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"*

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. Each transaction running on the Tangle contains several fiels: the receiving address, the amount... A field of particular interest is the *SignatureFragments* one. This field allows one to add a message to the transaction, encoded in *trytes*. Alike *bytes* that are composed of *bits*, a tryte is composed of several *trits*. These trytes carry information, and in our case we use them to carry the item's state.

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", channel= "yahooweather:weather:berlin:temperature" }
```

And in `demo.things`:

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

Since we defined the metadata `"iota=yes"`, 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, as a message of an IOTA transaction.

2.2.2 Retrieving an item's state from the Tangle

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 `Iota.java`

```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) {}

    /**
     * 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) {}

    /**
     * Deactivate the component
     */
    protected void deactivate() {}

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

3.2 `IotaApiConfiguration.java`

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

}
```

3.3 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) {}

}
```

3.4 IotaItemStateChangeListener.java

```java
/**
 * Listens for changes to the state of registered items.
 * @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) {}

}
```

3.5 IotaMetadataProvider.java

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

}
```

3.6 IotaService.java

```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) {}

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

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

}
```

3.7 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) {}
}
```

3.8 IotaUtils.java

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

        /**
         * Cut the fragment message contained in the IOTA transaction to remove the
         * padded 9's and isolate the message
         */
        protected String revealMessage(String trytes) {}

        /**
         * Attach an item's state to the Tangle
         */
        protected void publishState(@NonNull IotaAPI bridge, @NonNull Item item,
        State state, String to, String seed) {}

        /**
         * Returns the state of an item, contained in an IOTA transaction
         */
        protected String getStateFromTransaction(String[] transactions,
        @NonNull IotaAPI bridge) {}

}
```