

Introduction to oFMS Features

Hannu Järvinen
hannu.jarvinen@tml.hut.fi

January 2, 2007

1 open Facility Management Server

Open Facility Management Server offers oBIX services over HTTP. It is developed to enable legacy equipments to join the network of equipments. It has been build according to oBIX specification [1] and it implements many of the functionalities defined.

The idea of oFMS is to offer an oBIX interface to equipments, user agents and other oBIX servers. Server is basically just storing information in oBIX format and offering an interface for accessing it.

The oFMS is an HTTP server for storing oBIX information. HTTP clients communicating with the server can be user agents or equipment adapters. All the clients are able to do the same things by using the services provided by oFMS. They can add their own data to the oFMS, they can read data, write data, or invoke operations. All the main services available can be read from the Lobby object in a server. Lobby object address is the only thing that client has to know when connecting to the server. It tells all the other important addresses.

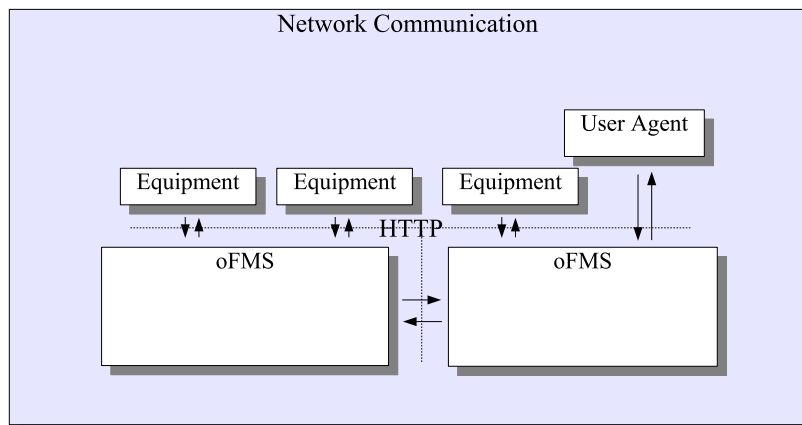


Figure 1: Network Communication

2 HTTP Methods

Server is using HTTP -protocol for network communication. HTTP defines GET, PUT and POST methods. In oBIX these three methods are mapped to Read, Write and Invoke Requests.[1]

2.1 Read

Read request is used to read any object which has an address. This means an object that has an href.

2.2 Write

Write request is used to change any object that has an href and writable tag is true. Equipment can thus modify its own data on the server to update it according to the current state. User Agent also uses write request to change the state of an equipment.

2.3 Invoke

Invoke request is used to invoke any op object. New equipment makes a new watch with an invoke request and adds its own data to it. That is how the equipment knows if it should change its state. This happens when some other client, for example User Agent, has made changes to the equipment data in the oFMS by using write request. To see if changes has been made, clients polls the watch URI using pollChanges operation.

3 Equipment Adapters

In order to connect legacy equipments to the system we need to implement equipment adapters. Equipment adapter is a piece of software that connects the device to the oFMS. This adapter is needed because equipment does not have its own support for oBIX. How the adapter communicates with the equipment is not interesting, it depends on the equipment. To the oFMS the adapter is an HTTP client which writes information to the server using write requests and polls commands using watches.

General behaviour for equipment adapter designed for oFMS goes as follows. First the adapter reads the Lobby object and stores it. Next step is to sign up using the signUp address from Lobby object. Then the adapter creates a new watch and adds its own data objects to it. Now the initialization is ready and normal polling for the watch can begin. Adapter only polls for changes and sometimes refreshes using pollRefresh. If some data in the server has been changed it redirects commands to the equipment. It also writes new values to its own objects in the server if their states have been changed at the equipment.

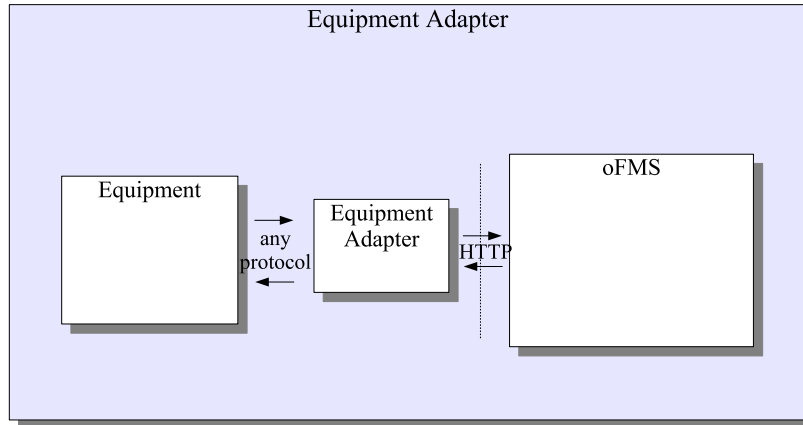


Figure 2: Equipment Adapter

4 Lobby

Only thing that client has to know when connecting to the server is the address of the Lobby object. By convention the address is `http://server/obix`, but servers are also free to use any other address. By reading the Lobby object, clients can determine other important addresses. In addition to features defined in oBIX specification oFMS Lobby object defines SignUp address. This address is used by equipments when registering to the server for the first time. Actual operation is adding the data about the equipment to the database.

4.1 Example

Equipment adapter is started and it reads the Lobby object to find out addresses of the important objects and services.

Command:

```
client.read("http://server/obix")
```

Response:

```
1 <obj name="lobby" href="http://server/obix">
2   <ref name="about" href="/about" is="obix:About"/>
3   <op name="batch" href="/batch" in="obix:BatchIn" out="obix:BatchOut"/>
4   <ref name="watchService" href="/watchService" is="obix:WatchService"/>
5   <op name="signUp" href="/signUp" in="obix:obj" out="obix:obj"/>
6 </obj>
```

Now the adapter stores addresses that are defined in the Lobby object. After this it can directly use basic functionalities by using these addresses.

5 About

About object contains general information about the oBIX server.

5.1 Example

In some cases there can be need to know more about the oBIX server itself. For example in more advanced equipment adapters there can be support for different oBIX versions. Currently supported version number can be found in servers about object. In this example equipment adapter reads the Lobby object for server related information.

Command:

```
client.read("http://server/about")
```

Response:

```
1 <obj name="about" href="http://server/about">
2   <str name="obixVersion" val="0.11"/>
3   <str name="serverName" val="oFMS"/>
4   <abstime name="serverTime" val="2006-11-27T12:24:07.749+02:00"/>
5   <abstime name="serverBootTime" val="2006-11-27T12:24:03.031+02:00"/>
6   <str name="vendorName" val="Acme Corp."/>
7   <uri name="vendorUrl" val=""/>
8   <str name="productName" href="/about/productName" val="oFMS" writable="true"/>
9   <str name="productVersion" href="/about/productVersion" val="beta" writable="true"/>
10  <uri name="productUrl" href="/about/productUrl" val="" writable="true"/>
11 </obj>
```

At line 4 equipment adapter can see the current time on the server. At line 5 is the time when the server was last booted.

6 Batch

Batch operation can be used to bunch network requests together. All the requests are packed into batch object which is sent to the server. Response object contains responses for all requests.

6.1 Example

Lets consider an example about equipment which acts like an alarm clock. At a given time equipment wants to wake up user by turning on the television. When turning on the TV it also makes sure that it is not muted. The equipment adapter could easily make two requests to the server and do the same thing but just to minimize the network load it has been implemented to use batch operation.

Command:

```
client.invoke("http://server/batch",
1 <list name="in" is="obix:BatchIn">
2   <uri is="obix:Read" val="/tv/muteSwitch" />
3   <uri is="obix:Write" val="/tv/onOffSwitch">
4     <bool name="in" val="true" />
5   </uri>
6 </list>);
```

Response:

```
7 <list is="obix:BatchOut">
8   <bool name="muteSwitch" href="/tv/muteSwitch" val="false" writable="true"/>
9   <bool name="onOffSwitch" href="/tv/onOffSwitch" val="true"
10   writable="true"/>
11 </list>
```

The alarm clock equipment can now read from the response message that the TV is not muted and is now turned on.

7 WatchService

Watches offer a simple way to poll changes in certain objects. The goal is to make it simple and minimize network load. When using watches only the changes are sent in response message and unchanged objects are ignored. All the available watch operations are defined in oBIX specification. In addition all the URIs added to the watch are listed in a child object of a watch.

7.1 Example

In this example we jump to the other side to think about the equipment adapter of the TV set from the previous example. The TV set should know when to turn itself on. The adapter has already added its data to the server and now wants to poll it for the changes. First the adapter has to read the watchService object.

Command:

```
client.read("http://server/watchService");
```

Response:

```
1 <obj name="watchService" href="http://server/watchService">
2   <op name="make" href="/watchService/make" in="obix:nil" out="obix:Watch"/>
3 </obj>
```

Now the adapter has an address to the operation that is able to create a watch. Creating it is the next step.

Command:

```
client.invoke("http://server/watchService/make");
```

Response:

```
4 <obj name="http://server/watch26" href="http://server/watch26" is="obix:Watch">
5   <reltime name="lease" href="http://server/watch26/lease" val="PT0S" writable="true"
6     min="PT60S"/>
7   <op name="add" href="http://server/watch26/add" in="obix:WatchIn"
8     out="obix:WatchOut"/>
9   <op name="remove" href="http://server/watch26/remove" in="obix:WatchIn"
10    out="obix:obj"/>
11  <op name="pollChanges" href="http://server/watch26/pollChanges" in="obix:obj"
12    out="obix:WatchOut"/>
13  <op name="pollRefresh" href="http://server/watch26/pollRefresh" in="obix:obj"
14    out="obix:WatchOut"/>
15  <op name="delete" href="http://server/watch26/delete" in="obix:obj" out="obix:obj"/>
16  <list name="uris" href="http://server/watch26/uris"/>
17 </obj>
```

Server creates a new empty watch and returns it as an HTTP response. Now it is time to add all the addresses that adapter wants to watch. In this example the adapter watches the changes in muteSwitch and onOffSwitch objects.

Command:

```
client.invoke("http://server/watch26/add",
18 <obj name="watchIn" is="obix:WatchIn">
19 <list name="hrefs">
20 <uri val="/tv/muteSwitch" />
21 <uri val="/tv/onOffSwitch" />
22 </list>
23 </obj>);
```

The server returns current values of the objects.

Response:

```
24 <list name="values">
25 <bool name="muteSwitch" href="/tv/muteSwitch" val="false" writable="true"/>
26 <bool name="onOffSwitch" href="/tv/onOffSwitch" val="false" writable="true"/>
27 </list>
```

Now the adapter periodically polls changes in the objects by polling only the watches pollChanges address. As the line 28 shows no changes in these objects came up this time.

Command:

```
client.invoke("http://server/watch26/pollChanges");
```

Response:

```
28 <list name="watchOut"/>
```

8 SignUp

By signing up clients can add their own data to the server. Sign-up operation is not defined in oBIX specification but is an addition to the feature set offered by oFMS. The idea is that an equipment first writes its own data to the server by signing up and then polls changes in that data. Other clients or servers can control the equipment by altering the data on the server.

8.1 Example

In this example we have an equipment adapter of the one-room lamp control unit. After the adapter has read Lobby it wants to add its own data to the server so that it can be controlled by any other client or server. It invokes the SignUp operation.

Command:

```
client.invoke("http://server/signUp",
1  <obj name="bedroomLampDevice" href="/bedroom/lampDevice">
2    <obj href="/bedroom/lampDevice/lamp1" displayName="Desk lamp" is="obix:Point"
3      name="4R1" writable="true">
4      <bool href="/bedroom/lampDevice/lamp1/switch" displayName="Lamp Switch"
5        name="4R1Value" val="" writable="true"/>
6      <obj name="status">
7        <bool name="overridden" val="false"/>
8        <bool name="disabled" val="false"/>
9        <bool name="fault" val="false"/>
10       <bool name="down" val="false"/>
11       <bool name="inAlarm" val="false"/>
12       <bool name="unackedAlarm" val="false"/>
13       <bool name="historyStart" val="false"/>
14       <bool name="historyEnd" val="false"/>
15     </obj>
16     <abstime name="timestamp" val="1970-01-01T02:00:00.000+02:00"/>
17   </obj>
18 </obj>);
```

Server returns the added data object.

Response:

```
19 <obj name="bedroomLampDevice" href="/bedroom/lampDevice">
20   <obj name="4R1" href="/bedroom/lampDevice/lamp1" is="obix:Point"
21     displayName="Desk lamp" writable="true">
22     <bool name="4R1Value" href="/bedroom/lampDevice/lamp1/switch" val="false"
23       displayName="Lamp Switch" writable="true"/>
24     <obj name="status">
25       <bool name="overridden" val="false"/>
26       <bool name="disabled" val="false"/>
27       <bool name="fault" val="false"/>
28       <bool name="down" val="false"/>
29       <bool name="inAlarm" val="false"/>
30       <bool name="unackedAlarm" val="false"/>
31       <bool name="historyStart" val="false"/>
32       <bool name="historyEnd" val="false"/>
33     </obj>
34     <abstime name="timestamp" val="1970-01-01T02:00:00.000+02:00"/>
35   </obj>
36 </obj>
```

Now the added data on the server can be read or manipulated normally.

Command:

```
client.read("http://server/bedroom/lampDevice/lamp1");
```

Response:

```
37 <obj name="4R1" href="http://server/bedroom/lampDevice/lamp1" is="obix:Point"
38   displayName="Desk lamp" writable="true">
39   <bool name="4R1Value" href="/bedroom/lampDevice/lamp1/switch" val="false"
40     displayName="Lamp Switch" writable="true"/>
41   <obj name="status">
42     <bool name="overridden" val="false"/>
43     <bool name="disabled" val="false"/>
44     <bool name="fault" val="false"/>
45     <bool name="down" val="false"/>
46     <bool name="inAlarm" val="false"/>
47     <bool name="unackedAlarm" val="false"/>
48     <bool name="historyStart" val="false"/>
49     <bool name="historyEnd" val="false"/>
50   </obj>
51   <abstime name="timestamp" val="1970-01-01T02:00:00.000+02:00"/>
52 </obj>
```

References

- [1] oBIX Specification Committee Draft 02, June 30, 2006;
OASIS Open Building Information Exchange TC