## Summary

The ODMI application is a tool to allow DLC to expand the usage of their Openway data collection engine to additional users by allowing them to query meters.

## Components

The ODMI application will consist the following components:

### App

A web based application that will allow the user to search for addresses, accounts, and meters, and then query reading information from the meter.

### Middle Tier

The middle tier will be a set of web services that the application calls to perform searches and query the meters. It consists of a set of web services and a database to store the results.

### SOA

The middle tier will call services within the SOA to perform searches, then combine that with results from the database and return to the application.

### Openway

The middle tier will call services within Openway to retrieve information from the meters.

## Simulators

At a minimum one simulator will be used to simulate Openway responses. This will allow the middleware to be developed with controlled responses. In addition, we might make a middle tier simulator to allow development of the App to proceed, but probably will not need to.

## Standards

### Date Formats

### All dates will in the interfaces will follow ISO 8601 standards. Ie. 2019-07-22T20:16:00-05:00

## Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Screen Panel | Interfaces/Field |  |
| ESN |  |  | Used as key to meter in Openway |
| Search Field | SR |  |  |
| panelId |  |  | 2 or 3 digit id of the panels:   * SR – search results * RQ - requests * CI – customer information * MI – meter information * RI – reading information * AAG – at a glance * PR – ping requests |
| Customer Name | SR, CI |  |  |
| Meter Number | SR, MI |  |  |
| Address | SR, CI |  |  |
| City | SR, CI |  |  |
| State | SR, CI |  |  |
| Zip | SR, CI |  |  |
| Prev 5 Ping Dates & Times | SR |  | Date & Time of previous 5 pings. |
| Prev 5 Ping Request Id’s |  |  | Request id of previous 5 pings, can be used to request the results from the previous pings. |
| Meter Type | MI |  |  |
| Device Class | MI |  |  |
| Device Types | MI |  |  |
| Meter Form | MI |  |  |
| CCB Device Type | MI |  |  |
| MI Refresh Date & Time | MI |  |  |
| CI Refresh Date & Time | CI |  |  |
| Switch State | AAG |  | Open or Closed - Whether the load side voltage is on or not. |
| Ping Result | AAG |  |  |
| PR Refresh Date & Time | RR |  |  |
| Ph A Voltage | RI |  |  |
| KWH Reading | RI |  |  |
| KW Reading | RI |  |  |
| RI Refresh Date & Time | RI |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Interfaces

### Middle Tier

The interfaces to the middle tier will be REST format, passing parameters in the URL, and returning JSON.

#### Search

The search API will take a search string as input, and return results in JSON including an array of search results.

Parameters

* Search Field
* maxResults – maximum number of results per page, defaults to 25
* page – 1 based page count, used to retrieve further pages of results if the result set is greater than the maximum. Defaults to 1

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| totalResults | Integer | Total number of results in the query |
| numResults | Integer | Contains the number of results returned |
| results | Array | Array containing one object for each result, the names of the fields within the object would match the column names defined in the utilities schema. |
| columnName1 | String |  |
| columnName2 | String |  |
| … | String |  |
| columnNamexx | String |  |
|  |  |  |

#### Requests

This API will return requests that have already been initiated, either by this user or all users.

Parameters

* userid – guid of user (required)
* requestId – if requestId is sent, then that request will be the only one returned. Otherwise the following fields will be used to filter results
  + ownership – either ‘mine’ or ‘all’, defaults to ‘mine’ if not supplied
  + period – either ‘last10’, ‘today’, or ‘custom’. Defaults to ‘last10’. If custom than startDate and endDate are required.
  + startDate – start date if period = ‘custom’
  + endDate – end date if period = ‘custom’
  + maxResults – maximum number of results per page, defaults to 25
  + page – 1 based page count, used to retrieve further pages of results if the result set is greater than the maximum. Defaults to 1

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| totalResults | Integer | Total number of results in the query |
| numResults | Integer | Contains the number of results returned |
| results | Array | Array containing one object for each result, note that this data is snapshotted when each request is made. |
| columnName1 | String |  |
| columnName2 | String |  |
| … | String |  |
| columnNamexx | String |  |
| customerInfoRefreshDate | String | ISCO 8601 UTC |
| prevPings | Array | Array containing objects for the previous 5 pings |
| requestDate | String | ISO 8601 UTC |
| requestId | String | Unique Identifier (guid) of the previous ping |

#### Request

The request API will take an ESN as input, and return results in JSON. There will be additional parameters asking for which web services the Ping should initiate. The web service will initiate the specified calls to Open Way and returns JSON with the request id’s to use to retrieve the data.

Parameters

* ESN (string) – ESN that should have the following web services called
* performPing (char) – Y or N, defaults to Y
* performVolt (char) – Y or N, defaults to Y
* performRead (char) – Y or N, defaults to Y

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| requestTime | String | ISO 8601 Date format |
| requestId | Guid | Id that can be used to look up information about the request. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### Ping

The Ping API will take a requestId as input, and return results in JSON. If the results are not available yet, the API will wait until they are available, or a configured timeout.

Parameters

* requestId – guid to the request id, which will be used to look up the pingId.

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| pingDateTime | String | ISO 8601 Date format – the time the ping was completed in Openway |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### Voltage

The Voltage API will take a requestId as input, and return results in JSON. If the results are not available yet, the API will wait until they are available, or a configured timeout.

Parameters

* requestId – guid to the request id, which will be used to look up the voltId.

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| voltageDateTime | String | ISO 8601 Date format – the time the voltage request was completed in Openway. |
| switchState | String | Open or Closed – whether the load side has voltage or not. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### Readings

The Readings API will take a requestId as input, and return results in JSON. If the results are not available yet, the API will wait until they are available, or a configured timeout.

Parameters

* requestId – guid to the request id, which will be used to look up the readsId.

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| readsDateTime | String | ISO 8601 Date format – the time the reading request was completed in Openway |
| Readings | Array | An array of reading objects |
| regName | String | Name of the reading |
| regValue | String | Value of the reading |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Note that we need to have DLC run the readings API to see what actual results are returned. And whether the phase a voltages are in the readings.

#### Schema

The Schema API will take a company name as input, and return results in JSON.

Parameters

* Company – the company identifier to return the schema for. This will generally be a short name for the company, ie. DLC for Duquense Light Company

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| schema | Object | JSON object representing the schema. See the schema section below for definition. |
|  |  |  |

#### Authenticate

The Authenticate API will return security information and configuration. This is not authenticating against active directory, as the app will do that outside the middleware. It just receives information from the middleware about this user.

Parameters

* company – the company identifier to return the schema for. This will generally be a short name for the company, ie. DLC for Duquense Light Company
* userId – the user id for the user
* roleName – the roll name that active directory returned for the user.

Output

|  |  |  |
| --- | --- | --- |
| status | String | Status of the search, values include:   * Success * Error |
| errorMessage | String | If status is Error, then contains the error message. |
| roleSecurity | Object | JSON object representing the roles security |
| roleName | String |  |
| adminRoles | Boolean |  |
| adminSchema | Boolean |  |
| roleConfig | Object | Object representing the screen configuration and availability of columns for the role, see the section on role configuration. |
| userConfig | Object | JSON object representing the users configuration, see the section on user configuration for a description. |
|  |  |  |

### SOA

This section will contain information on the services within SOA that will be used to perform the search and retrieve customer and meter data.

### Openway

This section will contain information on the services within Openway that will be used to retrieve information from the meters.

## Middle Tier

The middle tier will be a C# Web API using REST format web services. The basic flow will be this:

* Openway Requests
  + Request is initiated by ODMI App
  + The request will be entered in the database.
  + For each web service that the request needs, OPENWAY will be called, the request id’s for retrieving results will be added to the request in the database.
  + Threads will be spawned for each web service to request the results from Openway.
  + When the results arrive they will be added to the database as documents.
* Results
  + When a request comes in for results, the database will be queried and the results returned. If the results are not yet available, then the web service will wait for the results or a timeout period.
* Search Requests
  + The results will be returned either from database lookup or a web service call that has to be defined. Need info from DLC.

## App

### Config JSON

Each role will have configuration that defines the panel order and size, along with which fields are available to that role. Each user will then have their own configuration that defines the same characteristics for the user. The JSON format is:

|  |  |  |
| --- | --- | --- |
|  | Array | The JSON will be an array containing objects for each panel. Each panel will include information as well as an array of fields for that panel. Each field will include whether it is visible, and the order of the array will control the order that the fields are displayed. |
| panelId | String | If status is Error, then contains the error message. |
| results | Array | Array containing one object for each column that is part of this panel. The order of the items in the array is the order in which they are to be displayed. |
| columnName | String |  |
| columnVisible | String |  |
|  |  |  |

Note that columnName would match either the name of the column from the utilities schema or one of the following internal column names:

* esn
* ping.result
* voltage.switch\_state
* readings.<Reg Name> - where <Reg Name> is one of the register names returned by the readings API. Ie. readings.kwh or readings.kw

### Screen Panels

The final version of the application will consist of multiple screen panels that the user will have the capability to arrange. The POC will have a fixed layout using the same screen panels.

#### Search (SR)

There will be a google like search bar that allows the user to enter text. % can be used as a wild card, and whatever text is entered will have a % added on each side when the search is run. The following fields will be searched.

* Meter Number
* Account
* Street Name
* Customer Name ( future )
* Primary phone number ( future )

Integrated search bar should have three small buttons at the end of the search box that is integrated into box. Button 1 = search (Blue Magnifying glass icon) Button 2 = Grey clock icon to show last five search strings Button 3 = A button with a RED X to clear all the previous results that are listed below. Make it easy with an X in the edit box to clear the text entered without hitting backspace. Hitting enter should start the search. If the search returns only one result a PING request should automatically initiated.

This panel will displays the results of the search above. Results displayed include:

* Customer Name
* Meter Number
* Address
* City, State, Zip
* There would be a dropdown with an action for each result. Currently only the ping action will be supported. In the future we might have further actions. The dropdown button will be replaced with a ping graphic most likely.

#### Request (RQ)

This panel will display the current and past requests. The user can choose between seeing just their requests or all requests (probably include security in the future on this so it would be all requests that they have the right to see).

* Customer Name
* Meter Number
* Address
* City, State, Zip
* Status – In Progress, Complete, or Failed
* Dropdown showing last 5 ping dates & times

#### Meter Information (MI)

This component displays information about the meter including:

* Meter Number (Badge Number)
* Meter Type
* Device Class
* Device Types
* Meter Form
* CCB Device Config Type
* Meter Info Refresh Date & Time

#### Customer Information (CI)

This component displays information about the customer including:

* Customer Name
* Account Number
* City
* State
* Zip
* Phone
* Customer Info Refresh Date & Time

#### At a GlancePing (AaG)

This panel displays information about the latest ping including:

* Switch State – displayed as one of 2 icons, open or closed
* Ping Result – displayed as Success in Green or No Response in red.
* Voltage – displayed as a gauge showing min, actual, and max voltages. With anything under min or max being in the red range. If actual is in between min and max, then it will be in the green range.
* kWH and kW readings

#### Reading Information (RI)

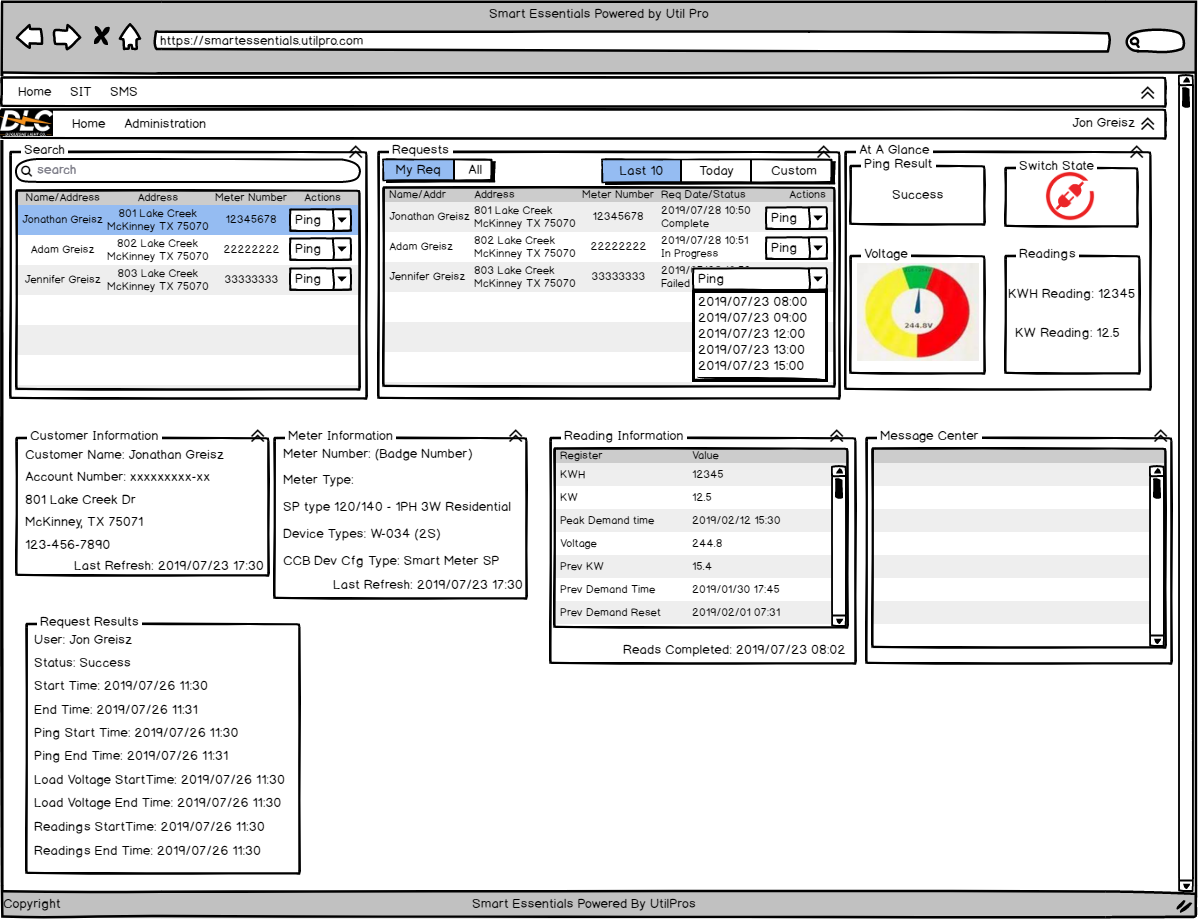
This panel displays the latest reading information for the meter using a data grid including register name and value:

* KWH Reading (Delivered)
* KW Reading (Delivered)
* Peak Demand Time & Date
* Voltage Phase A
* Power Factor
* Previous KW Reading
* Previous Max KW Time & Date
* Previous Demand Reset Time & Date
* KWH Reading (Received)
* KW Reading (Received)
* Peak Demand Received Time & Date
* Previous KW Reading (Received)
* Previous Max KW (Received) Time & Date
* Previous Demand (Received) Reset Time & Date
* Reading Information Refresh Date

Note that all of the above register values need to be confirmed as available. The users role will define which of these elements are available, but the user should have the capability to decide which of the available ones they want to see, and the order.

### Wireframe

Below is a first pass at a mockup. Note that all the panels will be movable by the user, as well as controlling what content is displayed. When the search is performed, the results will be displayed in the view on the left. Each result will have a ping button that can be pressed. When pressed the meter will be moved to the requests panel. The ping button on the requests panel will be a dropdown that can be used to display the results from the last 5 ping attempts.



### Headers

There will be 2 header lines, both of which will include arrows to close the header and make more room.

#### Header 1

Header 1 is the top band of the Page, which holds the module names. Currently we have two modules on SIT (Smart meter interrogation tool) and SMS (Smart meter simulator) . We will add more and more modules as the product grows. By clicking on the module name the user will be see the main page of that module. This bar also will have a Home option to get to the home page of the Smart essentials. Right now, we only have one page and pressing the home button will put on that one page.

#### Header 2

Header Band 2 is a band for the selected module, which may show the main menu items for that module that they are working with. For now it will consists of these

1. Customer Name / Logo
2. Home ( which will bring it to the home of the module )
3. User name who logged in with an arrow and drop down menu that will have user related functions such as logout etc.
4. Date and time stamp ( System date)

#### Footer 1

This is the last band of the page, which shows copyright information in the left and product logo (Smart Essentials Powered by UtilPros logo. The logo can be a link to the UtilPros website,

### Administrative Menu

The administrative menu will only be visible to users that have a role granted access to at least one administrative function. The functions will be menu items that show up if the users role includes that function. They include:

* Schema Definition – allows the user to define which columns the utility uses, and which columns are part of the search.
* Role Maintenance – allows the user to define whether roles have access to role maintenance, schema definition, as well as panel definition for that role.

## Database

The database will be a SQL server database consisting of tables for requests as well as the results. It will be fleshed out as we define the storage requirements. Currently the thought is that all results will be stored as JSON documents. Below is a quick conceptual model.

### 

### Utility Data

In order to make the system usable by multiple utilities, utility data will be stored in a generic format. Each utility will have a schema that defines the format, then each record will be stored as JSON, along with some index columns for searching.

#### Utility Data Schema

The utility data schema will be JSON that defines column data, labels, and which columns are indexed. There will be an administrative menu item for administering the schema. JSON format is below.

|  |  |  |
| --- | --- | --- |
| lastUpdateTime | String | ISO format datetime |
| indexCol1Name | String | Column name to use for 1st index |
| indexCol2Name | String | Column name to use for 2nd index |
| indexCol3Name | String | Column name to use for 3rd index |
| indexCol4Name | String | Column name to use for 4th index |
| indexCol5Name | String | Column name to use for 5th index |
| columns | Array | Array containing one object for each column |
| columnName | String |  |
| columnLabel | String |  |
|  |  |  |

#### Utility Data

The utility data will be stored in a table that includes 5 indexed columns, along with the ESN to use for that meter, and a JSON field that stores the generic data. The schema for the utility will be used to extract the data for the indexed columns and store it within those columns. When searches are run, the system will UNION the search on the results of like comparisons with each of the data elements.

##### Utility Data JSON

The actual data for each utilities meter will be stored as JSON with a flat structure, as an object with column names and data. Each data value will be a string. Example:

{

“meterId” : “12345678”,

“account” : “10-1234567”,

…

“fieldxx” : “Value”

}

### Readings Data

The readings data will be stored in JSON as an array of objects. Each object will have a regName and regValue.

[

{ “regName” : “kwh”, “regValue” : “12345”,

{ “regName” : “kw”, “regValue” : “12.5”

]

## Security

The system will use a role based security system. Active Directory will be used for authentication, picking up the users login credentials from windows, then validating that they are in a role that includes access to Smart Essentials. The role would then be used to determine what features that they have access to within the application.