XEP-0347: Internet of Things - Discovery

Peter Waher
mailto:peterwaher@hotmail.com
xmpp:peter.waher@jabber.org
http://www.linkedin.com/in/peterwaher

Ronny Klauck
mailto:rklauck@informatik.tu-cottbus.de
xmpp:TBD
http://www-rnks.informatik.tu-cottbus.de/~rklauck

2018-11-03
Version 0.5.1

Status	Type	Short Name
Deferred	Standards Track	iot-discovery

This specification describes an architecture based on the XMPP protocol whereby Things can be installed and safely discovered by their owners and connected into networks of Things.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

When installing massive amounts of Things into public networks care has to be taken to make installation simple, but at the same time secure so that the Things cannot be hijacked or hacked, making sure access to the Thing is controlled by the physical owner of the Thing. One of the main problems is how to match the characteristics of a Thing, like serial number, manufacturer, model, etc., with information automatically created by the Thing itself, like perhaps its JID, in an environment with massive amount of Things without rich user interfaces. Care has also to be taken when specifying rules for access rights and user privileges.
This document provides a network architecture based on the XMPP protocol that provides a means to safely install, configure, find and connect massive amounts of Things together, and at the same time minimizing the risk that Things get hijacked. It also provides information how each individual step in the process can be performed with as little manual intervention as possible, aiming where possible at zero-configuration networking. Furthermore, this document specifies how to create a registry that allows simple access to public Things without risking their integrity unnecessarily.
Internet of Things contains many different architectures and use cases. For this reason, the IoT standards have been divided into multiple XEPs according to the following table:

XEP	Description
xep-0000-IoT-BatteryPoweredSensors	Defines how to handle the peculiars related to battery powered devices, and other devices intermittently available on the network.
xep-0000-IoT-Events	Defines how Things send events, how event subscription, hysteresis levels, etc., are configured.
xep-0000-IoT-Interoperability	Defines guidelines for how to achieve interoperability in Internet of Things, publishing interoperability interfaces for different types of devices.
xep-0000-IoT-Multicast	Defines how sensor data can be multicast in efficient ways.
xep-0000-IoT-PubSub	Defines how efficient publication of sensor data can be made in Internet of Things.
xep-0000-IoT-Chat	Defines how human-to-machine interfaces should be constructed using chat messages to be user friendly, automatable and consistent with other IoT extensions and possible underlying architecture.
XEP-0322	Defines how to EXI can be used in XMPP to achieve efficient compression of data. Albeit not an Internet of Things specific XEP, this XEP should be considered in all Internet of Things implementations where memory and packet size is an issue.

XEP	Description
XEP-0323	Provides the underlying architecture, basic operations and data structures for sensor data communication over XMPP networks. It includes a hardware abstraction model, removing any technical detail implemented in underlying technologies. This XEP is used by all other Internet of Things XEPs.
XEP-0324	Defines how provisioning, the management of access privileges, etc., can be efficiently and easily implemented.
XEP-0325	Defines how to control actuators and other devices in Internet of Things.
XEP-0326	Defines how to handle architectures containing concentrators or servers handling multiple Things.
XEP-0331	Defines extensions for how color parameters can be handled, based on Data Forms (XEP-0004) XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
XEP-0336	Defines extensions for how dynamic forms can be created, based on Data Forms (XEP-0004) XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>., Data Forms Validation (XEP-0122) XEP-0122: Data Forms Validation <https://xmpp.org/extensions/xep-0122.html>., Publishing Stream Initiation Requests (XEP-0137) XEP-0137: Publishing Stream Initiation Requests <https://xmpp.org/extensions/xep-0137.html>. and Data Forms Layout (XEP-0141) XEP-0141: Data Forms Layout <https://xmpp.org/extensions/xep-0141.html>..
XEP-0347	This specification. Defines the peculiars of Thing discovery in Internet of Things. Apart from discovering Things by JID, it also defines how to discover Things based on location, etc.

2 Glossary

The following table lists common terms and corresponding descriptions.

Actuator Device containing at least one configurable property or output that can and should be controlled by some other entity or device.

Authority Used synonymously with Provisioning Server.

Computed Value A value that is computed instead of measured.

Concentrator Device managing a set of devices which it publishes on the XMPP network.

Field One item of sensor data. Contains information about: Node, Field Name, Value, Precision, Unit, Value Type, Status, Timestamp, Localization information, etc. Fields should be unique within the triple (Node ID, Field Name, Timestamp).

Field Name Name of a field of sensor data. Examples: Energy, Volume, Flow, Power, etc.

Field Type What type of value the field represents. Examples: Momentary Value, Status Value, Identification Value, Calculated Value, Peak Value, Historical Value, etc.

Historical Value A value stored in memory from a previous timestamp.

Identification Value A value that can be used for identification. (Serial numbers, meter IDs, locations, names, etc.)

Localization information Optional information for a field, allowing the sensor to control how the information should be presented to human viewers.

Meter A device possible containing multiple sensors, used in metering applications. Examples: Electricity meter, Water Meter, Heat Meter, Cooling Meter, etc.

Momentary Value A momentary value represents a value measured at the time of the readout.

Node Graphs contain nodes and edges between nodes. In Internet of Things, sensors, actuators, meters, devices, gateways, etc., are often depicted as nodes whereas links between sensors (friendships) are depicted as edges. In abstract terms, it's easier to talk about a Node, rather than list different possible node types (sensors, actuators, meters, devices, gateways, etc.). Each Node has a Node ID.

Node ID An ID uniquely identifying a node within its corresponding context. If a globally unique ID is desired, an architecture should be used using a universally accepted ID scheme.

Parameter Readable and/or writable property on a node/device. The XEP-0326 Internet of Things - Concentrators (XEP-0326) XEP-0326: Internet of Things - Concentrators <https://xmpp.org/extensions/xep-0326.html>. deals with reading and writing parameters on nodes/devices. Fields are not parameters, and parameters are not fields.

Peak Value A maximum or minimum value during a given period.

Provisioning Server An application that can configure a network and provide services to users or Things. In Internet of Things, a Provisioning Server knows who knows whom, what privileges users have, who can read what data and who can control what devices and what parts of these devices.

Precision In physics, precision determines the number of digits of precision. In sensor networks however, this definition is not easily applicable. Instead, precision determines, for example, the number of decimals of precision, or power of precision. Example: 123.200 MWh contains 3 decimals of precision. All entities parsing and delivering field information in sensor networks should always retain the number of decimals in a message.

Sensor Device measuring at least one digital value (0 or 1) or analog value (value with precision and physical unit). Examples: Temperature sensor, pressure sensor, etc. Sensor values are reported as fields during read-out. Each sensor has a unique Node ID.

SN Sensor Network. A network consisting, but not limited to sensors, where transport and use of sensor data is of primary concern. A sensor network may contain actuators, network applications, monitors, services, etc.

Status Value A value displaying status information about something.

Timestamp Timestamp of value, when the value was sampled or recorded.

Thing Internet of Things basically consists of Things connected to the Internet. Things can be any device, sensor, actuator etc., that can have an Internet connection.

Thing Registry A registry where Things can register for simple and secure discovery by the owner of the Thing. The registry can also be used as a database for meta information about Things in the network.

Token A client, device or user can get a token from a provisioning server. These tokens can be included in requests to other entities in the network, so these entities can validate access rights with the provisioning server.

Unit Physical unit of value. Example: MWh, l/s, etc.

Value A field value.

Value Status Status of field value. Contains important status information for Quality of Service purposes. Examples: Ok, Error, Warning, Time Shifted, Missing, Signed, etc.

Value Type Can be numeric, string, boolean, Date & Time, Time Span or Enumeration.

WSN Wireless Sensor Network, a sensor network including wireless devices.

XMPP Client Application connected to an XMPP network, having a JID. Note that sensors, as well as applications requesting sensor data can be XMPP clients.

3 Use Cases

The life cycle of a Thing can be divided into multiple steps. The following sections will list many of these steps in possible order of occurrence during the life cycle of the Thing.

3.1 Production

During production of a Thing, decisions have to be made whether the following parameters should be pre-configured, manually entered after installation or automatically found and/or created by the device if possible (zero-configuration networking):

- Address and domain of XMPP Server.
- JID of the Thing.
- JID of Thing Registry, if separate from the XMPP Server.
- JID of first Provisioning Server, if separate from Thing Registry or XMPP Server.

A decision has to be made at this point if global/manufacturer/customer servers should be used, or if local resources should be searched for and used if found. The first option is easy to configure in a production environment and might have commercial significance, but cannot use local resources where available. The second leaves much responsibility to the Thing for finding local resources, but has the advantage of allowing for a more decentralized network architecture. A detailed discussion of the two alternatives goes beyond the scope of this specification, and will not be presented here.

3.2 Installation

Apart from physical installation, and connection to power and communication infrastructure, the installation phase of a Thing might also require manual entry of values that could not be set in the production environment. Since Things might have very limited human user interfaces, external tools might be required to provide this information. Due to its complexity, any manual entry of configuration parameters should be avoided, if possible. However, manual entry of some parameters might allow for Things to use local resources where such cannot be found nor set in a production environment.

3.3 Finding XMPP Server

If the address of an XMPP Server is not preconfigured, the Thing must attempt to find one in its local surroundings. This can be done using one of several methods:

- DHCP
- Multicast DNS
- SSDP/UPnP

The following sections describe them in more detail.

3.3.1 DHCP

DHCP offers an internal structure for advertising configuration information to clients in a network. This includes configuration parameters and other control elements, which are transmitted in special marked data elements, called 'options', as described in RFC 3942 [1].
Dynamic Host Configuration Protocol (DHCP) and Bootstrap Protocol (BOOTP) Parameters lists currently assigned 'options' by IANA. **Note:** There does exist no 'option' for XMPP at the moment. Options 224 to 254 are marked as 'site-specific option range' to support local (to a site) configuration options (i.e., reserved as 'Private Use').
Possible codes for the XMPP server option:

- Use 'site-specific option range'. Use of 'option-code' 224.
- **TBD:** Define and register DHCP and BOOTP option as described in Parameters for IoT Discovery. Use of 'option-code' 84.

This option specifies the name of the XMPP server. The name may or may not be qualified with the local domain name. See RFC 1035 [2] for character set restrictions.
The code for this option is 224 (for 'site-specific option range') or 84 (for DHCP and BOOTP Parameters for IoT Discovery), and its minimum length is 1.

Listing 1: IoT Discovery DHCP Option

```
<option code> <data length> machine
```

So, for example, if the machine name is "pronto", the code for the option is 224, the XMPP server option would be as follows:

Listing 2: IoT Discovery DHCP Option Example

```
224 12 pronto.local
```

The following parameters in use as of MONTH 201x. Refer to the DHCP and BOOTP parameters itself for a complete and current list of parameters (this specification might or might not be revised when new parameters are registered).

[1] RFC 3942: Reclassifying Dynamic Host Configuration Protocol version 4 (DHCPv4) Options <http://tools.ietf.org/html/rfc3942>.
[2] RFC 1035: Domain Names - Implementation and Specification <http://tools.ietf.org/html/rfc1035>.

Listing 3: IoT Discovery DHCP and BOOTP Parameters Registry

```
<tag>84</tag>
   <name>XMPP server</name>
   <data length>N</data length>
   <meaning>XMPP Servers DHCP Option</meaning>
   <reference>[RFC6120]</reference>
```

3.3.2 Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD)

An introduction of mDNS/DNS-SD (e.g., how it works and terminology) is described in Link-Local Messaging (XEP-0174) [3] (i.e., sections [1.2] and [2]). For the purpose of IoT Discovery we are interested only in the "xmpp-client" service. An XMPP server MUST publish four different kinds of DNS records to advertise its availability using the services of type "xmpp-client". An XMPP chat client (actually its mDNS daemon) can send out multicast DNS queries for services of type "xmpp-client". **Note:** the service of type "xmpp-client" is the reservered name for client-to-server connections by IANA, as described in RFC 6120 [4].
In order to advertise its availability, a server MUST publish four different kinds of DNS records:

1. A PTR record of the following form:

Listing 4: PTR record

```
_xmpp-client._tcp.local. PTR machine._xmpp-client._tcp.local.
```

2. An address ("A" or "AAAA") record of the following form (where the IP address can be either an IPv4 address or an IPv6 address):

Listing 5: A record

```
machine.local. A ip-address
```

3. A SRV record of the following form:

Listing 6: SRV record

```
machine._xmpp-client._tcp.local <ttl> SRV <priority> <weight>
    port-number machine.local.
```

4. A TXT record whose name is the same as the SRV record and whose value follows the format described in the TXT Record section of this document, consisting of a set of strings that typically represent a series of key-value pairs such as the following:

[3] XEP-0174: Link-Local Messaging <https://xmpp.org/extensions/xep-0174.html>.
[4] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

Listing 7: TXT record

```
txtvers=1
   ordom=example.com
   regis=registry
   provis=provisioning
```

Note: The DNS-SD specification stipulates that the TXT record MUST be published, but that it MAY contain no more than a single zero byte (e.g., if the server does not wish to publish any personal information).

For the purpose of IoT Discovery we are interested only in the ”xmpp-client” service. An XMPP server MUST publish four different kinds of DNS records to advertise its availability using the services of type ”xmpp-client”. An XMPP chat client (actually its mDNS daemon) can send out multicast DNS queries for services of type ”xmpp-client”. **Note:** the service of type ”xmpp-client” is the reservered name for client-to-server connections by IANA, as described in RFC 6120 [5].
So, for example, if the machine name is ”pronto”, the IP address is ”10.2.1.188”, and the personal information, the DNS records would be as follows:

Listing 8: IoT Discovery DNS Records Example

```
_xmpp-client._tcp.local. PTR pronto._xmpp-client._tcp.local.

   pronto._xmpp-client._tcp.local. SRV 5222 pronto.local.

   pronto.local. A 10.2.1.188

   pronto._xmpp-client._tcp.local. IN TXT
   ”txtvers=1”
   ”ordom=example.com”
   ”regis=registry”
   ”provis=provisioning”
```

The IPv4 and IPv6 addresses associated with a machine might vary depending on the local network to which the machine is connected. For example, on an Ethernet connection the physical address might be ”192.168.0.100” but when the machine is connected to a wireless network the physical address might change to ”10.10.1.188”. See RFC 3927 [6] for details.
If the machine name asserted by a client is already taken by another machine on the network, the client MUST assert a different machine name, which SHOULD be formed by adding the character ”-” and digit ”1” to the end of the machine name string (e.g., ”pronto-1”), adding the character ”-” and digit ”2” if the resulting machine name is already taken (e.g., ”pronto-2”), and similarly incrementing the digit until a unique machine name is constructed.
Note: DNS-SD enables service definitions to include a TXT record that specifies parameters

[5] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[6] RFC 3927: Dynamic Configuration of IPv4 Link-Local Addresses <http://tools.ietf.org/html/rfc3927>.

to be used in the context of the relevant service type. For detailed information refer to Link-Local Messaging (XEP-0174) [7] (Link-Local Messaging - TXT Record).
The registration process is described in Link-Local Messaging (XEP-0174) [8] (Link-Local Messaging - Registration Process).
The following submission registers parameters in use as of MONTH 201x. Refer to the registry itself for a complete and current list of parameters (this specification might or might not be revised when new parameters are registered).

Listing 9: IoT Discovery TXT Record Parameters Registry

```
<param>
      <name>ordom</name>
      <desc>The "origin_domain" of the XMPP service.</desc>
      <status>recommended</status>
   </param>

   <param>
      <name>regis</name>
      <desc>
         The username portion of the JID to Thing Registry;
         can contain a space-separated list of more than one JID.
      </desc>
      <status>optional</status>
   </param>

   <param>
      <name>provis</name>
      <desc>
         The username portion of the JID to provisioning server;
         can contain a space-separated list of more than one JID.
      </desc>
      <status>optional</status>
   </param>
```

3.3.3 SSDP/UPnP

TBD
Note: If server-less messaging is to be used, as described in Link-Local Messaging (XEP-0174) [9] this step can be used to find the Thing Registry and optionally the Provisioning Server and other peers it want to connect to. The next section can thus be skipped.

[7] XEP-0174: Link-Local Messaging <https://xmpp.org/extensions/xep-0174.html>.
[8] XEP-0174: Link-Local Messaging <https://xmpp.org/extensions/xep-0174.html>.
[9] XEP-0174: Link-Local Messaging <https://xmpp.org/extensions/xep-0174.html>.

3.4 Connection to XMPP Server

Once an XMPP Server has been found, a connection can be made. If multiple XMPP Servers are found, the client is free to choose the one that best suits its purposes.
If the Thing does not have an account already, an account can be registered along what is specified in In-Band Registration (XEP-0077) [10]. If multiple servers are available, the first XMPP server that allows account creation can be used.

3.5 Finding Thing Registry

If a Thing Registry is not preconfigured, one must be found. A Thing Registry can be hosted either as a server component using Jabber Component Protocol (XEP-0114) [11] or as an XMPP Client accessible through a JID. The following lists methods to obtaining the Component Address or JID for the Thing Registry. Note that the last one has security considerations that need to be taken into account, if implemented.

1. Preconfigured Component Address of Thing Registry. A Component address is normally a subdomain to the domain of the XMPP Server that hosts the component.
2. Preconfigured bare JID of Thing Registry.
3. Preconfigured subdomain part of Component Address. This will be added to the domain of the XMPP Server used to connet to.
4. Preconfigured user name of JID. This will be added to the domain of the XMPP Server used to connected to.
5. Searching through Server Components on the XMPP Server currently connected to, as described in Determining Support.

3.6 Registering Thing

Once a Thing Registry has been found and been befriended, the Thing can register itself with the registry, as follows:

Listing 10: Register Thing

```
<iq type='set'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='1'>
      <register xmlns='urn:xmpp:iot:discovery'>
          <str name='SN' value='394872348732948723'/>
```

[10] XEP-0077: In-Band Registration <https://xmpp.org/extensions/xep-0077.html>.
[11] XEP-0114: Jabber Component Protocol <https://xmpp.org/extensions/xep-0114.html>.

```
        <str name='MAN' value='www.ktc.se'/>
        <str name='MODEL' value='IMC'/>
        <num name='V' value='1.2'/>
        <str name='KEY' value='4857402340298342'/>
    </register>
</iq>

<iq type='result'
    from='discovery.example.org'
    to='thing@example.org/imc'
    id='1'/>
```

There are two types of tags: Tags with string values and tags with numerical values. The distinction is important, since the type of value affects how comparisons are made later when performing searches in the registry.
The Thing should only register parameters required to be known by the owner of the Thing. Dynamic meta information must be avoided at this point. To claim the ownership of the Thing, the owner needs to present the same meta information as registered by the Thing. Before an owner has claimed ownership of the Thing, it will not be returned in any search results. A list of predefined meta tag names can be found in the Meta Tags section below.
The Thing can register itself as many times as it wants, and the response is always empty. Only one record per resource-less JID must be created. A new registration overrides any previous information, including meta tags previously reported but not available in the new registration. Once a Thing has been claimed by an owner, it should not register itself again, unless it is reset and the installation process restarted.
If the Thing tries to register itself even though the Thing has already been claimed in the registry, the registry must not update any meta data in the registry, and instead respond with the following response. When the thing receives this, it can safely extract the JID of the owner and switch its internal state to claimed.

Listing 11: Registration response when alrady claimed

```
<iq type='result'
    from='discovery.example.org'
    to='thing@example.org/imc'
    id='1'>
   <claimed xmlns='urn:xmpp:iot:discovery' jid='owner@example.org'/
      >
</iq>
```

Note: Meta Tag names are case insensitive. In this document, all tag names have been written using upper case letters.

3.7 Register self-owned Thing

If a thing is self-owned, it can register itself with the Registry as normal, with the addition of setting the attribute **selfOwned** to **true**, as is shown below. This registers the Thing directly as PUBLIC CLAIMED, with no need for an owner to claim ownership of the device. This can be useful if installing Things that should be publically available.

Listing 12: Register self-owned Thing

```
<iq type='set'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='2'>
      <register xmlns='urn:xmpp:iot:discovery' selfOwned='true'>
          <str name='SN' value='394872348732948723'/>
          <str name='MAN' value='www.ktc.se'/>
          <str name='MODEL' value='IMC'/>
          <num name='V' value='1.2'/>
          <str name='KEY' value='4857402340298342'/>
      </register>
   </iq>

   <iq type='result'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='2'/>
```

3.8 Register Thing behind Concentrator

A Thing might reside behind a gateway or concentrator and might not be directly connected to the XMPP network itself, as is described in Internet of Things - Concentrators (XEP-0326) [12]. In these cases, there are three optional attributes that can be used to identify the Thing behind the JID: The **nodeId** attribute gives the ID of the Thing (a.k.a. "Node"). The Node might reside in specific Data Source (large systems might have multiple sources of nodes). In this case, the data source is specified in the **sourceId** attribute. Normally, the Node ID is considered to be unique within the concentrator. If multiple data sources are available, the Node ID is unique within the data source. However, a third attribute allows the uniqueness to be restricted to a given **cacheType**. Finally, it is the triple (**nodeId**, **sourceId**, **cacheType**) which guarantees uniqueness within the concentrator.
For a Thing controlled by a concentrator to register itself in the Thing Registry, it simply adds the optional attributes **nodeId**, **sourceId** and **cacheType** as appropriate to the registration request, as follows:

[12] XEP-0326: Internet of Things - Concentrators <https://xmpp.org/extensions/xep-0326.html>.

Listing 13: Register Thing behind Concentrator

```
<iq type='set'
       from='rack@example.org/plcs'
       to='discovery.example.org'
       id='3'>
      <register xmlns='urn:xmpp:iot:discovery' nodeId='imc1' sourceId=
         'MeteringTopology'>
          <str name='SN' value='394872348732948723'/>
          <str name='MAN' value='www.ktc.se'/>
          <str name='MODEL' value='IMC'/>
          <num name='V' value='1.2'/>
          <str name='KEY' value='4857402340298342'/>
      </register>
   </iq>

   <iq type='result'
       from='discovery.example.org'
       to='rack@example.org/plcs'
       id='3'/>
```

If the Thing behind the concentrator is self-owned, it simply adds the **selfOwned** attribute to the request and sets it to **true**.

3.9 Claiming Ownership of a Thing

As mentioned above, the owner of the Thing must provide the information provided by the Thing to the Registry, in order to claim ownership over it. To avoid the possibility that somebody can guess the information, the information must necessarily be long. This creates the problem of transfer of information. One method to solve this is through the use of QR-codes. Such codes can be either printed on a sticker and put on the Thing itself, its wrapping, or displayed on its display when not claimed. This QR-code can then be photographed by a smart phone or tablet, decoded and the information retrieved can be used in the ownership claim call.
If QR-codes are used to transfer Thing meta data for ownership claims, it must be generated as follows: To the string ”IoTDisco” is appended all meta tags in order. Each tag name is prefixed by a semi-colon (;), and if the tag is numeric, the tag is prefixed by an additional hash sign (#). Each tag value is prefixed by a colon (:). If the meta value contains semi-colons or back-slashes, each one is prefixed by a back-slash. When decoding the string, this allows the decoder to correctly differ between tag delimiters and characters belonging to tag values. A tag name must never contain colon, hash sign or white space characters.
The above meta data would therefore generate the string:

Listing 14: String to encode as a QR-code

```
IoTDisco;SN:394872348732948723;MAN:www.ktc.se;MODEL:IMC;#V:1.2;
   KEY:4857402340298342
```

Using UTF-8 encoding when generating the QR-code, this string returns the following QR-code:

Once the client has the required meta information about the Thing to claim ownership, it sends itself the following request to the Thing Registry:

Listing 15: Claim Ownership of public Thing

```
<iq type='set'
       from='owner@example.org/phone'
       to='discovery.example.org'
       id='4'>
      <mine xmlns='urn:xmpp:iot:discovery'>
          <str name='SN' value='394872348732948723'/>
          <str name='MAN' value='www.ktc.se'/>
          <str name='MODEL' value='IMC'/>
          <num name='V' value='1.2'/>
          <str name='KEY' value='4857402340298342'/>
      </mine>
   </iq>
```

If this claim is successful, the Thing is marked as a public claimed Thing. The thing can always be removed later, but after the claim, the Thing is public. If you want to claim a private Thing, you can add the **public** attribute with value **false** to the claim, as follows:

Listing 16: Claim Ownership of private Thing

```
<iq type='set'
       from='owner@example.org/phone'
       to='discovery.example.org'
       id='4'>
      <mine xmlns='urn:xmpp:iot:discovery' public='false'>
          <str name='SN' value='394872348732948723'/>
          <str name='MAN' value='www.ktc.se'/>
          <str name='MODEL' value='IMC'/>
          <num name='V' value='1.2'/>
          <str name='KEY' value='4857402340298342'/>
      </mine>
   </iq>
```

In this case, if the claim is successful, the Thing will not be made public in the Thing Registry, after the claim.
If a claim is successful, i.e. there's a Thing that has not been claimed with EXACTLY the same meta data (however, the order is not important), the Thing is marked in the Registry as

CLAIMED, and as public or private depending on the **public** attribute, and an empty result is returned as follows. If there's a claimed Thing with exactly the same meta data, and the JID of the claimant (without resource) matches the JID of the claimer (without resource), a success response is also returned, containing the resource-less JID of the Thing, as follows:

Listing 17: Ownership claim successful

```
<iq type='result'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='4'>
      <claimed xmlns='urn:xmpp:iot:discovery' jid='thing@example.org'/
         >
   </iq>
```

If the Thing that has been claimed resides behind a concentrator, the result will contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing in calls made using Internet of Things - Sensor Data (XEP-0323) [13], Internet of Things - Provisioning (XEP-0324) [14], Internet of Things - Control (XEP-0325) [15] and Internet of Things - Concentrators (XEP-0326) [16]. The following example illustrates a response where a Thing behind a Concentrator has been claimed:

Listing 18: Ownership claim of a Thing behind a concentrator successful

```
<iq type='result'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='4'>
      <claimed xmlns='urn:xmpp:iot:discovery' jid='rack@example.org/
         plcs' nodeId='imc1' sourceId='MeteringTopology'/>
   </iq>
```

If, on the other hand, no such Thing was found, or if such a Thing was found, but it is already claimed by somebody else, a failure response is returned. This response should avoid to inform the client in detail why the claim failed, as follows:

Listing 19: Ownership claim failure

```
<iq type='error'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='4'>
      <error type='cancel'>
```

[13] XEP-0323: Internet of Things - Sensor Data <https://xmpp.org/extensions/xep-0323.html>.
[14] XEP-0324: Internet of Things - Provisioning <https://xmpp.org/extensions/xep-0324.html>.
[15] XEP-0325: Internet of Things - Control <https://xmpp.org/extensions/xep-0325.html>.
[16] XEP-0326: Internet of Things - Concentrators <https://xmpp.org/extensions/xep-0326.html>.

```
        <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      </error>
   </iq>
```

When the Thing has been successfully claimed, the Registry sends information about this to the Thing, to inform it that it has been claimed and the resource-less JID of owner. After receiving this information, it doesn't need to register itself with the Registry anymore.

Listing 20: Ownership claimed

```
<iq type='set'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='5'>
      <claimed xmlns='urn:xmpp:iot:discovery' jid='owner@example.org'/
         >
   </iq>
```

If the Thing was claimed as a private Thing, this is shown using the **public** attribute in the response, as follows:

Listing 21: Ownership claim of private Thing successful

```
<iq type='set'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='5'>
      <claimed xmlns='urn:xmpp:iot:discovery' jid='owner@example.org'
         public='false'/>
   </iq>
```

If the **public** attribute is present and has value **false**, it means no further meta data updates are necessary, since the device is not searchable through the Thing Registry.
If the Thing resides behind a concentrator, the request must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 22: Ownership of Thing behind concentrator claimed

```
<iq type='set'
       from='discovery.example.org'
       to='rack@example.org/plcs'
       id='5'>
      <claimed xmlns='urn:xmpp:iot:discovery' jid='owner@example.org'
         nodeId='imc1' sourceId='MeteringTopology'/>
   </iq>
```

The Thing simply returns an empty response to acknowledge the receipt of the information, as follows:

Listing 23: Ownership claimed acknowledged

```
<iq type='result'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='5'/>
```

After receiving this, the thing knows it can accept friendship requests from the corresponding owner. It can also safely send a friendship request to the owner.
Note: Meta Tag names are case insensitive. In this document, all tag names have been written using upper case letters.

3.10 Removing Thing from Registry

After a Thing has been claimed and is registed as a PUBLIC CLAIMED Thing in the Registry, it implies the Thing is available in searches. The owner can choose to remove the Thing from the Registry, to avoid that the Thing appears in searches. To remove a Thing from the Registry the owner simply sends a removal request to the Registry with the resource-less JID of the Thing to remove, as follows:

Listing 24: Remove Thing

```
<iq type='set'
       from='owner@example.org/phone'
       to='discovery.example.org'
       id='6'>
      <remove xmlns='urn:xmpp:iot:discovery' jid='thing@example.org'/>
   </iq>
```

If the Thing resides behind a concentrator, the request must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 25: Remove Thing behind concentrator

```
<iq type='set'
       from='owner@example.org/phone'
       to='discovery.example.org'
       id='6'>
      <remove xmlns='urn:xmpp:iot:discovery' jid='rack@example.org/
         plcs' nodeId='imc1' sourceId='MeteringTopology'/>
   </iq>
```

If such a Thing is found and is owned by the caller, it is removed from the Registry, and an empty response is returned, to acknowledge the removal of the Thing from the registry, as follows:

Listing 26: Thing removed

```
<iq type='result'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='6'/>
```

However, if such a thing is not found, or if the thing is owned by another, an **item-not-found** error is returned, as follows:

Listing 27: Removal failure

```
<iq type='error'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='6'>
      <error type='cancel'>
         <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      </error>
   </iq>
```

After successfully removing a Thing from the Registry, and if the Thing is friend to the Registry, the Registry informs the Thing it has been removed from the Registry. It does this, so the Thing can remove the friendship and stop any meta data updates to the Registry.

Listing 28: Thing removed from registry by owner

```
<iq type='set'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='7'>
      <removed xmlns='urn:xmpp:iot:discovery'/>
   </iq>

   <iq type='result'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='7'/>
```

If the Thing lies behind a concentrator, the removal request would look as follows:

Listing 29: Thing behind concentrator removed from registry by owner

```
<iq type='set'
       from='discovery.example.org'
       to='rack@example.org/plcs'
       id='7'>
      <removed xmlns='urn:xmpp:iot:discovery' nodeId='imc1' sourceId='
         MeteringTopology'/>
   </iq>
```

The Thing acknowledges the removal request by returning an empty response, as follows:

Listing 30: Removal acknowledgement

```
<iq type='result'
      from='thing@example.org/imc'
      to='discovery.example.org'
      id='7'/>
```

3.11 Finding Provisioning Server

Up to this point only basic configuration and ownership and visibility of a Thing has been covered. For more advanced operations, a Thing might be required to use a Provisioning Server to whom it can delegate trust and allow making decisions, controlling access rights and privileges for the Thing, as described in Internet of Things - Provisioning (XEP-0324) [17]. If a Provisioning Server is not preconfigured, one must be found. The following lists methods to obtaining the JID for the Provisioning Server.

1. Preconfigured Component Address of Provisioning Server. A Component address is normally a subdomain to the domain of the XMPP Server that hosts the component.
2. Preconfigured bare JID of Provisioning Server.
3. Preconfigured subdomain part of Component Address. This will be added to the domain of the XMPP Server used to connet to.
4. Preconfigured user name of JID. This will be added to the domain of the XMPP Server used to connected to.
5. The Thing Registry itself can be a Provisioning Server. This can be found out by sending a discovery request to the Thing Registry, as described in Determining Support.
6. The Owner itself can be a Provisioning Server. This can be found out by sending a discovery request to the Owner, as described in Determining Support.
7. Searching through Server Components on the XMPP Server currently connected to, as described in Determining Support.

3.12 Delegating Trust

Once a Provisioning Server has been found and been befriended, the Thing can delegate its trust to it, according to Internet of Things - Provisioning (XEP-0324) [18].

[17] XEP-0324: Internet of Things - Provisioning <https://xmpp.org/extensions/xep-0324.html>.
[18] XEP-0324: Internet of Things - Provisioning <https://xmpp.org/extensions/xep-0324.html>.

3.13 Update Meta Information about Thing in Registry

Once a Thing has been claimed and chooses to reside as a public Thing in the registry, it can update its meta information at any time. This meta information will be available in searches made to the registry by third parties and is considered public. However, the Thing should be connected to a provisioning server at this point, so that correct decisions can be made regarding to friendship, readout and control requests made by parties other than the owner. Meta information updated in this way will only overwrite tags provided in the request, and leave other tags previously reported as is. To remove a string-valued tag, it should be updated with an empty value. It is also recommended that key meta information required to claim ownership of the Thing after a factory reset is either removed, truncated or otherwise modified after it has been claimed so that third parties with physical access to a public Thing cannot hijack it by searching for it, extracting its meta information from the registry, then resetting it and then claiming ownership of it.
To update meta data about itself, a Thing simply sends a request to the Thing Registry, as follows:

Listing 31: Update Meta Data request

```
<iq type='set'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='8'>
      <update xmlns='urn:xmpp:iot:discovery'>
          <str name='KEY' value='{}'/>
          <str name='CLASS' value='PLC'/>
          <num name='LON' value='-71.519722'/>
          <num name='LAT' value='-33.008055'/>
      </update>
   </iq>
```

If the Thing resides behind a concentrator, the request must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 32: Update Meta Data of Thing behind concentrator

```
<iq type='set'
       from='rack@example.org/plcs'
       to='discovery.example.org'
       id='8'>
      <update xmlns='urn:xmpp:iot:discovery' nodeId='imc1' sourceId='
         MeteringTopology'>
          <str name='KEY' value='{}'/>
          <str name='CLASS' value='PLC'/>
          <num name='LON' value='-71.519722'/>
          <num name='LAT' value='-33.008055'/>
      </update>
```

```
    </iq>
```

If the Thing is found in the registry and it is claimed, the registry simply acknowledges the update as follows:

Listing 33: Update Meta Data request acknowledgement

```
<iq type='result'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='8'/>
```

However, if the Thing is not found in the registry, probably because the owner has removed it from the registry, an error response is returned. When receiving such a response, the Thing should assume it is the owner who has removed it from the registry, and that further meta data updates are not desired. The Thing can then unfriend the registry and stop further meta data updates. The error response from the registry would look as follows:

Listing 34: Update Meta Data request failure

```
<iq type='error'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='8'>
      <error type='cancel'>
         <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      </error>
   </iq>
```

If the Thing on the other hand is found in the Registry, but is not claimed, the registry must not update any meta data in the registry, and instead respond with the following response. When the thing receives this, the Thing can assume it has been disowned, and perform a new registration in the Registry so that it can be re-claimed.

Listing 35: Update Meta Data response to request from disowned Thing

```
<iq type='result'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='8'>
      <disowned xmlns='urn:xmpp:iot:discovery'/>
   </iq>
```

Note: Meta Tag names are case insensitive. In this document, all tag names have been written using upper case letters.

3.14 Owner updating Meta Information about Thing in Registry

An owner of a thing can also update the metadata of a thing it has claimed. To do this, you simply add a **jid** attribute containing the JID of the thing to the **update** element. (If this attribute is not present, the JID is assumed to be that of the sender of the message.)

Listing 36: Owner requests an update of Meta Data of Thing

```
<iq type='set'
       from='owner@example.org/1234'
       to='discovery.example.org'
       id='8'>
      <update xmlns='urn:xmpp:iot:discovery' jid='thing@example.org'>
          <str name='ROOM' value='...'/>
          <str name='APT' value='...'/>
          <str name='BLD' value='...'/>
          <str name='STREET' value='...'/>
          <str name='STREETNR' value='...'/>
          <str name='AREA' value='...'/>
          <str name='CITY' value='...'/>
          <str name='REGION' value='...'/>
          <str name='COUNTRY' value='...'/>
      </update>
   </iq>
```

The owner can update metadata of things behind concentrators also. To do this, the corresponding attributes **nodeId**, **sourceId** and **cacheType** must be used to identify the thing, as follows:

Listing 37: Owner requests an update of Meta Data of Thing behind concentrator

```
<iq type='set'
       from='owner@example.org/1234'
       to='discovery.example.org'
       id='8'>
      <update xmlns='urn:xmpp:iot:discovery' jid='rack@example.org'
         nodeId='imc1' sourceId='MeteringTopology'>
          <str name='ROOM' value='...'/>
          <str name='APT' value='...'/>
          <str name='BLD' value='...'/>
          <str name='STREET' value='...'/>
          <str name='STREETNR' value='...'/>
          <str name='AREA' value='...'/>
          <str name='CITY' value='...'/>
          <str name='REGION' value='...'/>
          <str name='COUNTRY' value='...'/>
      </update>
   </iq>
```

If the Thing is found in the registry and it is claimed by the sender of the current message (i.e. owner is the sender), the registry simply acknowledges the update as follows:

Listing 38: Owner updating thing Meta Data request acknowledgement

```
<iq type='result'
       from='discovery.example.org'
       to='owner@example.org/1234'
       id='8'/>
```

But if the owner is not the sender of the current message (i.e. owner is somebody else), or if the thing is not found at all, the server must report the node as **not existing** (i.e. not existing among the set of things claimed by the owner).

Listing 39: Owner updating thing Meta Data request failure

```
<iq type='error'
       from='discovery.example.org'
       to='owner@example.org/1234'
       id='8'>
      <error type='cancel'>
         <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      </error>
   </iq>
```

3.15 Search for Public Things in Registry

It is possible for anyone with access to the Thing Registry to search for public Things that have been claimed, including self-owned Things. Such searches will never return things that have not been claimed or have been removed from the registry.
A search is performed by providing one or more comparison operators in a search request to the registry. If more than one comparison operator is provided, the search is assumed to be performed on the intersection (i.e. AND) of all operators. If the union (i.e. OR) of different conditions is desired, multiple consecutive searches have to be performed.
The following table lists available search operators, their element names and meanings:

Element	Type	Operator	Description
strEq	String	tag = c	Searches for string values tags with values equal to a provided constant value.
strNEq	String	tag <> c	Searches for string values tags with values not equal to a provided constant value.

Element	Type	Operator	Description
strGt	String	tag > c	Searches for string values tags with values greater than a provided constant value.
strGtEq	String	tag >= c	Searches for string values tags with values greater than or equal to a provided constant value.
strLt	String	tag < c	Searches for string values tags with values lesser than a provided constant value.
strLtEq	String	tag <= c	Searches for string values tags with values lesser than or equal to a provided constant value.
strRange	String	min <(=) tag <(=) max	Searches for string values tags with values within a specified range of values. The endpoints can be included or excluded in the search.
strNRange	String	tag <(=) min OR tag >(=) max	Searches for string values tags with values outside of a specified range of values. The endpoints can be included or excluded in the range (and therefore correspondingly excluded or included in the search).
strMask	String	tag LIKE c	Searches for string values tags with values similar to a provided constant value including wildcards.
numEq	Numeric	tag = c	Searches for numerical values tags with values equal to a provided constant value.

Element	Type	Operator	Description
numNEq	Numeric	tag <> c	Searches for numerical values tags with values not equal to a provided constant value.
numGt	Numeric	tag > c	Searches for numerical values tags with values greater than a provided constant value.
numGtEq	Numeric	tag >= c	Searches for numerical values tags with values greater than or equal to a provided constant value.
numLt	Numeric	tag < c	Searches for numerical values tags with values lesser than a provided constant value.
numLtEq	Numeric	tag <= c	Searches for numerical values tags with values lesser than or equal to a provided constant value.
numRange	Numeric	min <(=) tag <(=) max	Searches for numerical values tags with values within a specified range of values. The endpoints can be included or excluded in the search.
numNRange	Numeric	tag <(=) min OR tag >(=) max	Searches for numerical values tags with values outside of a specified range of values. The endpoints can be included or excluded in the range (and therefore correspondingly excluded or included in the search).

The following example shows how a search for specific devices within a specific geographic area can be found. More precisely, it searches for a certain kind of PLC produced by a certain manufacturer, but only versions 1.0 <= v < 2.0 and with serial numbers beginning with 39487.

The PLCs must also lie within latitude 33 ad 34 degrees south and between longitude 70 and 72 west.

Listing 40: Searching for Things

```
<iq type='get'
       from='curious@example.org/client'
       to='discovery.example.org'
       id='9'>
      <search xmlns='urn:xmpp:iot:discovery' offset='0' maxCount='20'>
          <strEq name='MAN' value='www.ktc.se'/>
          <strEq name='MODEL' value='IMC'/>
          <strMask name='SN' value='39487*' wildcard='*'/>
          <numRange name='V' min='1' minIncluded='true' max='2'
              maxIncluded='false'/>
          <numRange name='LON' min='-72' minIncluded='true' max='-70'
              maxIncluded='true'/>
          <numRange name='LAT' min='-34' minIncluded='true' max='-33'
              maxIncluded='true'/>
      </search>
   </iq>
```

The **offset** attribute tells the registry the number of responses to skip before returning found things. It provides a mechanism to page result sets that are too large to return in one response. the **maxCount** attribute contains the desired maximum number of things to return in the response. The registry can lower this value, if it decides the requested maximum number is too large.
If tag names are not found corresponding to the names provided in the search, the result set will always be empty. There's a reserved tag named **KEY** that can be used to provide information shared only between things and their owners. If a search contains an operator referencing this tag name, the result set must also always be empty. Searches on **KEY** MUST never find things. Furthermore, search results must never return **KEY** tags.
The registry returns any things found in a response similar to the following:

Listing 41: Search result

```
<iq type='result'
       from='discovery.example.org'
       to='curious@example.org/client'
       id='9'>
      <found xmlns='urn:xmpp:iot:discovery' more='false'>
          <thing owner='owner@example.org' jid='thing@example.org'>
             <str name='SN' value='394872348732948723'/>
             <str name='MAN' value='www.ktc.se'/>
             <str name='MODEL' value='IMC'/>
             <num name='V' value='1.2'/>
             <str name='CLASS' value='PLC'/>
             <num name='LON' value='-71.519722'/>
```

```
            <num name='LAT' value='-33.008055'/>
        </thing>
        ...
    </found>
</iq>
```

If a Thing resides behind a concentrator, the response must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 42: Search result containing Thing behind a concentrator

```
<iq type='result'
    from='discovery.example.org'
    to='curious@example.org/client'
    id='9'>
    <found xmlns='urn:xmpp:iot:discovery' more='false'>
        <thing owner='owner@example.org' jid='rack@example.org'
            nodeId='imc1' sourceId='MeteringTopology'>
            <str name='SN' value='394872348732948723'/>
            <str name='MAN' value='www.ktc.se'/>
            <str name='MODEL' value='IMC'/>
            <num name='V' value='1.2'/>
            <str name='CLASS' value='PLC'/>
            <num name='LON' value='-71.519722'/>
            <num name='LAT' value='-33.008055'/>
        </thing>
        ...
    </found>
</iq>
```

If more results are available in the search (accessible by using the **offset** attribute in a new search), the **more** attribute is present with value **true**.
Note: Meta Tag names are case insensitive. In this document, all tag names have been written using upper case letters.

3.16 Unregistering Thing from Registry

A thing can unregister itself from the Registry. This can be done in an uninstallation procedure for instance. To unregister from the registry, it simply sends an un-registration request to the registry as follows.

Listing 43: Unregister Thing

```
<iq type='set'
    from='thing@example.org/imc'
    to='discovery.example.org'
    id='10'>
```

```
      <unregister xmlns='urn:xmpp:iot:discovery'/>
   </iq>
```

If the Thing resides behind a concentrator, the request must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 44: Unregistring Thing behind concentrator

```
<iq type='set'
       from='rack@example.org/plcs'
       to='discovery.example.org'
       id='10'>
      <unregister xmlns='urn:xmpp:iot:discovery' nodeId='imc1'
         sourceId='MeteringTopology'/>
   </iq>
```

The registry always returns an empty response, simply to acknowledge the receipt of the request.

Listing 45: Unregister Thing acknowledgement

```
<iq type='result'
       from='discovery.example.org'
       to='thing@example.org/imc'
       id='10'/>
```

3.17 Disowning Thing

The owner of a Thing can disown the Thing, returning it to a state without owner. This is done by sending the following request to the Thing Registry:

Listing 46: Disowning Thing

```
<iq type='set'
       from='owner@example.org/phone'
       to='discovery.example.org'
       id='11'>
      <disown xmlns='urn:xmpp:iot:discovery' jid='thing@example.org'/>
   </iq>
```

If the Thing resides behind a concentrator, the request must contain those of the attributes **nodeId**, **sourceId** and **cacheType** that are required to access the Thing, as follows:

Listing 47: Disowning Thing behind concentrator

```
<iq type='set'
       from='owner@example.org/phone'
```

```
      to='discovery.example.org'
      id='11'>
    <disown xmlns='urn:xmpp:iot:discovery' jid='rack@example.org/
       plcs' nodeId='imc1' sourceId='MeteringTopology'/>
  </iq>
```

If such a Thing is not found, or if the thing is not owned by the caller, an **item-not-found** error is returned, as follows:

Listing 48: Failure to disown Thing - Not Found

```
<iq type='error'
      from='discovery.example.org'
      to='owner@example.org/phone'
      id='11'>
    <error type='cancel'>
      <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    </error>
  </iq>
```

If such a Thing is found, and it is owned by the caller, but not online as a friend, the Thing cannot be disowned, since it would put the Thing in a state from which it cannot be re-claimed. Therefore, the Thing Registry must respond in the following manner:

Listing 49: Failure to disown Thing - Offline

```
<iq type='error'
      from='discovery.example.org'
      to='owner@example.org/phone'
      id='11'>
    <error type='cancel'>
      <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    </error>
  </iq>
```

Before returning a response to the caller, the Thing Registry informs the Thing it has been disowned. It does this, so the Thing can remove the friendship to the owner, and perform a new registration.

Listing 50: Thing disowned in registry by owner

```
<iq type='set'
      from='discovery.example.org'
      to='thing@example.org/imc'
      id='12'>
    <disowned xmlns='urn:xmpp:iot:discovery'/>
  </iq>
```

If the Thing lies behind a concentrator, the disowned request would look as follows:

Listing 51: Thing behind concentrator disowned in registry by owner

```
<iq type='set'
       from='discovery.example.org'
       to='rack@example.org/plcs'
       id='12'>
      <disowned xmlns='urn:xmpp:iot:discovery' nodeId='imc1' sourceId=
         'MeteringTopology'/>
   </iq>
```

The Thing acknowledges that it has been disowned by returning an empty response, as follows:

Listing 52: Acknowledging disownment

```
<iq type='result'
       from='thing@example.org/imc'
       to='discovery.example.org'
       id='12'/>
```

When receiving the acknowledgement from the Thing, the Thing is set as an unclaimed Thing in the Registry. Furthermore, all tags corresponding to the Thing are removed from the registry, and a random KEY tag is added of sufficient complexity to make sure other clients cannot claim the Thing by guessing. Finally, an empty response is returned, to acknowledge that the Thing has been disowned, as follows:

Listing 53: Thing disowned

```
<iq type='result'
       from='discovery.example.org'
       to='owner@example.org/phone'
       id='11'/>
```

If for any reason, the Thing does not acknowledge the disowned request, or an error occurs, the Registry returns the same error as if the Thing would have been offline.

4 Determining Support

If an entity is a Thing Registry and supports the protocol specified herein, it MUST advertise that fact by returning a feature of "urn:xmpp:iot:discovery" in response to Service Discovery (XEP-0030) [19] information requests.

[19] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

Listing 54: Service discovery information request

```
<iq type='get'
       from='device@example.org/device'
       to='provisioning@example.org'
       id='13'>
      <query xmlns='http://jabber.org/protocol/disco#info'/>
   </iq>
```

Listing 55: Service discovery information response

```
<iq type='result'
       from='provisioning@example.org'
       to='device@example.org/device'
       id='13'>
     <query xmlns='http://jabber.org/protocol/disco#info'>
        ...
        <feature var='urn:xmpp:iot:discovery'/>
        ...
     </query>
   </iq>
```

To search for a Thing Registry hosted as a component on an XMPP Server, you first request a list of available components, as follows:

Listing 56: Checking if server supports components

```
<iq from='device@example.org/device' to='example.org' type='get' id='
   14'>
      <query xmlns="http://jabber.org/protocol/disco#info"/>
   </iq>
```

Listing 57: Response confirming support for components

```
<iq type="result" id="14" from="example.org" to="device@example.org/
   device">
      <query xmlns="http://jabber.org/protocol/disco#info">
         ...
         <feature var="http://jabber.org/protocol/disco#items"/>
         ...
      </query>
   </iq>
```

If components (items) are supported, a request for available components is made:

Listing 58: Requesting list of server components

```
<iq from='device@example.org/device' to='example.org' type='get' id='
   15'>
      <query xmlns="http://jabber.org/protocol/disco#items"/>
   </iq>
```

Listing 59: Response containing list of server components

```
<iq type="result" id="15" from="example.org" to="995
    fab3dd759452ca9c370647323af0c@example.org/ebe2348e">
      <query xmlns="http://jabber.org/protocol/disco#items">
        ...
          <item jid="discovery.example.org" name="Registro de cosas"/>
        ...
      </query>
   </iq>
```

The client then loops through all components (items) and checks what features they support, until a Thing Registry is found:

Listing 60: Service discovery information request made to each component

```
<iq type='get'
       from='device@example.org/device'
       to='discovery.example.org'
       id='16'>
      <query xmlns='http://jabber.org/protocol/disco#info'/>
   </iq>
```

Listing 61: Service discovery information response from each component

```
<iq type='result'
       from='discovery.example.org'
       to='device@example.org/device'
       id='16'>
      <query xmlns='http://jabber.org/protocol/disco#info'>
         ...
         <feature var='urn:xmpp:iot:discovery'/>
         ...
      </query>
   </iq>
```

5 Implementation Notes

5.1 JID vs Component Thing Registries

A client must treat the connection between a Thing Registry differently if it is hosted as a client, having a JID, or if it is hosted as a Jabber Server Component. If it is hosted as a server component, there's no need for the thing to become friends with the Thing Registry. Messages and requests can be made directly to the server component without having to add it to the roster or request presence subscriptions. If the Thing Registry is hosted as a client, having a JID (@ in the address), the Thing Registry must be added to the roster of the client before the client can communicate with the Thing Registry.

5.2 Meta Tags

This document does not limit the number or names of tags used by Things to register meta information about themselves. However, it provides some general limits and defines the meaning of a few tags that must have the meanings specified herein.
The maximum length of a tag name is **32** characters. Tag names must not include colon (:), hash sign (#) or white space characters. String tag values must not exceed 128 characters in length.
The following table lists predefined tag names and their corresponding meanings.

Tag Name	Type	Description
ALT	Numeric	Altitude (meters)
APT	String	Apartment associated with the Thing
AREA	String	Area associated with the Thing
BLD	String	Building associated with the Thing
CITY	String	City associated with the Thing
CLASS	String	Class of Thing
COUNTRY	String	Country associated with the Thing
KEY	String	Key, shared between thing and owner.
LAT	Numeric	Latitude (degrees)
LON	Numeric	Longitude (degrees)
MAN	String	Domain name owned by the Manufacturer
MLOC	String	Meter Location ID
MNR	String	Meter Number
MODEL	String	Name of Model
NAME	String	Name associated with the Thing
PURL	String	URL to product information for the Thing.
REGION	String	Region associated with the Thing
ROOM	String	Room associated with the Thing
SN	String	Serial Number
STREET	String	Street Name
STREETNR	String	Street Number
V	Numeric	Version Number

It is up to the Thing Registry to choose which tags it persists and which tags it doesn't. To avoid the possibility of malicious reporting of tags, some limit should be imposed on what tags are supported. As a minimum, a Thing Registry must support all predefined tags, as listed above.
Note: Meta Tag names are case insensitive. In this document, all tag names have been written

using upper case letters.

5.3 Friendships between Things and Registry

In the case the Thing Registry is not the XMPP Server to which the Thing is connected, a friendship relationship between the Thing and the Thing Registry needs to be handled. To minimize the number of concurrent friends the Thing Registry needs to maintain, a Thing must only maintain an active friendship with the registry if it needs to communicate with the registry. This means that unless updating meta data frequently, the Thing must unfriend the Registry when done with its communication. If only updating meta data intermittently, the friendship can be reestablished when needed, and removed when done.

6 Security Considerations

6.1 Jabber Components Protocol

The Jabber Component Protocol (XEP-0114) [20] provides an elegant way to introduce external services as server components using a third port into the server (the first two being the client-to-server port and the server-to-server port). But since XEP-0114 is historical, meaning it is not guaranteed to conform to v1.0 of the XMPP specification, it has some serious security issues:

1. It lacks SSL/TLS support, or the starttls element to switch to TLS after connecting. This makes it possible to sniff traffic in this port.
2. It lacks SASL authentication. Instead a simple handshake is performed
3. There is no way to actually verify that the server is the server. This makes it possible to create a simple Man-in-the-middle attack.

For these reasons, it is not recommended that a Thing Registry service, publishing itself as a Jabber Server Component, does so from outside of the network. Instead, the Thing Registry should be installed on the same server or on a server in the same local area network, so that the Jabber Component protocol port is closed to the Internet.
Since it is not guaranteed that an XMPP Server operator allows installation of third party products (such as a Thing Registry), the option to host a Thing Registry using a normal JID is still available. It can be used in proof of concepts, etc. For scalability issues it is recommended that the Thing Registry be hosted as a Jabber Server Component when the population of Things grows.

[20] XEP-0114: Jabber Component Protocol <https://xmpp.org/extensions/xep-0114.html>.

6.2 Hijacking predefined JIDs

If using predefined user names when searching for a Thing Registry or Provisioning Server, care must be taken to which XMPP Server things connect. It might be possible for third parties to register these predefined account names, and pretend to be a Thing Registry or Provisioning Server and in this way hijack unsuspecting Things. If installing things using this method of finding a Thing Registry or Provisioning Server, these accounts must be registered beforehand, to make sure the things cannot be hijacked.

6.3 Hijacking things in public areas

The combination of visible key meta information (perhaps in a visible QR-code) and a factory default reset button on a Thing, opens up the possibility to hijack the Thing. To avoid this, at least one of the two should be removed after successful installation. Either the key meta information (QR-code) should be placed on the package or separate paper and not on the thing itself, or the factory default reset button should be sealed or hidden and only accessible by licensed maintenance personell. If using an electronic means to present the key meta information (for instance by displayed a QR-code on a display on the thing), care should be taken so that the information cannot be displayed without breaking a seal, or other means to protect the Thing.
Regardless the above security measures, a Thing can be hijacked by a third party in the time window between successful installation of the device and until the correct owner has claimed ownership of the device. Minimizing this time window, and using a shared secret (KEY tag) between the Thing and its owner, decreases the possibility of getting the thing hijacked.

6.4 Key meta information in searches

Care should be taken what key meta information is used to accept an ownership claim. After a successful claim, this meta information is still available in the registry, at least until the Thing is removed from the registry. While public in the registry, the meta information can be searched and presented to third parties. Access to this information can help third parties to hijack Things, if they can reset them to factory default settings.
To avoid this, the Thing can do three things after a successful ownership claim:

- Including a **KEY** tag in the key meta information. The **KEY** tag is not searchable nor presented in search results.
- Remove, truncate or change some key meta information after a successful ownership claim. Partial information is not sufficient for a successful ownership claim.
- Remove the Thing from the registry.

6.5 KEY tag

The **KEY** tag is unique in that it is not searchable nor available is search results. For this reason it is ideal for providing secrets shared only between the Thing and the owner. By providing a sufficiently long KEY value in the key meta information required to claim the Thing, guessing the information even though the other meta information is available, will be sufficiently hard to make it practically impossible.
Even though the **KEY** tag is not searchable or available in search results, it should be emptied by the Thing after a successful claim, just to make sure the key cannot be learned by looking into the database of the registry, or by some other means.

6.6 Tag name spam

This document does not limit tag names or the number of tags that can be used by Things. This opens up the possibility of tag spam. Malicious things could fill the database of the registry by reporting random tag names until the database is full.
To prevent such malicious attacks, the registry could limit the tags it allows to be stored in the database. The registry must however allow the storage of the predefined tag names defined in this document. If it has a configurable list of approved tags that can be stored, or if it allows any tags is an implementation decision.

6.7 External services for creating QR-codes

If using external services when creating QR-codes, like the Google Charts API used in this document, make sure HTTPS is used and certificates validated. If HTTP is used, metadata tags used in Thing Registry registrations can be found out by sniffing the network, making it possible to hijack the corresponding devices.

6.8 DHCP Security Considerations

TBD

6.9 DNS Security Considerations

TBD

6.10 UPnP Security Considerations

TBD

7 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [21].

8 XMPP Registrar Considerations

The protocol schema needs to be added to the list of XMPP protocol schemas.

9 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:iot:discovery'
    xmlns='urn:xmpp:iot:discovery'
    elementFormDefault='qualified'>

    <xs:element name='register' type='Register'/>
    <xs:element name='mine' type='Mine'/>
    <xs:element name='update' type='Update'/>

    <xs:element name='claimed' type='Claimed'/>
    <xs:element name='remove' type='Jid'/>
    <xs:element name='removed' type='NodeInfo'/>
    <xs:element name='unregister' type='NodeInfo'/>
    <xs:element name='disown' type='Jid'/>
    <xs:element name='disowned' type='NodeInfo'/>

    <xs:element name='search'>
        <xs:complexType>
            <xs:choice minOccurs='1' maxOccurs='unbounded'>
                <xs:element name='strEq' type='StrTag'/>
                <xs:element name='strNEq' type='StrTag'/>
                <xs:element name='strGt' type='StrTag'/>
                <xs:element name='strGtEq' type='StrTag'/>
                <xs:element name='strLt' type='StrTag'/>
                <xs:element name='strLtEq' type='StrTag'/>
                <xs:element name='strRange' type='StrRange'/>
                <xs:element name='strNRange' type='StrRange'/>
                <xs:element name='strMask'>
```

[21] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

```
                    <xs:complexType>
                        <xs:attribute name='name' type='xs:string' use
                            ='required'/>
                        <xs:attribute name='value' type='xs:string'
                            use='required'/>
                        <xs:attribute name='wildcard' type='xs:string'
                             use='required'/>
                    </xs:complexType>
                </xs:element>
                <xs:element name='numEq' type='NumTag'/>
                <xs:element name='numNEq' type='NumTag'/>
                <xs:element name='numGt' type='NumTag'/>
                <xs:element name='numGtEq' type='NumTag'/>
                <xs:element name='numLt' type='NumTag'/>
                <xs:element name='numLtEq' type='NumTag'/>
                <xs:element name='numRange' type='NumRange'/>
                <xs:element name='numNRange' type='NumRange'/>
            </xs:choice>
            <xs:attribute name='offset' type='xs:nonNegativeInteger'
                use='optional' default='0'/>
            <xs:attribute name='maxCount' type='xs:positiveInteger'
                use='optional'/>
        </xs:complexType>
    </xs:element>

    <xs:element name='found'>
        <xs:complexType>
            <xs:sequence minOccurs='0' maxOccurs='unbounded'>
                <xs:element name='thing'>
                    <xs:complexType>
                        <xs:choice minOccurs='0' maxOccurs='unbounded'
                            >
                            <xs:element name='str' type='StrTag'/>
                            <xs:element name='num' type='NumTag'/>
                        </xs:choice>
                        <xs:attribute name='owner' type='xs:string'
                            use='required'/>
                        <xs:attribute name='jid' type='xs:string' use=
                            'required'/>
                        <xs:attributeGroup ref='nodeInfo'/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name='more' type='xs:boolean' use='optional'
                 default='false'/>
        </xs:complexType>
    </xs:element>

    <xs:attributeGroup name='nodeInfo'>
```

```
        <xs:attribute name='nodeId' type='xs:string' use='optional'/>
        <xs:attribute name='sourceId' type='xs:string' use='optional'/
            >
        <xs:attribute name='cacheType' type='xs:string' use='optional'
            />
    </xs:attributeGroup>

    <xs:complexType name='MetaData'>
        <xs:choice minOccurs='0' maxOccurs='unbounded'>
            <xs:element name='str' type='StrTag'/>
            <xs:element name='num' type='NumTag'/>
        </xs:choice>
    </xs:complexType>

    <xs:complexType name='MetaDataNodeInfo'>
        <xs:complexContent>
            <xs:extension base='MetaData'>
                <xs:attributeGroup ref='nodeInfo'/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name='Register'>
        <xs:complexContent>
            <xs:extension base='MetaDataNodeInfo'>
                <xs:attribute name='selfOwned' type='xs:boolean' use='
                    optional' default='false'/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name='Mine'>
        <xs:complexContent>
            <xs:extension base='MetaDataNodeInfo'>
                <xs:attribute name='public' type='xs:boolean' use='
                    optional' default='true'/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name='Update'>
        <xs:complexContent>
            <xs:extension base='MetaDataNodeInfo'>
                <xs:attribute name='jid' type='xs:string' use='
                    optional'/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

```
    <xs:complexType name='Claimed'>
        <xs:complexContent>
            <xs:extension base='Jid'>
                <xs:attribute name='public' type='xs:boolean' use='
                    optional' default='false'/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name='Jid'>
        <xs:attribute name='jid' type='xs:string' use='required'/>
        <xs:attributeGroup ref='nodeInfo'/>
    </xs:complexType>

    <xs:complexType name='NodeInfo'>
        <xs:attributeGroup ref='nodeInfo'/>
    </xs:complexType>

    <xs:complexType name='StrTag'>
        <xs:attribute name='name' type='xs:string' use='required'/>
        <xs:attribute name='value' type='xs:string' use='required'/>
    </xs:complexType>

    <xs:complexType name='NumTag'>
        <xs:attribute name='name' type='xs:string' use='required'/>
        <xs:attribute name='value' type='xs:double' use='required'/>
    </xs:complexType>

    <xs:complexType name='StrRange'>
        <xs:attribute name='name' type='xs:string' use='required'/>
        <xs:attribute name='min' type='xs:string' use='required'/>
        <xs:attribute name='minIncluded' type='xs:boolean' use='
            optional' default='true'/>
        <xs:attribute name='max' type='xs:string' use='required'/>
        <xs:attribute name='maxIncluded' type='xs:boolean' use='
            optional' default='true'/>
    </xs:complexType>

    <xs:complexType name='NumRange'>
        <xs:attribute name='name' type='xs:string' use='required'/>
        <xs:attribute name='min' type='xs:double' use='required'/>
        <xs:attribute name='minIncluded' type='xs:boolean' use='
            optional' default='true'/>
        <xs:attribute name='max' type='xs:double' use='required'/>
        <xs:attribute name='maxIncluded' type='xs:boolean' use='
            optional' default='true'/>
    </xs:complexType>

</xs:schema>
```

10 For more information

For more information, please see the following resources:

- The Sensor Network section of the XMPP Wiki contains further information about the use of the sensor network XEPs, links to implementations, discussions, etc.
- The XEP's and related projects are also available on github, thanks to Joachim Lindborg.
- A presentation giving an overview of all extensions related to Internet of Things can be found here: `http://prezi.com/esosntqhewhs/iot-xmpp/`.

11 Acknowledgements

Thanks to Eelco Cramer, Henrik Svedlund, Ivan Vučica, Joachim Lindborg, Joakim Eriksson, Joakim Ramberg, Johannes Hund, Karin Forsell, Kevin Smith, Lance Stout, Lars Åkerskog, Olof Zandrén, Philipp Hancke, Steffen Larsen, Teemu Väisänen and Yusuke Doi for all valuable feedback.