Eclipse Smarthome
IOTA Integration

Théo Giovanna

May 2018

1 Global Configuration

1.1 Metadata

This add-on allows you to expose your SmartHome system to the IOTA blockchain, **Tangle**. Using this add-on, you will be able to configure for each *Thing* of your choice some *metadata*. Metadata are extra-information that can be attached to an *item*. This add-on offers two types of metadata:

- `value:` *"yes"* — `label:` *"Share the item's state with the Tangle"*
- `value:` *"no"* — `label:` *"Don't share the item's state with the Tangle"*

The *"yes"* value allows for some more fine tuning:

- `param:` *"mode"* — `public, private, restricted`
- `param:` *"key"* — `Optional key for restricted mode`
- `param:` *"seed"* — `Optional seed. Allows one to publish on an existing channel`

For information about what *Things*, *Channels* and *Items* are, please refer to the official ESH documentation. To be able to configure metadata for your items, you must ensure that the option `Simple Mode` under `Configuration > System > Item Linking` is turned off.

1.2 IOTA

Besides configuring the Item metadata, one must configure the IOTA API settings, by which information will transit. As described in the JAVA wrapper documentation for IOTA, an `IotaAPI` instance must be created in order to interact with the Tangle. This instance has three settings: *protocol, host, port*. By default, this add-on is configured with the following settings:

- Protocol: `https`
- Host: `iotanode.us`
- Port: `443`

Each one of these settings can be edited at any time through the PaperUI, under `Configuration > Services`. You are free to use any remote nodes running the IOTA protocol, as well as local nodes.

2 Technical information

The purpose of this section is to describe how the add-on works, as well as giving some hints on why IOTA was chosen and how this can be beneficial for your smart home system.

2.1 The Tangle

The Tangle isn't a blockchain per se, as it is block-less. Instead of miners solving cryptographic puzzles in order to verify new blocks, the Tangle is quite different: there are no miners. Instead, the users themselves verify the transaction: to create a new transaction, one needs to validate two other transactions. This has the additional benefit of speeding up the network the more one uses it. Moreover, Tangle is fee-less which makes it a perfect fit for micro-transactions. IOTA is the name of the cryptocurrency that is based upon Tangle.

In a world where the *Internet of Things*, or *IoT*, is taking more and more breadth, the Tangle has a key role to play. The main challenge isn't in creating smart devices or machines, but rather it is at the communication level. The Tangle enables a future in which devices can securely communicate with and autonomously compensate each other. For more information on how the Tangle works at his core, I encourage you to read this series of articles published on their blog. You can also view their official white-paper here.

2.2 ESH IOTA Add-on

The basic idea behind this add-on is to use the Tangle capabilities in order to share any item's state on it. Lately, IOTA revealed a new feature running on the Tangle: **Masked Authenticated Messaging**, or MAM for short. MAM is a second layer data communication protocol which adds functionality to emit and access an encrypted data stream, like RSS, over the Tangle, regardless of the size or cost of device. IOTA's consensus protocol adds integrity to these message streams. Information on MAM can be found on their official blog. This add-on therefore uses MAM as a mean of communication between any ESH thing (providers) and entities that want to know the states of these things (subscribers).

2.2.1 Publishing an item's state to the Tangle

The iota add-on implements several interfaces from ESH, some of which are enumerated and explained below:

- `RegistryChangeListener<Metadata>`, implemented by `IotaService`: Each time a metadata is added, updated or removed, the implementing class is notified.
- `ItemRegistryChangeListener`, implemented by `IotaItemRegistryListener`: Each time an item is added, updated or removed, the implementing class is notified.
- `StateChangeListener`, implemented by `IotaItemStateChangeListener`: Each time an item sees its state changed or updated, the implementing class if notified.
- `MetadataConfigDescriptionProvider`, implemented by `IotaMetadataProvider`: Allows one to offer different configuration for metadata in the PaperUI.

The flow is therefore as follow: anytime one adds metadata to an existing item offered by this add-on or that a new item is created with metadata "iota" with value "yes", the add-on will pick up the event and monitor future state changes of this item, by associating to it a `StateListener` as described above.

For instance, consider the following item, defined in `demo.items`:

```
Number:Temperature Weather_Temperature "OutsideTemperature[%.1f°C]" <temperature>
(Weather_Chart) {iota="yes" [mode="public", seed="yourSeed", key="null"],
channel= "yahooweather:weather:berlin:temperature" }
```

And in `demo.things`:

```
yahooweather:weather:berlin [ location=638242 ]
```

MAM allows for three different publishing modes: `public, private, restricted`. These options can be configured in PaperUI through metadata. More information on what achieves each mode can be found in the blog link given above. Each item may also be published on a given MAM channel, through *seeds*. In such scenario, the flow is as follow: let X be an ESH instance with items A, B, C, D. Let Y and Z be to other ESH instances. X may choose to share on a `public, private or restricted` channel A and B's state with Y. Therefore, all states will be published on seed s_0 with root r_0. Now, X may also choose to share with Z on a separate channel (`public, private or restricted`) C and D's state, on seed s_1. X is now broadcasting data on two different channels, each belonging to a seed: s_0, s_1. To add new item's state to any of these streams, say the first one broadcasting to Y, X just has to configure a new item's metadata with the seed he's using for Y, in this case s_0.

In the previously defined item, we defined the metadata `"iota=yes"` with several parameters. The add-on will detect it and associate a state listener to this item. From now on, every update of this item will be detected and pushed on the Tangle. In MAM, the first transaction emitted has a `Root` address, which is derived from the **seed**. Any sub-sequent message (i.e. any new update on the item's state) will be published on the `NextRoot` address, which is contained in any transaction retrieved from MAM. Given the first `root` address, or any `nextRoot` address that follows, one can see all the states shared from this address onward using the official MAM-Explorer. For instance:


```
[
  {
    "NAME": "YAHOOWEATHER_WEATHER_BERLIN_HUMIDITY",
    "STATUS": {
      "TOPIC": "HUMIDITY",
      "STATE": "51.0",
      "TIME": "2018-06-19T17:49:03.273Z"
    }
  }
]
```


Figure 1: Retrieving item states through MAM

3 Javadoc

Note: getters and setters are left out of this java documentation, which is here just to give the user a grasp of how this add-on is organized and what are the responsibilities of each class.

3.1 Debouncer.java

```
/**
 * Implements a debounce mechanism.
 *
 * @author Theo Giovanna - Initial Contribution
 */
public class Debouncer {
        /**
         * Debounces {@code callable} by {@code delay}, i.e., schedules it to be
         * executed after {@code delay}, or cancels its execution if the method
         * is called with the same key within the {@code delay} again.
         * @author simon04
         * https://stackoverflow.com/questions/4742210/implementing-debounce-in-java
         */
         public void debounce(final Object key, final Runnable runnable, long delay,
         TimeUnit unit) {}
}
```

3.2 Iota.java

```
/**
 * Set up the IOTA API and the metadata listener
 * @author Theo Giovanna - Initial Contribution
 */
public class Iota {

    protected synchronized void activate(Map<String, Object> data) {}
    protected void deactivate() {}

    /**
     * This function is called every time that the user updates the API parameters
     * in Paper UI. The Iota API instance is then updated accordingly
     */
    protected synchronized void modified(IotaApiConfiguration config) {}

    /**
     * Called when the component is started. Set up the IOTA API
     */
    private void start() {}
}
```

3.3 IotaApiConfiguration.java

```java
/**
 * Configuration parameters of the {@link Iota}.
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaApiConfiguration {

}
```

3.4 IotaItemRegistryListener.java

```java
/**
 * Listens for changes to the item registry.
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaItemRegistryListener implements ItemRegistryChangeListener {

        /**
         * Called when items are added, removed or updated
         */
        public void added(Item element) {}
        public void removed(Item element) {}
        public void updated(Item oldElement, Item element) {}
        public void allItemsChanged(Collection<String> oldItemNames) {}

        /**
         * Add a registry change listener
         */
        public synchronized void setItemRegistry(ItemRegistry itemRegistry) {}

}
```

3.5 IotaItemStateChangeListener.java

```
/**
 * Listens for changes to the state of registered items.
 * Each item wishing to publish its state on the Tangle has a seed associated to its
 * UID. A custom seed may be given to some item. This way, any ESH instance is able
 * to select which item's state to share on which channel.
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaItemStateChangeListener implements StateChangeListener {

        /**
         * Called when a registered item sees its state changed or updated
         */
        public void stateChanged(@NonNull Item item, @NonNull State oldState,
        @NonNull State newState) {}
        public void stateUpdated(@NonNull Item item, @NonNull State state) {}

        /**
         * Constructs a JSON object with all item names and states that will
         * be published on the Tangle.
         */
         public synchronized void addToStates(@NonNull Item item,
         @NonNull State state) {}

         /**
         * Cleaning json struct: an item has been removed in the Paper UI, therefore
         * its state will not be published to the Tangle anymore
         */
         public void removeItemFromJson(@NonNull Item item) {}

}
```

3.6 IotaMetadataProvider.java

```
/**
 * Describes the metadata for the "iota" namespace.
 * @author Theo Giovanna - initial contribution
 */
public class IotaMetadataProvider implements MetadataConfigDescriptionProvider {

}
```

3.7 IotaService.java

```
/**
 * Listens for changes to the metadata registry.
 * This class will allow items to be listened to through the
 * IotaItemStateChangeListener class if they contain the metadata IOTA, when
 * created.
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaService implements RegistryChangeListener<Metadata> {

        /**
         * Called when metadata are added, removed or updated
         */
        public void added(Metadata element) {}
        public void removed(Metadata element) {}
        public void updated(Metadata oldElement, Metadata element) {}

        /**
        * Updates the hashmaps in the {@link IotaStateListener} class.
        */
        public void updateMaps(Item item, String seed) {}

        /**
         * Add a registry change listener
         */
        public void setMetadataRegistry(MetadataRegistry metadataRegistry) {}

        /**
         * Remove the registry change listener
         */
        public void stop() {}

}
```

3.8 IotaSettings.java

```
/**
 * Provides the configured and static settings for the IOTA add-on
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaSettings {

        /**
         * Sets the config. parameters
         * Adapted from org.openhab.io.homekit.internal
         * @author Andy Lintner
         */
        public void fill(IotaApiConfiguration config) throws UnknownHostException {}
        private static String getOrDefault(Object value, String defaultValue) {}
}
```

3.9 IotaUtils.java

```
/**
 * Provides utils methods to work with IOTA transactions
 * @author Theo Giovanna - Initial Contribution
 */
public class IotaUtils {

        /**
         * Attach an item's state to the Tangle
         */
        protected void publishState(JsonElement jsonElement, String mode,
        String key) {}

        /**
         * Retrieve an item state from the Tangle, through MAM
         */
         public String fetchFromTangle(int refresh, String root, String mode) {}

         /**
          * Checks the IOTA API
          */
          public boolean checkAPI() {}


}
```