English

Asema IoT Central

Concept demos manual

ASEMA

Table of Contents

1. Introduction

Asema IoT Central lets you create web apps that use adaptive pricing, i.e. custom logic for pricing your products and services. For example, you can vary the ice cream price depending on the weather, or raise the price of the cinema tickets as the seats are gradually sold out. Which price depends on what measurement is up for you to decide and the Asema IoT pricing design tools let you design these correlations.

The Adapt concept demos are made to illustrate in practice how you could embed adaptive pricing in the functionality of your website. The source for the demo applications is included, making it possible to modify and expand them from simple re-branding to full-blown logic customization.

There are a total of eight demos available:

- Art Museum – a museum that prices exhibition tickets dynamically depending on the amount of pollution in the air in the city.
- Diva beauty salon – a salon that sets service prices dynamically depending on plastic recycling success.
- Acme Parking parking operator – a parking firm that offers dynamic discounts on parking, increasing the discount as people park their cars and walk instead.
- Duckston – a town that sells tickets to cultural events at dynamic discount depending on how many people choose to ride a bike instead of taking a car.
- Wish clothing store – an e-commerce site that offers dynamic discounts on clothing when more clothes are received by recycling centres.
- Luigi's pizza – an Italian restaurant that supports city biking with a special dynamically priced dessert menu.
- Sunset Canoes – a canoe rental where the rental prices depend on how far you want to take the canoe.
- Marco's Burgers – a fastfood restaurant that attracts people from further away by offering a discount on pre-orders that is higher when the order is made from far away.
- Ice Cream Stand – an ice cream sales site where the price of goods depends on the weather.

These demos show the use of adaptive pricing in a range of business environments from renting equipment to selling food and offering beauty services. Do note that the demo cases are illustrative and concept-only: they just offer ideas on how to use such pricing in various business contexts. Some of the concepts may not be applicable to real life businesses without customization and tweaking as well as safeguards for ensuring that prices cover sufficient sales margin in various situations. These are however applied to case-by-case when the pricing is taken into use. The pricing tools of Asema IoT let's business managers define and enforce such safeguards whenever needed.

> **Note**
> You may study and adapt the source of the concept demos as you please. However, note that most of them have graphics and layout that are based on Creative Commons 3.0 license from Colorlib. If you use the code as such for commercial purposes, you do need to purchase a commercial license from Colorlib.

1.1. Runtime environment

With the exception of the Ice Cream Stand, all concept demos run with the HTTP server of Asema IoT Central. All you need to test and customize these demos is Asema IoT Central software. The Ice Cream Stand demonstrates the use of node.js, which in this case serves the demo website. To use it, you will need a node.js instance and nginx proxy server.

All demos are installed the same way, except for Ice Cream Stand, due to differences in the used techonologies.

> **Note**
> Note that the free demonstration license of Asema IoT Central is enough to try out the demos. However, as there is a limit on the number of objects in the demo license, you may not be able to configure all the demos at the same time. So with the demo license you can run **any** of the demos but not **all** of the demos.

1.2. Functionality of the demos

Each demo presents the use of adaptive pricing where a price of one item depends on the state of some other item. In some demos there may be multiple elements on both sides, i.e. the prices of multiple items depend on the states of other items.

With the exception of the Ice Cream Stand, which is built with the React components, the demos use the templating system of Asema IoT Central. Both approaches work with Asema IoT Central equally well and the choice between them depends on the implementation goals. A component approach gives superior code control and reusability especially in modern single-page applications. Then again, planning and designing such an approach may take more time. Templating works well on multi-page "traditional" HTML5 websites which are fast to assemble using templates. The speed of getting things running with plain HTML5 then comes with the expense of code management later.

Each demo features a top bar for simulating the object values. This way the effect of measured properties can be easily demonstrated and played around with. The use of the simulation top bar is optional: the demo works equally well by changing the same values with the object dashboard of Asema IoT Central admin interface, by connecting the objects to actual measurement devices, or by changing values with the JSON and Smart API interfaces. Either way, changing the property values the prices depend on will change the prices on the demos.

If you use the simulation top bar, there are two types of items the demo needs to load when the page is loaded into the browser:

- The objects that drive the prices. These are loaded into the top bar for simulation purposes.
- The objects that are priced. These are loaded into the actual demo content.

If you don't use the top bar, naturally you only need the latter.

1.3. Using the demos

Each demo application uses a certain set of objects and specific template. You can find these details in the 3, Configuring the demos chapter. However, the procedure for using the demos is the same:

1. Install any required additional software components i.e.
   - Asema IoT Central (required by all demos)
   - node.js ja nginx (only applies to React/node.js demos)
2. Copy the demo source code into required locations (either to Asema IoT Central or node.js).
3. Configure the objects in Asema IoT Central.
4. Add the identifiers of the classes you've created to the demo templates.
5. Tweak the look and feel of the demo and expand it as you please.

1.3.1. Copying demo code into place

To copy the demo source code:

1. Find the common top template files `iotccontrolpanel.html` and `iotcmappanel.html` and copy them to the templates directory of your installation, which is usually `public_html/templates` (see 2, Installing the demos for details).

2. Find the folder of the concept demo itself. Then,

   - If it is a standard templated application, copy it to the `public_html` folder of your installation.

   - If it is a React component application, create a new application with node.js, then copy the files of the demo to the folder **of this node.js application** (not to Asema IoT Central).

1.3.2. Configuring the objects

The number of objects and their IDs depend on the specific demo. You will find those details in 3, Configuring the demos chapter that outlines the details for each demo. For specifics on how to add and configure objects in Asema IoT Central, read the corresponding chapter in Asema IoT Central manual.

To configure and add the objects:

1. Create objects that drive the price.

   First, create the object schema: in Asema IoT Central, go to Objects > Generic objects > Generic schemas. The instructions of each demo in 3, Configuring the demos will tell you what properties this schema should have.

   Next, create an object under Objects > Generic objects > Generic objects . Give it the identifier required by the particular demo (see 3, Configuring the demos). Link it to the schema you just created.

2. Create a price list.

   First, create a price component (or components) under Pricing > Price components > Property dependent. In the price component, select the object and the property you just created. Then set thresholds to the price (the ones you find appropriate).

   Next, create a price list under Pricing > Price lists. Name it and add the created components(s) to it.

3. Create the objects that are priced. You'll find the number of objects for each demo in the demo description (see 3, Configuring the demos). The generic object type is just fine for this task.

4. Assing the price list to the objects you just created. In Objects > Configure, find your object and click the gear icon in the Pricing column. Choose Charge a price and then Variable price from a pricelist. Then choose the price list you created from the dropdown.

All done! Now the necessary configuration steps have been taken.

> **Important**
> Remember that you always need to create three types of objects:
>
> 1. An object that drives the price
>
> 2. A price list object that depends on the object that drives the price
>
> 3. An object that the price list is attached to i.e. the object that gets the actual price.

1.3.3. Loading the objects to simulate

> **Note**
>
> The objects to load to the demo depend on the demo and the configuration you've made. This section describes the procedure, as it is commmon for all demos and works the same way with all the demos. The specifics of each demo such as identifier are explained in the last chapter.

To load the simulated objects to the demo, edit the header/footer template of the demo. You can find it in the `templates` directory of each demo. At the end of the template, there is a section that looks like this:

```
{% block controlpanelitems %}
 var iotControlPanelItems = {
  byComponentId: {
   "no2Emissions": {
    properties: ["particles"]
   }
  },
  byGid: {

  }
 }
{% endblock %}
```

This section defines the `iotControlPanelItems` variable with lists of items to load. They are recognized either by their component ID or their GID. `byComponentId` contains items to be loaded by component ID. `byGid` contains those to be loaded by GID. Each identifier is a key in the respective directory.

For example, to load three items, two of them with component IDs "my_compo_1" and "my_compo_2" and one with GID "abcd1234", edit the `iotControlPanelItems` variable to look like this:

```
var iotControlPanelItems = {
 byComponentId: {
  "my_compo_1": {
   properties: []
  },
  "my_compo_2": {
   properties: []
  }
 },
 byGid: {
  "abcd1234": {
   properties: []
  }
 }
}
```

The configuration above only loads the objects but does not yet create any controls in to the simulation top bar. To do so, you still need to add info on which properties of those objects you want to simulate and control. `properties` is a list of the properties for which a slider should be created as a controller of the values.

For example, to control the `height` and `width` properties of `my_compo_1`, add them to the properties list like this:

```
var iotControlPanelItems = {
 byComponentId: {
  "my_compo_1": {
   properties: ["height", "width"]
  },
  ...
}
```

1.3.4. Loading the objects to display and price

The objects that are priced are loaded by one or several of the demo pages. This is demo-dependent (see 3, Configuring the demos for details). However, again, all demos use the same procedure which is the Jarvis library and its `ObjectManager`.

In the templated demos, one `ObjectManager` is created by the main template that does the simulation. There is no need to create another manager: assigning callbacks to the main one is enough. The callback will run when the `ObjectManager` is ready to act and load your objects. The code for making and registering callbacks looks like this:

```
var onCommonLoadedObjectsReady = function(propertyName) {
 // object processing code goes here. This is initiated once the object has loaded.
}

var onCommonManagerReady = function() {
 // object loading code goes here. This is initiated first.
}

var objectManagerReadyCallbacks = [onCommonManagerReady];
var objectManagerObjectsReadyCallbacks = [onCommonLoadedObjectsReady];
```

Both the loading code (`onCommonManagerReady`) and the processing code (`onCommonLoadedObjectsReady`) are already included in the demos. If you use the identifiers from the 3, Configuring the demos chapter, you don't need to do anything. However, if you use different identifiers or would like to change what happens once the objects load (e.g. change the items that are generated and their markup), then this is the code to modify.

When the common `ObjectManager` is ready, invoke the load commands for the objects. For example, to load three objects, two of them with component IDs and one with a GID, the code would look like this:

```
var onCommonManagerReady = function() {
 iotControlPanelObjectManager.findObjectByComponentId("firstObject", "myFirstObjectIdentifier");
 iotControlPanelObjectManager.findObjectByComponentId("secondObject", "mySecondObjectIdentifier");
 iotControlPanelObjectManager.findObjectByGid("thirdObject", "abcdefgh12345");
}
```

Once the object loads, it can be rendered. The demo components take the object as an argument and then render its content. This approach makes it easy to react to the changes in prices and other properties and render the changes on screen. This is what creating components can look like:

```
var onCommonLoadedObjectsReady = function(propertyName) {
 if (propertyName === "firstObject") {
  var obj = iotControlPanelObjectManager.objects[propertyName].list[0];
  var item = new RenderedBoxFromItem(obj);
  $("#items").append(item.getMarkup());
 }

 else if (propertyName === "secondObject") {
  var obj = iotControlPanelObjectManager.objects[propertyName].list[0];
  var item = new RenrederCircleFromItem(obj);
  $("#items").append(item.getMarkup());
 }
}
```

`RenderedBoxFromItem` is some JavaScript function that invokes HTML5 or React component rendering.

1.4. Viewing the demos

The concept demos, expect for the Ice Cream Stand, install as web applications into Asema IoT Central. To view the actual demo, you need to make sure Asema IoT Central is running and then access the demo with a web browser. Assuming you've installed the demo in the public_html forlder off you installation,

the demo is available at the `pub` path of the installation. Do for example, if you copied the "Wish" clothing store to a subdirectory called "Wish" and you run Asema IoT Central in localhost at default port, you can see the demo at http://127.0.0.1:8080/p/applications/pub/Wish/

Note that if you've copied the `iotwebapp.xml` file with the demo, Asema IoT Central will also list the demo automatically in the Applications section of the web admin interface. So in Asema IoT Central web admin go to Applications > Web apps and then click on the eye icon to view the demo.

The default configuration of the Ice Cream Sales app assumes that you have the nginx web proxy installed. To view this demo, you need to set up nginx properly and then point your browser to the port where nginx runs. For example, if nginx runs in port 8000, then the demo can be viewed at http://127.0.0.1:8000

2. Installing the demos

2.1. Installing the files for plain vanilla JavaScript demos

The files for the plain vanilla demos comprise: the template files (common to all demos) and the demo files (files for each individual demo).

Once your unpacked the demo package, you find the templates under `common/controlbar/`. Copy `common/controlbar/iotccontrolpanel.html` and `common/controlbar/iotcmappanel.html` to the `public_html/templates` directory of your Asema IoT Central installation.

Next, install the web application files. To do this, all you need to do is copy the files of the concept demo to the `public_html` directory of your Asema IoT Central installation.

2.2. Installing the React demos

To run the React demos, you first need the required software i.e. node.js and nginx. Installing these will depend on the type of operating system you are using. On most Linux distributions they are both available in the package managers. For other operating systems, follow the instructions and downloads available at https://nodejs.org/en/ and https://www.nginx.com/.

Once you have the software installed, do the necessary configuration. For nginx, you need to edit the nginx.conf to property forward 1) calls to Asema IoT Central libraries, 2) calls to Asema IoT Central websocket server, 3) calls to the node.js HTTP server, 4) calls to the node.js websocket server. Assuming you run node.js in port 3000, the example configuration below does the trick.

```
server {
        listen       8000;
        server_name  localhost;

        location / {
            proxy_set_header X-Real-IP  $remote_addr;
            proxy_set_header X-Forwarded-For $remote_addr;
            proxy_set_header Host $host;

            proxy_buffering off;
            proxy_pass http://127.0.0.1:3000;
        }

        location /sockjs-node {
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_pass http://127.0.0.1:3000;
        }

        location /iot {
            proxy_set_header X-Real-IP  $remote_addr;
            proxy_set_header X-Forwarded-For $remote_addr;
            proxy_set_header Host $host;

            proxy_buffering off;
            proxy_pass http://127.0.0.1:8080;
        }

        location /json {
            proxy_set_header X-Real-IP  $remote_addr;
            proxy_set_header X-Forwarded-For $remote_addr;
            proxy_set_header Host $host;

            proxy_buffering off;
            proxy_pass http://127.0.0.1:8080;
        }

         location /ws {
                proxy_pass http://127.0.0.1:8081;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
        }
    }
```

Once the configuration has been changed, restart nginx.

Next, install React and create an empty application. To do so, run the following commands in a terminal

```
> mkdir nodejs
> cd nodejs
> npm install react
> npm install create-react-app
> npx create-react-app icecreamapp
```

Once the steps above are finished, find the nodejs/icecreamapp folder and two of its subfolders "public" and "src". Replace the contents of these folders with the contents found in the demo samples package for Ice Cream Sales app.

Once done, start node.js

```
> cd icecreamapp
> npm start
```

Once you have all the three pieces of software running (Asema IoT Central, node.js, nginx), the demo will be available at http://127.0.0.1:8000, assuming you used the nginx configuration as shown above.

3. Configuring the demos

3.1. Art Museum (Adapt/artmuseum)

The objects are loaded and used by the `ticket.html` file.

Objects that drive prices: To set the pricing, create an object with "no2Emissions" componentId. Assing it a schema with the "particles" property.

Objects that are priced: To display pricing, create an object with the "cheap" componentId and an object with "expensive" componentId.

Once the objects are created, create a price list and link it to the cheap and expensive objects as explained in the previous chapter.

3.2. Diva (Adapt/beautysalon)

The objects are loaded and used by the file `services.html`.

Objects that drive prices: To set the pricing, create an object the "plasticRecycling" componentId. Assing it a schema with the "weight" property.

Objects that are priced: To display pricing, create an object with the "beautyservice" componentId.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.3. Acme Parking (Adapt/carparking)

The objects are loaded and used by the file `index.html`.

Objects that drive prices: To set the pricing, create an object with the "duckstonCyclists" componentId. Assing it a schema with the "count" property.

Objects that are priced: To display pricing, create an object with the "park1" componentId and an object with the "park2" componentId.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.4. Duckston (Adapt/city)

The objects are loaded and used by the file `events.html`.

Objects that drive prices: To set the pricing, create an object with "duckstonCyclists" componentId. Assing it a schema with the "count" property.

Objects that are priced: To display pricing, create an object with the "speakers" componentId, an object with the "lecture" componentId and an object with the "music" componentId.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.5. Wish (Adapt/clothing)

The objects are loaded and used by the file `index.html`.

Objects that drive prices: To set the pricing, create an object with the "electricityConsumption" componentId. Assing it a schema with the "energy" property.

Objects that are priced: To display pricing, create an object with the "clothingDiscount" componentId.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.6. Luigi's (Adapt/restaurant)

The objects are loaded and used by the file `index.html`.

Objects that drive prices: To set the pricing, create an object with the "foodWaste" componentId. Assing it a schema with the "weight" property.

Objects that are priced: To display pricing, create one custom metatype class and give it the "demo:dessert" ID. Then create any number of objects that represent the dessert food items. Edit their metadata and assing each of them to the "demo:dessert" class.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.7. Sunset Canoes (CanoeRental)

The objects are loaded and used by the file `index.html`.

Objects that drive prices: To set the pricing, create an object with the "rentalCanoe" componentId.

Once the objects are created, create a price list and link it to the objects as explained in the previous chapter.

3.8. Ice Cream Stand (IceCreamSalesApp)