**PROTOCOLS FOR THE MFW**

**Teemu Kanstrén, VTT**

**v0.2, 03.04.2011**

**v0.2.1 04.04.2011: added history request**

**v0.2.2 04.04.2011: moved data type to bm list, value type is by reference to this**

**v0.2.3 12.05.2011: changed measurement content and authentication (http-header)**

# Introduction

This document describes the protocol for communication between the different internal and external entities of the MFW. This includes communication between probe-agents, server-agents, and clients. The probe-agents are the entities responsible for controlling a set of probes providing measurements. The server-agents are the entities to which the probe-agents provide their measurements, and to which external clients connect to request and use the measurements.

Figure 1 shows a layered view of the different components both internal and external to the MFW, where different communication protocols need to be considered. Several client applications can connect to a single server-agent to make use of available measurements. Currently only a single server-agent is supported per deployment.

In Figure 1 the internal components of the MFW are the server-agent and probe-agents. External components that the MFW needs to communicate with are the clients and the probes. The following are the types of protocols between these different elements:

* Clients – Server Agent: REST based XML over HTTP.
* Server Agent – Probe Agent: REST based XML over HTTP.
* Probe-Agents – Probes: Custom implementation for each probe by the MFW user.
* Probes – ToM: Probe and ToM specific.

Within the MFW, REST over HTTP is the preferred protocol here to allow for simple cross-platform implementations and to avoid the overhead of more complex solutions such as web-services.



Figure 1. Components of the MFW and its operational environment.

# General notes

This section briefly describe some general aspects of the different interfaces.

## Measure URI

The probes provide measurements, which are considered raw measurements and termed as “base measure” here. A measurement request and response for a base measure is defined by a “MeasureURI”, which consists of four elements:

* Target name
* Target type
* BM class
* BM name

Of these, target properties describe the target of measurement, that is, the target that the base measure is measuring. The BM properties are base measure properties the describe the base measure that the measure is for. The definition of what these name/type/class properties mean is not exactly specified in order to give the user freedom to use them as best seen fit to represent their measurement needs. However, the split to two different properties for both target and base measure is to give a chance for finer granularity. For example, target could have a type of “mobile push mail client” and a name “bob’s phone”. Similarly, the BM class could be “authentication strength” and name “webmail login authentication”. But these are just examples and not specific rules for what they should be used for.

## Message types

The interfaces described in this document are always between two specific elements. One makes a request and gets some type of a response. Sometimes this will trigger further messaging later, such as subscriptions triggering measurement values later. However, these are still separate messages with both request and reply. The responses can be either successes or failures. This section describes each of these options.

A typical case of a successful response is providing a HTTP 200 OK response. The specifics are then the possible XML values returned in the response body.

For errors, the basic error is the HTTP 404 NOT FOUND error when a requested resource is not found (the message URL is not supported). A second type of generic error is that of unauthorized access attempt. In this case, the error is of “HTTP 401 Authorization Required”. This can happen if a client tries to register with an erroneous authorization code to initiate a session, or to use a session with a wrong session id/key.

In addition to these, specific error codes can be delivered for each request. These are identified by the suitable HTTP error code in the response. Additionally, these contain a specific error code values in the body XML response. This is tagged as <error> in the response body.

## Data Types

The general data types supported are given in . Each base measure data type is defined when the base measure definitions are requested by the client. These definitions are in the <type> tags in the message described in section . The values for this <type> tag are in the “Type string” column of Table 1. When the client receives some measurements with the message defined in sections 3.3.1, they know how to parse this information based on these definitions. This also applies to the history requests described in section 3.2.3.

Table 1. Data types.

|  |  |  |
| --- | --- | --- |
| Type | Description/format. | Type string |
| Timestamp | Number of milliseconds from 1.1.2010. Java long datatype. | timestamp |
| Boolean | “true”/”false”. | boolean |
| String | UTF8 character stream. | string |
| Number | Double precision floating point, e.g. ”11.423”. Point separated (“.”). | number |

# Interfaces between the clients and the server-agent

This section describes the interfaces between the server-agent and its clients.

## Rendezvous

To communicate with the MFW, each client must first register to it. If the client is not registered, any communication is to be rejected. An authenticated client is identified by it providing an authentication token with each request it makes. To be identified as being a valid MFW, the MFW must also provide a similar token when communicating with each client. These authentication tokens are to be negotiated during the registration of a client with the MFW. This could be implemented using, for example, predefined and distributed certificates, or tokens signed with a predefined private key for the server-agent.

A single server-agent needs to support several clients. Thus it must be able to identify each client in each request. For this reason, each client needs to negotiate a suitable identifier for communication with the MFW. This can be the authentication token as this needs to be negotiated and always provided, but this also means that each client needs to have a unique authentication token.

The rendezvous is expected to take the following form:

The client sends a “register” message to the server-agent. This message includes the initial authentication token of the client, signed with a certificate that is supported by the MFW. Additional measures could also be taken, such as defining the address range from which a given client with a given token is allowed to connect from.

The registration sequence thus takes the form described in . This excludes the fact that the server-agent must keep track of all registrations and session id’s for future communications. When a client has not been active for a defined period of time, the session is ended and the session id is no longer valid. When the client receives a response of HTTP 401 “unauthorized”, it needs to re-register for a new session. When a session is invalidated or ended, all state related to that session, such as subscription, is also invalidated.



Figure 2. Client registration message sequence.

The format of the HTTP message to perform the registration is shown in Table 2.

Table 2. Registration message format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | <http://localhost:8080/rest/client/register> |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”). * endpoint: the address where the client REST service is found (see section 3.3) |
| Returns: | session id: to be used in further communications |
| Example: | Request:  POST /rest/client/register HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <endpoint>http://localhost:8088</endpoint>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: 0  <sessionId>new session id</sessionId>  Error response:  Error response as defined in 2.2. |

When a client registers, all subscriptions and any other previous state in the MFW for it will be removed. This is to avoid confusion between the client believing to have a different set of subscriptions vs. what the MFW believes it has. After re-registration, the client needs to re-build its state.

## Interface from the client to the server-agent

This section describes the communication in the direction of client to the server-agent.

### Requesting a list of available measurement information

The client can request a list of all available measurements from the MFW. This list will include the base measures, probes providing these measurements, and the targets that are being measured.

The client can request a list of all available base measures from the MFW. The difference with probes is that there can be some overlap with what base measures different probes provide. Several probes may provide the same base measures but with different properties such as precision. In this case, the MFW picks the best one. However, this request will only list the set of available base measures regardless of their other properties (such as precision).

The format of the HTTP message to perform the registration is shown in Table 3.

Table 3. Probe availability request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP GET |
| URL: | http://localhost:8080/rest/client/probes |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”). |
| Returns: | List of registered probes, their measurement targets and provided base measures.   * target: information for each measurement target is enclosed in this tag. * name: target or bm name, part of metadata for measureURI. * type: target type, part of metadata for measureURI. * class: bm class, part of metadata for measureURI. * probe: Information for each probe is enclosed in this tag. * id: Unique identifier (string) for each target, probe, or bm. Used as referce in other requests. * targetid: References to a target id. * bmid: Reference to a base measure id. |
| Example: | Request:  GET /rest/client/probes HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  <target>  <id>target1</id>  <name>bob’s phone</name>  <type>email client</type>  </target>  <target>  <id>target2</id>  <name>company email server</name>  <type>email server</type>  </target>  <bm>  <id>bm1</id>  <name>email authentication</name>  <class>authentication mechanism list</class>  <description>list of supported mechanisms<description>  <targetid>target2</targetid>  <datatype>string</datatype>  </bm>  <bm>  <id>bm2</id>  <name>bob’s email client</name>  <class>email client name</class>  <description>software name<description>  <targetid>target2</targetid>  <datatype>string</datatype>  </bm>  <bm>  <id>bm3</id>  <name>email server errors</name>  <class>error log</class>  <description>number of errors in syslog<description>  <targetid>target2</targetid>  <datatype>string</datatype>  </bm>  <probe>  <id>probe1</id>  <name>hello</name>  <bmid>bm1</bmid>  <bmid>bm2</bmid>  </probe>  <probe>  <id>probe2</id>  <name>world</name>  <bmid>bm1</bmid>  <bmid>bm3</bmid>  </probe>  Error response:  Error response as defined in 2.2. |

### Requesting a base measure (once)

The client can request the MFW to provide the current value of a base measure. This is provided once, after which the request is discarded.

Table 4. Base measure one time request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/bmrequest/{bmid} |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”). * bmid: identifier for requested base measure. |
| Returns: | ok/error |
| Example: | Request:  POST /rest/client/bmrequest/bm1 HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

### Requesting measurement history

The client can ask the MFW to provide the measurement history for a specific time period. The MFW reads this from the database and provides all the measurements for that time period in the response.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/history |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”). * bms: list of base measure to provide * bmid: identifier for requested base measure. * starttime: defines the start of the time period. See section 2.3. * endtime: defines the end of time period. See section 2.3. |
| Returns: | * bmid: base measure identifier as given in message defined in section 3.2.1. * string: UTF8 string * number: double precision floating point,e.g. ”11.423” * boolean: “true” or “false”. See section 2.3. * timestamp: when was the value measured? See section 2.3. * see 3.3.1 for more information. |
| Example: | Request:  POST /rest/client/bmrequest/bm1 HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  <bms>  <bmid>1</bmid>  <bmid>2</bmid>  </bms>  <starttime>52534554</starttime>  <endtime>54663364</endtime>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  <values>  <measurement>  <bmid>1</bmid>  <value>hello</value>  <timestamp>1342430</timestamp>  </measurement>  <measurement>  <bmid>2</bmid>  <value>world</value>  <timestamp>1342430</timestamp>  </measurement>  </values>  Error response:  Error response as defined in 2.2. |

### Subscribing to a base measure (continuously)

The client can subscribe to a base measure, in which case the MFW will keep providing the value of the base measure over time.

Table 5. Base measure subscription format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/bmsubscription/{bmid} |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”). * bmid: identifier for requested base measure. * frequency: interval for how often the value should be retrieved (in seconds). |
| Returns: | ok/error |
| Example: | Request:  POST /rest/client/bmsubscripton/bm1 HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  <interval>5000</interval>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

### Unsubscribing to a base measure

When the client no longer wishes to receive values for a given base measure, this message is used to stop the subscription in an active session.

As noted before, all subscriptions are cancelled when a session ends (client shutdown, exit, timeout, server shutdown, etc.).

Table 6. Base measure unsubscribe request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP DELETE |
| URL: | http://localhost:8080/rest/client/bmsubscription/{bmid} |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”).bmid: identifier for requested base measure. |
| Returns: | ok/error |
| Example: | Request:  DELETE /rest/client/bmsubscription/bm1 HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

### Requesting a probe configuration

The client can request for the configuration of a given probe, if the probe supports this.

Table 7. Probe configuration read request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP GET |
| URL: | http://localhost:8080/rest/client/probeconfiguration/{probeid} |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”).probeid: the id for the targeted probe. |
| Returns: | parameters as key/value pairs. |
| Example: | Request:  GET /rest/client/probeconfiguration/probe1 HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  <parameter>  <key>key1</key>  <value>hello</value>  </parameter>  <parameter>  <key>key2</key>  <value>world</value>  </parameter>  Error response:  Error response as defined in 2.2. |

### Setting a probe configuration

The client can set the configuration for a given probe if the probe supports this.

Table 8. Probe configuration set request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/probeconfiguration/{probeid} |
| Parameters: | * authentication: A pre-deployed authentication token. This is in http authentication header (“username:password”).probeid: referenced probe id. * key: parameter key * value: parameter value |
| Returns: | ok/error |
| Example: | Request:  POST /rest/client/probes HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  <key>key1</key>  <value>hola</value>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

### Getting MFW information

The client can request the MFW to provide it with some basic information about the MFW it is connected to.

Table 9. MFW general information request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP GET |
| URL: | http://localhost:8080/rest/client/mfwinformation |
| Parameters: | none. allowed without session. |
| Returns: | information describing the mfw. |
| Example: | Request:  GET /rest/client/mfwinformation HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  <mfwInformation>  <id>mfw1</id>  <name>company x security measurements</name>  </mfwInformation>  Error response:  Error response as defined in 2.2. |

## Interface from the server-agent to the client

This interface describes the messages the server-agent sends to the client. The difference to the previous section is that in this case the server-agent (MFW) is the initiating party.

### Providing base measure values

The MFW will provide to the client base measures according to subscriptions and requests through this message interface. These measures are provided in a defined interval, for example, every 5 seconds. All new measures received in this interval are delivered. If no measurements are received in this time, this message is provided with empty set of values.

Table 10. Base measure value post request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/measurements |
| Parameters: | * session id * bmid: base measure identifier as given in message defined in section 3.2.1. * string: UTF8 string * number: double precision floating point,e.g. ”11.423” * boolean: “true” or “false”. See section 2.3. * timestamp: when was the value measured? See section 2.3.   CONSTRAINT: The request needs to have one value type and can only have one. These types of values are of type “string”, “number” or “boolean”. If several are included, the behavior is undefined (the other end may pick any one). |
| Returns: | ok/error |
| Example: | Request:  POST /rest/client/bmvalue/{bmid} HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  <values>  <measurement>  <bmid>1</bmid>  <value>hello</value>  <timestamp>1342430</timestamp>  </measurement>  <measurement>  <bmid>2</bmid>  <value>world</value>  <timestamp>1342430</timestamp>  </measurement>  </values>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

### Providing events

The MFW will provide to the client events related to the measurement infrastructure based on this interface.

Table 11. Event nofitication request format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/client/event |
| Parameters: | * session id * type: event type from a predefined list, or a custom type * description: free form text describing the event |
| Returns: | ok/error |
| Example: | Request:  POST /rest/client/event HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <sessionId>xxx</sessionId>  <type>event type</type>  <description>event description</description>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: <...>  Error response:  Error response as defined in 2.2. |

List of specific events defined:

* probe added: “P1”
* probe deleted: “P2”
* bm added: “BM1”
* bm deleted: “BM2”
* bm precision changed: “BM3”

**Providing availability change notifications**

The MFW will provide the client notifications related to probe availability through this interface.

# Internal interfaces between MFW agents

This section describes the interfaces between the server-agent and the probe agents.

## Rendezvous

Similar to the MFW clients, the probe-agents must also register with the server-agent prior to being allowed to perform any other operations with relation to the server-agent. If a probe-agent is not registered, any communication is to be rejected. An authenticated probe-agent is identified by it providing an authentication token with each request it makes. To be identified as being a valid MFW, the MFW must also provide a similar token when communicating with each probe-agent. These authentication tokens are to be negotiated during the registration of a probe-agent with the MFW. This could be implemented using, for example, predefined and distributed certificates, or tokens signed with a predefined private key for the server-agent.

In case of the probe-agent providing an authentication token, this token needs to match the metadata of the base measure provided. Additionally, connections can be restricted by other means, such as only allowing certain base measures from certain address range.

The rendezvous is expected to take the following form:

The probe-agent sends a “register” message to the server-agent. This message includes the initial authentication token of the probe-agent, signed with a certificate that is supported by the MFW. Additional measures could also be taken, such as defining the address range from which a given client with a given token is allowed to connect from.

The registration sequence thus takes the form described in . This excludes the fact that the server-agent must keep track of all registrations and tokens for future communications.



Figure 3. Client registration message sequence.

The format of the HTTP message to perform the registration is shown in Table 2.

Table 12. Registration message format and example.

|  |  |
| --- | --- |
| Request type: | HTTP POST |
| URL: | http://localhost:8080/rest/probe/register |
| Parameters: | <authentication>: The pre-deployed authentication token. |
| Returns: | session id: to be used in further communications |
| Example: | Request:  POST /rest/client/register HTTP/1.0  Content-Type: text/xml  Content-Length: <...>  <authentication>authentication info</authentication>  OK response:  HTTP/1.0 200 OK  Date: Fri, 31 Dec 1999 23:59:59 GMT  Content-Type: text/xml  Content-Length: 0  <sessionId>new session id</sessionId>  Error response:  Error response as defined in 2.2. |

## From the probe-agent to the server-agent

This section describes the messages from the probe-agent to the server-agent.

### Providing measurement values

Basic types that are supported:

* String (UTF8 character streams)
* Numbers (double precision floating points)
* Booleans (true and false)

### Error notification

The probes can also provide events to the server-agent.

### Keep-alive messages

The server-agent needs to know when a connection to a probe-agent is lost and to assure this information is available, the probe-agent must keep sending keep-alive messages to the server-agent at predefined intervals.

### Unregister

The probe-agent can unregister itself if being shut down to avoid the need for keep-alive termination. This is a form of a managed shutdown.

### Check subscriptions

When started, the probe-agent must check which subscriptions it needs to uphold from the server-agent.

## From the server-agent to the probe-agent

messages from the server-agent to the probe-agent.

### Measurement subscription

Subscribing to measurements and requesting one-time measurements.

### Requesting probe configuration

The server-agent requests the probe configurations to enable clients to work with them.

### Setting probe configuration

The server-agent requests the probe-agent to set the probe configuration according to the request from the client.