ（" python "を設定）
    },
  ]
}
```

"datasource"以降の設定が存在しない場合は追加する必要はありません。（config¥datasources フォルダの説明を参照）

同一のデータソースでも複数接続先が存在する場合は個別に設定を作成する必要があります。

トークン設定は、Designer でアクションクラスターのアクション種別：Gateway 連携　トークン設定と合わせる必要があります。

default.json を変更した場合は ConMas Gateway の再起動が必要になります。

ConMas Gateway との SSL 通信については、default.json で ssl パラメータを設定する方法と、上位の WEBサーバーでリバースプロキシを構成して Node.js をWEBサーバー上でホストする方法の2種類が利用可能です。

config¥datasources フォルダ

ConMas Gateway の config¥datasources フォルダ内のデータソース設定ファイルは、データソースを分離するために利用します。datasources フォルダ内に接続先が存在する場合は、default.json で設定されている接続先よりも優先されます。また、default.json 内のデータソースを変更した場合は ConMas Gateway の再起動が必要ですが、datasources フォルダ内の接続先を変更した場合は再起動の必要はありません。

datasources フォルダ内に設定ファイルを作成する場合は、default.json で指定した"name"項目がファイル名になります。

例えば、default.json 内の

```
"datasource": [
  { "name": "conmasgwdb",   <---------- データソース名（任意の名前）:PostgreSQL 連携
    "type": "postgreSQL",   <---------- データソースタイプ（"postgreSQL"を設定）
    "user": "postgres",     <---------- ログインユーザー
    "host": "localhost",    <---------- 接続先
    "database": "gwdb",              <---------- 接続 DB
    "password": "cimtops",  <---------- ログインパスワード
    "port": 5432            <---------- 接続ポート
  }
```

を分離させる場合のファイル名は、¥ConMas¥gateway¥config¥datasources¥**conmasgwdb**.json になります。

【PostgreSQL 設定（¥ConMas¥gateway¥config¥datasources¥conmasgwdb.json）】

```
{
  "type": "postgreSQL",     <---------- データソースタイプ（"postgreSQL"を設定）
  "user": "postgres",       <---------- ログインユーザー
  "host": "localhost",      <---------- 接続先
  "database": "gwdb",       <---------- 接続 DB
  "password": "cimtops",    <---------- ログインパスワード
  "port": 5432              <---------- 接続ポート
}
```

【Microsoft SQL Server 設定（¥ConMas¥gateway¥config¥datasources¥gwdb.json）】

```
{
  "type": "mssql",                  <---------- データソースタイプ(”mssql”を設定)
  "host": " localhost ",            <---------- 接続先
  "user": "sa",                     <---------- ログインユーザー
  "password": "cimtops",            <---------- ログインパスワード
  "trustServerCertificate": true,   <---------- Microsoft SQL 接続用 trustServerCertificate プロパティ
  "encrypt": true,                  <---------- Microsoft SQL 接続 encrypt プロパティ
  "instanceName": "MSSQLSERVER",    <---------- インスタンス名(インスタンスが無い場合は行を削除してください)
  "database": "gwdb"                <---------- 接続 DB
}
```

【Oracle 設定(¥ConMas¥gateway¥config¥datasources¥oragwdb.json)】

```
{
  "type": "oracle",                 <---------- データソースタイプ(”oracle”を設定)
  "user": "system",                 <---------- ログインユーザー
  "password": "cimtops",                  <---------- ログインパスワード
  "connectString": "localhost:1521/orcl"          <---------- 接続文字列
}
```

【MySQL 設定(¥ConMas¥gateway¥config¥datasources¥mysqlgwdb.json)】

```
{
  "type": "mysql",                  <---------- データソースタイプ(”mysql”を設定)
  "user": "root",                   <---------- ログインユーザー
  "password": "cimtops",            <---------- ログインパスワード
  "host": "localhost",              <---------- 接続先
  "port": "3306",                   <---------- 接続ポート
  "database": "gwdb"                <---------- 接続 DB
}
```

【i-Reporter 設定(¥ConMas¥gateway¥config¥datasources¥irepodb.json)】

```
{ "name": "irepodb",                <---------- データソース名(任意の名前):i-Reporter DB 連携
  "type": "postgreSQL",             <---------- データソースタイプ(”postgreSQL”を設定)
  "user": "postgres",               <---------- ログインユーザー
  "host": "localhost",              <---------- 接続先
  "database": "irepodb",            <---------- 接続 DB
  "password": "cimtops",            <---------- ログインパスワード
  "port": 5432                      <---------- 接続ポート
}
```

【IoT Data Share 読込設定(¥ConMas¥gateway¥config¥datasources¥plcdata.json)】

```
{
  "type": "iotdswebapi",            <----------データソースタイプ(” iotdswebapi”を設定)
  "url": "http://localhost:5225/api/v1/getvalues", <----------IoT Data Share 接続先
  "self_signed_certificate": true, <----------HTTPS 通信時の自己署名証明書可否
  "method": "POST",                  <----------固定設定値
```

```
    "headers": {
      "Content-Type": "application/json",
      "Authorization": "Bearer XXXXXXXXXXXXXXXXXXXXXXXXX"
    }                                   <----------セキュリティートークン（IoT Data Share 側で設定時に有効）
  }
```

【IoT Data Share 書込設定（¥ConMas¥gateway¥config¥datasources¥plcput.json）】

```
  {
    "type": "iotdswebapi",              <----------データソースタイプ（" iotdswebapi"を設定）
    "url": "http://localhost:5225/api/v1/putvalues", <---------- PLC 書込み用 IoT Data Share 接続先
    "self_signed_certificate": true, <----------HTTPS 通信時の自己署名証明書可否
    "method": "POST",                   <----------固定設定値
    "headers": {
      "Content-Type": "application/json",
      "Authorization": "Bearer XXXXXXXXXXXXXXXXXXXXXXXXX"
    }                                   <----------セキュリティートークン（IoT Data Share 側で設定時に有効）
  }
```

【Python スクリプト設定（¥ConMas¥gateway¥config¥datasources¥script.json）】

```
{
  "type": "python"                      <----------データソースタイプ（" python "を設定）
}
```

【接続テスト設定（¥ConMas¥gateway¥config¥datasources¥ test1.json）】

```
  {
    "type": "static"                    <----------データソースタイプ（" static "を設定）
  }
```

3. アクションファイルについて

actions フォルダ

i-Reporter のクラスターと PostgreSQL, Microsoft SQL、Oracle, Python スクリプト、WebAPI（IoT Data Share）と連携させる動作を設定した JSON ファイルが配置されています。

●PostgreSQL 接続テスト用 JSON ファイル

例： **i-Reporter** アクションクラスター設定が下記の場合に、

アクション種別： 【Gateway 連携】

URL： http://localhost:3000/api/v1/getvalue/pgtest?user_id=conmasadmin

トークン： xxxxxxxxxxxxxxxxx

1 2

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の pgtest.json を呼び出します。
2. user_id という変数に conmasadmin という文字列が渡されます。

【pgtest.json】の内容

```
{
  "datasource": "irepodb",
  "basequery": "select * from mst_user ",
  "where": "",
  "params": [
    {
      "name": "user_id",
      "type": "string"
    }
  ],
  "mappings": [
    {
      "item": "user_name",
      "sheet": 1,
      "cluster": 1,
      "type": "string",
      "value": ""
    }  ]
}
```

default.json 内の"irepodb"項目の設定が呼び出されます

基本クエリー： **SQL** の基本部分

固定条件： このアクションでの固定条件 **(ex) where active = true**

可変条件： パラメーターで指定できる可変条件。
（複数の場合は **AND** のみ）
"name" ・・・テーブルのフィールド名
"type" ・・・データ型

クラスターマッピング： 取得した値をクラスターへ挿入する設定
"item" ・・・テーブルのフィールド名
"sheet" ・・・帳票のシート No
シート番号の指定が無く、クラスター番号のみ指定される場合はアクティブなシートのクラスター番号指定でマッピングされます。
"cluster" ・・・帳票のクラスターインデックス
"type" ・・・（"string"または”image”）
"value" ・・・取得した値が格納される。
設定ファイルにあらかじめ入力し、
クエリーで取得しなければ固定値としても利用可

Fig.4-3-1 i-Reporter アクションクラスター設定

第5章　サンプルの利用方法

1.　サンプルの準備

ConMas Gateway パッケージをサポート WEB よりダウンロードいただき、解凍後に¥ConMas¥samples フォルダ内のサブフォルダにサンプルがございます。

¥samplepy_graph

PostgreSQL 連携、Microsoft SQL Server 連携、Python スクリプト連携（線グラフ、棒グラフ、円グラフ、QR コード、バーコード）、i-Reporter 連携

¥samplepy_read

Python スクリプト連携（Excel、kintone）

¥samplepy_Excel_function

Python スクリプト連携（Excel 関数を Python スクリプトで作成）

¥samplepy_GET_POST

Python スクリプト連携（PostgreSQL からのデータ取得・計算・書き込み）

Python スクリプト連携（Microsoft SQL Server からのデータ取得・計算・書き込み）

Python スクリプト連携（EXCEL からのデータ取得・計算・書き込み）

¥samplepy_post

Python スクリプト連携（POST リクエスト用サンプル）

各フォルダ内にサンプル定義がございます。

Microsoft SQL Server 連携を利用される場合は事前にテーブル作成が必要です。

mssql_table.sql　・・・・・・・**SQLServer 用テーブル作成スクリプト**

Oracle 連携を利用される場合は事前にテーブル作成が必要です。

oracle_table.sql　・・・・・・・**Oracle 用テーブル作成スクリプト**

PostgreSQL 連携を利用される場合は事前にテーブル作成が必要です。

pgsql_table.sql　・・・・・・・**PostgreSQL 用テーブル作成スクリプト**

各フォルダ内に利用方法についての Readme_xxxxxx.md が存在します。

本マニュアルと合わせてご確認ください。

Readme_xxxxxx.md ・・・・・・・**サンプル利用方法について**

¥sample_select

単一選択クラスター連携の選択肢取得

¥samples¥sample_table_format

Python スクリプト連携（複数行の検索結果を複数ページの帳票に展開するサンプル）

¥SetFocus

クラスターをフォーカスする機能

¥sample_custom_master

マスター選択クラスターの連携先設定

¥sample_AutoGenerate

Python スクリプト連携（外部連携 API(自動帳票作成)を実行するサンプル）

¥sample_custom_master_py

Python スクリプト連携（マスター選択クラスターの連携先設定）

2. サンプル定義の準備

【ConMas Gateway】サンプル定義.xml を Designer で読込み利用できるように準備します。

Designer を起動します。

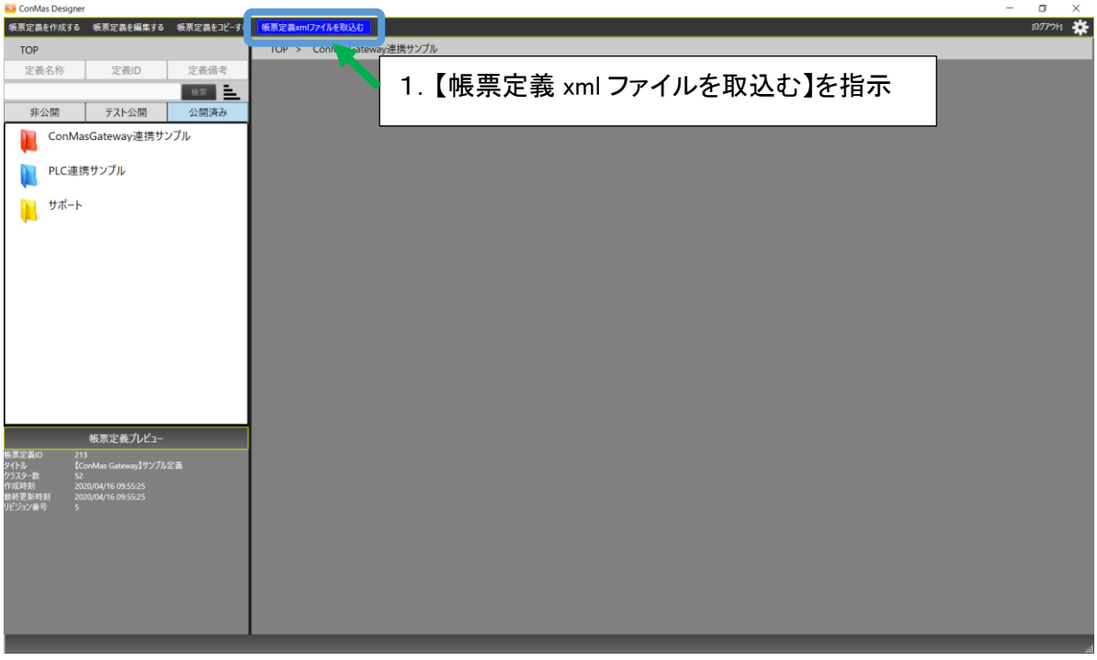


Fig.5-2-1 サンプル定義の準備（Step1）

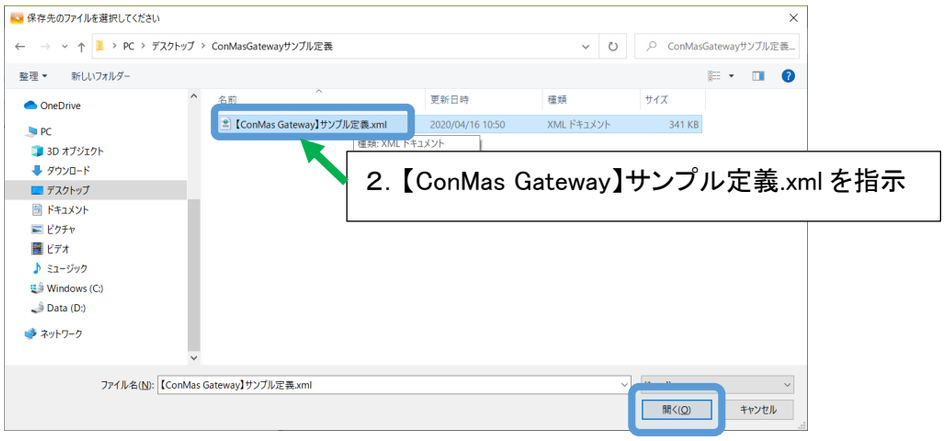


Fig.5-2-2 サンプル定義の準備（Step2）

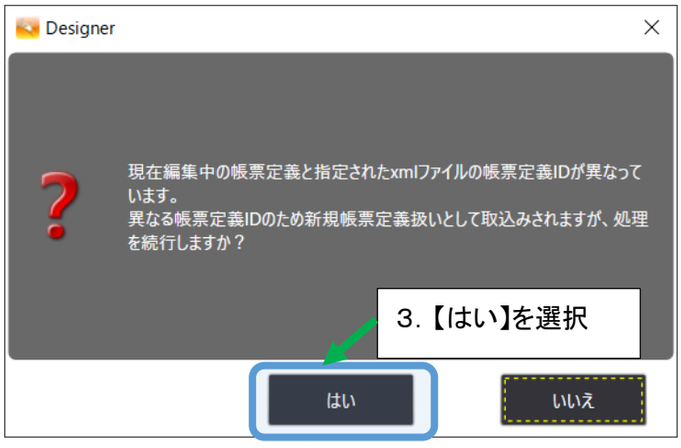


Fig.5-2-3 サンプル定義の準備（Step3）

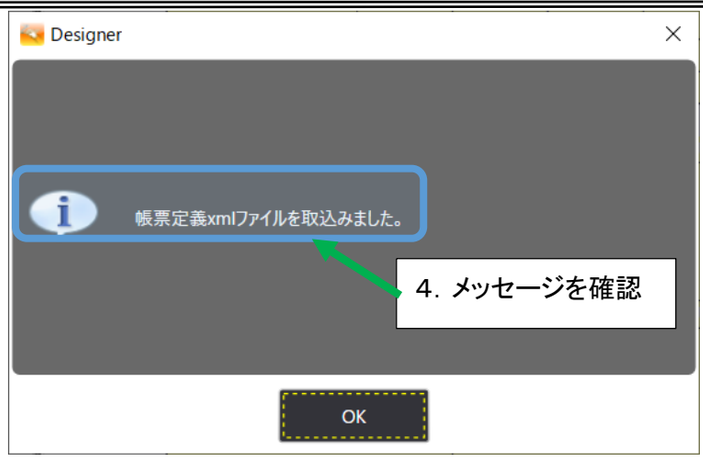


Fig.5-2-4 サンプル定義の準備（Step4）

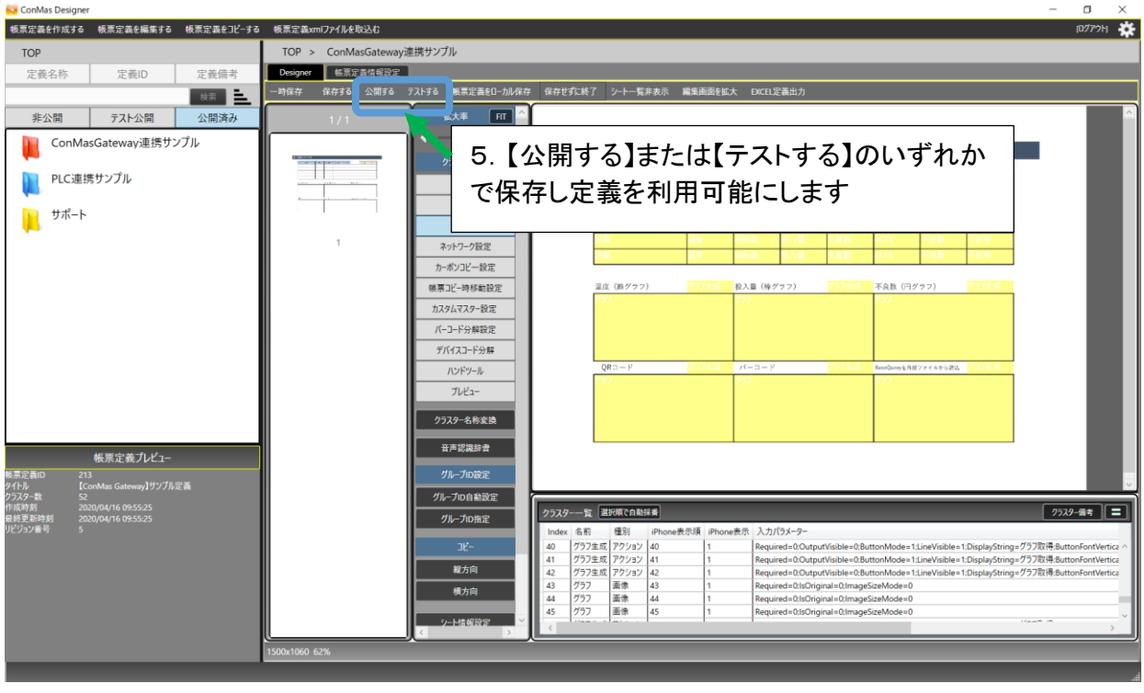


Fig.5-2-5 サンプル定義の準備（Step5）

3.　タイマー起動を利用した自動取得設定

アクション種別を【タイマー】に設定したアクションクラスターを配置することにより、アクションクラスターを押下することなく指定時間でアクションクラスターの起動が可能です。

3-1. タイマー起動用アクションクラスターの配置と設定

連携する各クラスターの詳細の入力パラメータ設定を行います。

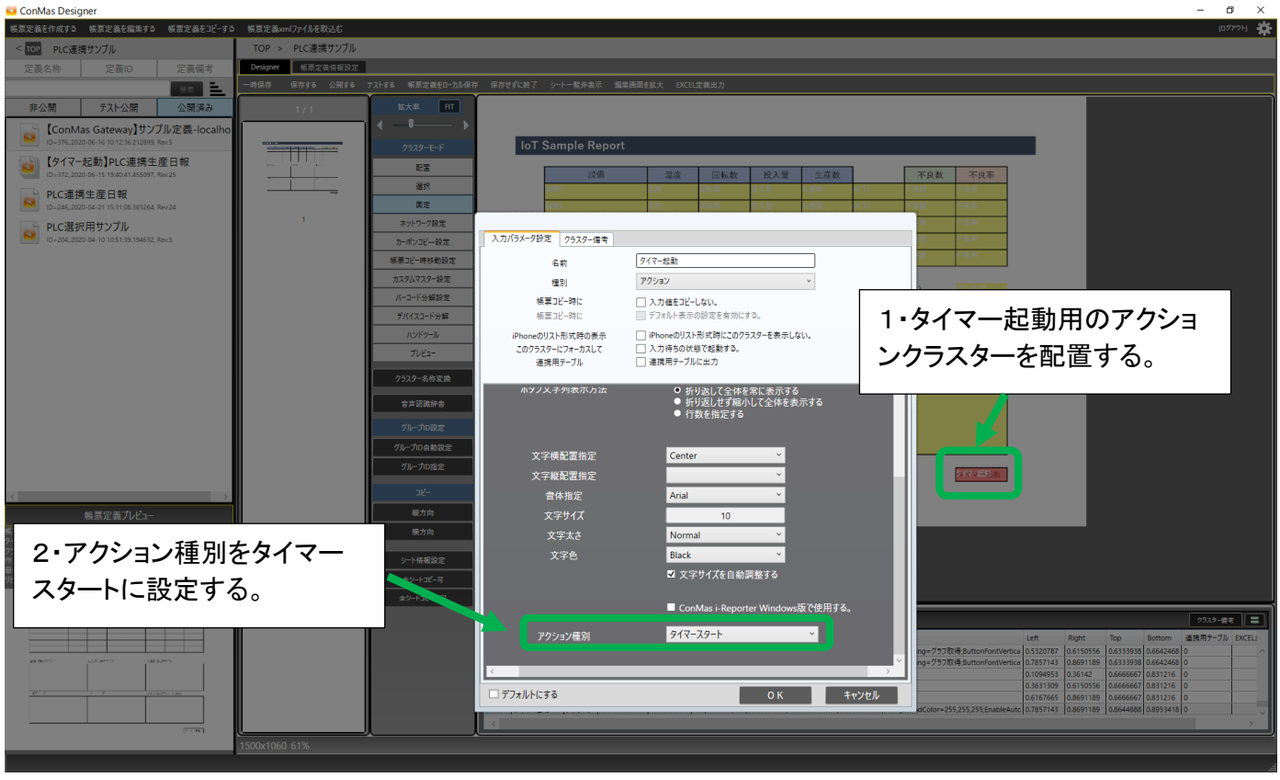


Fig.5-3-1 入力パラメータ設定

【操作手順】

1. EXCEL COM Add-in でタイマー起動用のアクションクラスターを配置します。
2. アクション種別をタイマースタートに設定します。

タイマー起動はアクション種別「Gateway 連携」のスケジュール時刻とセットで使用します。
タイマー起動を実行するとタイマーモードになり、タイマーモードを解除するまでは一切の帳票編集が不可になります。この状態で「Gateway 連携」のスケジュール時刻で設定しておいた時刻になると、該当の「Gateway 連携」を自動で実行します。なお、タイマーモード中に動作するようなネットワーク設定が行われていた場合はネットワーク機能の動作保証はされません。

3-2. アクションクラスターの設定

起動する時刻を各アクションクラスターに設定します。

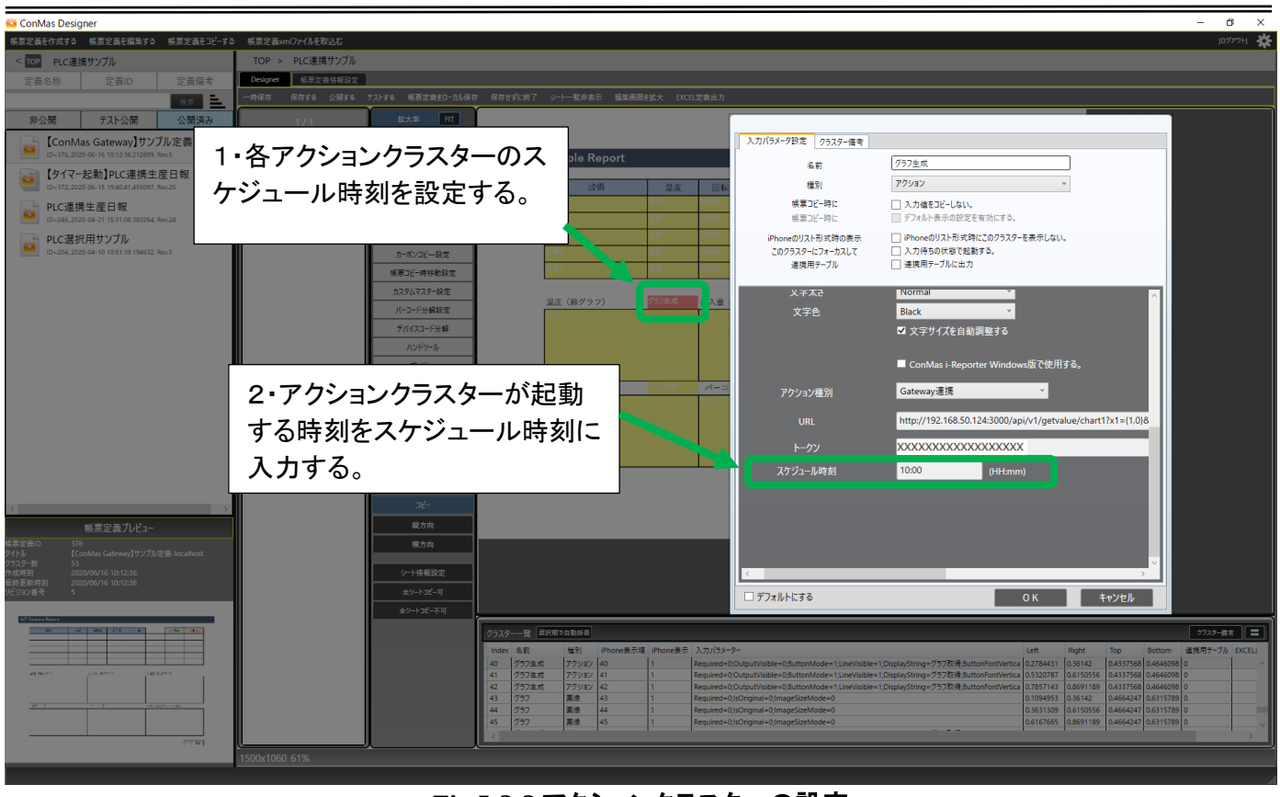


Fig.5-3-2 アクションクラスターの設定

【操作手順】

1. 各アクションクラスターのスケジュール時刻を設定します。
2. アクションクラスターが起動する時刻をスケジュール時刻に入力します。

タイマー起動用アクションクラスターが配置されているシート以外のアクションクラスターも設定可能です。その場合、起動時刻になったアクションクラスターが配置されているシートに自動移動します。

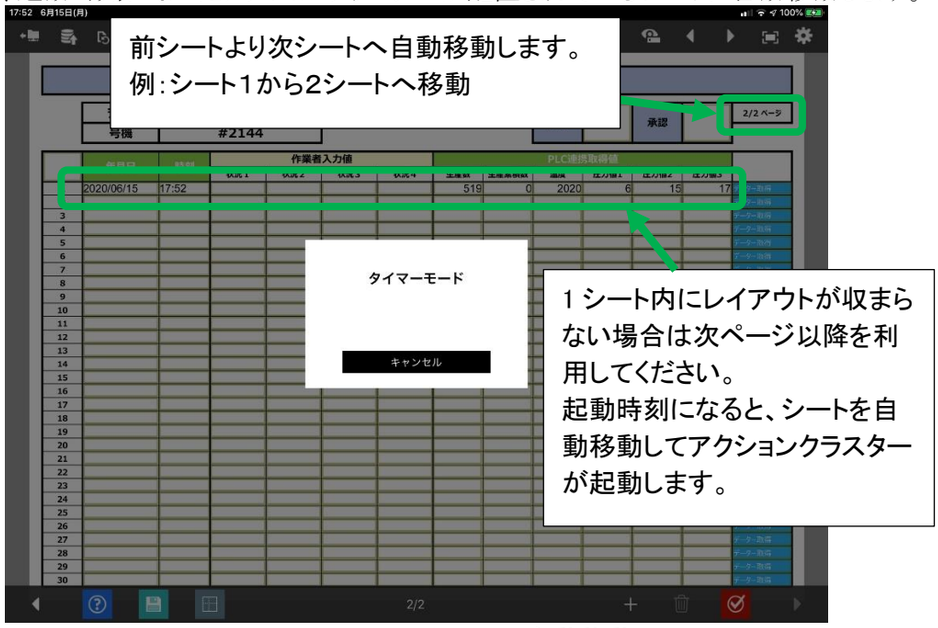


Fig.5-3-3 シートの自動移動

アクションクラスターが起動するスケジュール時刻の間隔は最低10分以上が必要になります。

4.　タイマー起動のアプリでの動作確認

タイマー起動の動作確認を行います。

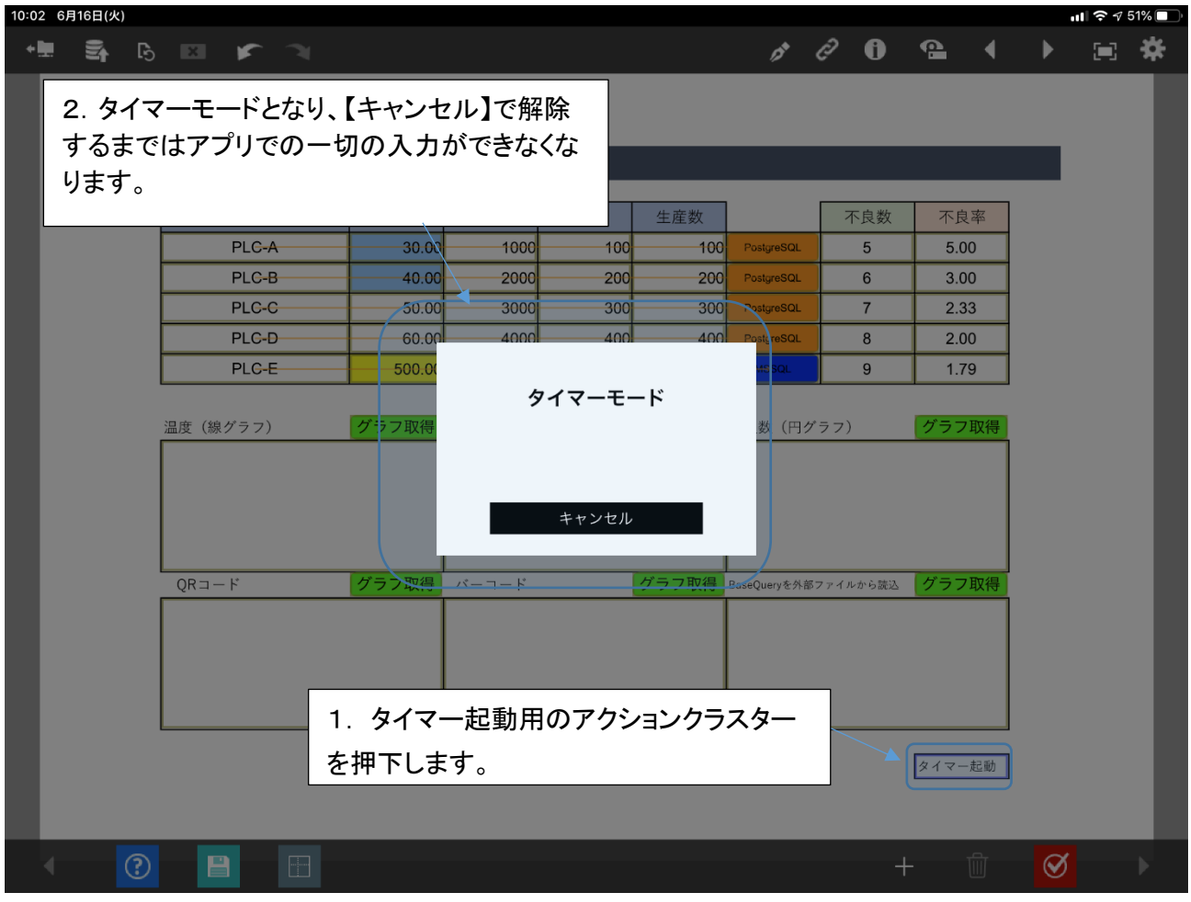


Fig.5-4-1 動作確認1（iOS アプリ）

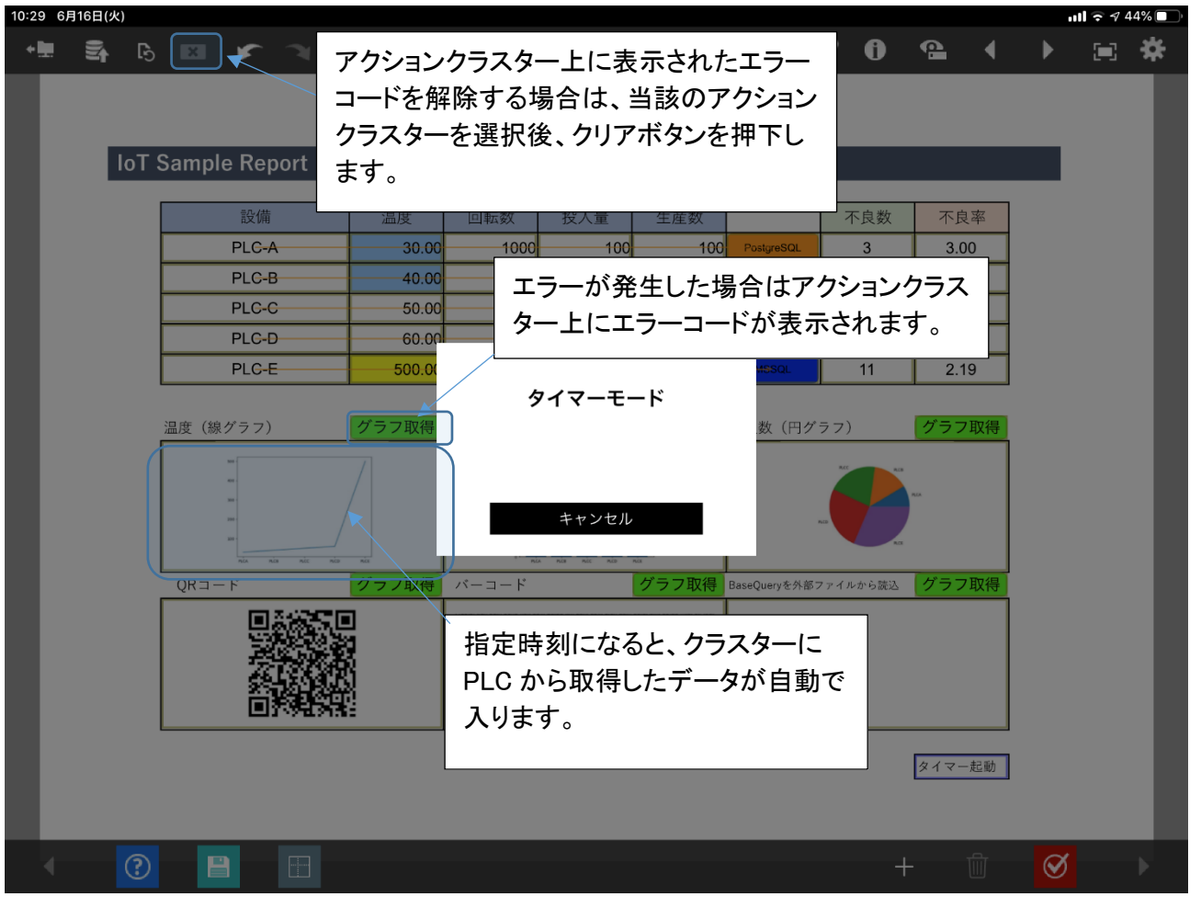


Fig.5-4-2 動作確認2（iOS アプリ）

- シートコピーを利用した場合、タイマー起動をコピー元シートで行った場合はコピー先シートのアクションクラスターは自動起動しません。
- シートコピーを利用した場合、タイマー起動をコピー先シートで行った場合は、タイマー起動を実施したコピー先シートのアクションクラスターのみ自動起動されます。
- iOS アプリでは、アプリが非アクティブになったとき、アプリをバックグラウンドにした時、アプリを終了した時は、タイマーが解除されます。

第6章　各連携のサンプル設定

1.　PostgreSQL 連携

【ConMas Gateway】サンプル定義を利用して PostgreSQL 連携について説明します。

Fig.6-1-1 PostgreSQL 連携

【動作概要】

1. 単一選択クラスターより設備を選択。
2. ネットワーク接続されたアクションクラスターが起動。
3. 【温度】、【回転数】、【投入数】、【生産数】を検索条件からテーブルから呼出し各クラスターへ値を挿入。

【アクションクラスター設定】

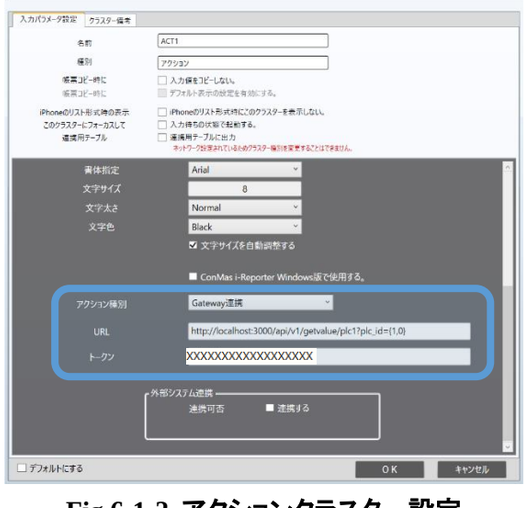
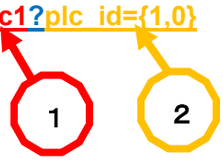


Fig.6-1-2 アクションクラスター設定

アクション種別:　　【Gateway 連携】

URL：　http://localhost:3000/api/v1/getvalue/plc1?plc_id={1,0}

トークン：　xxxxxxxxxxxxxxxxx



localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の plc1.json を呼び出します。
2. plc_id という変数にシート番号：1、クラスターID：0 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

4. テーブル内の【温度】、【回転数】、【投入数】、【生産数】を検索条件から呼出し、各クラスターへ戻値が挿入されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥plc1.json）】

```
{
  "datasource": "conmasgwdb",
  "basequery": "select temp, rpm, in_amount, out_amount from plcdata1",
  "where": "",
  "params": [
    { "name": "plc_id", "type": "string"},
    { "name": "rpm", "type": "string"}
  ],
  "mappings": [
    { "item": "temp",       "sheet": 1, "cluster": 2, "type": "string", "value" : ""},
    { "item": "rpm",        "sheet": 1, "cluster": 1, "type": "string", "value" : ""},
    { "item": "in_amount",  "sheet": 1, "cluster": 3, "type": "string", "value" : ""},
    { "item": "out_amount", "sheet": 1, "cluster": 4, "type": "string", "value" : ""}
  ]
}
```

default.json 内の"conmasgwdb"項目の設定が呼び出されます

基本クエリー： SQL の基本部分

検索条件： plc id 変数で検索

CHAR 型の場合は char、
日付型は date、
それ以外は string

テーブルのフィールド名

i-Reporter のクラスター番号

i-Reporter のシート番号

画像の場合は image、
それ以外は string

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "conmasgwdb",<---------- データソース名（任意の名前）:PostgreSQL 連携
  "type": "postgreSQL",
  "user": "postgres",
  "host": "localhost",
  "database": "gwdb ",
  "password": "cimtops",
  "port": 5432
},
```

連携先としてこのデータソースを利用します

2. Microsoft SQL Server 連携

【ConMas Gateway】サンプル定義を利用して Microsoft SQL Server 連携について説明します。

Fig.6-2-1 Microsoft SQL Server 連携

【動作概要】

1. 単一選択クラスターより設備を選択。
2. ネットワーク接続されたアクションクラスターが起動。
3. 【温度】、【回転数】、【投入数】、【生産数】を検索条件からテーブルから呼出し各クラスターへ値を挿入。

【アクションクラスター設定】

Fig.6-2-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：　http://localhost:3000/api/v1/getvalue/mssql1?plc_id={1,32}

トークン：　XXXXXXXXXXXXXXXXX

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の mssql1.json を呼び出します。
2. plc_id という変数にシート番号:1、クラスターID:32 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3.. テーブル内の【温度】、【回転数】、【投入数】、【生産数】を検索条件から呼出し、各クラスターへ戻値が挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥ mssql1.json)】

```
{
  "datasource": " gwdb",
  "basequery": " select temp, rpm, in_amount, out_amount from plcdata1"
  "where": "",
  "params": [
    { "name": "plc_id", "type": "string"},
  ],
  "mappings": [
    { "item": "temp"       "sheet": 1, "cluster": 33, "type": "string" "value" : ""},
    { "item": "rpm"        "sheet": 1, "cluster": 34, "type": "string" "value" : ""},
    { "item": "in_amount"  "sheet": 1, "cluster": 35, "type": "string" "value" : ""},
    { "item": "out_amount", "sheet": 1, "cluster": 36, "type": "string" "value" : ""}
  ]
}
```

default.json 内の"gwdb"項目の設定が呼び出されます

検索条件： plc id 変数で検索

基本クエリー： SQL の基本部分

テーブルのフィールド名

i-Reporter のクラスター番号

文字列型、日付型は string、数値型は numeric

i-Reporter のシート番号

画像の場合は image、それ以外は string

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "gwdb",<---------- データソース名(任意の名前):Microsoft SQL 連携
    "type": "mssql",
    "host": "192.168.50.125",
    "user": "sa",
    "password": "cimtops",
    "trustServerCertificate": true,
    "encrypt": true,
    "instanceName": "MSSQLSERVER",
    "port": 1433,
    "database": "gwdb"
  },
```

instanceName(インスタンス名)、port(ポート番号)はいずれかのみ指定可能です。

3. Oracle 連携

【ConMas Gateway】サンプル定義の MSSQL ボタンを Oracle 用に変更して Oracle 連携について説明します。

Fig.6-3-1 Oracle 連携

【動作概要】

1. 単一選択クラスターより設備を選択。
2. ネットワーク接続されたアクションクラスターが起動。
3. 【温度】、【回転数】、【投入数】、【生産数】を検索条件からテーブルから呼出し各クラスターへ値を挿入。

【アクションクラスター設定】

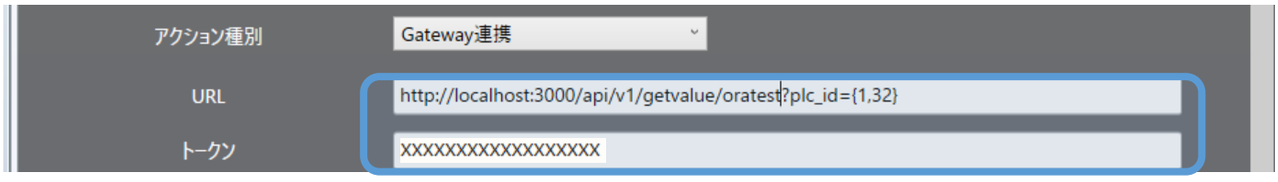


Fig.6-3-2 アクションクラスター設定

アクション種別: 【Gateway 連携】

URL: http://localhost:3000/api/v1/getvalue/oratest?plc_id={1,32}

トークン: xxxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

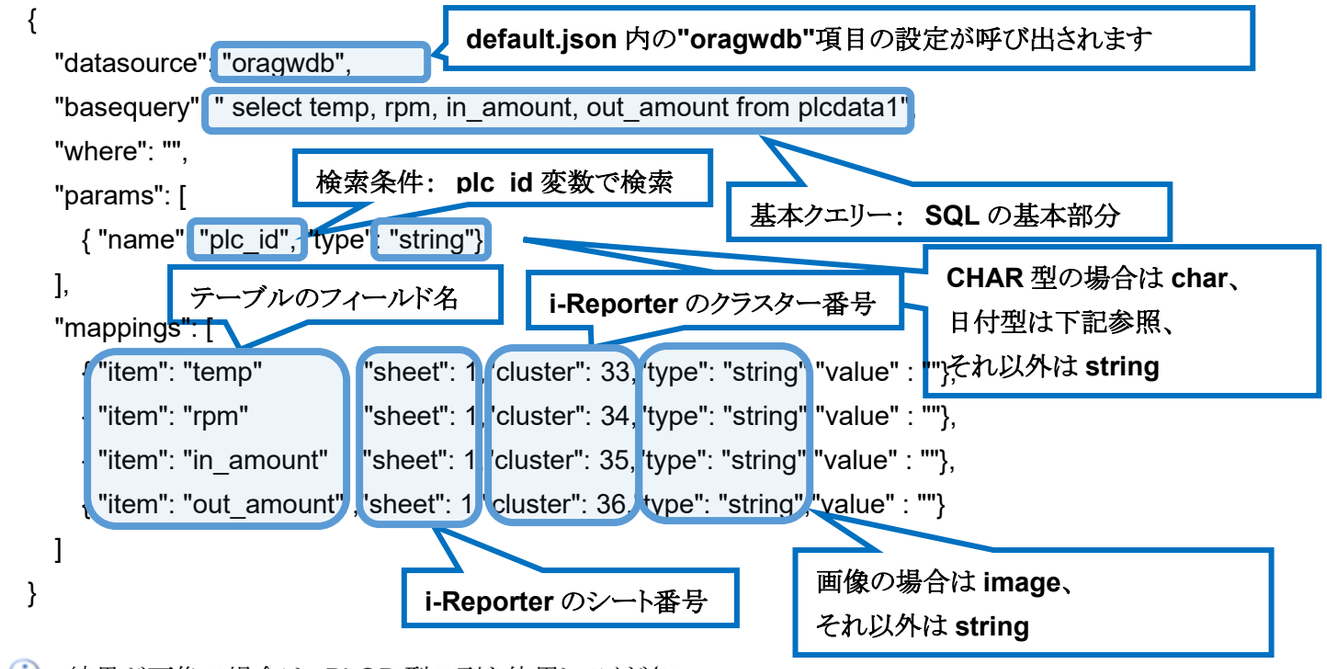
1. アクションフォルダ内の oratest.json を呼び出します。
2. plc_id という変数にシート番号:1、クラスターID:32 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. テーブル内の【温度】、【回転数】、【投入数】、【生産数】を検索条件から呼出し、各クラスターへ戻値が挿入されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥oratest.json）】

```
{
  "datasource": "oragwdb",
  "basequery" " select temp, rpm, in_amount, out_amount from plcdata1"
  "where": "",
  "params": [
    { "name" "plc_id", "type" : "string"}
  ],
  "mappings": [
    { "item": "temp"       "sheet": 1, "cluster": 33, "type": "string" "value" : ""},
    { "item": "rpm"        "sheet": 1, "cluster": 34, "type": "string" "value" : ""},
    { "item": "in_amount"  "sheet": 1, "cluster": 35, "type": "string" "value" : ""},
    { "item": "out_amount", "sheet": 1, "cluster": 36, "type": "string", "value" : ""}
  ]
}
```



結果が画像の場合は、BLOB 型の列を使用してください。

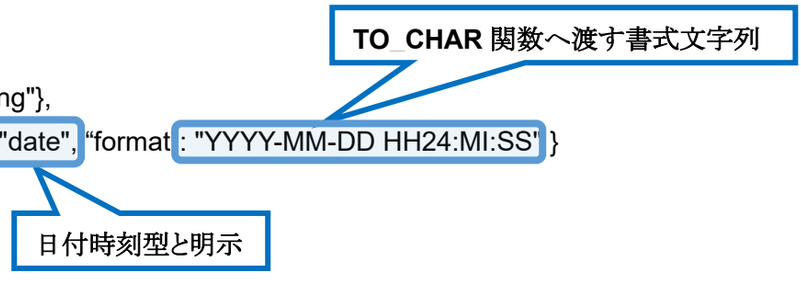
【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "oragwdb",<---------- データソース名（任意の名前）:Oracle 連携
  "type": "oracle",
  "user": "system",
  "password": "cimtops",
  “connectString”: “localhost:1521/orcl”
},
```

日付時刻列を条件列に含める際に文字列比較を行っています。日付時刻列については、書式指定をしないと意図する検索結果を得ることができません。その場合のアクションファイル内 params に指定する方法は次のようになります。

【アクションファイル設定】

```
"params": [
  { "name": "plc_id", "type": "string"},
  { "name": "start_time", "type": "date", “format” : "YYYY-MM-DD HH24:MI:SS" }
],
```



4. MySQL 連携

【ConMas Gateway】サンプル定義の MSSQL ボタンを MySQL 用に変更して MySQL 連携について説明します。

Fig.6-4-1 MySQL 連携

【動作概要】

1. 単一選択クラスターより設備を選択。
2. ネットワーク接続されたアクションクラスターが起動。
3. 【温度】、【回転数】、【投入数】、【生産数】を検索条件からテーブルから呼出し各クラスターへ値を挿入。

【アクションクラスター設定】

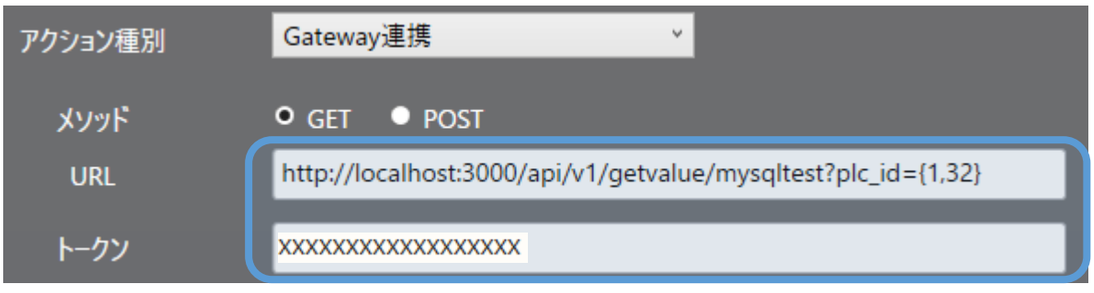


Fig.6-4-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：　http://localhost:3000/api/v1/getvalue/mysqltest?plc_id={1,32}

トークン：　XXXXXXXXXXXXXXXXX

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の mysqltest.json を呼び出します。
2. plc_id という変数にシート番号：1、クラスターID：32 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. テーブル内の【温度】、【回転数】、【投入数】、【生産数】を検索条件から呼出し、各クラスターへ戻値が挿入されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥mysqltest.json）】

```
{
  "datasource": "mysqlgwdb",
  "basequery" " select temp, rpm, in_amount, out_amount from plcdata1"
  "where": "",
  "params": [
    { "name" "plc_id", "type' : "string"}
  ],
  "mappings": [
    { "item": "temp"       "sheet": 1, cluster": 33, 'type": "string" "value" : ""},
    { "item": "rpm"        "sheet": 1, 'cluster": 34, 'type": "string" "value" : ""},
    { "item": "in_amount"  "sheet": 1 'cluster": 35, 'type": "string" "value" : ""},
    { "item": "out_amount", 'sheet": 1 'cluster": 36, type": "string", "value" : ""}
  ]
}
```

default.json 内の"oragwdb"項目の設定が呼び出されます

検索条件： plc id 変数で検索

基本クエリー： SQL の基本部分

テーブルのフィールド名

i-Reporter のクラスター番号

CHAR 型の場合は char、
日付型は date、
それ以外は string

i-Reporter のシート番号

画像の場合は image、
それ以外は string

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "mysqlgwdb",<---------- データソース名（任意の名前）:MySQL 連携
  "type": "mysql",
  "user": "root",
  "password": "cimtops",
  "host": "localhost",
  "port": "3306",
  "database": "gwdb"
},
```

5. Python スクリプト連携（線グラフ）

【ConMas Gateway】サンプル定義を利用して Python スクリプト連携で線グラフを表示する方法について説明します。

Fig.6-5-1 Python スクリプト連携（棒グラフ）

【動作概要】

1.【温度】列の数値を入力します。

2.【グラフ取得】アクションクラスターを起動します。

3. Python スクリプトで作成された線グラフが表示されます。

【アクションクラスター設定】

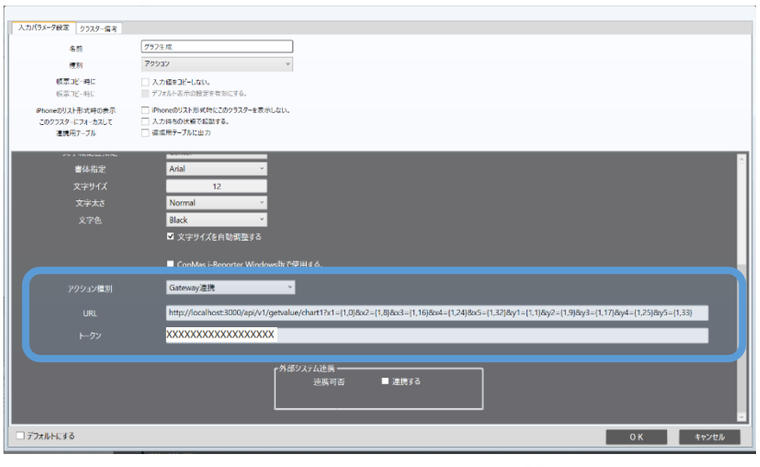


Fig.6-5-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/chart1?x1={1,0}&x2={1,8}&x3={1,16}&x4={1,24}&x5={1,32}&y1={1,1}&y2={1,9}&y3={1,17}&y4={1,25}&y5={1,33}

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart1.json を呼び出します。
2. x1 という変数にシート番号:1、クラスターID:0 のクラスター値
   x2 という変数にシート番号:1、クラスターID:8 のクラスター値
   x3 という変数にシート番号:1、クラスターID:16 のクラスター値
   x4 という変数にシート番号:1、クラスターID:24 のクラスター値
   x5 という変数にシート番号:1、クラスターID:32 のクラスター値
   y1 という変数にシート番号:1、クラスターID:1 のクラスター値
   y2 という変数にシート番号:1、クラスターID:9 のクラスター値
   y3 という変数にシート番号:1、クラスターID:17 のクラスター値
   x4 という変数にシート番号:1、クラスターID:25 のクラスター値
   x5 という変数にシート番号:1、クラスターID:33 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【温度】列の各クラスター値を Python スクリプトへ渡しグラフが表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥chart1.json)】

```
{
  "datasource": "script ",
  "script": " scripts/line.py",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の line.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):Python スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥line.py)】

```
import sys
import matplotlib.pyplot as plt
import uuid
import json
import base64
import os

try:
  #引数を処理
  #
  # ex)
```

```
#    http://localhost:3000/api/v1/getvalue/chart1?x1=A&x2=B&x3=C&x4=D&x5=E&y1=1&y2=2&y3=3&y4=4&y5=2
#
#    上記のリクエストの場合、以下のような JSON で渡される
#
# { data:             ← 固定
#     [
#         'A',       ← 以下、クエリー引数の値が配列で格納されている
#         'B',
#         'C',
#         'D',
#         'E',
#         '1',
#         '2',
#         '3',
#         '4',
#         '2'
#     ]
# }
jsonData = json.loads(sys.stdin.readline())
chartDatas = jsonData['data']
x = []
y = []
x = x + [chartDatas[0]]
x = x + [chartDatas[1]]
x = x + [chartDatas[2]]
x = x + [chartDatas[3]]
x = x + [chartDatas[4]]
y = y + [float(chartDatas[5])]
y = y + [float(chartDatas[6])]
y = y + [float(chartDatas[7])]
y = y + [float(chartDatas[8])]
y = y + [float(chartDatas[9])]

plt.plot(x, y)

#被らないファイル名で画像を一時保存
fileId = uuid.uuid4()
plt.savefig(fileId.hex + '.png')

#画像ファイルを base64 文字列に encode
file = open(fileId.hex + '.png', 'rb').read()
enc_file = base64.b64encode( file ).decode('utf-8')
os.remove(fileId.hex + '.png')

#JSON で返す
```

```
    mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 1,"cluster": 43,"type": "image","value" :
enc_file}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

6. Python スクリプト連携（棒グラフ）

【ConMas Gateway】サンプル定義を利用して Python スクリプト連携で棒グラフを表示する方法について説明します。

Fig.6-6-1 Python スクリプト連携（棒グラフ）

【動作概要】

1. 【投入数】列の数値を入力します。
2. 【グラフ取得】アクションクラスターを起動します。
3. Python スクリプトで作成された棒グラフが表示されます。

【アクションクラスター設定】

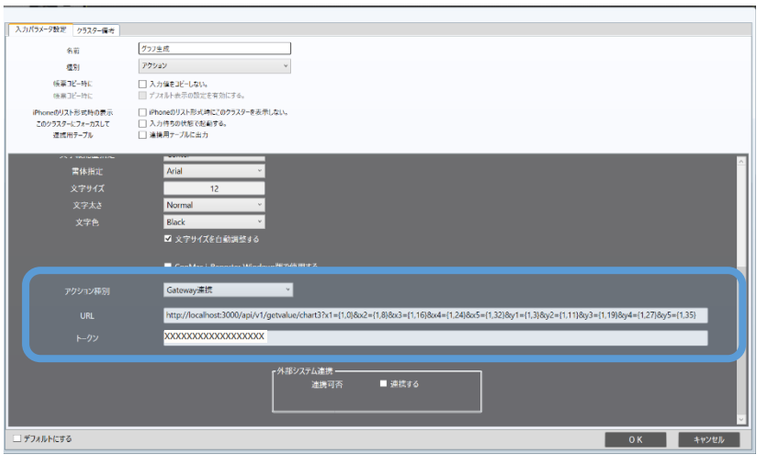


Fig.6-6-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：

1　　2

http://localhost:3000/api/v1/getvalue/chart3?x1={1,0}&x2={1,8}&x3={1,16}&x4={1,24}&x5={1,32}&y1={1,3}&y2={1,11}&y3={1,19}&y4={1,27}&y5={1,35}

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart3.json を呼び出します。
2. x1 という変数にシート番号:1、クラスターID:0 のクラスター値
   x2 という変数にシート番号:1、クラスターID:8 のクラスター値
   x3 という変数にシート番号:1、クラスターID:16 のクラスター値
   x4 という変数にシート番号:1、クラスターID:24 のクラスター値
   x5 という変数にシート番号:1、クラスターID:32 のクラスター値
   y1 という変数にシート番号:1、クラスターID:3 のクラスター値
   y2 という変数にシート番号:1、クラスターID:11 のクラスター値
   y3 という変数にシート番号:1、クラスターID:19 のクラスター値
   x4 という変数にシート番号:1、クラスターID:27 のクラスター値
   x5 という変数にシート番号:1、クラスターID:35 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【投入数】列の各クラスター値を Python スクリプトへ渡しグラフが表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥chart3.json)】

```
{
  "datasource": "script ",
  "script": " scripts/bar.py",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **bar.py** が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥bar.py)】

```
import sys
import matplotlib.pyplot as plt
import uuid
import json
import base64
import os

try:
  #引数を処理
  #
  # ex)
  #   http://localhost:3000/api/v1/getvalue/chart1?x1=A&x2=B&x3=C&x4=D&x5=E&y1=1&y2=2&y3=3&y4=4&y5=2
```

```
#
#   上記のリクエストの場合、以下のような JSON で渡される
#
# { data:             ← 固定
#    [
#       'A',      ← 以下、クエリー引数の値が配列で格納されている
#       'B',
#       'C',
#       'D',
#       'E',
#       '1',
#       '2',
#       '3',
#       '4',
#       '2'
#    ]
# }
jsonData = json.loads(sys.stdin.readline())
chartDatas = jsonData['data']
x = []
y = []
x = x + [chartDatas[0]]
x = x + [chartDatas[1]]
x = x + [chartDatas[2]]
x = x + [chartDatas[3]]
x = x + [chartDatas[4]]
y = y + [float(chartDatas[5])]
y = y + [float(chartDatas[6])]
y = y + [float(chartDatas[7])]
y = y + [float(chartDatas[8])]
y = y + [float(chartDatas[9])]

plt.bar(x, y)

#被らないファイル名で画像を一時保存
fileId = uuid.uuid4()
plt.savefig(fileId.hex + '.png')

#画像ファイルを base64 文字列に encode
file = open(fileId.hex + '.png', 'rb').read()
enc_file = base64.b64encode( file ).decode('utf-8')
os.remove(fileId.hex + '.png')

#JSON で返す
mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 1,"cluster": 44,"type": "image","value" :
```

```
enc_file}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

7. Python スクリプト連携（円グラフ）

【ConMas Gateway】サンプル定義を利用して Python スクリプト連携で円グラフを表示する方法について説明します。

Fig.6-7-1 Python スクリプト連携（円グラフ）

【動作概要】

1.【不良数】列の数値を入力します。

2.【グラフ取得】アクションクラスターを起動します。

3. Python スクリプトで作成された円グラフが表示されます。

【アクションクラスター設定】

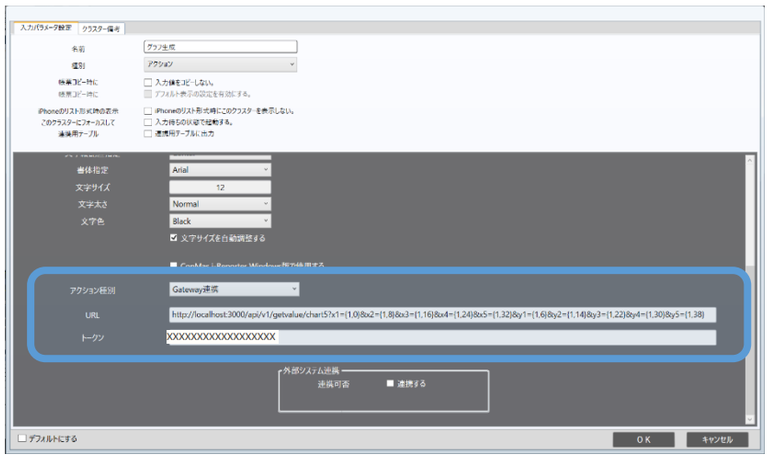


Fig.6-7-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/chart5?x1={1,0}&x2={1,8}&x3={1,16}&x4={1,24}&x5={1,32}&y1={1,6}&y2={1,14}&y3={1,22}&y4={1,30}&y5={1,38}

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart5.json を呼び出します。
2. x1 という変数にシート番号:1、クラスターID:0 のクラスター値
   x2 という変数にシート番号:1、クラスターID:8 のクラスター値
   x3 という変数にシート番号:1、クラスターID:16 のクラスター値
   x4 という変数にシート番号:1、クラスターID:24 のクラスター値
   x5 という変数にシート番号:1、クラスターID:32 のクラスター値
   y1 という変数にシート番号:1、クラスターID:6 のクラスター値
   y2 という変数にシート番号:1、クラスターID:14 のクラスター値
   y3 という変数にシート番号:1、クラスターID:22 のクラスター値
   x4 という変数にシート番号:1、クラスターID:30 のクラスター値
   x5 という変数にシート番号:1、クラスターID:38 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【不良数】列の各クラスター値を Python スクリプトへ渡しグラフが表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥chart5.json)】

```
{
   "datasource": "script ",
   "script": " scripts/pie.py",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **pie.py** が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥pie.py)】

```
import sys
import matplotlib.pyplot as plt
import uuid
import json
import base64
import os

try:
  #引数を処理
  #
  # ex)
  #   http://localhost:3000/api/v1/getvalue/chart1?x1=A&x2=B&x3=C&x4=D&x5=E&y1=1&y2=2&y3=3&y4=4&y5=2
```

```
#
#   上記のリクエストの場合、以下のような JSON で渡される
#
# { data:             ← 固定
#    [
#       'A',      ← 以下、クエリー引数の値が配列で格納されている
#       'B',
#       'C',
#       'D',
#       'E',
#       '1',
#       '2',
#       '3',
#       '4',
#       '2'
#    ]
# }
jsonData = json.loads(sys.stdin.readline())
chartDatas = jsonData['data']
x = []
y = []
x = x + [chartDatas[0]]
x = x + [chartDatas[1]]
x = x + [chartDatas[2]]
x = x + [chartDatas[3]]
x = x + [chartDatas[4]]
y = y + [float(chartDatas[5])]
y = y + [float(chartDatas[6])]
y = y + [float(chartDatas[7])]
y = y + [float(chartDatas[8])]
y = y + [float(chartDatas[9])]

plt.pie(y, labels=x)

#被らないファイル名で画像を一時保存
fileId = uuid.uuid4()
plt.savefig(fileId.hex + '.png')

#画像ファイルを base64 文字列に encode
file = open(fileId.hex + '.png', 'rb').read()
enc_file = base64.b64encode( file ).decode('utf-8')
os.remove(fileId.hex + '.png')

#JSON で返す
mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 1,"cluster": 45,"type": "image","value" :
```

```
enc_file}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

8. Python スクリプト連携（QR コード）

【ConMas Gateway】サンプル定義を利用して Python スクリプト連携で QR コードを表示する方法について説明します。

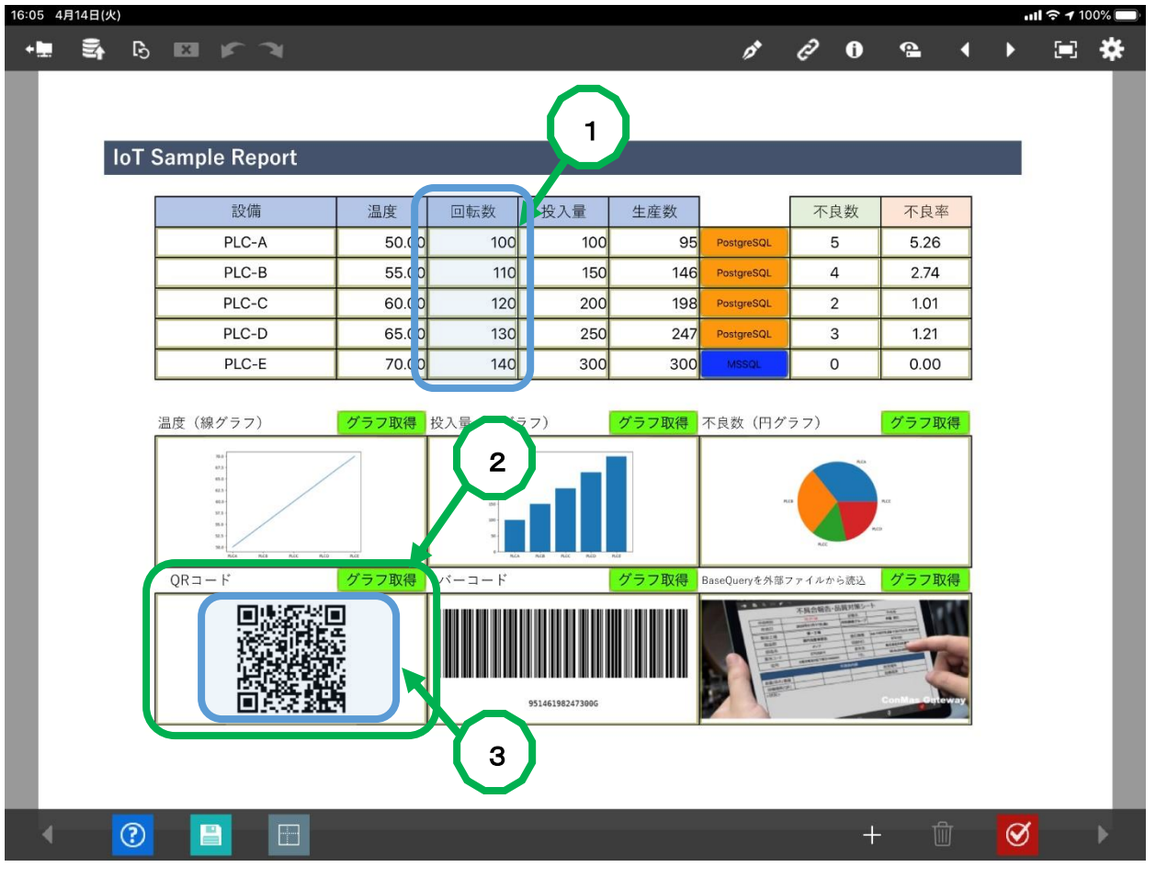


Fig.6-8-1 Python スクリプト連携（QR コード）

【動作概要】

1. 【回転数】列の数値を入力します。
2. 【グラフ取得】アクションクラスターを起動します。
3. Python スクリプトで作成された QR コードが表示されます。

【アクションクラスター設定】

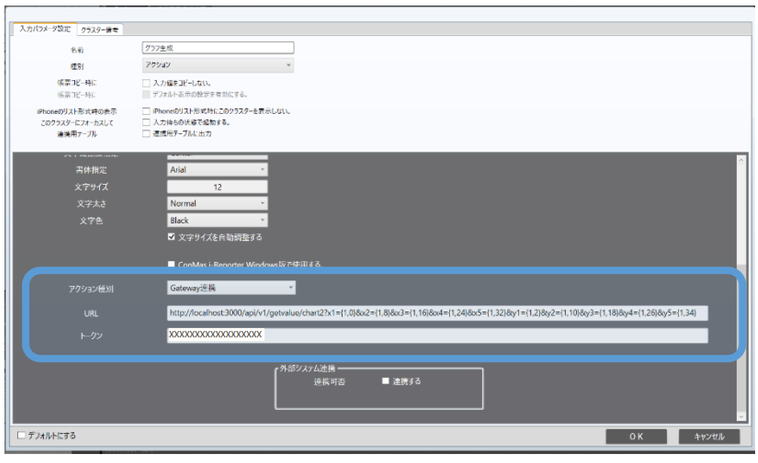


Fig.6-8-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/chart2?x1={1,0}&x2={1,8}&x3={1,16}&x4={1,24}&x5={1,32}&y1={1,2}&y2={1,10}&y3={1,18}&y4={1,26}&y5={1,34}

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart2.json を呼び出します。
2. x1 という変数にシート番号:1、クラスターID:0 のクラスター値
   x2 という変数にシート番号:1、クラスターID:8 のクラスター値
   x3 という変数にシート番号:1、クラスターID:16 のクラスター値
   x4 という変数にシート番号:1、クラスターID:24 のクラスター値
   x5 という変数にシート番号:1、クラスターID:32 のクラスター値
   y1 という変数にシート番号:1、クラスターID:2 のクラスター値
   y2 という変数にシート番号:1、クラスターID:10 のクラスター値
   y3 という変数にシート番号:1、クラスターID:18 のクラスター値
   x4 という変数にシート番号:1、クラスターID:26 のクラスター値
   x5 という変数にシート番号:1、クラスターID:34 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【回転数】列の各クラスター値を Python スクリプトへ QR コードが表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥chart2.json)】

```
{
   "datasource": "script ",
   "script": " scripts/qr.py",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の qr.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥qr.py)】

```
import sys
import qrcode
import uuid
import json
import base64
import os

try:

  img = qrcode.make(sys.stdin.readline())

  fileId = uuid.uuid4()
```

```
    img.save(fileId.hex + '.png')


    #画像ファイルを base64 文字列に encode
    file = open(fileId.hex + '.png', 'rb').read()
    enc_file = base64.b64encode( file ).decode('utf-8')
    os.remove(fileId.hex + '.png')

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 1,"cluster": 49,"type": "image","value" :
enc_file}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

9. Python スクリプト連携（バーコード）

【ConMas Gateway】サンプル定義を利用して Python スクリプト連携でバーコードを表示する方法について説明します。

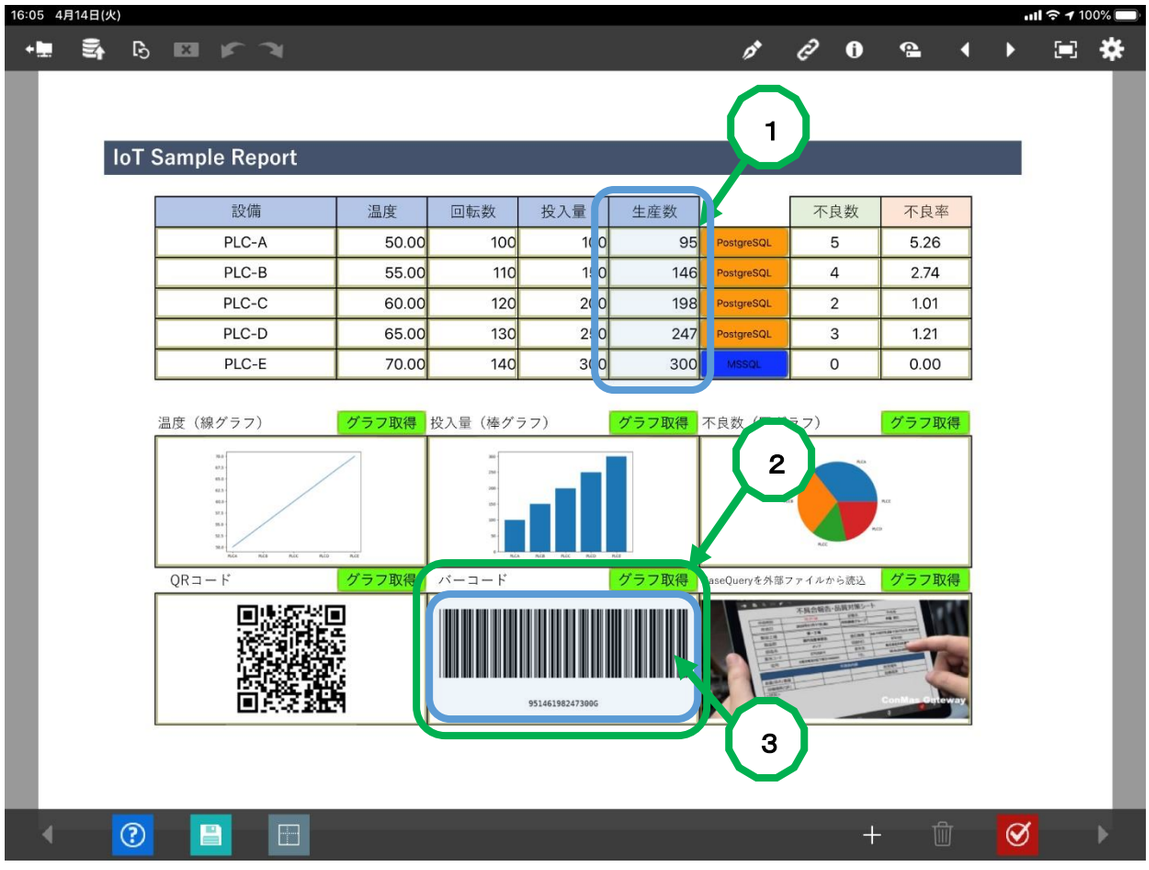


Fig.6-9-1 Python スクリプト連携（バーコード）

【動作概要】

1. 【回転数】列の数値を入力します。
2. 【グラフ取得】アクションクラスターを起動します。
3. Python スクリプトで作成されたバーコードが表示されます。

【アクションクラスター設定】

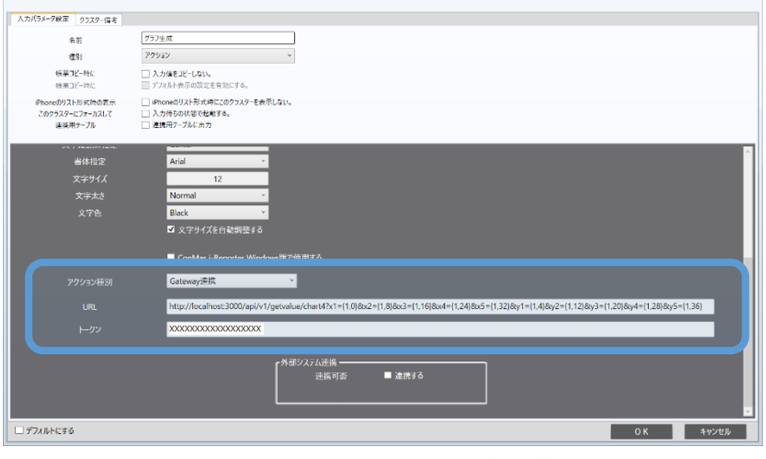


Fig.6-9-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/chart4?x1={1,0}&x2={1,8}&x3={1,16}&x4={1,24}&x5={1,32}&y1={1,4}&y2={1,12}&y3={1,20}&y4={1,28}&y5={1,36}

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart4.json を呼び出します。
2. x1 という変数にシート番号:1、クラスターID:0 のクラスター値
   x2 という変数にシート番号:1、クラスターID:8 のクラスター値
   x3 という変数にシート番号:1、クラスターID:16 のクラスター値
   x4 という変数にシート番号:1、クラスターID:24 のクラスター値
   x5 という変数にシート番号:1、クラスターID:32 のクラスター値
   y1 という変数にシート番号:1、クラスターID:4 のクラスター値
   y2 という変数にシート番号:1、クラスターID:12 のクラスター値
   y3 という変数にシート番号:1、クラスターID:20 のクラスター値
   x4 という変数にシート番号:1、クラスターID:28 のクラスター値
   x5 という変数にシート番号:1、クラスターID:36 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【生産数】列の各クラスター値を Python スクリプトへバーコードが表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥chart4.json)】

```
{
  "datasource": "script ",
  "script": " scripts/barcd.py",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **barcd.py** が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥barcd.py)】

```
import sys
import barcode
import uuid
import json
import base64
import os

try:
  #引数を処理
  #
  # ex)
  #   http://localhost:3000/api/v1/getvalue/chart1?x1=A&x2=B&x3=C&x4=D&x5=E&y1=1&y2=2&y3=3&y4=4&y5=2
```

```
#
#    上記のリクエストの場合、以下のような JSON で渡される
#
# { data:              ← 固定
#     [
#         'A',       ← 以下、クエリー引数の値が配列で格納されている
#         'B',
#         'C',
#         'D',
#         'E',
#         '1',
#         '2',
#         '3',
#         '4',
#         '2'
#     ]
# }

jsonData = json.loads(sys.stdin.readline())
chartDatas = jsonData['data']
y = ''
y = y + chartDatas[5]
y = y + chartDatas[6]
y = y + chartDatas[7]
y = y + chartDatas[8]
y = y + chartDatas[9]

CODE39 = barcode.get_barcode_class('code39')
from barcode.writer import ImageWriter
img = CODE39(y, writer=ImageWriter())

#被らないファイル名で画像を一時保存
fileId = uuid.uuid4()
fullname = img.save(fileId.hex)

#画像ファイルを base64 文字列に encode
file = open(fullname, 'rb').read()
enc_file = base64.b64encode( file ).decode('utf-8')
os.remove(fullname)

#JSON で返す
mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 1,"cluster": 50,"type": "image","value" :
enc_file}]}
print(json.dumps(mappings))
```

```
except Exception as e:
    mappings = {"error": "Python でエラー：" + str(e)}
    print(json.dumps(mappings))
```

10. i-Reporter 連携

【ConMas Gateway】サンプル定義を利用して BaseQuery を外部ファイルから読み込む機能と PostgreSQL の Bytea 型を画像データとして扱う機能（i-Reporter データベースにある他の帳票の画像を取得）の方法について説明します。

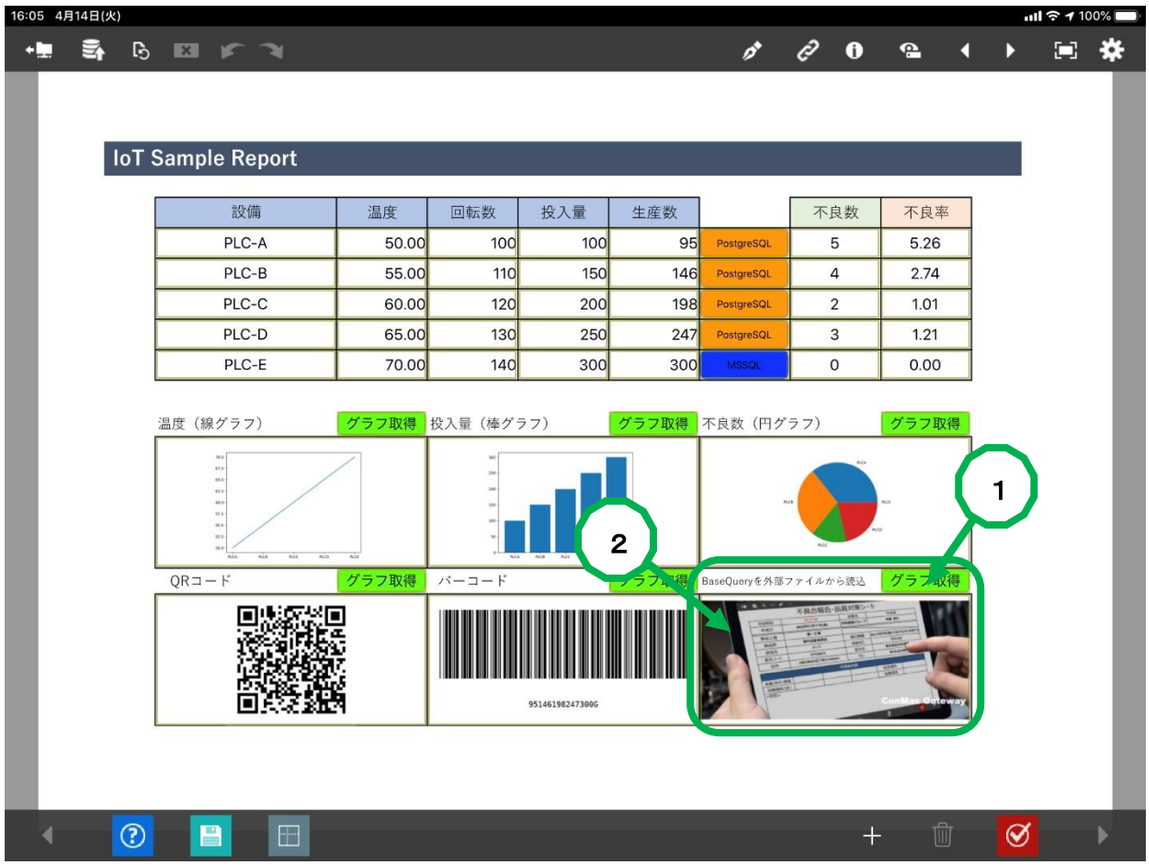


Fig.6-10-1 i-Reporter 連携

【動作概要】

1.【グラフ取得】アクションクラスターを起動します。

2. i-Reporter データベースにある他の帳票の画像を取得し表示します。

【アクションクラスター設定】

Fig.6-10-2 アクションクラスター設定

アクション種別： 【Gateway 連携】

URL： http://localhost:3000/api/v1/getvalue/chart6?

トークン： xxxxxxxxxxxxxxxxxx

1

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の chart6.json を呼び出します。

【アクションファイル設定（D:¥ConMas¥gateway2¥actions¥chart6.json）】

```
{
  "datasource": "irepodb",
  "basetype": "file",
  "basequery": "sql/image.sql",
  "where": "",
  "params": [
    { "name": "plc_id", "type": "string"}
  ],
  "mappings": [
    { "item": "image_file"       ,"sheet": 1,"cluster": 51,"type": "image","value" : ""}
  ]
}
```

default.json 内の**"irepodb"**項目の設定が呼び出されます

”file”で固定

D:¥ConMas¥gateway¥sql フォルダ内の **image.sql** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "irepodb",
   "type": "postgreSQL",
   "user": "postgres",
   "host": "localhost",
   "database": "irepodb",
   "password": "cimtops",
   "port": 5432
},
```

<---------- データソース名（任意の名前）**: i-Reporter DB** 連携

11. Python スクリプト連携（Excel）

Python スクリプト連携で Excel シート上のデータを検索し表示する方法について説明します。

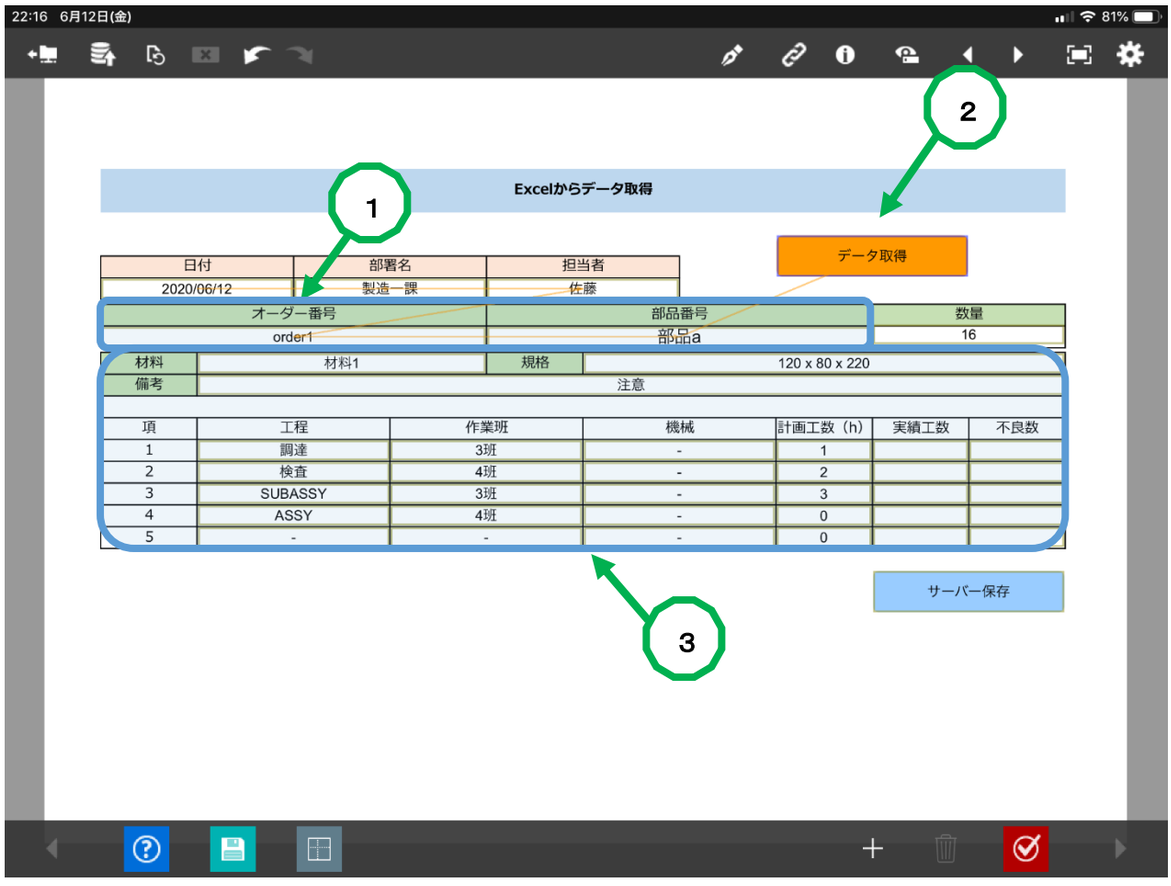


Fig.6-11-1 Python スクリプト連携（Excel）

【動作概要】

1.【オーダー番号】【部品番号】欄に値を入力します。

2.【データ取得】アクションクラスターを起動します。

3. Python スクリプトで Excel ファイル内の【オーダー番号】と同名のシートから、【部品番号】の入力値に相当する部品の行データを表示します。

【アクションクラスター設定】

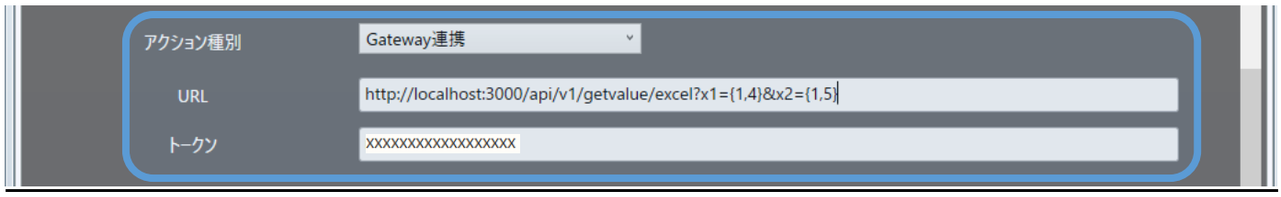


Fig.6-11-2 アクションクラスター設定

アクション種別：　【Gateway 連携】

URL：　http://localhost:3000/api/v1/getvalue/excel?order_no={1,4}&buhin_no={1,5}

トークン：　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の excel.json を呼び出します。

2. order_no という変数にシート番号：1、クラスターID：4 のクラスター値

buhin_no という変数にシート番号：1、クラスターID：5 のクラスター値
が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【オーダー番号】名称のシート内で A 列（部品番号）が【部品番号】の値に一致する値が帳票に表示されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥excel.json）】

```
{
   "datasource": "script ",
   "script": " scripts/excel.py",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **excel.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<-----------データソース名（任意の名前）：**Python** スクリプト連携

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥excel.py）】

```
import sys
import json
import os
import uuid
import shutil
from pandas import DataFrame, Series, ExcelFile

try:

  jsonData = json.loads(sys.stdin.readline())
  reqdata = jsonData['data']

  # ========================================
  #  読み込ファイル情報
  # ========================================

  #元データのファイル path
  filepath ="ex/"
  #対象ファイル
  exfile = "(demo_sample)工程表.xlsx"
  #テンポラリーディレクトリ
  tempdir = "ex/temp_data/"
  #読み込み先シート
  readsheet = reqdata[0]
```

```
#被らない名前で対象ファイルをコピー
fileId = uuid.uuid4()
tempfile = shutil.copy(filepath + exfile,tempdir + fileId.hex + exfile)

# ================================================
# Excel データ取得
# ================================================
def isExcelFilePath(filepath: str) -> bool:
    return (filepath.endswith('.xlsx')
        or filepath.endswith('.xls'))

def getExcelFile(filepath: str) -> ExcelFile:
    if (not os.path.exists(filepath)
        or (not isExcelFilePath(filepath))):
            return None
    return ExcelFile(filepath)

def getDataFromExcelFile(excelFile: ExcelFile) -> DataFrame:
    return excelFile.parse(
        sheet_name = reqdata[0],
        index_col=None,
        skiprows=1,
        header=None
    )

dataFrame = getDataFromExcelFile(
    getExcelFile(tempfile)
)
# ========================================
#  指定する部品名の行データを抽出
# ========================================
partsid = reqdata[1]

#  検索するカラムを列番号で指定し対象の行を取得
search_data = dataFrame[dataFrame[0] == partsid]

# index を振りなおす
i = search_data.reset_index(drop=True)
resdata = i.at

#  各データを(列,行)で指定し SON で返す
mappings = {"error": "", "mappings":[
    { "item": "chart1","sheet": 1,"cluster": 6    ,"type" : "string","value" : str(resdata[0,3])}#数量
    ,{ "item": "chart1","sheet": 1,"cluster": 7    ,"type" : "string","value" : str(resdata[0,1])}#材料
```

```
    ,{ "item": "chart1","sheet": 1,"cluster": 8    ,"type" : "string","value" : str(resdata[0,2])}#規格
    ,{ "item": "chart1","sheet": 1,"cluster": 9    ,"type" : "string","value" : str(resdata[0,24])}#備考

    #工程 1
    ,{ "item": "chart1","sheet": 1,"cluster": 10 ,"type" : "string","value" : str(resdata[0,4])}#工程
    ,{ "item": "chart1","sheet": 1,"cluster": 11 ,"type" : "string","value" : str(resdata[0,5])}#作業班
    ,{ "item": "chart1","sheet": 1,"cluster": 12 ,"type" : "string","value" : str(resdata[0,6])}#機械
    ,{ "item": "chart1","sheet": 1,"cluster": 13 ,"type" : "string","value" : str(resdata[0,7])}#工数

    #工程 2
    ,{ "item": "chart1","sheet": 1,"cluster": 16 ,"type" : "string","value" : str(resdata[0,8])}#工程
    ,{ "item": "chart1","sheet": 1,"cluster": 17 ,"type" : "string","value" : str(resdata[0,9])}#作業班
    ,{ "item": "chart1","sheet": 1,"cluster": 18 ,"type" : "string","value" : str(resdata[0,10])}#機械
    ,{ "item": "chart1","sheet": 1,"cluster": 19 ,"type" : "string","value" : str(resdata[0,11])}#工数

    #工程 3
    ,{ "item": "chart1","sheet": 1,"cluster": 22 ,"type" : "string","value" : str(resdata[0,12])}#工程
    ,{ "item": "chart1","sheet": 1,"cluster": 23 ,"type" : "string","value" : str(resdata[0,13])}#作業班
    ,{ "item": "chart1","sheet": 1,"cluster": 24 ,"type" : "string","value" : str(resdata[0,14])}#機械
    ,{ "item": "chart1","sheet": 1,"cluster": 25 ,"type" : "string","value" : str(resdata[0,15])}#工数

    #工程 4
    ,{ "item": "chart1","sheet": 1,"cluster": 28 ,"type" : "string","value" : str(resdata[0,16])}#工程
    ,{ "item": "chart1","sheet": 1,"cluster": 29 ,"type" : "string","value" : str(resdata[0,17])}#作業班
    ,{ "item": "chart1","sheet": 1,"cluster": 30 ,"type" : "string","value" : str(resdata[0,18])}#機械
    ,{ "item": "chart1","sheet": 1,"cluster": 31 ,"type" : "string","value" : str(resdata[0,19])}#工数

    #工程 5
    ,{ "item": "chart1","sheet": 1,"cluster": 34 ,"type" : "string","value" : str(resdata[0,20])}#工程
    ,{ "item": "chart1","sheet": 1,"cluster": 35 ,"type" : "string","value" : str(resdata[0,21])}#作業班
    ,{ "item": "chart1","sheet": 1,"cluster": 36 ,"type" : "string","value" : str(resdata[0,22])}#機械
    ,{ "item": "chart1","sheet": 1,"cluster": 37 ,"type" : "string","value" : str(resdata[0,23])}#工数

  ]}
  print(json.dumps(mappings))

  #コピーしたファイルを削除
  os.remove(tempfile)

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

12. Python スクリプト連携（Kintone）

Kintone アプリケーションに対して Python スクリプト連携により検索コマンドを発行し、検索結果の行データを表示する方法について説明します。

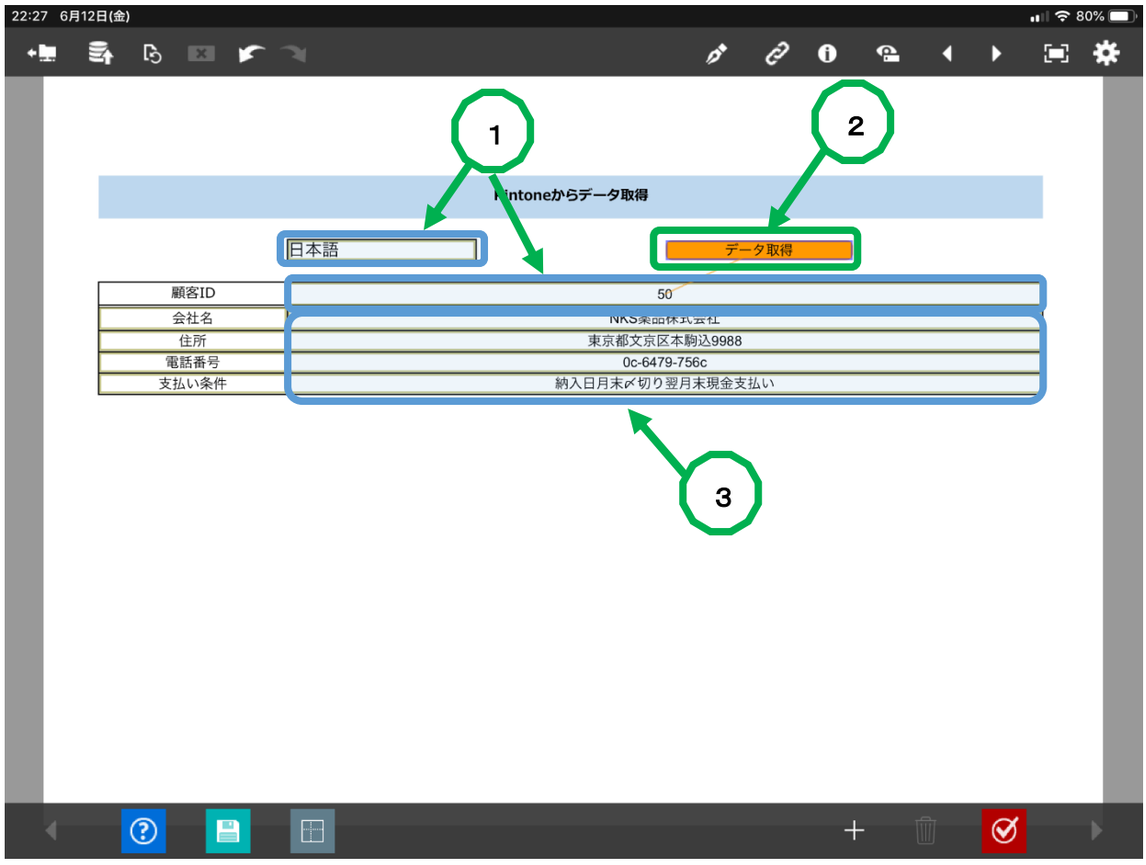


Fig.6-12-1 Python スクリプト連携（Kintone）

【動作概要】

1. 【言語】と【顧客 ID】を入力します。
2. 【データ取得】アクションクラスターを起動します。
3. Python スクリプトで取得した顧客情報が指定した言語で表示されます。

【アクションクラスター設定】

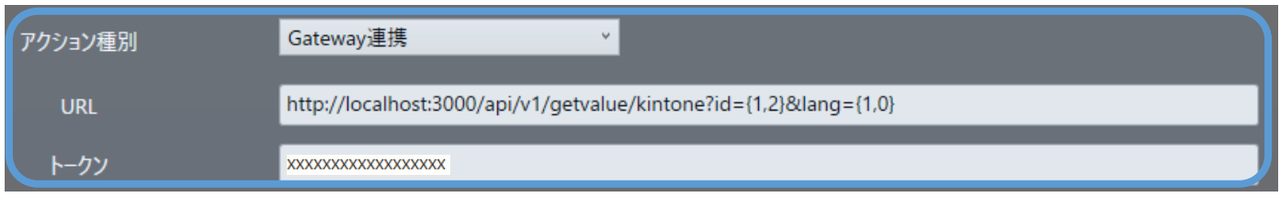


Fig.6-12-2 アクションクラスター

アクション種別：　【Gateway 連携】

URL：　http://localhost:3000/api/v1/getvalue/kintone?id={1,2}&lang={1,0}

トークン：　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の kintone.json を呼び出します。
2. id という変数にシート番号：1、クラスターID：2 のクラスター値

lang という変数にシート番号：1、クラスターID：0 のクラスター値
が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【生産数】列の各クラスター値を Python スクリプトへバーコードが表示されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥kintone.json）】

```
{
   "datasource": "script ",
   "script": " scripts/kintone.py",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の kintone.py が

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<----------データソース名（任意の名前）：Python スクリプト連携

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥kintone.py）】

```
import sys
import requests
import json

try:

   jsonData = json.loads(sys.stdin.readline())
   reqdata = jsonData['data']

   #Kintone 接続先情報
   domain = "AAAAAA"
   app = 123
   record = reqdata[0]
   API_TOKEN    = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
   URL = "https://%s.cybozu.com/k/v1/record.json?app=%s&id=%s"%(domain,app,record)
   headers = {"X-Cybozu-API-Token": API_TOKEN}

   #Kintone からデータ取得する関数
   def get_kintone(url, api_token):
      resp = requests.get(url, headers=headers)
      return resp

   #データ取得
   data = get_kintone(URL, API_TOKEN).json()
   resp_data = data["record"]
```

```
    #mapping
    if reqdata[1] == "日本語":
        mappings = {"error": "", "mappings":[
            { "item": "chart1","sheet": 1,"cluster": 4    ,"type" : "string","value" : resp_data["フィールドコード 1"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 6    ,"type" : "string","value" : resp_data["フィールドコード 2"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 8    ,"type" : "string","value" : resp_data["フィールドコード 3"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 10 ,"type" : "string","value" : resp_data["フィールドコード 4"]["value"]}
        ]}
        print(json.dumps(mappings))
    else:
        mappings = {"error": "", "mappings":[
            { "item": "chart1","sheet": 1,"cluster": 4    ,"type" : "string","value" : resp_data["フィールドコード 5"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 6    ,"type" : "string","value" : resp_data["フィールドコード 6"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 8    ,"type" : "string","value" : resp_data["フィールドコード 7"]["value"]}
            ,{ "item": "chart1","sheet": 1,"cluster": 10 ,"type" : "string","value" : resp_data["フィールドコード 8"]["value"]}
        ]}
        print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

13.　Python スクリプト連携（POST リクエスト用サンプル）

POST リクエスト用サンプル定義を利用して Python スクリプト連携で帳票内のクラスター値を POST リクエストし Python スクリプト内で処理する方法について説明します。

GET リクエストの URL 文字数制限の解除、セキュリティー強化、画像クラスターや大量データの送信が i-Reporter から可能になります。

POST リクエストで受信したデータを Python スクリプトへ連携することにより、外部データソースとの連携で画像クラスターや大量データの利用が可能になりました。

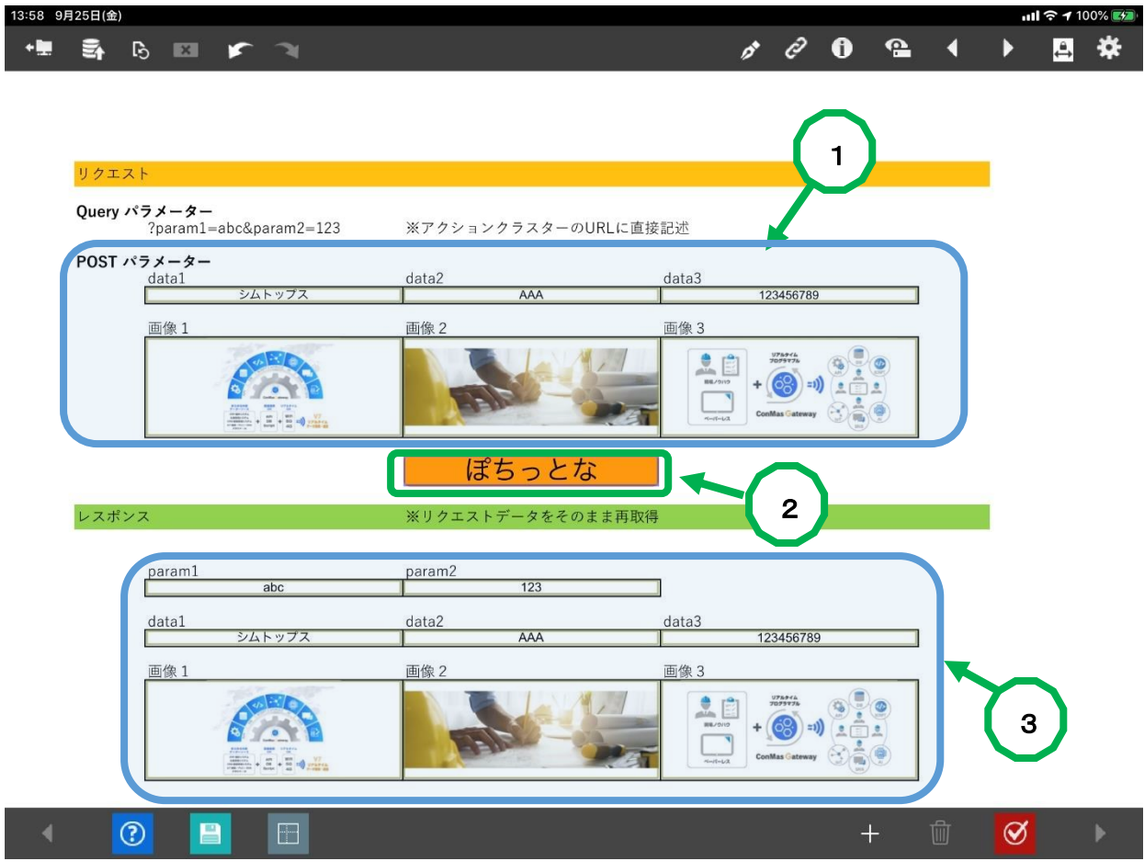


Fig.6-13-1 Python スクリプト連携（POST リクエスト用サンプル）

【動作概要】

1. Python スクリプトに渡すクラスターを入力します。
2. アクションクラスターを起動します。
3. 1で入力したクラスター値が Python スクリプトを経由し、そのままレスポンス用のクラスターに表示されます。

【アクションクラスター設定】

Fig.6-13-2 アクションクラスター設定

アクション種別: 【Gateway 連携】

メソッド: 【POST】

URL: http://localhost:3000/api/v1/upload/testpost?param1=abc&param2=123

トークン: xxxxxxxxxxxxxxxxx

POST 設定: POST リクエストを行うクラスターのシート番号、クラスターID を指定します。

POST 設定で指定可能なクラスター種別は、アクション、ピン打ち、ピン No.配置、ピン No.、承認、査閲、作成以外になります。

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の testpost.json を呼び出します。
2. param1 という変数に abc という値、param2 に 123 という値が渡されます。

URL 内に記述された変数は GET リクエストされます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥testpost.json）】

```
{
  "datasource": "script ",
  "script": " scripts/post.py",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）:Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post.py）】

```
import sys
import matplotlib.pyplot as plt
import uuid
import json
import base64
import os

try:
   jsonData = json.loads(sys.stdin.readline())
   query = jsonData['query']
   post = jsonData['post']
   path = jsonData['path']

   # file = open(path + post["clusters"][3]["value"], 'rb').read()
   # enc_file1 = base64.b64encode( file ).decode('utf-8')
   # file.close()

   # file = open(path + post["clusters"][4]["value"], 'rb').read()
   # enc_file2 = base64.b64encode( file ).decode('utf-8')
   # file.close()
   file = open(path + "/" + post["clusters"][3]["value"], 'rb').read()
   enc_file1 = base64.b64encode( file ).decode('utf-8')

   file = open(path + "/" + post["clusters"][4]["value"], 'rb').read()
   enc_file2 = base64.b64encode( file ).decode('utf-8')

   file = open(path + "/" + post["clusters"][5]["value"], 'rb').read()
   enc_file3 = base64.b64encode( file ).decode('utf-8')
   # file.close()
   # enc_file3 = path + "/" + post["clusters"][5]["value"]

   mappings = {"error": "", "mappings": [
      { "item": "chart1","sheet": 1,"cluster":   7,"type": "string","value":query["param1"]},
      { "item": "chart1","sheet": 1,"cluster":   8,"type": "string","value":query["param2"]},
      { "item": "chart1","sheet": 1,"cluster":   9,"type": "string","value":post["clusters"][0]["value"]},
      { "item": "chart1","sheet": 1,"cluster": 10,"type": "string","value":post["clusters"][1]["value"]},
      { "item": "chart1","sheet": 1,"cluster": 11,"type": "string","value":post["clusters"][2]["value"]},
      { "item": "chart1","sheet": 1,"cluster": 12,"type": "image","value":enc_file1},
      { "item": "chart1","sheet": 1,"cluster": 13,"type": "image","value":enc_file2},
      { "item": "chart1","sheet": 1,"cluster": 14,"type": "image","value":enc_file3}
      ]}
   print(json.dumps(mappings))

except Exception as e:
   #標準出力 JSON で返す（エラー）
   mappings = {"error": "Python でエラー:" + str(e)}
   print(json.dumps(mappings))
```

【アプリからの POST データ例】

```
{
  "query": {                                  <----------------------クエリストリングの情報
    "param1": "abc",                          <----------------------パラメータ名1:値
    "param2": "123"                           <----------------------パラメータ名2:値
  },
  "post": {                                   <---------------------- POST 情報
    "userId": "takahashi",                    <----------------------ユーザーID
    "terminalId": "45BBBBF5-C502-4DD9-A606-D151BD461D58",<----------------------端末物理 ID
    "defTopOrg": "438",                       <----------------------元定義 ID
    "publicStatus": "2",                      <----------------------公開ステータス
    "updateUser": "",                         <----------------------最新更新ユーザー
    "updateUserName": "",                     <----------------------最終更新ユーザー名称
    "updateTime": "",                         <----------------------更新日時
    "appVersion": "8.1.23061",                <----------------------バージョン
    "remarksValues": {                        <----------------------備考情報の配列
      "remarksValue9": "",     <----------------------備考情報9
      "remarksValue5": "",     <----------------------備考情報5
      "remarksValue1": "",     <----------------------備考情報1
      "remarksValue6": "",     <----------------------備考情報6
      "remarksValue2": "",     <----------------------備考情報2
      "remarksValue10": "",    <----------------------備考情報10
      "remarksValue7": "",     <----------------------備考情報7
      "remarksValue3": "",     <----------------------備考情報3
      "remarksValue8": "",     <----------------------備考情報8
      "remarksValue4": ""      <----------------------備考情報4
    },
    "repTopName": "POST リクエスト用サンプル_192.168.11.250",<----------------------TOP 帳票名称
    "registUser": "",                         <----------------------初回登録ユーザー
    "os": "iOS",                             <----------------------  プラットフォーム
    "registUserName": "",                     <----------------------初回登録ユーザー名称
    "registTime": "",                         <----------------------初回登録日時
    "systemKeys": {                           <----------------------システムキーの配列
      "systemKey4": "",        <----------------------システムキー4
      "systemKey2": "",        <----------------------システムキー2
      "systemKey5": "",        <----------------------システムキー5
      "systemKey3": "",        <----------------------システムキー3
      "systemKey1": ""         <----------------------システムキー1
    },
    "defTopId": "1320",                       <----------------------定義 ID
    "repTopId": "0",                          <----------------------TOP 帳票 ID
    "defTopName": "POST リクエスト用サンプル_192.168.11.250",       <----------------------TOP 定義名称
    "remarksNames": {                         <----------------------備考名称の配列
      "remarksName1": "帳票備考1", <----------------------備考名称1
```

```
    "remarksName8": "帳票備考8",  <----------------------備考名称8
    "remarksName2": "帳票備考2",  <----------------------備考名称2
    "remarksName9": "帳票備考9",  <----------------------備考名称9
    "remarksName3": "帳票備考3",  <----------------------備考名称3
    "remarksName10": "帳票備考10",<----------------------備考名称10
    "remarksName4": "帳票備考4",  <----------------------備考名称4
    "remarksName5": "帳票備考5",  <----------------------備考名称5
    "remarksName6": "帳票備考6",  <----------------------備考名称6
    "remarksName7": "帳票備考7"   <----------------------備考名称7
},
"clusters": [                                  <----------------------クラスター情報の配列
  {
    "gps": {                                   <----------------------GPS 情報の配列
      "lat": "35.743941",  <----------------------緯度
      "lon": "139.937116",<----------------------経度
      "alt": "22.574715"   <----------------------高度（※Win アプリでは常に0となります）
    },
    "verified": "0",                           <----------------------確認済フラグ
    "clusterId": "0",                          <----------------------クラスターID
    "type": "KeyboardText",                    <----------------------クラスター種別
    "editUser": "takahashi",                   <----------------------編集ユーザーID
    "value": "ABC",                            <----------------------入力値
    "sheetNo": "1",                            <----------------------シート NO
    "remarksValues": {                         <----------------------備考情報の配列
      "remarksValue9": "",<----------------------備考情報9
      "remarksValue5": "",<----------------------備考情報5
      "remarksValue1": "",<----------------------備考情報1
      "remarksValue6": "",<----------------------備考情報6
      "remarksValue2": "",<----------------------備考情報2
      "remarksValue10": "",<----------------------備考情報10
      "remarksValue7": "",<----------------------備考情報7
      "remarksValue3": "",<----------------------備考情報3
      "remarksValue8": "",<----------------------備考情報8
      "remarksValue4": ""<----------------------備考情報4
    },
    "editUserName": "Masaru Takahashi",<----------------------編集ユーザー名
    "displayValue": "ABC",                    <----------------------表示用文字列
    "editTime": "2023/10/04 15:48:25",         <----------------------編集日時
    "name": "data1"                            <----------------------クラスター名称
  },
  {
    "gps": {
      "lat": "35.743941",
      "lon": "139.937116",
      "alt": "22.574715"
```

```
            },
            "verified": "0",
            "clusterId": "3",
            "type": "Image",
            "editUser": "takahashi",
            "value": "img_sheet_1_cluster_3.jpg",
            "sheetNo": "1",
            "remarksValues": {
               "remarksValue9": "",
               "remarksValue5": "",
               "remarksValue1": "",
               "remarksValue6": "",
               "remarksValue2": "",
               "remarksValue10": "",
               "remarksValue7": "",
               "remarksValue3": "",
               "remarksValue8": "",
               "remarksValue4": ""
            },
            "editUserName": "Masaru Takahashi",
            "displayValue": "",
            "editTime": "2023/10/04 15:48:47",
            "name": "画像1"
         },
      ],
      "userName": "Masaru Takahashi",                    <----------------------ユーザー名
      "editStatus": "0",                                  <----------------------編集ステータス
      "gps": {                                            <----------------------GPS 情報の配列
         "lat": "35.743941",                  <----------------------緯度
         "lon": "139.937116",                 <----------------------経度
         "alt": "22.574715"                   <----------------------高度（※Win アプリでは常に0となります）
      }
   }
   "path": "./uploads/CB84F90E-C0A6-47C5-AB04-0009C6C0B7F0"     <------------------  内部使用アップロードパス
}
```

14.　Python スクリプト連携（PostgreSQL からのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で PostgreSQL からのデータ取得・計算・書き込みを行う方法について説明します。

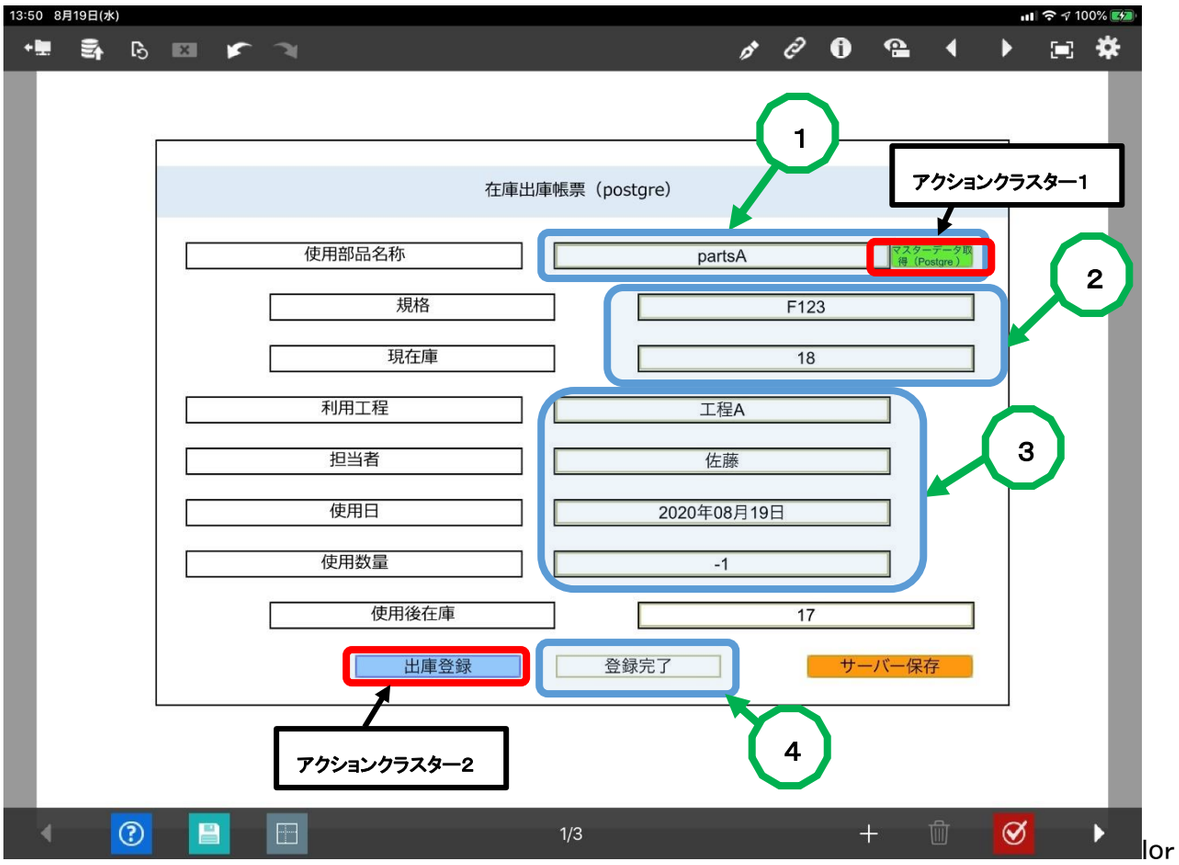


Fig.6-14-1 Python スクリプト連携（PostgreSQL からのデータ取得・計算・書き込み）

【動作概要】

1. 【使用部品名称】を選択し、Python スクリプトへ値を渡します。
2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれのテーブルからデータを取得します。
   - stock_mst
   - stock_trn

   stock_trn で抽出した複数行のデータを積算し stock_mst のデータを更に積算します。
   取得したデータ及び計算結果をそれぞれクラスターへ戻します。
3. 【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。
   書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量

   （※本サンプルでは上記3つ）
   stock_trn テーブルへ書き込む
4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

Fig.6-14-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　**http://localhost:3000/api/v1/getvalue/getstock_pg?parts_name={1,0}**

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_pg.json を呼び出します。
2. parts_name という変数にシート番号:1、クラスターID:0 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに各テーブルから検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_pg.json)】

```
{
   "datasource": "script ",
   "script": " scripts/ getstock_pg.py ",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_pg.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<----------データソース名(任意の名前):Python スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_pg.py)】

```
import sys
import json
import numpy
import psycopg2
from psycopg2.extras import DictCursor

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    # postgreSQL
    server = 'localhost'
    database = 'gwdb_stock'
    username = 'postgres'
    password = 'cimtops'
    port = '5432'
```

```
    # query
    Getmstquery = "SELECT stock_standard,stock_qua FROM stock_mst where parts_name ='%s'"%(reqdata[0],)
    Gettrnquery = "SELECT stock_qua FROM stock_trn where parts_name = '%s'"%(reqdata[0],)

    # get(dict)
    with psycopg2.connect('postgresql://'+username+':'+password+'@'+server+':'+port+'/'+database) as conn:
        with conn.cursor(cursor_factory=DictCursor) as cur:
            # get(mst)
            cur.execute(Getmstquery)
            rows_mst = cur.fetchall()[0]

            # get(trn)
            cur.execute(Gettrnquery)
            rows_trn = cur.fetchall()

    cur.close()
    conn.close()

    # トランザクションデータの数量を集計
    trnsum = numpy.sum(rows_trn)

    # マスターとトランザクションの数量の合計
    stock_plan = trnsum + rows_mst[1]

    # JSON で返す
    mappings = {"error": "", "mappings": [
        { "item": "chart1"      ,"sheet": 1,"cluster": 2,"type": "string","value" : str(rows_mst[0])},
        { "item": "chart1"      ,"sheet": 1,"cluster": 3,"type": "string","value" : str(stock_plan)}
        ]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

【アクションクラスター2　設定】

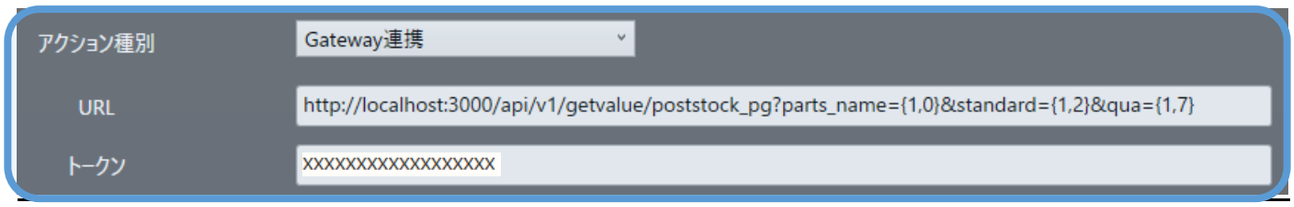


Fig.6-14-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

URL:

http://localhost:3000/api/v1/getvalue/poststock_pg?parts_name={1,0}&standard={1,2}&qua={1,7}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_pg.json を呼び出します。
2. parts_name という変数にシート番号:1、クラスターID:0 のクラスター値
   standard という変数にシート番号:1、クラスターID:2 のクラスター値
   qua という変数にシート番号:1、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称

- 規格
- 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥post_pg.json）】

```
{
   "datasource": "script ",
   "script": " scripts/post_pg.py ",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post_pg.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）:Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_pg.py）】

```
import numpy
import importlib
import json
import csv
import sys
import psycopg2
from psycopg2.extras import DictCursor

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    # postgreSQL
    server = 'localhost'
    database = 'gwdb_stock'
    username = 'postgres'
    password = 'cimtops'
    port = '5432'

    query = "INSERT INTO stock_trn    VALUES ('%s','%s','%s',true)"%(pdata[0],pdata[1],pdata[2])

    with psycopg2.connect('postgresql://'+username+':'+password+'@'+server+':'+port+'/'+database) as conn:
        with conn.cursor() as cur:
            # insert
            cur.execute(query)
            conn.commit()
    cur.close()
    conn.close()

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1","sheet": 1,"cluster": 10,"type": "string","value" : "登録完了
"}]}
    print(json.dumps(mappings))
```

```
except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

15.　Python スクリプト連携（MS SQL Server からのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で Microsoft SQL Server からのデータ取得・計算・書き込みを行う方法について説明します。

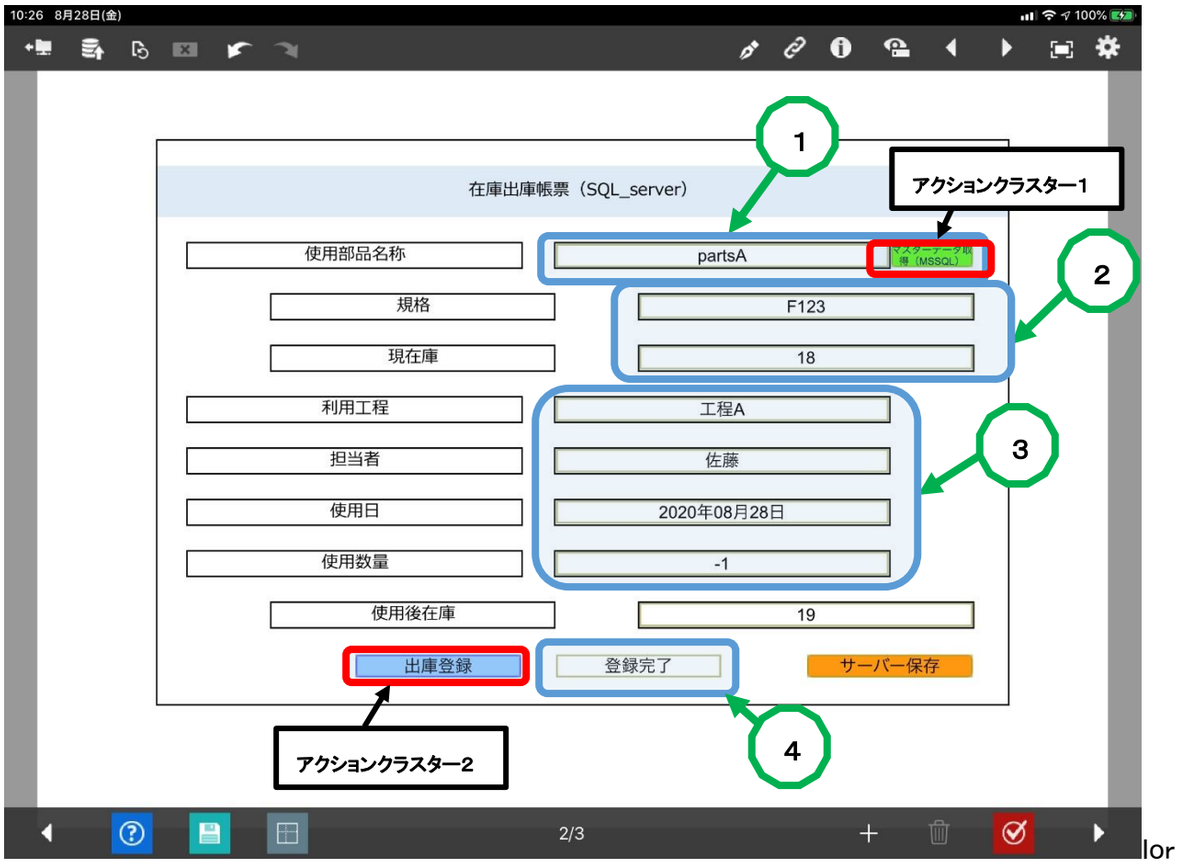


Fig.6-15-1 Python スクリプト連携（Microsoft SQL Server からのデータ取得・計算・書き込み）

【動作概要】

1.【使用部品名称】を選択し、Python スクリプトへ値を渡します。

2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれのテーブルからデータを取得します。

　- stock_mst

　- stock_trn

　stock_trn で抽出した複数行のデータを積算し stock_mst のデータを更に積算します。

　取得したデータ及び計算結果をそれぞれクラスターへ戻します。

3.【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。

　書き込むデータを python へ渡す

　- 使用部品名称

　- 規格

　- 使用数量

　（※本サンプルでは上記3つ）

　stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

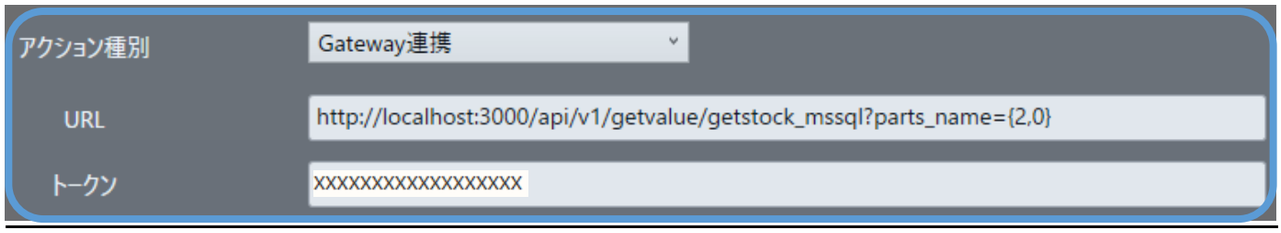


Fig.6-15-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http://localhost:3000/api/v1/getvalue/getstock_mssql?parts_name={2,0}

トークン:　xxxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_ mssql.json を呼び出します。
2. parts_name という変数にシート番号:2、クラスターID:0 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに各テーブルから検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_mssql.json)】

```
{
    "datasource": "script",
    "script": "scripts/getstock_mssql.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_mssql.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):**Python** スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_mssql.py)】

```
import sys
import json
import numpy
import pyodbc

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    # SQL Server
    server = 'localhost'
    database = 'gwdb_stock'
    username = 'sa'
    password = 'cimtops'

    # query
```

```
    Getmstquery = "SELECT stock_standard,stock_qua FROM stock_mst where parts_name ='%s'"%(reqdata[0],)
    Gettrnquery = "SELECT stock_qua FROM stock_trn where parts_name = '%s'"%(reqdata[0],)

    with pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password) as conn:
        with conn.cursor() as cur:
            # get(mst)
            cur.execute(Getmstquery)
            rows_mst = cur.fetchall()[0]

            # get(trn)
            cur.execute(Gettrnquery)
            rows_trn = cur.fetchall()

    cur.close()
    conn.close()

    # トランザクションデータの数量を集計
    trnsum = numpy.sum(rows_trn)

    # マスターとトランザクションの数量の合計
    stockplan = trnsum + rows_mst[1]

    # JSON で返す
    mappings = {"error": "", "mappings": [
        { "item": "chart1"      ,"sheet": 2,"cluster": 2,"type": "string","value" : str(rows_mst[0])},
        { "item": "chart1"      ,"sheet": 2,"cluster": 3,"type": "string","value" : str(stockplan)}
        ]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

【アクションクラスター2　設定】

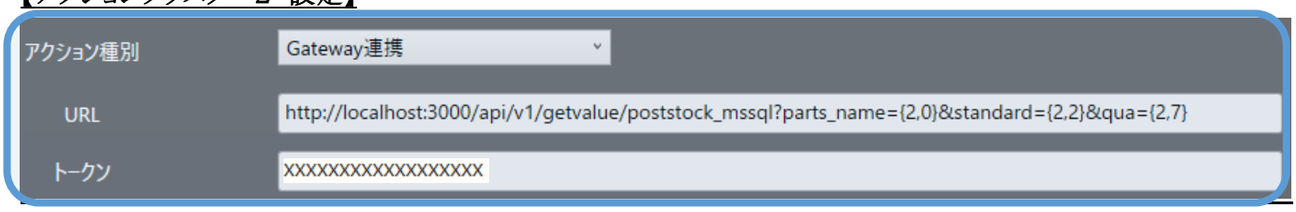


Fig.6-15-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

1　2

URL:http://localhost:3000/api/v1/getvalue/poststock_ mssql ?parts_name={2,0}&standard={2,2}&qua={2,7}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_mssql.json を呼び出します。
2. parts_name という変数にシート番号:2、クラスターID:0 のクラスター値
   standard という変数にシート番号:2、クラスターID:2 のクラスター値
   qua という変数にシート番号:2、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥post_mssql.json）】

```
{
   "datasource": "script ",
   "script" " scripts/post_mssql.py ",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post_mssql.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）:Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_mssql.py）】

```
import sys
import json
import pyodbc

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    # SQL Server
    server = 'localhost'
    database = 'gwdb_stock'
    username = 'sa'
    password = 'cimtops'

    # query
    query = "INSERT INTO stock_trn    VALUES ('%s','%s','%s','1',
CURRENT_TIMESTAMP)"%(pdata[0],pdata[1],pdata[2])

    with pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password) as conn:
        with conn.cursor() as cur:
            cur.execute(query)
            conn.commit()

    # JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": 2,"cluster": 10,"type": "string","value" : "登録
完了"}]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

16.　Python スクリプト連携（Excel からのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で Microsoft Excel からのデータ取得・計算・書き込みを行う方法について説明します。

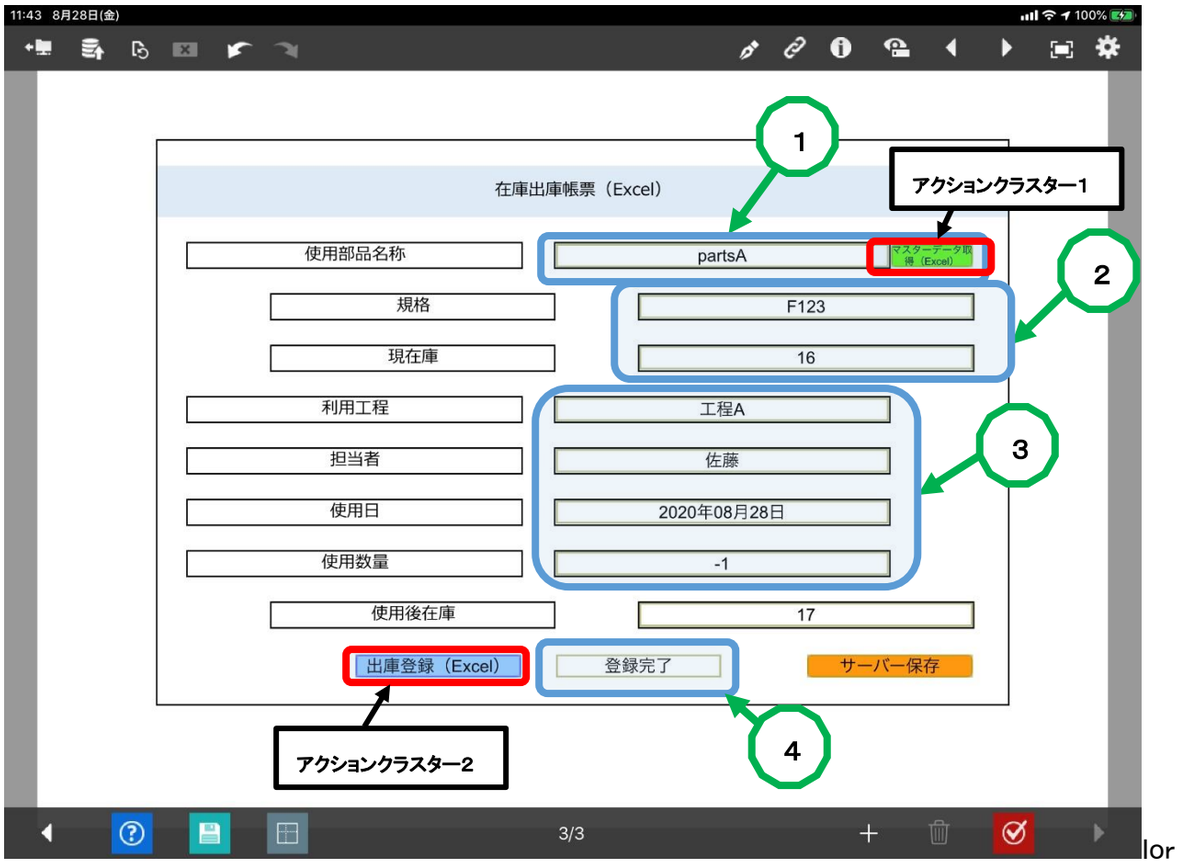


Fig.6-16-1 Python スクリプト連携（Microsoft Excel からのデータ取得・計算・書き込み）

【動作概要】

1. 【使用部品名称】を選択し、Python スクリプトへ値を渡します。
2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれの Excel からを取得します。

   - stock_mst.xlsx
   - stock_trn.xlsx

   stock_trn.xlsx で抽出した複数行のデータの指定列を積算し stock_mst.xlsx のデータを更に積算します。
   取得したデータ及び計算結果をそれぞれクラスターへ戻します。
3. 【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。
   書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量

   （※本サンプルでは上記3つ）
   stock_trn.xlsx へ書き込み日時を付与して Excel の最終行へ書き込む
4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

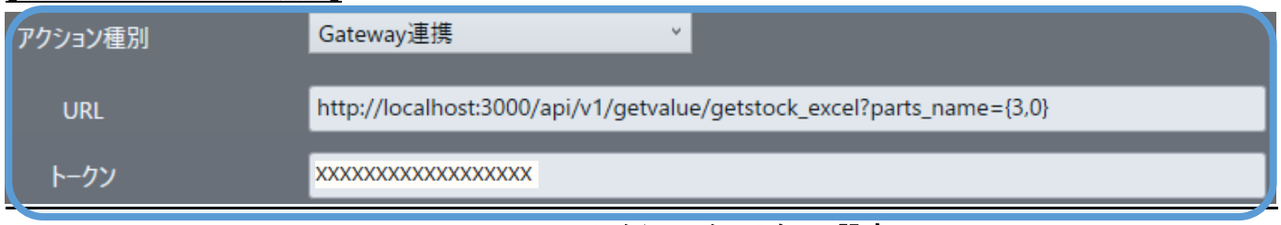


Fig.6-16-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http://localhost:3000/api/v1/getvalue/getstock_excel?parts_name={3,0}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_ excel.json を呼び出します。
2. parts_name という変数にシート番号:3、クラスターID:0 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに Excel から検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_excel.json)】

```
{
    "datasource": "script",
    "script": "scripts/getstock_excel.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_excel.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_excel.py)】

```
import os
import sys
import json
import uuid
import shutil
import openpyxl
import numpy as np

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    #------------------------------------------------------
    # 読み込み先の Excel と検索条件
    #------------------------------------------------------
```

```
filepath = "stock_ex/"
tempdir = "stock_ex/temp_data/"

# excel の book と shhet を指定
mstbook    = "stock_mst.xlsx"
mstsheet = "Sheet1"
trnbook    = "stock_trn.xlsx"
trnsheet = "Sheet1"

#  検索したい列を指定
i = 0

#  検索データ
reqdata = reqdata[0]

# excel からデータ取得する関数
def getfunc(book,sheet,reqdata):
    wb = openpyxl.load_workbook(book)
    ws = wb[sheet]

    # excel の 2 行目以降のデータを取得
    mstlist = []
    for row in ws.iter_rows(min_row=2):
        values = []
        for c in row:
            values.append(c.value)
        mstlist.append(tuple(values))

    #  検索する部品名
    partsid = reqdata

    #  検索
    resultparts = []
    for parts in mstlist:
        #  行番号を指定して列から検索
        if parts[i] in partsid:
            resultparts.append(parts)

    resdata = resultparts
    return resdata

#被らない名前で対象ファイルをコピー
fileId = uuid.uuid4()
tempmstbook = shutil.copy(filepath + mstbook,tempdir + fileId.hex + mstbook)
temptrnbook = shutil.copy(filepath + trnbook,tempdir + fileId.hex + trnbook)

# mstbook(stock_mst.xlsx)からデータ取得
mstdata = getfunc(tempmstbook,mstsheet,reqdata)

# trnbook(stock_trn.xlsx)からデータ取得
trndata = getfunc(temptrnbook,trnsheet,reqdata)

#-------------------------------------------------------
#  取得したデータの列を指定して数値を計算する
#-------------------------------------------------------

#  列抽出(計算する列を index で指定)
row = ((np.array(trndata))[:,2])

#  型変換
rowtype = list(map(lambda x: int(x) ,row))

#  計算(trn データの合計値と mst データの数量の和)
sumdata = np.sum(rowtype) + mstdata[0][2]

# JSON で返す
mappings = {"error": "", "mappings": [
    { "item": "chart1"       ,"sheet": 3,"cluster": 2,"type": "string","value" : str(mstdata[0][1])},
```

```
            { "item": "chart1"       ,"sheet": 3,"cluster": 3,"type": "string","value" : str(sumdata)}
            ]}
        print(json.dumps(mappings))
        os.remove(tempmstbook)
        os.remove(temptrnbook)

except Exception as e:
        mappings = {"error": "Python でエラー:" + str(e)}
        print(json.dumps(mappings))
```

【アクションクラスター2　設定】

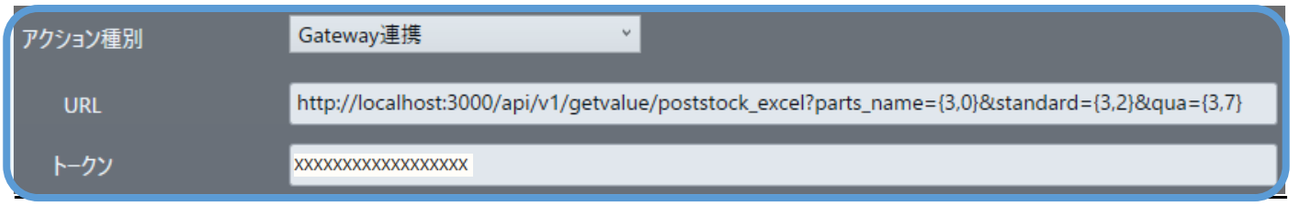


Fig.6-16-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

URL:http://localhost:3000/api/v1/getvalue/poststock_ excel ?parts_name={3,0}&standard={3,2}&qua={3,7}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_mssql.json を呼び出します。
2. parts_name という変数にシート番号:3、クラスターID:0 のクラスター値
   standard という変数にシート番号:3、クラスターID:2 のクラスター値
   qua という変数にシート番号:3、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量
   (※本サンプルでは上記3つ)
   stock_trn.xlsx へ書き込み日時を付与して Excel の最終行へ書き込む
4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥post_excel.json)】

```
{
   "datasource": "script ",
   "script" " scripts/post_excel.py ",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の post_excel.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
   "type": "python"
```

},

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_excel.py）】

```
import sys
import json
import openpyxl
import datetime

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    trnbook    = "stock_ex/stock_trn.xlsx"
    sheet = "Sheet1"

    wb = openpyxl.load_workbook(trnbook)
    ws = wb[sheet]

    # リストに現在日時を挿入
    writedata = pdata + [datetime.datetime.now()]

    ws.append(writedata)
    wb.save(trnbook)

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1","sheet": 3,"cluster": 10,"type": "string","value" : "登録完了
"}]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー：" + str(e)}
  print(json.dumps(mappings))
```

17.　Python スクリプト連携（Oracle からのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で Oracle からのデータ取得・計算・書き込みを行う方法について説明します。

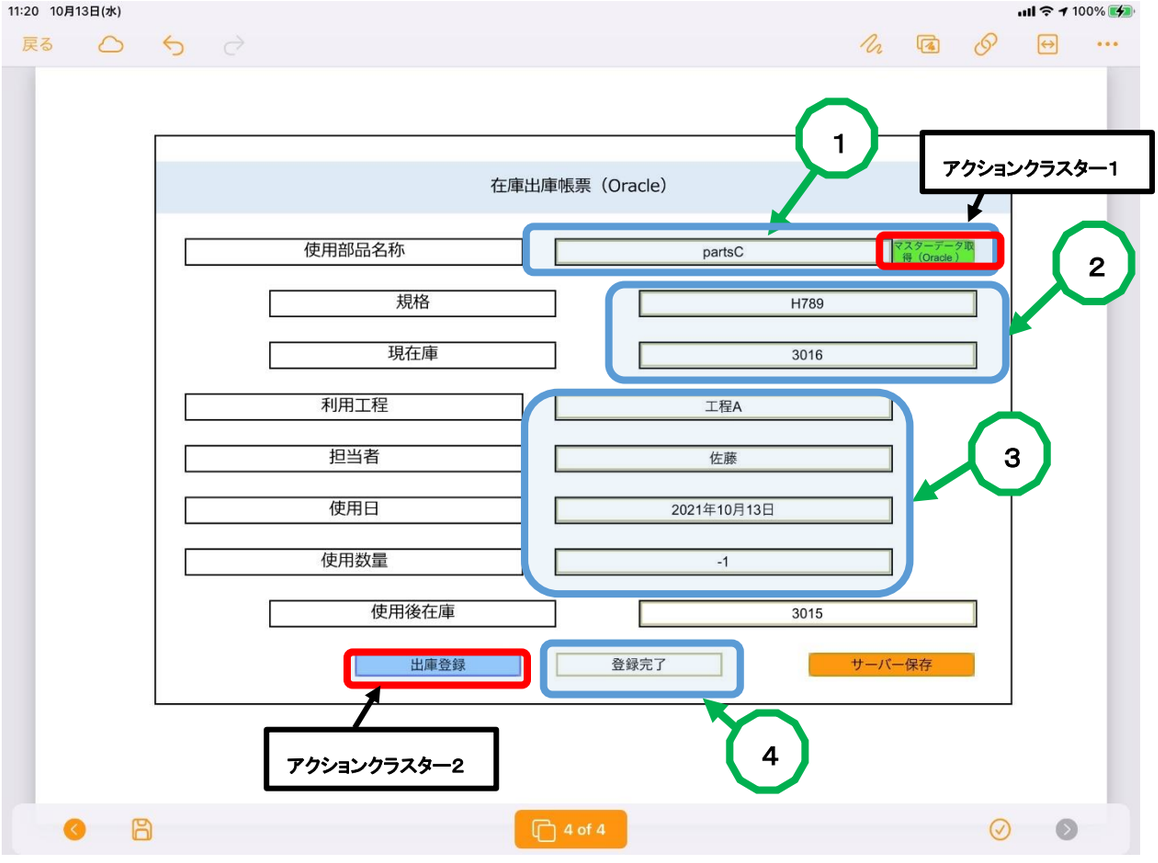


Fig.6-17-1 Python スクリプト連携（Oracle からのデータ取得・計算・書き込み）

【動作概要】

1.【使用部品名称】を選択し、Python スクリプトへ値を渡します。

2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれのテーブルからデータを取得します。

- stock_mst
- stock_trn

stock_trn で抽出した複数行のデータを積算し stock_mst のデータを更に積算します。

取得したデータ及び計算結果をそれぞれクラスターへ戻します。

3.【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。

書き込むデータを python へ渡す

- 使用部品名称
- 規格
- 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

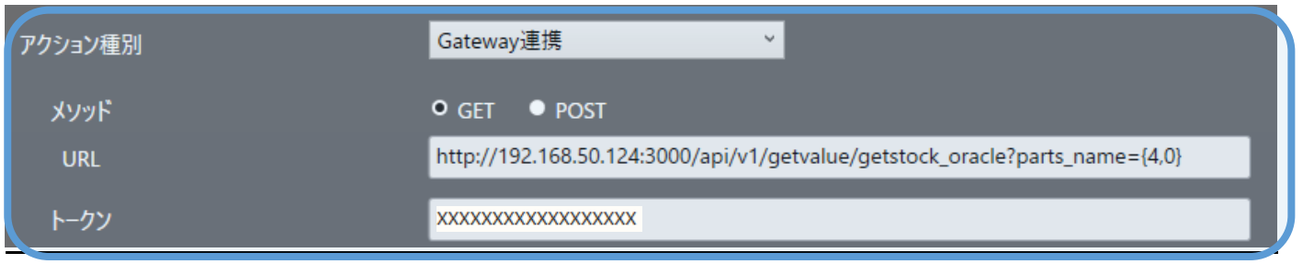


Fig.6-17-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http://192.168.50.124:3000/api/v1/getvalue/getstock_oracle?parts_name={4,0}

トークン:　xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_oracle.json を呼び出します。
2. parts_name という変数にシート番号:4、クラスターID:0 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに各テーブルから検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_oracle.json)】

```
{
    "datasource": "script",
    "script": "scripts/getstock_oracle.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_oracle.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):**Python** スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_oracle.py)】

```
import sys
import json
import numpy
import cx_Oracle

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']
    #Oracle 接続情報
    HOST = "192.168.50.125"
    PORT = 1521
    SVC_NM = "xe"
    USER = 'system'
    PASS = 'cimtops'
    #  接続記述子の生成
```

```
    dsn = cx_Oracle.makedsn(HOST, PORT, service_name = SVC_NM)
    # SQL 生成
    Getmstquery = "SELECT stock_standard,stock_qua FROM stock_mst where parts_name ='%s'"%(reqdata[0],)
    Gettrnquery = "SELECT stock_qua FROM stock_trn where parts_name = '%s'"%(reqdata[0],)
    # コネクションの確立
    with cx_Oracle.connect(USER, PASS, dsn, encoding = "UTF-8") as connection:
        # カーソル生成し# SQL 発行
        with connection.cursor() as cursor:
            # get(mst)
            cursor.execute(Getmstquery)
            rows_mst = cursor.fetchall()[0]
            # get(trn)
            cursor.execute(Gettrnquery)
            rows_trn = cursor.fetchall()

    # カーソル解放
    #cursor.close()
    # コネクションの解放
    #connection.close()
    # トランザクションデータの数量を集計
    trnsum = numpy.sum(rows_trn)
    # マスターとトランザクションの数量の合計
    stock_plan = trnsum + rows_mst[1]

    # JSON で返す
    mappings = {"error": "", "mappings": [
        { "item": "chart1"      ,"sheet": 4,"cluster": 2,"type": "string","value" : str(rows_mst[0])},
        { "item": "chart1"      ,"sheet": 4,"cluster": 3,"type": "string","value" : str(stock_plan)}
        ]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

【アクションクラスター2　設定】

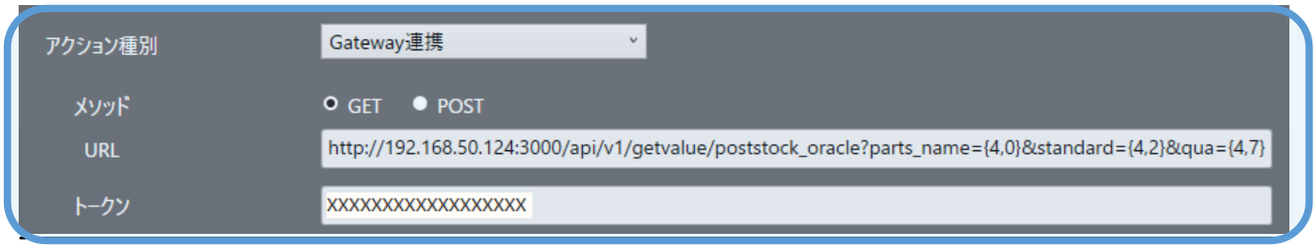


Fig.6-17-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

URL:http://localhost:3000/api/v1/getvalue/poststock_oracle ?parts_name={4,0}&standard={4,2}&qua={4,7}

トークン:　xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_oracle.json を呼び出します。
2. parts_name という変数にシート番号:4、クラスターID:0 のクラスター値
   standard という変数にシート番号:4、クラスターID:2 のクラスター値
   qua という変数にシート番号:4、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称

- 規格
- 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥poststock_oracle.json）】

```
{
   "datasource": "script ",
   "script": " scripts/post_oracle .py ",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post_oracle .py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<-----------データソース名（任意の名前）：**Python** スクリプト連携

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_oracle.py）】

```
import numpy
import json
import sys
import cx_Oracle

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    #Oracle 接続情報
    HOST = "192.168.50.125"
    PORT = 1521
    SVC_NM = "xe"
    USER = 'system'
    PASS = 'cimtops'

    # 接続記述子の生成
    dsn = cx_Oracle.makedsn(HOST, PORT, service_name = SVC_NM)
    # SQL 生成
    query = "INSERT INTO stock_trn (parts_name,stock_standard,stock_qua,akaden_flag) VALUES
('%s','%s','%s',1)"%(pdata[0],pdata[1],pdata[2])
    # コネクションの確立
```

```
    with cx_Oracle.connect(USER, PASS, dsn, encoding = "UTF-8") as connection:
        # カーソル生成し# SQL 発行
        with connection.cursor() as cursor:
            # insert
            cursor.execute(query)
            connection.commit()
    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1","sheet": 4,"cluster": 10,"type": "string","value" : "登録完了
"}]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

18.　Python スクリプト連携（CSV ファイルからのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で CSV ファイルからデータ取得・計算・書き込みを行う方法について説明します。

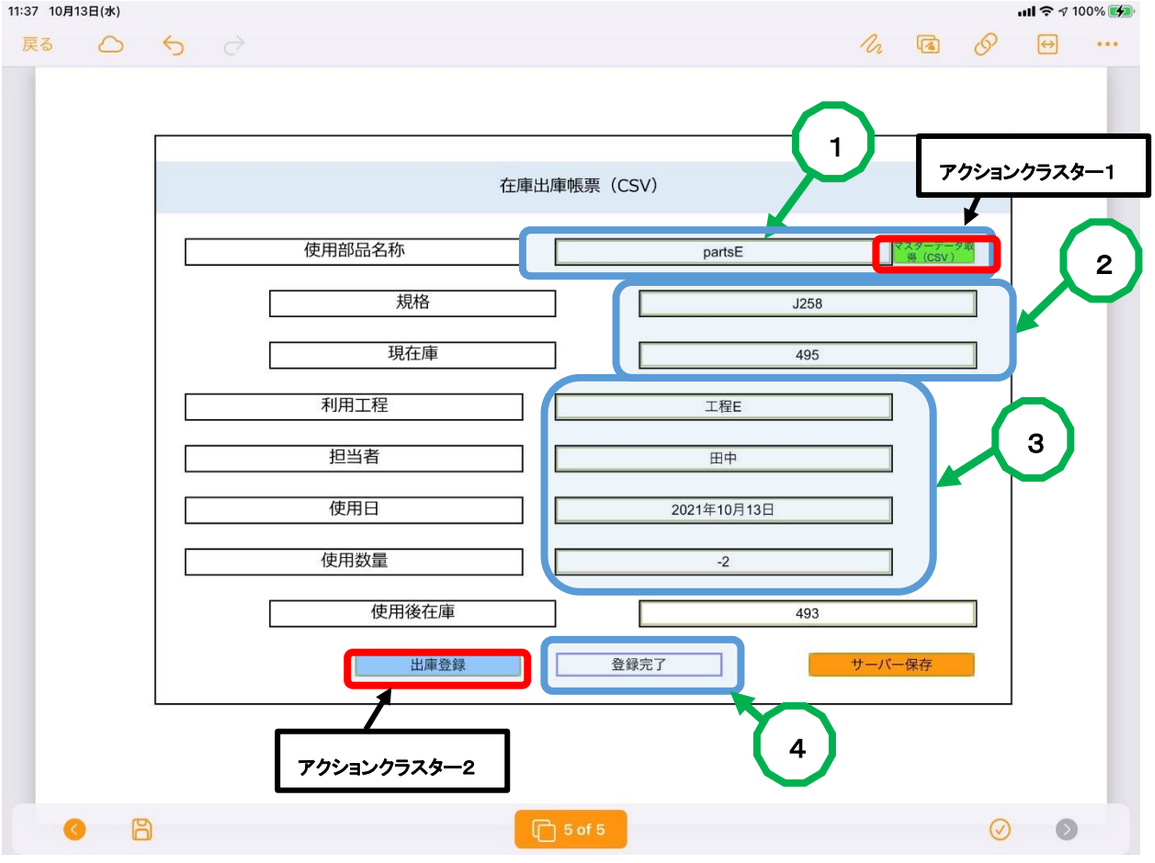


Fig.6-18-1 Python スクリプト連携（CSV ファイルからデータ取得・計算・書き込み）

【動作概要】

1. 【使用部品名称】を選択し、Python スクリプトへ値を渡します。
2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれの CSV からデータを取得します。
   - stock_mst.csv
   - stock_trn.csv

   stock_trn.csv で抽出した複数行のデータを総和し stock_mst.csv のデータから引きます。
   取得したデータ及び計算結果をそれぞれクラスターへ戻します。
3. 【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。
   書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量

   （※本サンプルでは上記3つ）
   stock_trn.csv へ書き込む
4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

Fig.6-18-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http://192.168.50.124:3000/api/v1/getvalue/getstock_csv?parts_name={5,0}

トークン:　xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_csv.json を呼び出します。
2. parts_name という変数にシート番号:5、クラスターID:0 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに CSV から検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_csv.json)】

```
{
    "datasource": "script",
    "script": "scripts/getstock_csv.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_csv.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
    "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_csv.py)】

```
import sys
import json
import pandas as pd

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']
```

```
    #マスターを検索
    csv_input = pd.read_csv(filepath_or_buffer="ex/stock_mst.csv", encoding="Shift-JIS", sep=",",quotechar="")
    data_mst = csv_input[csv_input["parts_name"] == reqdata[0]]
    #使用数量を抽出（単一選択クラスターで選択した値のみ抽出し合計）
    csv_output = pd.read_csv(filepath_or_buffer="ex/stock_trn.csv", names=["parts_name",
"stock_standard","number_of_use","update_time"],encoding="Shift-JIS", sep=",",quotechar="")
    number_of_use = csv_output[csv_output["parts_name"] == reqdata[0]].sum()
    # マスターと使用数量の合計
    total = int(data_mst.values[0,2] + number_of_use.values[2])
    #JSON で返す
    mappings = {"error": "", "mappings": [
        { "item": "item1"      ,"sheet": 5,"cluster": 2,"type": "string","value" : str(data_mst.values[0,1])},
        { "item": "item2"      ,"sheet": 5,"cluster": 3,"type": "string","value" : str(total)}
        ]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター2　設定】

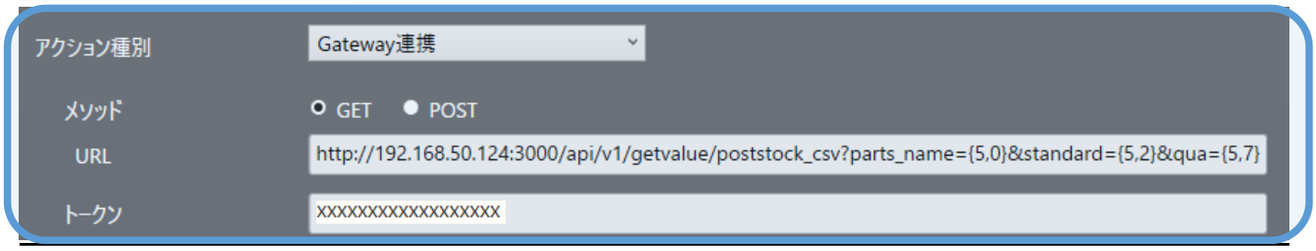


Fig.6-18-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

URL:http://localhost:3000/api/v1/getvalue/poststock_csv ?parts_name={5,0}&standard={5,2}&qua={5,7}

トークン:　xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_csv.json を呼び出します。
2. parts_name という変数にシート番号:5、クラスターID:0 のクラスター値
   standard という変数にシート番号:5、クラスターID:2 のクラスター値
   qua という変数にシート番号:5、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称
   - 規格
   - 使用数量

   （※本サンプルでは上記3つ）

stock_trn.csv フィルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥poststock_csv.json）】

```
{
   "datasource": "script ",
   "script": " scripts/post_csv .py ",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post_csv .py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）:Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_csv.py）】

```
import json
import csv
import sys
import datetime

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    f = open("ex/stock_trn.csv", "a", encoding="Shift-JIS", newline="")
    data = [str(pdata[0]), str(pdata[1]), str(pdata[2]), str(datetime.datetime.now())]
    writer = csv.writer(f, delimiter=',', quotechar="", quoting=csv.QUOTE_NONNUMERIC)
    writer.writerow(data)
    f.close()

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1","sheet": 1,"cluster": 10,"type": "string","value" : "登録完了
"}]}
    print(json.dumps(mappings))

except Exception as e:
  mappings = {"error": "Python でエラー:" + str(e)}
  print(json.dumps(mappings))
```

19.　Python スクリプト連携（MySQL からのデータ取得・計算・書き込み）

出庫帳票サンプル定義 を利用して Python スクリプト連携で MySQL からのデータ取得・計算・書き込みを行う方法について説明します。

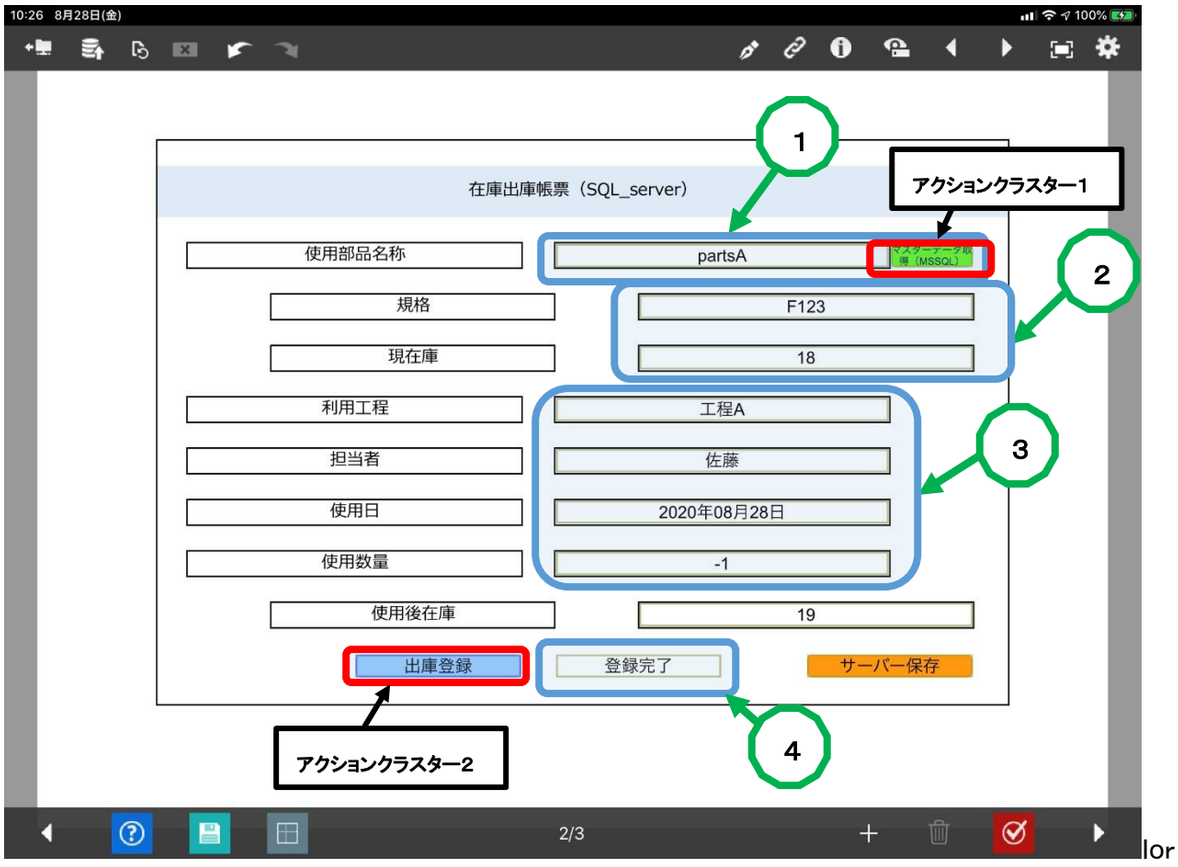


Fig.6-19-1 Python スクリプト連携（MySQL からのデータ取得・計算・書き込み）

【動作概要】

1.【使用部品名称】を選択し、Python スクリプトへ値を渡します。

2. コード内で受け取った部品名称に当てはまる行を下記のそれぞれのテーブルからデータを取得します。

- stock_mst
- stock_trn

stock_trn で抽出した複数行のデータを積算し stock_mst のデータを更に積算します。

取得したデータ及び計算結果をそれぞれクラスターへ戻します。

3.【利用工程】、【担当者】、【使用日】、【使用数量】を入力します。

書き込むデータを python へ渡す

- 使用部品名称
- 規格
- 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションクラスター1　設定】

Fig.6-19-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http://localhost:3000/api/v1/getvalue/getstock_mysql?parts_name={6,0}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の getstock_ mysql.json を呼び出します。
2. parts_name という変数にシート番号:6、クラスターID:0 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 帳票内の【規格】、【現在庫】、【使用後在庫】クラスターに各テーブルから検索した結果を挿入されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥getstock_mysql.json)】

```
{
    "datasource": "script",
    "script": "scripts/getstock_mysql.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の getstock_mysql.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):**Python** スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥getstock_mysql.py)】

```
import sys
import json
import numpy
import mysql.connector

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    # 接続情報
    HOST = "localhost"
    PORT = 3306
    USER = 'root'
    PASS = 'cimtops'
    DB = "gwdb"
```

```
    # SQL 生成
    Getmstquery = "SELECT stock_standard,stock_qua FROM stock_mst where parts_name = %s"
    Gettrnquery = "SELECT stock_qua FROM stock_trn where parts_name = %s"
    # コネクションの確立
    with mysql.connector.connect(
      host=HOST,
      port=PORT,
      user=USER,
      password=PASS,
      database=DB
    ) as connection:
        # カーソル生成し# SQL 発行
        with connection.cursor() as cursor:
            # get(mst)
            cursor.execute(Getmstquery, (reqdata[0],))
            rows_mst = cursor.fetchall()[0]
            # get(trn)
            cursor.execute(Gettrnquery, (reqdata[0],))
            rows_trn = cursor.fetchall()

    # トランザクションデータの数量を集計
    trnsum = numpy.sum(rows_trn)
    # マスターとトランザクションの数量の合計
    stock_plan = trnsum + rows_mst[1]

    # JSON で返す
    mappings = {"error": "", "mappings": [
        { "item": "chart1"      ,"sheet": 6,"cluster": 2,"type": "string","value" : str(rows_mst[0])},
        { "item": "chart1"      ,"sheet": 6,"cluster": 3,"type": "string","value" : str(stock_plan)}
        ]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター2　設定】

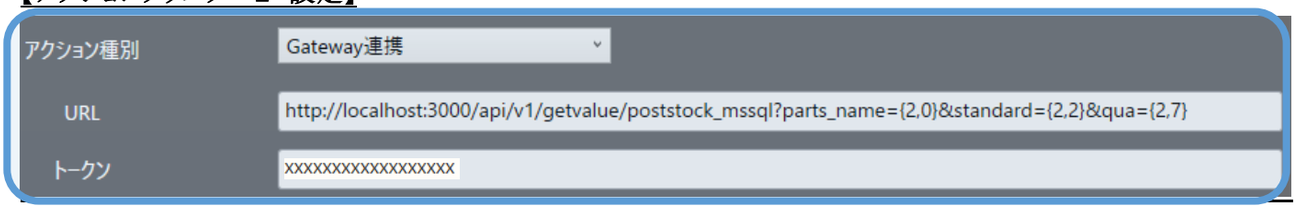


Fig.6-19-3 アクションクラスター2設定

アクション種別:　【Gateway 連携】

URL:http://localhost:3000/api/v1/getvalue/poststock_ mysql ?parts_name={6,0}&standard={6,2}&qua={6,7}

トークン:　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の poststock_mysql.json を呼び出します。
2. parts_name という変数にシート番号:6、クラスターID:0 のクラスター値
   standard という変数にシート番号:6、クラスターID:2 のクラスター値
   qua という変数にシート番号:6、クラスターID:7 のクラスター値
   が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. 書き込むデータを python へ渡す
   - 使用部品名称

- 規格
- 使用数量

（※本サンプルでは上記3つ）

stock_trn テーブルへ書き込む

4. 書き込みが完了後『登録完了』の文字列をクラスターへ戻します。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥poststock_mysql.json）】

```
{
   "datasource": "script ",
   "script": " scripts/post_mysql.py ",
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **post_mysql.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<-----------データソース名（任意の名前）：**Python** スクリプト連携

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥post_mysql.py）】

```
import sys
import json
import numpy
import mysql.connector

try:

    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    # 接続情報
    HOST = "localhost"
    PORT = 3306
    USER = 'root'
    PASS = 'Cimtops'
    DB = "gwdb"

    # SQL 生成
    query = "INSERT INTO stock_trn (parts_name,stock_standard,stock_qua,akaden_flag) VALUES (%s,%s,%s,1)"
    # コネクションの確立
    with mysql.connector.connect(
      host=HOST,
      port=PORT,
```

```
        user=USER,
        password=PASS,
        database=DB
    ) as connection:
        #  カーソル生成し# SQL 発行
        with connection.cursor() as cursor:
            # insert
            cursor.execute(query, (pdata[0],pdata[1],pdata[2]))
            connection.commit()

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1","sheet": 6,"cluster": 10,"type": "string","value" : "登録完了
"}]}
    print(json.dumps(mappings, ensure_ascii=False))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings, ensure_ascii=False))
```

20.　Python スクリプト連携（Excel 関数を Python スクリプトで作成）

i-Reporter 未対応関数サンプル定義 を利用して Python スクリプト連携で Excel 関数を Python スクリプトで作成する方法について説明します。

- SQRT 関数
- POWER 関数
- LOG 関数
- MOD 関数
- TEXT 関数（※全ての表示形式には対応しておりません）

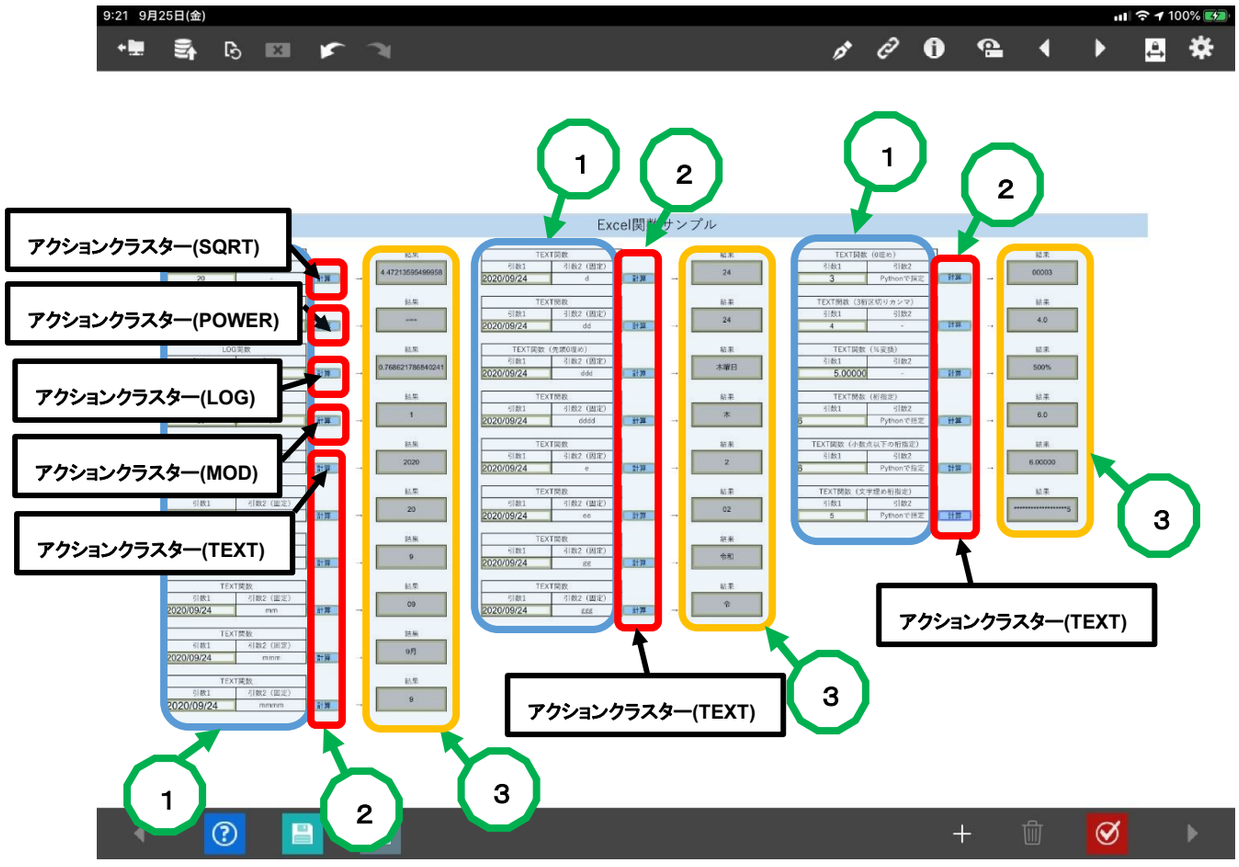


Fig.6-20-1 Python スクリプト連携（Excel 関数を Python スクリプトで作成）

【動作概要】

1. 各関数のパラメーターを設定します。Python スクリプトへ値を渡します。
2. アクションクラスターを起動して、1で設定したパラメーターを Python スクリプトへ値を渡します。
3. Python スクリプトより戻ってきた結果をクラスターに表示します。

【アクションクラスター（SQRT 関数）設定】

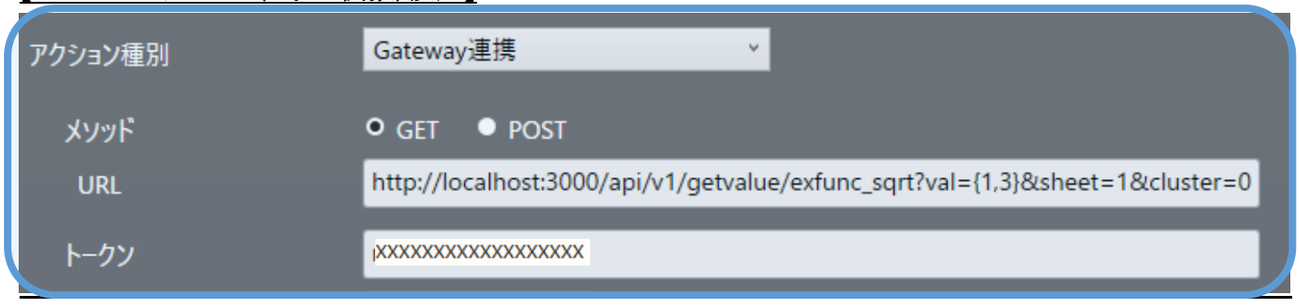


Fig.6-20-2 アクションクラスター（SQRT 関数）設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/exfunc_sqrt?val={1,3}&sheet=1&cluster=0

トークン：　xxxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の exfunc_sqrt.json を呼び出します。
2. val という変数にシート番号：1、クラスターID：3 のクラスター値
   Sheet という変数に1、cluster という変数に 0 が渡されます。
3. 帳票内のクラスターに結果が表示されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥exfunc_sqrt.json）】

```
{
    "datasource": "script",
    "script": "scripts/sqrt.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の sqrt.py が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）：Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥sqrt.py）】

```
import json
import sys
import math

try:

    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    resdata = math.sqrt(int(reqdata[0]))

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"      ,"sheet": int(reqdata[1]),"cluster": int(reqdata[2]),"type":
"string","value" : str(resdata)}]}
    print(json.dumps(mappings))
```

```
except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター（POWER 関数）設定】

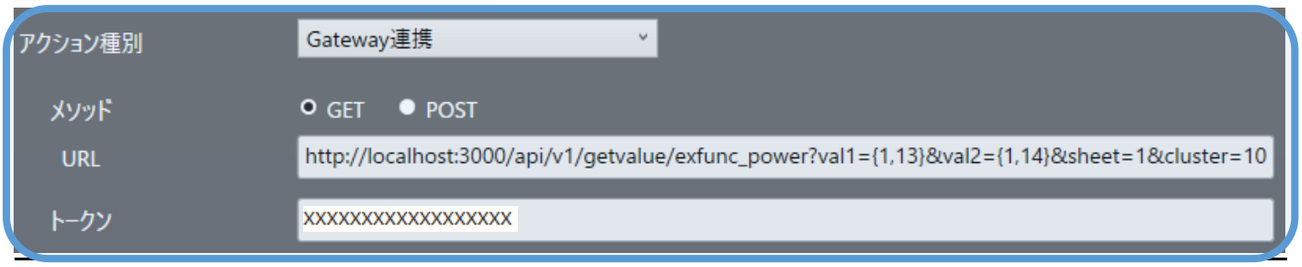


Fig.6-20-3 アクションクラスター（POWER 関数）設定

アクション種別：　【Gateway 連携】

URL：http://localhost:3000/api/v1/getvalue/exfunc_power?val1={1,13}&val2={1,14}&sheet=1&cluster=10

トークン：　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の exfunc_power.json を呼び出します。
2. val1 という変数にシート番号：1、クラスターID：13 のクラスター値
   val2 という変数にシート番号：1、クラスターID：14 のクラスター値
   sheet という変数に 1、cluster という変数に 10 が渡されます。
3. 帳票内のクラスターに結果が表示されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥exfunc_power.json）】

```
{
   "datasource": "script ",
   "script": " scripts/power.py ",
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の power.py が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）：Python スクリプト連携
   "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥power.py）】

```
import json
import sys
import math

try:
    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']
```

```
    resdata = pow(int(reqdata[0]),int(reqdata[1]))

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": int(reqdata[2]),"cluster": int(reqdata[3]),"type":
"string","value" : str(resdata)}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター(LOG 関数)設定】

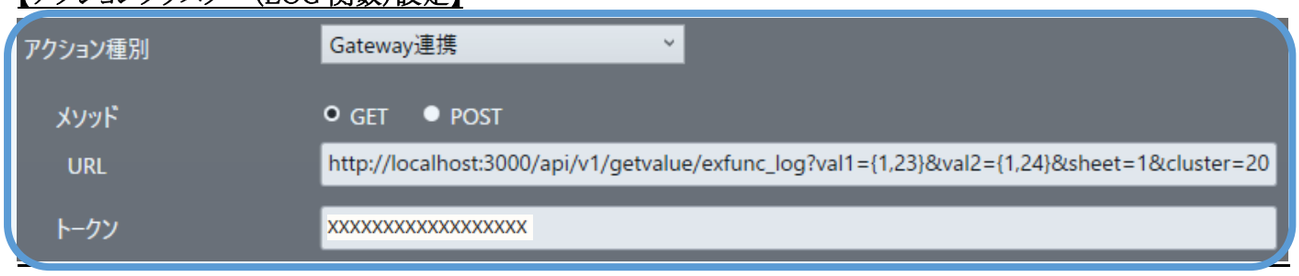


Fig.6-20-4 アクションクラスター(LOG 関数)設定

アクション種別:　【Gateway 連携】

URL:

http://localhost:3000/api/v1/getvalue/exfunc_log?val1={1,23}&val2={1,24}&sheet=1&cluster=20

トークン:　xxxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の exfunc_log.json を呼び出します。
2. val という変数にシート番号:1、クラスターID:23 のクラスター値
   val2 という変数にシート番号:1、クラスターID:24 のクラスター値
   Sheet という変数に1、cluster という変数に 20 が渡されます。
3. 帳票内のクラスターに結果が表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥exfunc_log.json)】

```
{
    "datasource": "script",
    "script": "scripts/log.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の log.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
    "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥log.py）】

```
import json
import sys
import math

try:
    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    resdata = math.log(int(reqdata[0]),int(reqdata[1]))

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"    ,"sheet": int(reqdata[2]),"cluster": int(reqdata[3]),"type": "string","value" : str(resdata)}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー：" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター（MOD 関数）設定】

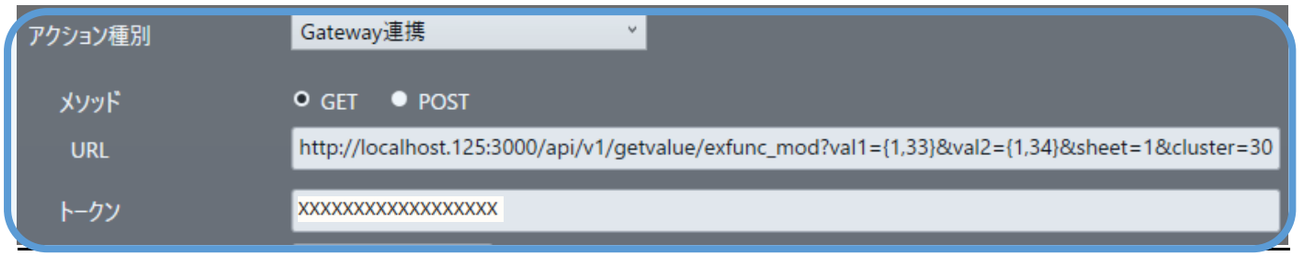


Fig.6-20-5 アクションクラスター（MOD 関数）設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/exfunc_mod?val1={1,33}&val2={1,34}&sheet=1&cluster=30

トークン：　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の exfunc_mod.json を呼び出します。
2. val という変数にシート番号：1、クラスターID：33 のクラスター値
   val2 という変数にシート番号：1、クラスターID：34 のクラスター値
   Sheet という変数に1、cluster という変数に 30 が渡されます。
3. 帳票内のクラスターに結果が表示されます。

【アクションファイル設定（D:¥ConMas¥gateway¥actions¥exfunc_mod.json）】

```
{
    "datasource": "script",
    "script": "scripts/mod.py"
}
```

default.json 内の**"script"**項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の **mod.py** が呼び出されます

【設定ファイル（D:¥ConMas¥gateway¥config¥default.json）内のデータソース設定】

```
{ "name": "script",<----------データソース名（任意の名前）:Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥mod.py）】

```
import json
import sys
import math

try:
    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    resdata = int(reqdata[0]) % int(reqdata[1])

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"    ,"sheet": int(reqdata[2]),"cluster": int(reqdata[3]),"type": "string","value" : str(resdata)}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター（TEXT 関数）設定】

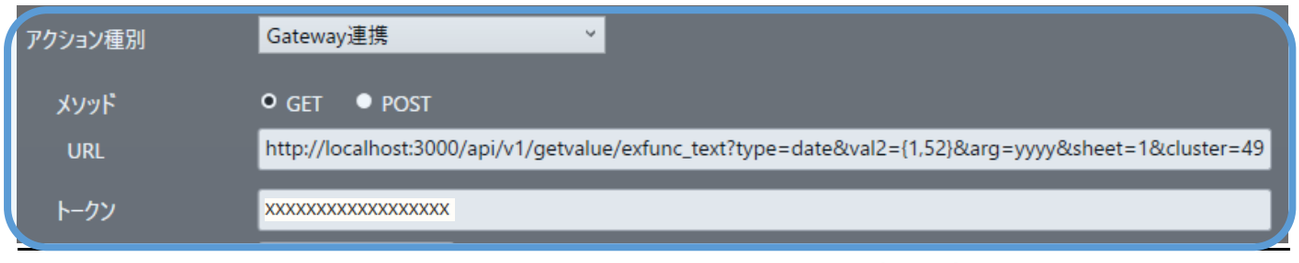


Fig.6-20-6 アクションクラスター（TEXT 関数）設定

アクション種別：　【Gateway 連携】

URL：

http://localhost:3000/api/v1/getvalue/exfunc_text?type=date&val2={1,52}&arg=yyyy&sheet=1&cluster=49

トークン:　　xxxxxxxxxxxxxxxxx

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の exfunc_text.json を呼び出します。
2. type という変数に date という値が渡されます
   val2 という変数にシート番号:1、クラスターID:52 のクラスター値
   arg という変数に yyyy、sheet という変数に 1、cluster という変数に 49 が渡されます。
3. 帳票内のクラスターに結果が表示されます。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥exfunc_text.json)】

```
{
    "datasource": "script",
    "script": "scripts/text.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の text.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",<----------データソース名(任意の名前):Python スクリプト連携
  "type": "python"
},
```

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥text.py)】

```
import json
import sys
import math
import datetime
from japanera import Japanera, EraDate
import locale

"""
月・曜日を日本語にしたい場合はロケールを変更してください
"""
#locale.setlocale(locale.LC_ALL, 'en_US.utf8')
#locale.setlocale(locale.LC_ALL, 'ja_JP.utf8')

# text_main_function
def function_text(type,val,arg):

    """
    value の型は、適宜変更してご利用ください
    """

    if type == 'date':
```

```
            result_date = function_date(val,arg)
            return result_date

        elif type == 'zero_pad':
            result_text_type = str(val).rjust(5,'0')

        elif type == 'comma':
            result_text_type = '{:,}'.format(float(val))

        elif type == 'per':
            result_text_type = '{:.0%}'.format(float(val))

        elif type == 'digit':
            result_text_type = '{:.5}'.format(float(val))

        elif type == 'digit_decimal':
            result_text_type = '{:.5f}'.format(float(val))

        elif type == 'digit_embedded':
            result_text_type = '{:*>20}'.format(str(val))
        return result_text_type

# date_function
def function_date(date,format):

        # year month day
        date_formatted = datetime.datetime.strptime(date, "%Y/%m/%d")
        result_date = date_formatted

        def_format = {'yyyy':'%Y', 'yy':'%y', 'm':'%m', 'mm':'%m', 'mmm':'%B', 'mmmm':'%b', 'd':'%d', 'dd':'%d', 'ddd':'%A',
'dddd':'%a'}

        if format in def_format:
            conv_val =    result_date.strftime(def_format[format])
            if format == 'd' or format == "m":
                conv_val = conv_val.lstrip("0")
        else:
            def_format_era = {'e':'%-o', 'ee':'%-o', 'gg':'%-E', 'ggg':'%-E'}
            era_date = EraDate(result_date.year, result_date.month, result_date.day)
            conv_val = era_date.strftime(def_format_era[format])
            if format == 'e':
                conv_val = conv_val.lstrip("0")
            elif format == 'ggg':
                conv_val = conv_val[0]
        return conv_val
```

```
try:
    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']

    result_data = function_text(reqdata[0],reqdata[1],reqdata[2])

    if reqdata[2].isdigit() or len(reqdata) !=5 :
        reqdata.insert(2,'')

    #JSON で返す
    mappings = {"error": "", "mappings": [{ "item": "chart1"       ,"sheet": int(reqdata[3]),"cluster": int(reqdata[4]),"type":
"string","value" : str(result_data)}]}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

21. Python スクリプト連携（クラスターをフォーカスする機能）

サンプル定義 を利用して Python スクリプト連携でクラスターをフォーカスする方法について説明します。

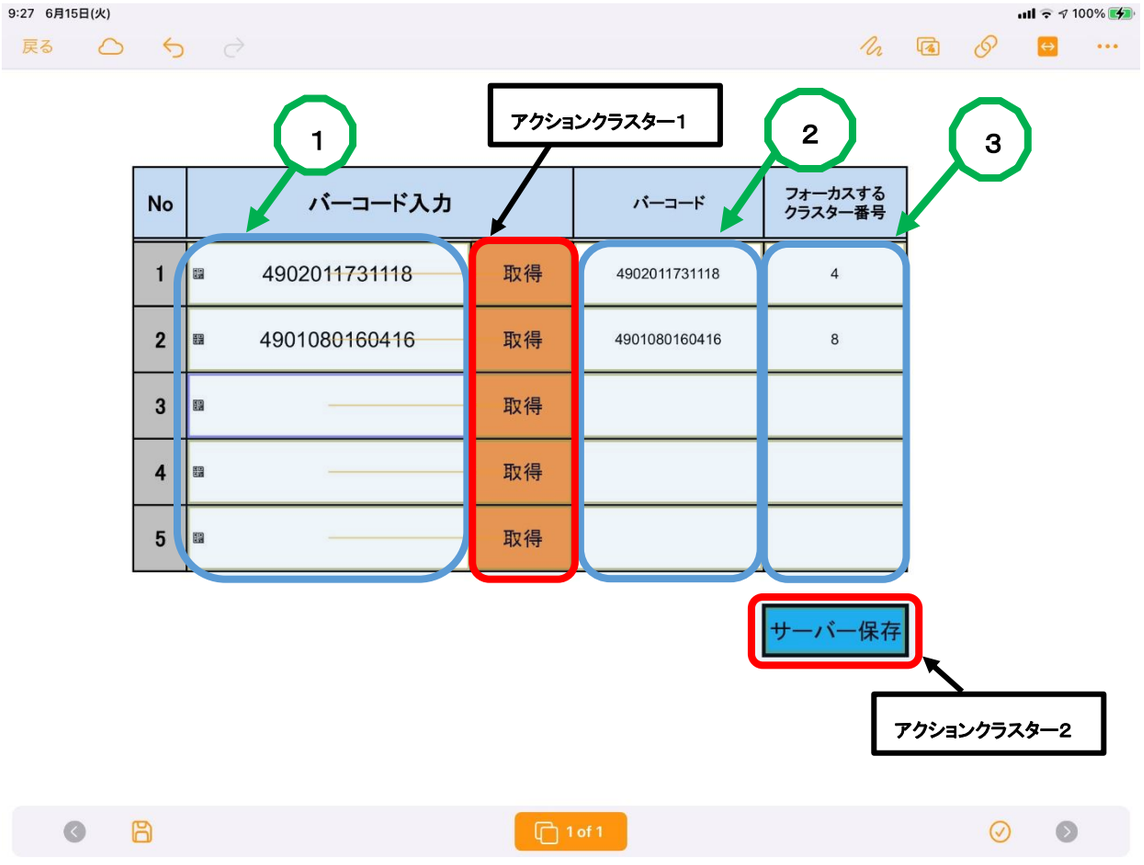


Fig.6-21-1 Python スクリプト連携（クラスターをフォーカスする機能）

【動作概要】

1. 【このクラスターにフォーカスして入力待ちの状態で起動する】設定で帳票を開くと1行目のバーコード入力クラスターが自動起動されます。バーコード読取後、ネットワーク設定の後続クラスター【アクションクラスター1】が起動され Python スクリプトへバーコード値と次にフォーカスするクラスターID 値を渡します。
2. Python スクリプトへ渡したバーコード値が表示されます。
3. Python スクリプトへ渡した次にフォーカスするクラスターID 値が表示されます。
4. 次にフォーカスするクラスターID が2行目のバーコード入力のため、2行目のバーコード入力が自動起動されます。この動作を5行目まで繰り返します。
4. 5行目のバーコード読取後、次にフォーカスするクラスターID が【アクションクラスター2】のためサーバーへ保存動作が起動されます。

このようにクラスターをフォーカスする機能を利用することにより、連続した操作を画面タップ無でクラスター起動することができ、操作の手間を省くことが可能になります。

【アクションクラスター1　設定】

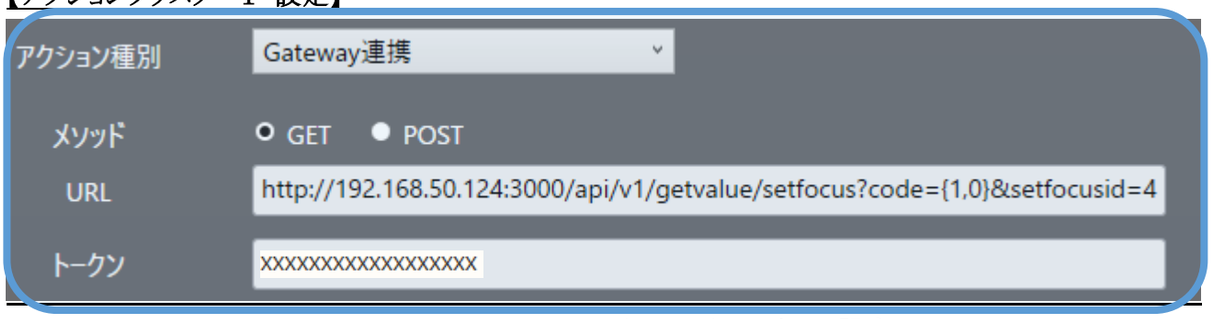


Fig.6-21-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http:// 192.168.50.124:3000/api/v1/getvalue/setfocus?code={1,0}&setfocusid=4

トークン:　xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の SetFocus.json を呼び出します。
2. code という変数にシート番号:1、クラスターID:0 のクラスター値が渡されます。
   setfocusid という変数に次にフォーカスするクラスターID を設定します。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥SetFocus.json)】

```
{
    "datasource": "script",
    "script": "scripts/SetFocus.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の SetFocus.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
   "type": "python"
},
```

<----------データソース名(任意の名前):Python スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥SetFocus.py)】

```
import sys
import json
import os

try:
    jsonData = json.loads(sys.stdin.readline())
    pdata = jsonData['data']

    # バーコードから読取った値を利用した処理コードなどを記述する
    code = pdata[0]
    line = int(pdata[1])

    # JSON で返す
    mappings = {"error": "", "mappings": [
            { "item": "" ,"sheet": 1,"cluster":   (line - 2),"type": "string","value" : code},
            { "item": "" ,"sheet": 1,"cluster":   (line - 1),"type": "string","value" : pdata[1]},
```

```
        { "sheet": 1,"cluster": line,"type": "SetFocus"}
    ]}

    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

クラスターをフォーカスするコード
Sheet:シート番号
Cluster:クラスター番号
type:"SetFocus"

【機能仕様】

アプリは Gateway 連携アクションをタップしレスポンス受信後、入力値セットや計算処理など、帳票上の一通りの処理が終わってから、指定されたクラスターをアクティブにする。

インプット部品を編集できる状態のクラスターはインプット部品表示する。
・編集できる状態とは、権限や機能でロックされていない、そのログインユーザーが編集可能なクラスターのこと。
・編集できないクラスターはフォーカス移動するのみとする。

“sheet”要素の省略による Current シートへの動作も可能。

シート移動などによる、通常の「クラスターの入力待ち機能」とバッティングした場合は本機能のフォーカス動作を優先する。

【機能制限】

"type": "SetFocus"　は 1 レスポンス内に 1 つとする。2 つあった場合はバリデーションエラーとする。

"type": "SetFocus"での指定先が、アクションクラスターの Gateway 連携のように循環されてしまうようなクラスターへの指定については Python スクリプト内で指定しないように制限とする。
※同じ Gateway 呼び出しであっても、必ず同じ SetFocus が返ってくるとは限らないため、i-Reporter 側で循環設定を判断することができないため。

iPhone リスト形式で表示されないクラスターにフォーカスした場合はエラーにはならないが機能しない。

【アクションクラスター2　設定】

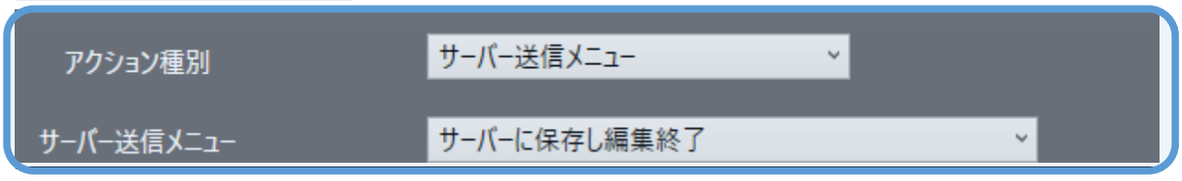


Fig.6-21-3 アクションクラスター2設定

22. Python スクリプト連携（複数行の検索結果を複数ページの帳票に展開するサンプル）

サンプル定義 を利用して Python スクリプト連携で複数行の検索結果を複数ページの帳票に展開する方法について説明します。

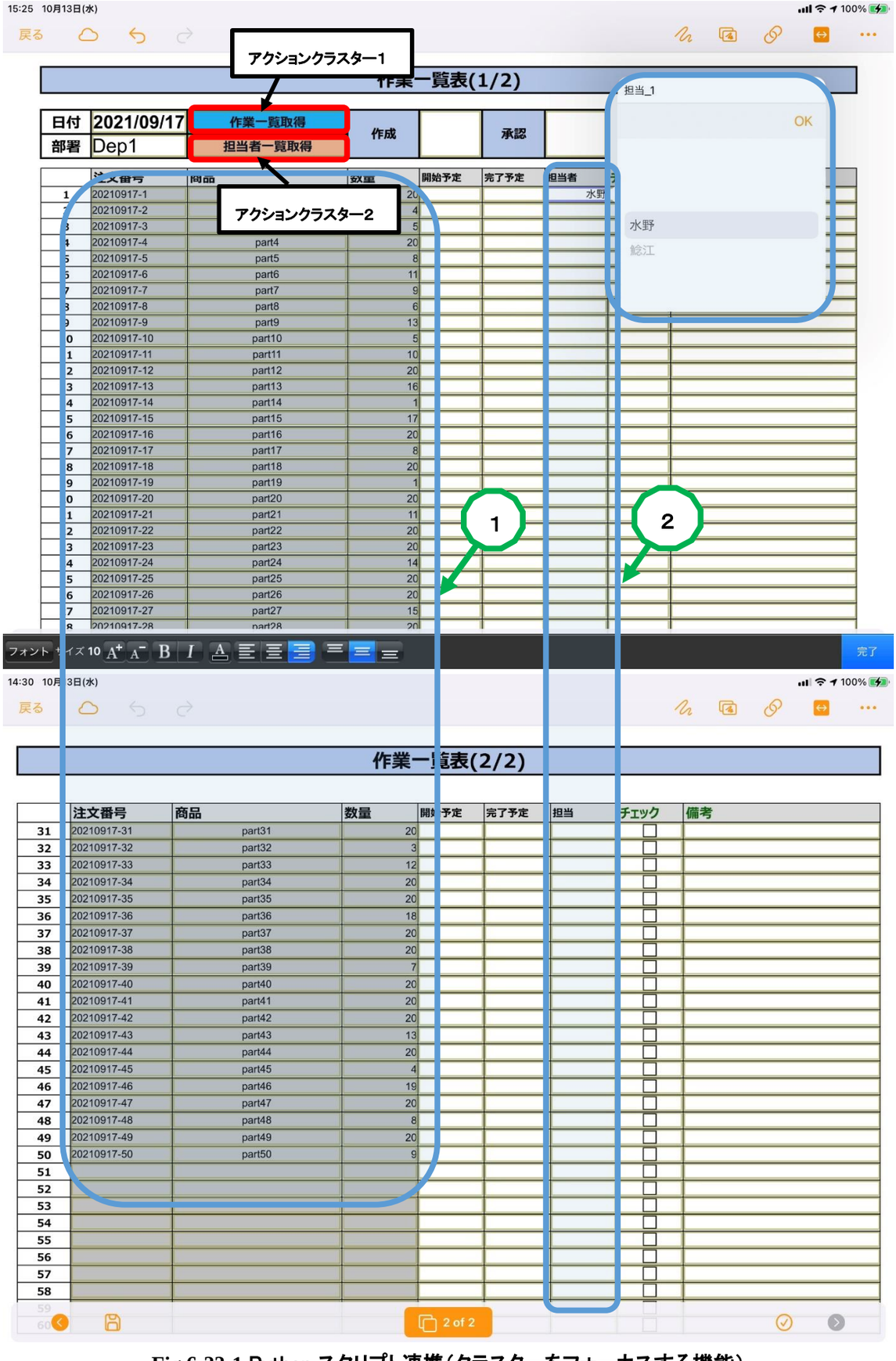


Fig.6-22-1 Python スクリプト連携（クラスターをフォーカスする機能）

【動作概要】

1. 【作業一覧取得】アクションクラスターを押下し、帳票内の【日付】を Python へ渡します。

ⓘ 作業一覧 CSV サンプルデータ(ex¥worklist.csv)には 2021/09/17,18 の2日間のみデータが存在します。いずれかの日を指定して確認してください。

受け取った【日付】で検索し当てはまる行を worklist.csv から抽出します。

【注文番号】、【商品】、【数量】列をそれぞれテキスト及び数値クラスターへ戻します。

2. 【担当者一覧取得】アクションクラスターを押下し、帳票内の【部署】を Python へ渡します。

受け取った【部署】で検索し所属している氏名を departmentlist.csv から抽出します。

抽出した担当者一覧を単一選択クラスターへ戻します。

【アクションクラスター1　設定】

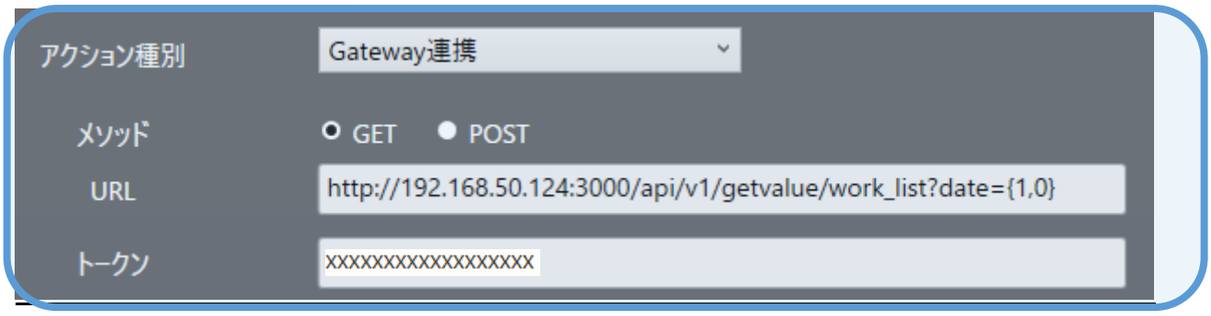


Fig.6-22-2 アクションクラスター1設定

アクション種別:　【Gateway 連携】

URL:　http:// 192.168.50.124:3000/api/v1/getvalue/work_list?date={1,0}

トークン:　xxxxxxxxxxxxxxxxx

ⓘ ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の work_list.json を呼び出します。
2. date という変数にシート番号:1、クラスターID:0 のクラスター値が渡されます。

ⓘ 単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥work_list.json)】

```
{
    "datasource": "script",
    "script": "scripts/work_list.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の work_list.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
    "type": "python"
},
```

<----------データソース名(任意の名前):Python スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥work_list.py)】

```
import sys
import json
import pandas as pd
import datetime

LINE_COUNT = 60                 #1ページ30行　X　2ページ分
LINE_ITEM_COUNT = 8             #1行のクラスター数
LINE_PAGE_COUNT = 30            #1ページ中の行数
CLUSTER_START_NUMBER = 6    #表となっている行中の開始クラスター番号
NUMBER_OF_RECIVE_COL = 3    #1行中の受信対象クラスター数

#文字列を日付に変換
def convert_date(x):
    date = datetime.datetime.strptime(x, '%Y/%m/%d')
    return date

try:
        #パラメータの取得
        jsonData = json.loads(sys.stdin.readline())
        reqdata = jsonData['data']
        #初期化
        mappings = ""
        result_arr = []

        #最初に空白行も含め初期化しておく
        colname = ["order_number", "product_name", "quantity"]
        result_arr = [
        {"item":    "{}_{}".format(colname[col], row % 30 + 1),
        "sheet": row // LINE_PAGE_COUNT + 1,
        "cluster": (row % LINE_PAGE_COUNT) * LINE_ITEM_COUNT + (CLUSTER_START_NUMBER if row //
LINE_PAGE_COUNT + 1 == 1 else 0)+ col,
        "type": "string",
        "value": ""}
        for row in range(LINE_COUNT)
        for col in range(NUMBER_OF_RECIVE_COL)]

        #CSV ファイルより日付で検索しレコードを抽出
        df = pd.read_csv(filepath_or_buffer="ex/worklist.csv", encoding="Shift-JIS", sep=",",quotechar="")
        df["date"] = df["date"].apply(convert_date)
        data_mst = df[df["date"] == datetime.datetime.strptime(reqdata[0], '%Y/%m/%d')]

        #取得した【注文番号】、【商品】、【数量】を value へ挿入
        cnt = 0
        for row in data_mst.itertuples():
            result_arr[cnt]["value"] = str(row[2])
```

```
        result_arr[cnt+1]["value"] = str(row[3])
        result_arr[cnt+2]["value"] = str(row[4])
        cnt += 3

    #Python オブジェクト mappings を作成
    mappings = {"error": "", "mappings": result_arr}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

【アクションクラスター2 設定】

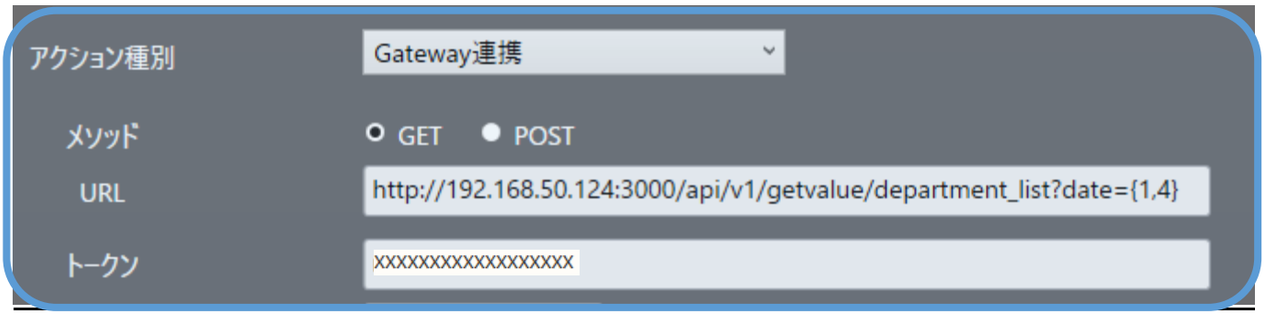


Fig.6-22-3 アクションクラスター2設定

アクション種別: 【Gateway 連携】

URL: **http:// 192.168.50.124:3000/api/v1/getvalue/department_list?date={1,4}**

トークン: xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の department_list.json を呼び出します。
2. date という変数にシート番号:1、クラスターID:4 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥department_list.json)】

```
{
    "datasource": "script",
    "script": "scripts/department_list.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の department_list.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):**Python** スクリプト連携

【Python スクリプトファイル（D:¥ConMas¥gateway¥scripts¥department_list.py）】

```
import sys
import json
import pandas as pd

LINE_COUNT = 60                #1ページ30行　X　2ページ分
LINE_ITEM_COUNT = 8            #1行のクラスター数
LINE_PAGE_COUNT = 30           #1ページ中の行数
CLUSTER_START_NUMBER = 6    #表となっている行中の開始クラスター番号

try:
    #パラメーターの取得
    jsonData = json.loads(sys.stdin.readline())
    reqdata = jsonData['data']
    #初期化
    mappings = ""
    result_arr = []

    #CSV ファイルより所属で検索しレコードを抽出
    df = pd.read_csv(filepath_or_buffer="ex/departmentlist.csv", encoding="Shift-JIS", sep=",",quotechar="")
    data_mst = df[df["department"] == str(reqdata[0])]

    #アクション用 JSON の作成
    result_arr = [
    {"item":    "list_{}".format(row + 1),
    "sheet": row // LINE_PAGE_COUNT + 1,
    "cluster": (row % LINE_PAGE_COUNT) * LINE_ITEM_COUNT + (CLUSTER_START_NUMBER if row //
LINE_PAGE_COUNT + 1 == 1 else 0)+ 5,
    "type": "SetItemsToSelect",
    "value": "",
    "selectItems": [{"item": str(row2[3]), "label": str(row2[3]), "selected": False} for row2 in data_mst.itertuples()]
    } for row in range(LINE_COUNT)]

    #Python オブジェクト mappings を作成
    mappings = {"error": "", "mappings": result_arr}
    print(json.dumps(mappings))

except Exception as e:
    mappings = {"error": "Python でエラー:" + str(e)}
    print(json.dumps(mappings))
```

23. Python スクリプト連携（外部連携 API（自動帳票作成）を呼び出す）

サンプル定義 を利用して Python スクリプト連携で外部連携 API を呼び出す方法について説明します。

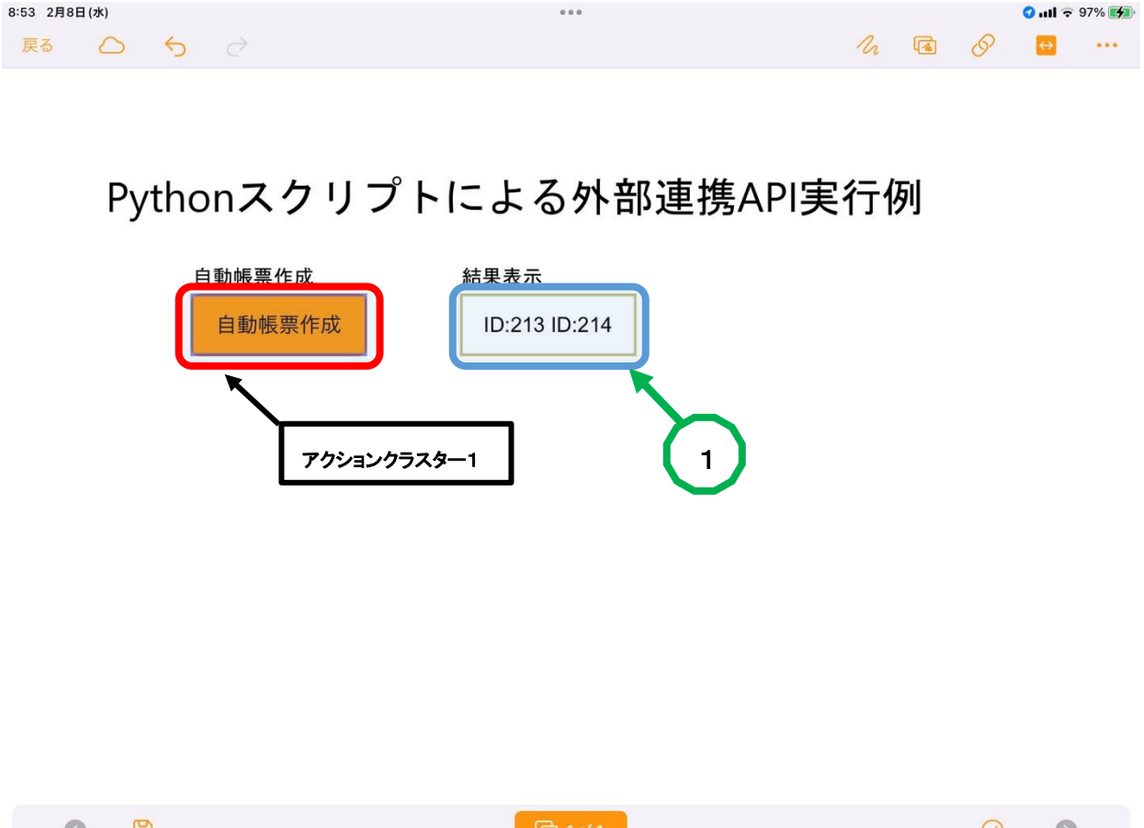


Fig.6-23-1 Python スクリプト連携（外部連携 API（自動帳票作成）を呼び出す）

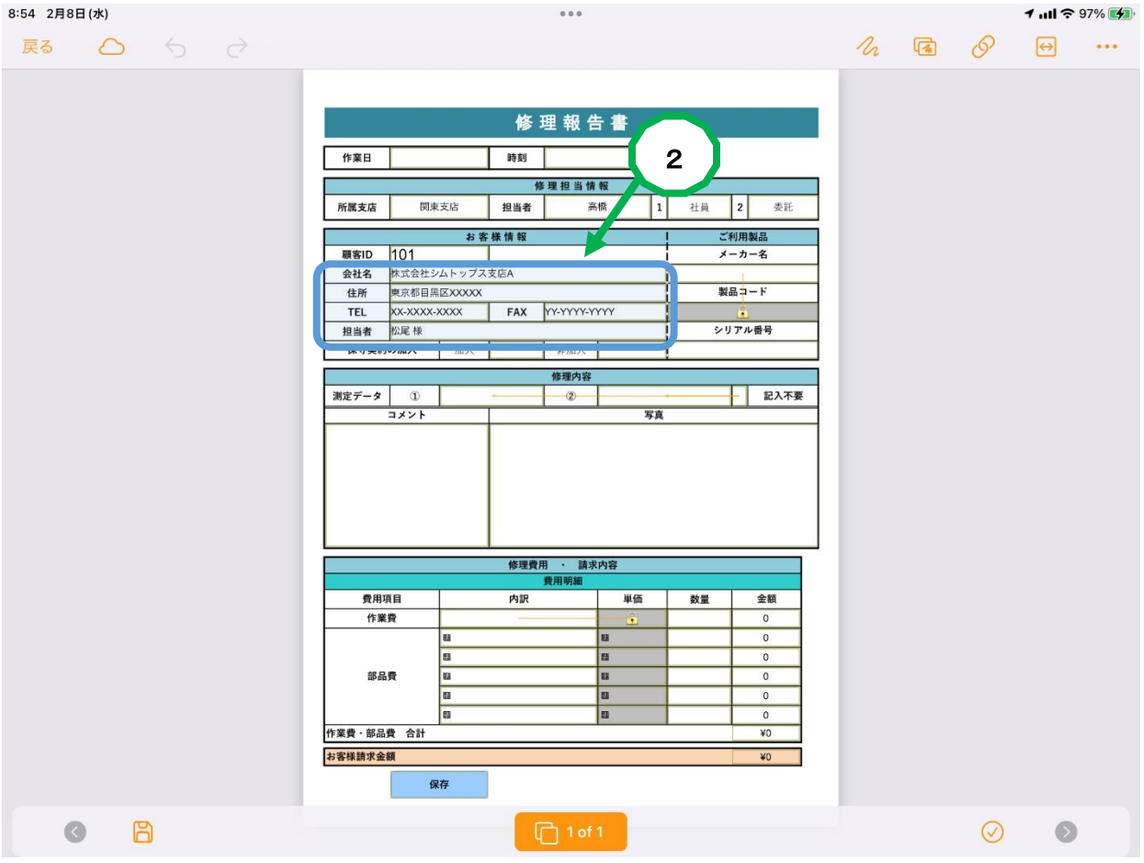


Fig.6-23-2 自動帳票作成 API で作成された入力前帳票

【動作概要】

1.【自動帳票作成】アクションクラスターを押下し、自動帳票作成 demo データ.csv の設定により入力前帳票を作成します。 作成された帳票 ID は結果表示クラスターに表示します。

自動帳票作成 demo データ.csv(ex¥自動帳票作成 demo データ.csv)は自動帳票作成 API で利用する CSV ファイルです。

2. 自動帳票作成 demo データ.csv に入力されている情報を各クラスターへ挿入した入力前帳票が作成されます。

【アクションクラスター1 設定】

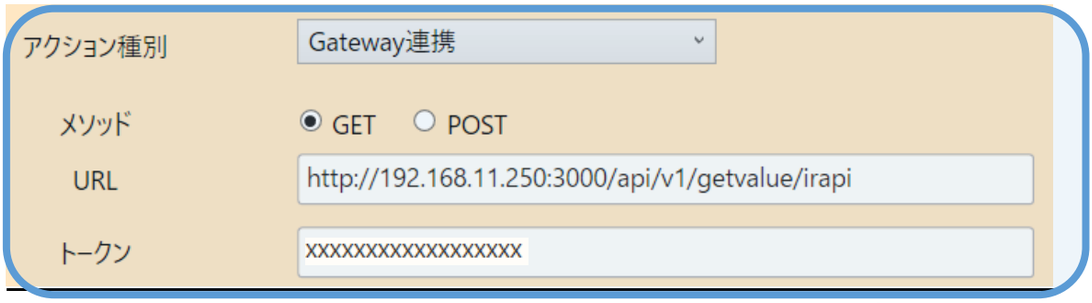


Fig.6-23-3 アクションクラスター1設定

1

アクション種別: 【Gateway 連携】

URL: http:// 192.168.11.250:3000/api/v1/getvalue/irapi

トークン: xxxxxxxxxxxxxxxxx

ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の irapi.json を呼び出します。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥irapi.json)】

```
{
    "datasource": "script",
    "script": "scripts/irapi.py"
}
```

default.json 内の"script"項目の設定が呼び出されます

D:¥ConMas¥gateway¥scripts フォルダ内の irapi.py が呼び出されます

【設定ファイル(D:¥ConMas¥gateway¥config¥default.json)内のデータソース設定】

```
{ "name": "script",
  "type": "python"
},
```

<-----------データソース名(任意の名前):**Python** スクリプト連携

【Python スクリプトファイル(D:¥ConMas¥gateway¥scripts¥irapi.py)】

```
# 外部連携 API 実行
# ConMas Gateway から py -X utf8 scripts¥irapi.py でも実行できることでしょう。
import os
import json
import requests
import xml.etree.ElementTree as ET
import traceback
import logging
```

```
if not os.path.exists("logs"):
   os.makedirs("logs")

logging.basicConfig(filename="./logs/irapi.log", level=logging.DEBUG,
   format="%(asctime)s - %(levelname)s - %(message)s")

baseUrl = "http://192.168.50.124/ConMasAPI/Rests/APIExecute.aspx?"

def main():
   with requests.Session() as s:

      params = {
            "command": "Login",
            "user": "user01",
            "password": "pass01"
      }

      res = s.post(baseUrl, params=params)
      logging.debug(res.text)
      root = ET.fromstring(res.text)

      for code in root.iter("code"):
         if code.text != "0":
            raise Exception(res.text)

      csvfile = "ex/自動帳票作成 demo データ.csv"
      csv = open(csvfile, "rb")
      files = {"dataFile": (csvfile, csv, "text/csv")}
      data = {
         "command": "AutoGenerate",
         "type": "csvSimple"
      }

      res = s.post(baseUrl, files=files, data=data)
      logging.debug(f"res={res}, text={res.text}")
      root = ET.fromstring(res.text)
      logging.debug(f"res={root}")

      clusters = []
      reports = ""
      for result in root.iter("result"):
         for code in result.iter("code"):
            if code.text != "0":
               raise Exception(f"error: {code.text}")
            else:
               reports += "ID:" + result[0].text + " "
      clusters.append({"item": "result", "cluster": 1, "type": "string", "value": reports})
      return {"error": "", "mappings": clusters}

mappings = {} # error 有無を含めた Python スクリプトからの返却値
try:
   mappings = main()
except Exception as e:
   logging.error(traceback.format_exc())
   mappings = {"error": f"Python でエラー: {str(e)}", "mappings": {}}
finally:
   print(json.dumps(mappings, ensure_ascii=False))
```

【自動帳票作成用 CSV ファイル（D:¥ConMas¥gateway¥ex¥自動帳票作成 demo データ.csv）】

```
"H","defTopId","S1C2","S1C3","S1C6","S1C7","S1C9","S1C10","S1C11","S1C13"
"R","1176","関東支店","高橋","101","株式会社シムトップス支店 A","東京都目黒区 XXXXX","XX-XXXX-XXXX","YY-YYYY-YYYY","松尾　様"
"R","1176","関東支店","高橋","102","株式会社シムトップス支店 B","東京都目黒区 XXXXX","XX-XXXX-XXXX","YY-YYYY-YYYY","堺　様"
```

第7章　接続データベースの設定

1.　バインド変数の利用法

1-1. 便利な利用場面について

従来、条件を持つサブクエリを含む SELECT 文を basequery に指定するのは不可能でした。また、列に対する範囲条件を指定するのも不可能でした。order by を伴う記述もできませんでした。その条件を params プロパティで与えることができなかったからです。例えば以下の例です。

範囲条件の例）select col1, col2 from table1 where key between 検索条件 1 and 検索条件 2

サブクエリと order by を伴う例) select input_date as last_date from (select * from table1 where input_date > 検索条件 1 order by input_date desc) where rownum <= 1 (Oracle)

※ここで、「検索条件 1」「検索条件 2」はクラスターから渡す条件値です。

これを解消するのがバインド変数となります。バインド変数の利用により、クエリ内の任意の箇所に条件値を埋め込むことが出来ますので、可能性が広がります。DB が本来持っているバインド変数の機構を名前付けバインド変数として扱いやすくご提供するものとなります。

1-2. 導入するための書式

先に説明した範囲条件の例をとり、アクションファイルの例を記述します。（i-Reporter のアクションクラスター設定については、これ以前の章の説明と同様です。）

【アクションファイル設定】

```
{
   "datasource": "conmasgwdb",
   "basequery" " select col1, col2 from table1 where key between :start and :end",
   "where": "",
   "params": [
      { "name" "start", "type": "string"},
      { "name" "end", "type": "string"}
   ],
   "mappings": [
      { "item": "col1"          ,"sheet": 1,"cluster": 0,"type": "string","value" : ""},
      { "item": "col2"          ,"sheet": 1,"cluster": 1,"type": "string","value" : ""}
   ]
}
```

検索条件：　バインド変数として使用できます

基本クエリー：　**SQL** の基本部分

params 要素の name プロパティの値（この場合、start, end）の先頭に ':'(コロン)を付けたもの（SQL Server は '@'(アットマーク)を付けたもの）を「バインド変数」と呼んでいます。

basequery 内の ":start" に対し、URL から start=検索値で渡した「検索値」が代入されることになり、":end"につ

いても同様です。

バインド変数を basequery 内で使用するのに特別な設定は不要で、ただ basequery 内にバインド変数参照があれば、上記のように解釈します。
複数テーブルを連結する場合は、SELECT 列には別名をつけてください。その別名は、mappings 配列の要素が持つ item プロパティの値と一致させる必要があります。

※バインド変数としてお使いいただける文字種類について:
アルファベット: 大文字（A から Z） 小文字（a から z) (ConMas Gateway では大文字小文字は区別されません)
数字: 0 から 9
記号: _（アンダースコア）

- 最初の文字は英字としてください。
- 各 DB の予約語は識別子としては使用できない制約がございます。
- アルファベットの大文字と小文字が区別されるかは、DB 依存の部分がございますが、
  ConMas Gateway では大文字小文字は区別されません。

2. チェッククラスターの利用法

2-1. マッピングの書式

クラスターマッピングについては、type を string とし、SQL で取得するマッピング値は”false”か”true”の文字列としてください。

例）

```
{
   "datasource": "conmasgwdb",
   "basetype": "file",
   "basequery": "./sql/custom_master/pg_check.sql",
   "params": [
      { "name": "orderno", "type": "string" }
   ],
   "mappings": [
      {
         "item": "check1",
         "sheet": 1,
         "cluster": 0,
         "type": "string",
         "value": "false"
      }
   ],
      {
         "item": "check2",
         "sheet": 1,
         "cluster": 1,
         "type": "string",
         "value": "true"
      }
   ]
}
```

2-2. SQL での取得方法

DB が PostgreSQL であり、bool 型である場合には、そのまま取得できます。ただし、null はマッピング出来ませんので適切な値に変換してください。

その他の DB で数値型や文字列型の場合、case, decode などの 条件分岐を使い 'false', 'true' という文字列へと変換し、列には別名を付けてください。

例） case col_check when 0 then 'false' else 'true' end check1

第8章　単一選択クラスターの選択肢取得

マスター選択クラスターの ConMas Gateway 対応定義(samples¥sample_select フォルダ内)を利用して、単一選択クラスターの選択肢を DB から取得する方法について説明します。

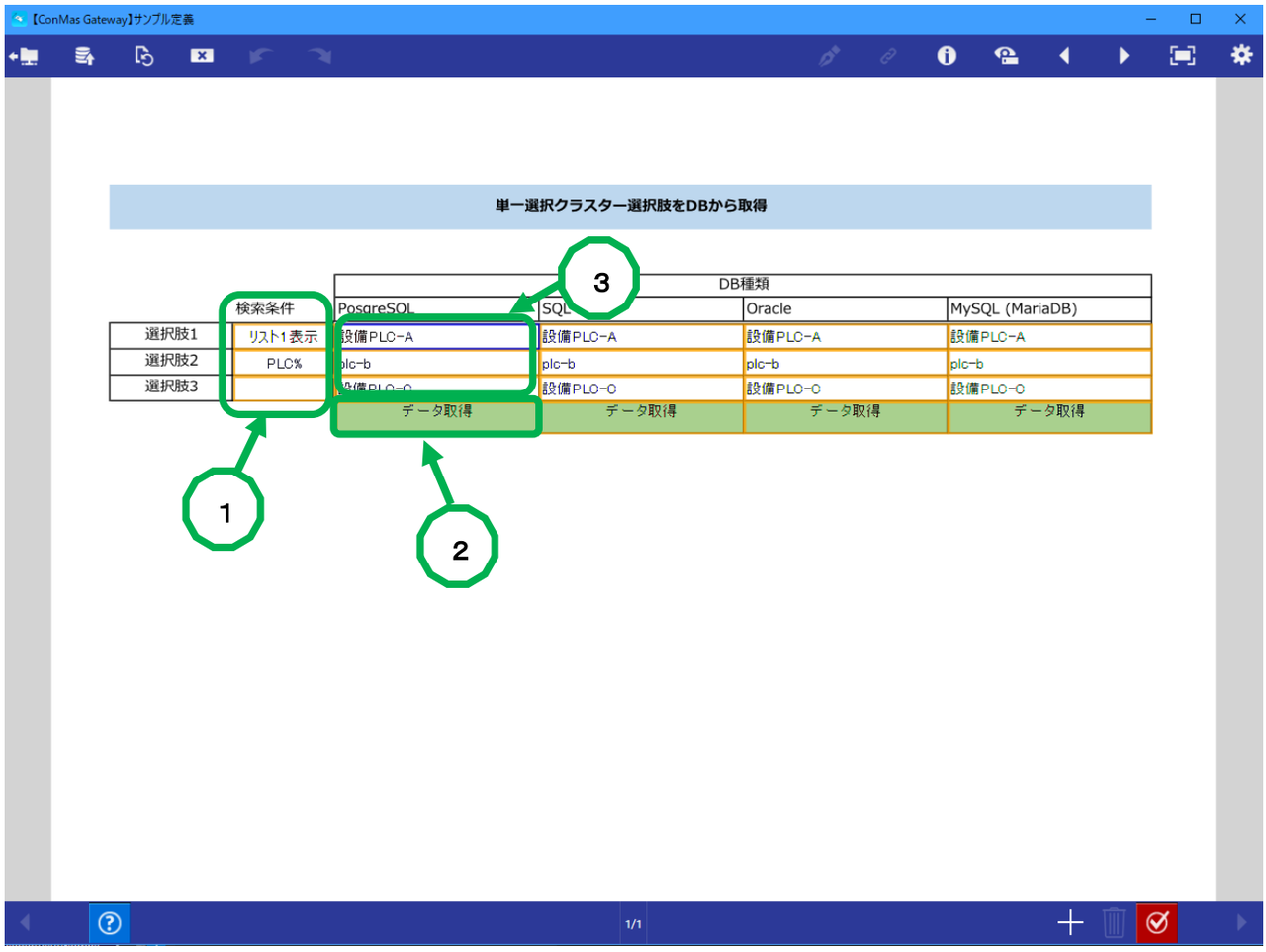


Fig.8-1　単一選択クラスター選択肢取得

【動作概要】

1. 検索条件を設定。
2. 各 DB 列のデータ取得ボタンを押下することで、アクションクラスターが起動。
3. 検索条件列の値をもとに該当 DB のテーブルから検索し、単一選択クラスターへリストを設定。

【単一選択クラスター設定】

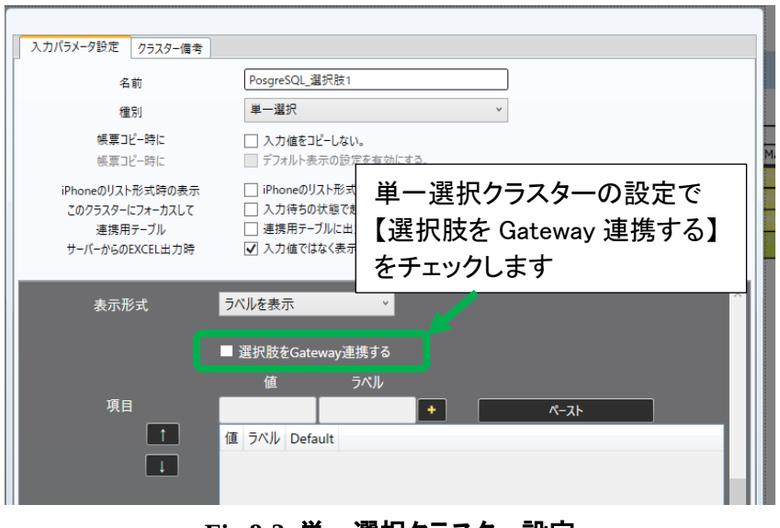


Fig.8-2　単一選択クラスター設定

【アクションクラスター設定】

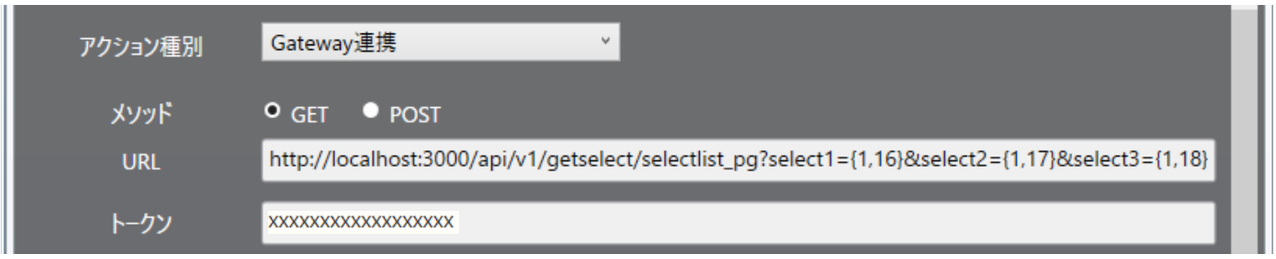


Fig.8-3 アクションクラスター設定

アクション種別:　【Gateway 連携】

URL:　http://localhost:3000/api/v1/getselect/selectlist_pg?select1={1,16}&select2={1,17}&select3={1,18}

トークン:　xxxxxxxxxxxxxxxxxx

1　2

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の selectlist_pg.json を呼び出します。
2. select1 という変数にシート番号:1、クラスターID:16 の、select2 という変数にシート番号:1、クラスターID:17 の、select3 という変数にシート番号:1、クラスターID:18 のクラスター値が渡されます。

単一選択クラスターで利用する場合は、ラベルでなく値が渡されますのでご注意ください。

3. テーブル内のデータを単一選択クラスターの値とラベルを検索条件から呼出し、各クラスターへ戻値が挿入されます。

値には"item"、ラベルには"label"と別名をつけてください。どちらも文字列型としてください。

【アクションファイル設定(D:¥ConMas¥gateway¥actions¥selectlist_pg.json)】

```
{
  "datasource": "conmasgwdb",
  "selectList": [
    {
      "name": "list1",
      "basequery": "select ¥"PLC アイディー¥" as ¥"item¥", ¥"PLC 名¥" as ¥"label¥" from public.¥"PLC マスター¥" where ¥"PLC 名¥" is not null and ¥"表示対象¥" = :select1 order by 1",
      "params": [
        { "name": "select1", "type": "string" }
      ]
    },
    {
      "name": "list2",
      "basequery": "select ¥"PLC アイディー¥" as ¥"item¥", name as ¥"label¥" from public.¥"PLC マスター¥" where name is not null and ¥"PLC アイディー¥" like :select2 order by 1",
      "params": [
        { "name": "select2", "type": "string" }
      ]
    }
  ],
  "mappings": [
    { "item": "list1", "sheet": 1, "cluster": 0, "type": "SetItemsToSelect", "value": "PLCA" },
```

default.json 内の"conmasgwdb"項目の設定が呼び出されます

選択肢リスト名:　基本クエリーで得られる集合に対する名前

基本クエリー:　**SQL** の基本部分

検索条件:　**URL** 内 **select1** の値を **basequery** 内バインド変数 **:select1** として使用する

選択肢に使う値の集合を指す **selectList** 内の選択肢リスト名

i-Reporter のクラスター番号

```
        { "item": "list2", "sheet": 1, "cluster": 4, "type": "SetItemsToSelect", "value": "PLCB"},
        { "item": "list1", "sheet": 1, "cluster": 8, "type": "SetItemsToSelect", "value": "PLCC"}
    ]
}
```

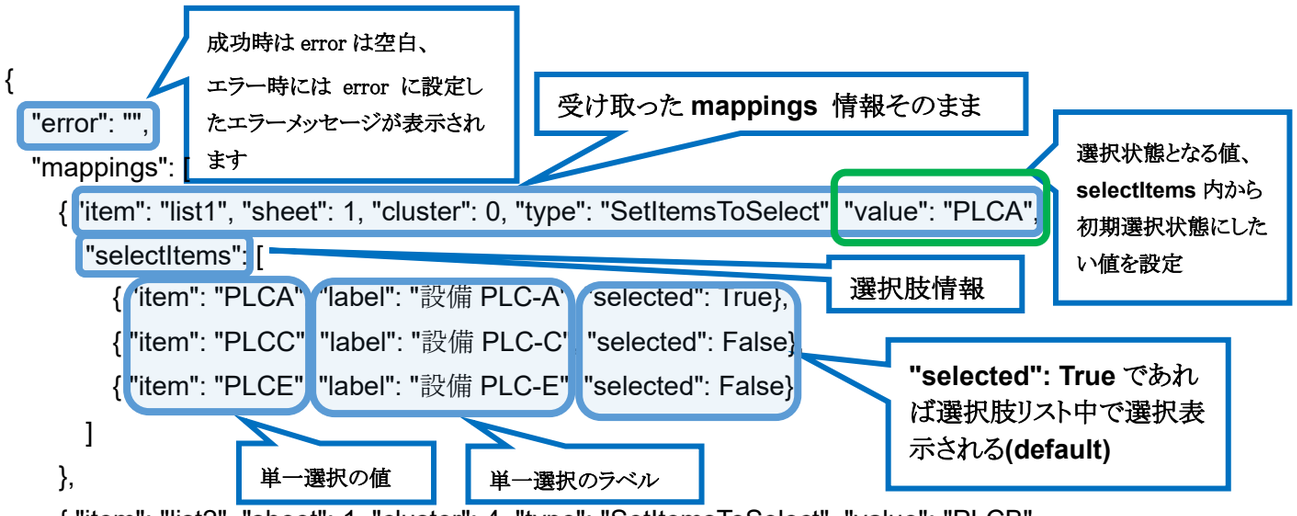
i-Reporter のシート番号

単一選択クラスターの場合は、SetItemsToSelect

デフォルト選択値、存在しないとエラー

【i-Reporter への戻値となる Python スクリプト用 JSON データ書式(成功時)】

Python スクリプト連携等の場合には、このような書式の戻値として返却することで、i-Reporter へ反映できます。
JSON での真偽値は false, true ですが、Python では False, True なので、コーディング時の便宜のために Python での表記にしてあります。




```
{
  "error": "",
  "mappings": [
    { "item": "list1", "sheet": 1, "cluster": 0, "type": "SetItemsToSelect", "value": "PLCA",
      "selectItems": [
        { "item": "PLCA", "label": "設備 PLC-A", "selected": True},
        { "item": "PLCC", "label": "設備 PLC-C", "selected": False},
        { "item": "PLCE", "label": "設備 PLC-E", "selected": False}
      ]
    },
    { "item": "list2", "sheet": 1, "cluster": 4, "type": "SetItemsToSelect", "value": "PLCB",
      "selectItems": [
        { "item": "PLCA", "label": "plc-a", "selected": False},
        { "item": "PLCB", "label": "plc-b", "selected": True},
        { "item": "PLCC", "label": "plc-c", "selected": False},
        { "item": "PLCD", "label": "plc-d", "selected": False},
        { "item": "PLCE", "label": "plc-e", "selected": False},
        { "item": "PLCF", "label": "plc-f", "selected": False}
      ]
    },
    { "item": "list1", "sheet": 1, "cluster": 8, "type": "SetItemsToSelect", "value": "PLCC",
      "selectItems": [
        { "item": "PLCA", "label": "設備 PLC-A", "selected": False},
        { "item": "PLCC", "label": "設備 PLC-C", "selected": True},
        { "item": "PLCE", "label": "設備 PLC-E", "selected": False}
      ]
    }
  ]
}
```

- 選択肢数は 1 個以上 1000 個以下まで
- 値とラベルと default の個数は一致していること
- 値とラベルと default はそれぞれ null 値は指定不可
- 同一の値は複数使用できない
- 同一のラベルは複数使用できない
- selected で True が指定できるのは1つのみ
- 単一選択クラスターの選択肢の「値」を数値として扱う設定の場合は戻値は数値変換可能なこと
- value で指定した値が select_value 内の item に存在すること
- value の空白指定は可能

第9章　マスター選択クラスターの連携先設定

【ConMas Gateway】サンプル定義(samples¥sample_custom_master フォルダ内)を利用して、マスター選択クラスターの連携先を ConMas Gateway に設定する方法について説明します。

ConMas Gateway 経由で、PostgreSQL、Microsoft SQL Server, Oracle, MySQL の DB 参照が利用可能です。ここではサンプル定義のシート No1　PostgreSQL について説明しますが、Microsoft SQL Server, Oracle, MySQL の DB 参照についてはシート No2～4及びサンプルフォルダ内の各設定を参照ください。

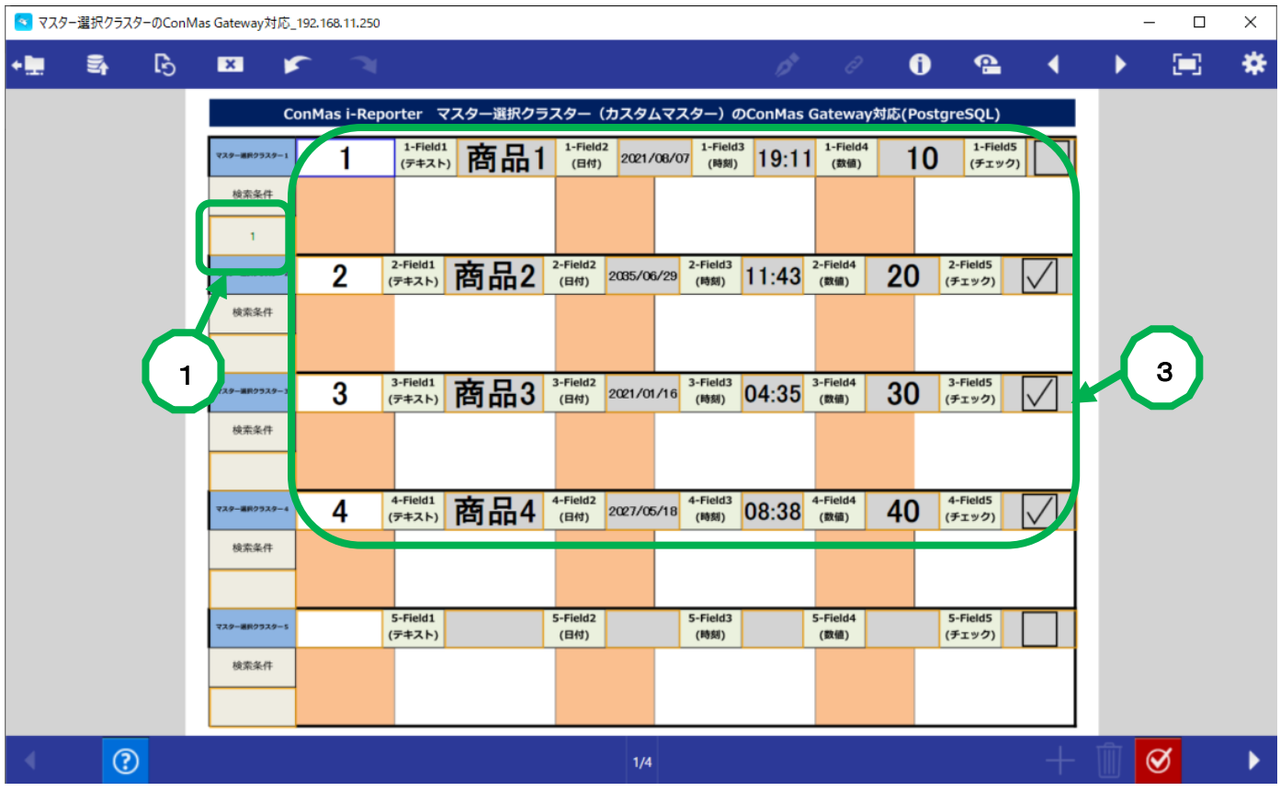


Fig.9-1 マスター選択クラスターの ConMas Gateway 連携

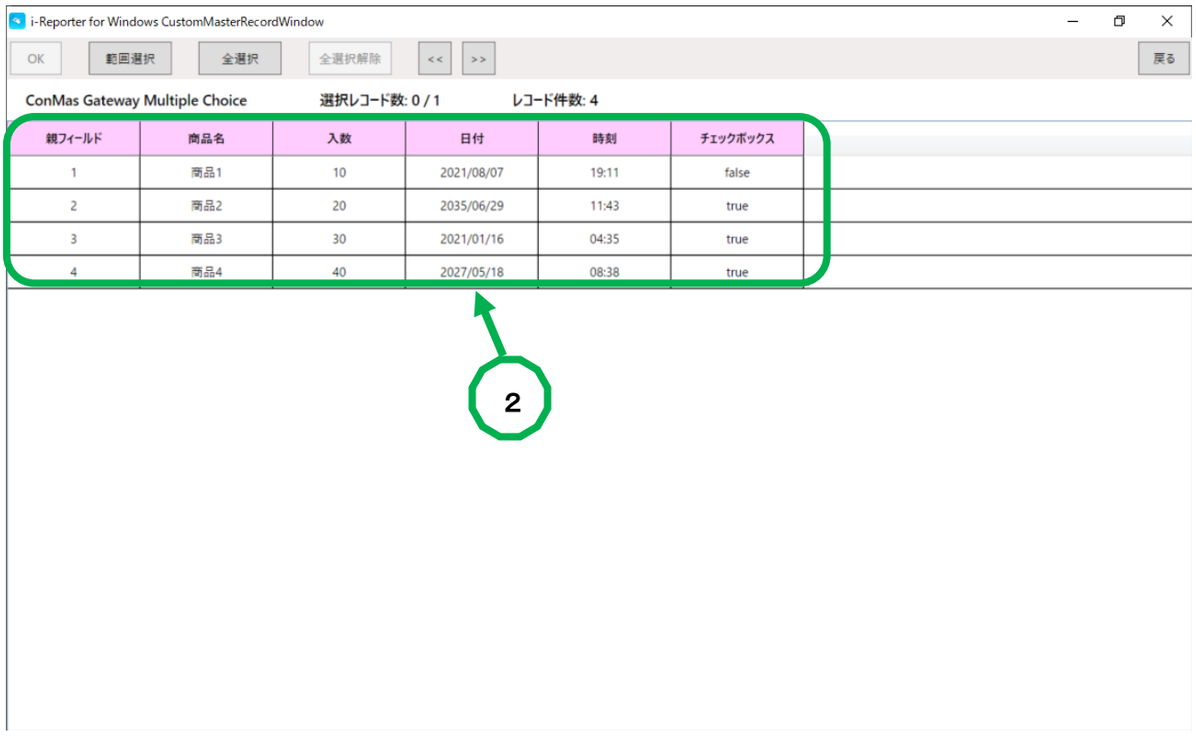


Fig.9-2 レコードー覧の選択

【動作概要】

1. 検索条件を設定。(※半角文字の 1～4 の範囲で入力してください)
2. 検索結果より設定するレコードを選択。(複数選択可)
3. カスタムマスター設定を実施している各クラスターに各フィールドの値を設定。

ⓘ 本サンプルではレコードを複数選択できるよう Designer で設定しています。
Designer の設定については下記マニュアルもご参照ください。
クラスター種別と入力パラメーター
https://cimtops-support.com/i-Reporter/ir_manuals/jp/designer/ClusterType_SettingProcedure_jp.pdf

ⓘ 検索結果のレコード数が default.json 内の customMaster 情報の limit を超えた場合はエラーとなります。

【マスター選択クラスター設定】

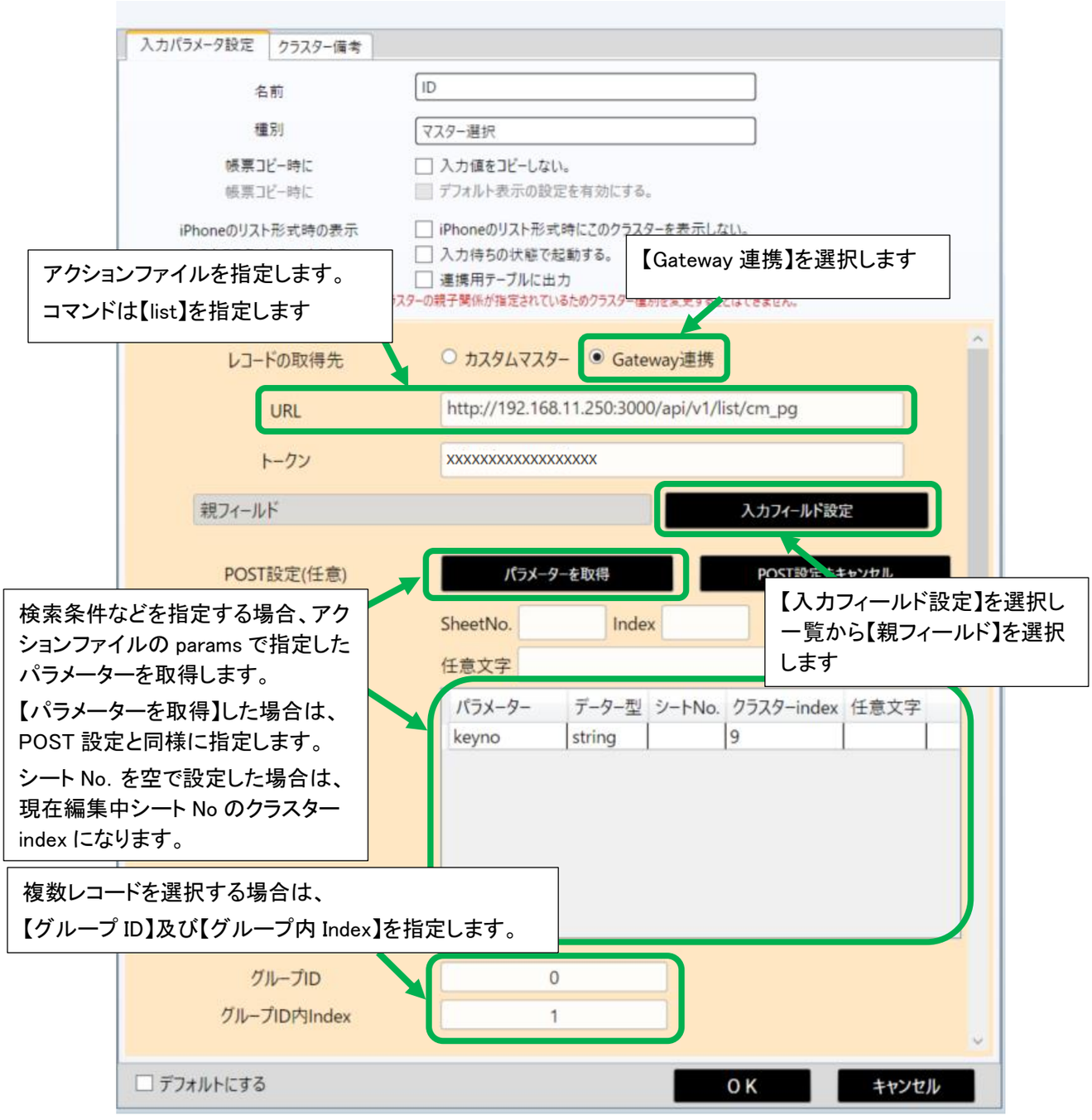


Fig.9-3 マスター選択クラスター入力パラメータ設定

アクション種別:　【Gateway 連携】

URL:　http://localhost:3000/api/v1/list/cm_pg

トークン:　xxxxxxxxxxxxxxxxx

1

localhost を ConMas Gateway をインストールしてある PC の IP アドレスに変更してください。

1. アクションフォルダ内の cm_pg.json を呼び出します。

URL 内にパラメーターは指定しません。

【アクションファイル設定(¥ConMas¥gateway¥actions¥cm_pg.json)】

```
{
  "datasource": "conmasgwdb",
  "basetype": "file",
  "basequery": "./sql/custom_master/pg.sql",
  "params": [
    { "name": "keyno", "type": "string" }
  ],
  "fields": [
    {
      "no": 1,
      "name": "親フィールド",
      "type": "numeric",
      "item": "SERIAL_NO"
    },
    {
      "no": 2,
      "name": "商品名",
      "type": "text",
      "item": "PRODUCT_NAME"
    },
    {
      "no": 3,
      "name": "入数",
      "type": "numeric",
      "item": "IN_AMOUNT"
    },
    {
      "no": 4,
      "name": "日付",
      "type": "date",
      "item": "IN_DATE"
    },
    {
      "no": 5,
```

default.json 内の"conmasgwdb"項目の設定が呼び出されます

/sql/custom_master/pg.sql ファイル内のクエリーが実行されます

Designer【パラメーターを取得】で渡す検索条件などのパラメーターを設定します。

マスター選択クラスターの「入力フィールド設定」およびカスタムマスターの親子設定時におけるフィールド指定の際、**fields** 情報は **Gateway** 連携にて取得を行います。

no:　1から開始される連番

name:　各フィールドのタイトル名

type:　**text**　テキストクラスター

numeric　数値クラスター

date　年月日クラスター

time　時刻クラスター

bool　チェッククラスター

(※現在のバージョンは画像クラスターは未対応です。)

item:　**item** は対応する **SELECT** 句の列名に合わせます。

```
            "name": "時刻",
            "type": "time",
            "item": "IN_TIME"
        },
        {
            "no": 6,
            "name": "チェックボックス",
            "type": "bool",
            "item": "WITH_TAX"
        }
    ],
    "defaultRecord": {
        "fields": [
            {
                "no": 1,
                "value": "-1"
            },
            {
                "no": 2,
                "value": "（商品無し）"
            },
            {
                "no": 3,
                "value": "0"
            },
            {
                "no": 4,
                "value": ""
            },
            {
                "no": 5,
                "value": ""
            },
            {
                "no": 6,
                "value": "false"
            }
        ]
    }
}
```

defaultRecord 情報は検索結果が0件の場合に返す値を設定します。

defaultRecord は必須ではありません。

defaultRecord が存在せず、検索結果が0の場合はアプリ側の一覧にはレコード0件で表示されます。

no: 1から開始される連番

value: 値

【SQL ファイル（¥ConMas¥gateway¥sql¥pg.sql）】

```
select
   serial_no
, col_text
, col_numeric
, to_char(col_date, 'YYYY/MM/DD') col_date /*  関数を使った際など、一意性が崩れるので別名必須  */
, to_char(col_time, 'HH24:MI') col_time
-- , col_image
, col_check
from custom_master
where key_no = to_number(:keyno, '999999')
order by order_no
```

【サンプルテーブル】

Data Output | Explain | Messages | Notifications

	serial_no [PK] integer	key_no numeric	order_no numeric	col_text text	col_numeric numeric	col_date date	col_time time without time zone	col_check boolean	col_image bytea	update_time timestamp with time zone
1	1	1	1	商品1	10	2021-08-07	19:11:00	false	[null]	2022-09-28 16:05:39.230707+09
2	2	1	2	商品2	20	2035-06-29	11:43:00	true	[null]	2022-09-28 16:05:39.230707+09
3	3	1	3	商品3	30	2021-01-16	04:35:00	true	[null]	2022-09-28 16:05:39.230707+09
4	4	1	4	商品4	40	2027-05-18	08:38:00	true	[null]	2022-09-28 16:05:39.230707+09
5	5	2	1	仕入先1	50	2035-04-22	13:41:00	true	[null]	2022-09-28 16:05:39.230707+09
6	6	2	2	仕入先2	60	2020-01-07	14:30:00	true	[null]	2022-09-28 16:05:39.230707+09
7	7	2	3	仕入先3	70	2022-06-01	17:47:00	true	[null]	2022-09-28 16:05:39.230707+09
8	8	2	4	仕入先4	80	2043-01-26	19:35:00	true	[null]	2022-09-28 16:05:39.230707+09
9	9	3	1	販売先1	90	2036-07-07	12:44:00	true	[null]	2022-09-28 16:05:39.230707+09
10	10	3	2	販売先2	100	2021-02-15	13:52:00	true	[null]	2022-09-28 16:05:39.230707+09
11	11	3	3	販売先3	110	2045-01-09	10:42:00	true	[null]	2022-09-28 16:05:39.230707+09
12	12	3	4	販売先4	120	2025-10-14	14:06:00	true	[null]	2022-09-28 16:05:39.230707+09
13	13	4	1	営業所1	130	2028-09-19	04:22:00	true	[null]	2022-09-28 16:05:39.230707+09
14	14	4	2	営業所2	140	2020-12-27	12:49:00	true	[null]	2022-09-28 16:05:39.230707+09
15	15	4	3	営業所3	150	2044-10-28	02:50:00	true	[null]	2022-09-28 16:05:39.230707+09
16	16	4	4	営業所4	160	2042-07-26	03:46:00	true	[null]	2022-09-28 16:05:39.230707+09

Fig.9-4 サンプルテーブル

第10章 使用上の注意事項

ConMas Gateway オンプレミス版は i-Reporter 標準機能です。

ConMas IoT（PLC 連携）はオプション機能です。

HTTPS 接続で ConMas Manager(i-Reporter Server)に接続する場合でも、
必ずしも ConMas Gateway に HTTPS 接続する必要はございません。
ConMas Gateway を AP サーバーとして扱い、リバースプロキシ経由のアクセスがお勧めです。

現在のバージョンでは、

- DB 参照は PostgreSQL、Microsoft SQL Server, Oracle, MySQL に対応
- 対応クラスター

キーボードテキスト、数値、年月日、時刻、画像、チェック、単一選択（※1）、マスター選択（※2）、バーコード（※3）

※1 単一選択クラスターは選択肢を含めた取得、Designer で設定した選択肢の両方で利用が可能です。Designer で設定した選択肢を利用する場合は、ConMas Gateway から“ラベル”でなく“値”を戻してください。バリューリンク設定されているクラスターには利用できません。

※2 マスター選択クラスターへ mappings で直接値を戻すことは iOS アプリのみ対応しています。
マスターレコードがローカル保存されていな場合は、カスタムマスターを利用できません。

※3 バーコード分解設定されている分解先クラスターに値は戻せません。

マスター選択クラスターのレコード取得先を【Gateway 連携】で設定する場合、カスタムマスター設定の親子関係で指定できるクラスター種類は、キーボードテキスト、数値、年月日、時刻、チェックになります。
画像クラスターには対応しておりません。

Python スクリプト連携のサンプルを実行するためには各種ライブラリが必要になります。
各サンプルに必要なライブラリについてはソースコードより確認をお願いします。
各サンプルは Python 3.10 で動作確認しております。

Python,PostgreSQL, Microsoft SQL Server, Oracle, MySQL など他社製品に関する個別のお問い合わせに関しては、弊社ではご回答ができない場合があります。

Oracle 接続時には、Oracle 社が提供する Oracle Instant Client が必要となります。ConMas Gateway インストール時、あるいは起動時に oracledb モジュール関連でエラーとなる場合、Node.JS のバージョンが新しすぎる場合があります。パッケージに同梱しているバージョンでご確認ください。

対応 Oracle DBMS サーバー: Oracle 10g(10.2.0.2)以降
必要な Oracle クライアント: Oracle Instant Client 11.2 以降

Microsoft SQL Server でマルチバイト文字を使用する列の照合順序がバイナリ系である場合に動作しないケースがあります。

PostgreSQL で、検索条件列が character であれば、type を char(10)あるいは character(10)など型設定する必要があります。（設定しない場合、末尾スペースが適切に比較されなくなります。）

MySQL は 5.7 及び 8.0 以降で動作確認しております。

default.json 及びアクションファイルなど ConMas Gateway が利用するファイルのエンコーディングは BOM 無しの UTF-8 で設定してください。

シート番号を削除した場合と削除しない場合のアクションファイル例です。

・シート番号を削除した場合

```
"mappings": [
  {
    "item": "user_name",
    "cluster": 1,
    "type": "string",
    "value": ""
  } ]
```

シート番号の指定が無く、クラスター番号のみ指定される場合はアクティブなシート（アプリで現在表示されているシート）のクラスター番号指定でマッピングされます。

・シート番号を削除しない場合

```
"mappings": [
{
  "item": "user_name",
  "sheet": 1,
  "cluster": 1,
  "type": "string",
  "value": ""
} ]
```

アクションファイルは、actions フォルダ配下のサブフォルダにも配置が可能です。フォルダ長の制限は、Windows プラットフォームの場合、フォルダ全体の長さの最大値は 247 文字であり、ファイルを含んだ最大のフルパスは 259 文字となります(2022 年4月現在)。

ConMas Gateway のバージョンは、**¥ConMas¥gateway¥package.json** 内の **version** プロパティとして記載してあります。

ConMas Gateway との SSL 通信については、default.json で https 通信の ssl パラメータを設定する方法と、上位のWEBサーバーでリバースプロキシを構成して Node.js をWEBサーバー上でホストする方法の2種類が利用可能です。

URL には RFC 3986 で制定された文字のみが使用可能となっております。
参照: https://www.ietf.org/rfc/rfc3986.txt
日本語や特殊記号をご使用の際には、JavaScript の encodeURI 関数を利用して、パーセントエンコーディングしていただく必要がございます。

年月日クラスターに対しては”YYYY/MM/DD”、時刻クラスターに対しては”HH:MM”といった書式をご使用ください。

トークン設定が default.son 内で設定しているトークン設定と違っており、認証エラーになった場合は下記メッセージが表示されます。

Content : {"result":{"code":-1,"description":"Authentication error."}}

ConMas Gateway に関する技術的なお問い合わせは、下記サポート WEB からお問い合わせが可能です。
https://cimtops-support.com/i-Reporter/ja/