Industry Marketplace
Technical Documentation

Version 1.0 - Sept. 20th, 2019 - Alexey Sobolev, Lisa Schneider

Summary

The next generation of industrial automation, Industry 4.0 (I4.0), is rapidly approaching. In tomorrow's world, devices will contain not only asset information, but proactive decision and optimization algorithms to enable goal-oriented behavior among their components. Such I4.0 devices can be viewed as autonomous independent economic agents that cooperate according to market economy principles.

The highly flexible value creation networks that result from I4.0 will require new forms of collaboration between companies - both at the national and global level. And the successful implementation of I4.0 will depend on the creation of a common global communication and computing infrastructure that allows economic relationships between machines.

By combining the latest technology with established standards and openly-developed specifications, the Industry Marketplace will provide this platform and enable the economy of things.

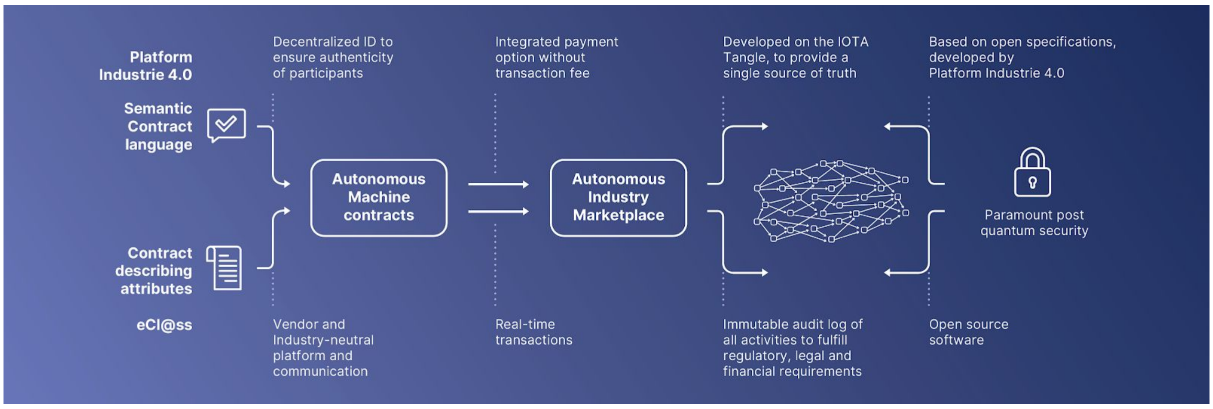
The Industry Marketplace will serve as a vendor and industry-neutral platform, automating the trading of physical and digital goods / services. Building on specifications developed by the Plattform Industrie 4.0 (Germany's central network for the advancement of digital transformation in manufacturing), the Industry Marketplace combines distributed ledger technology, immutable audit logs, and standardized, machine-readable contracts with an integrated decentralized identity system, to ensure the authenticity of all participants and enable secure communication and payments across the industry landscape.

The Industry Marketplace has been developed as an open source initiative and is free to join. A simple trial can be run at your office to explore its potential. We encourage open innovation with other industry partners to explore new business models and the many possibilities of industrial automation.

Key features

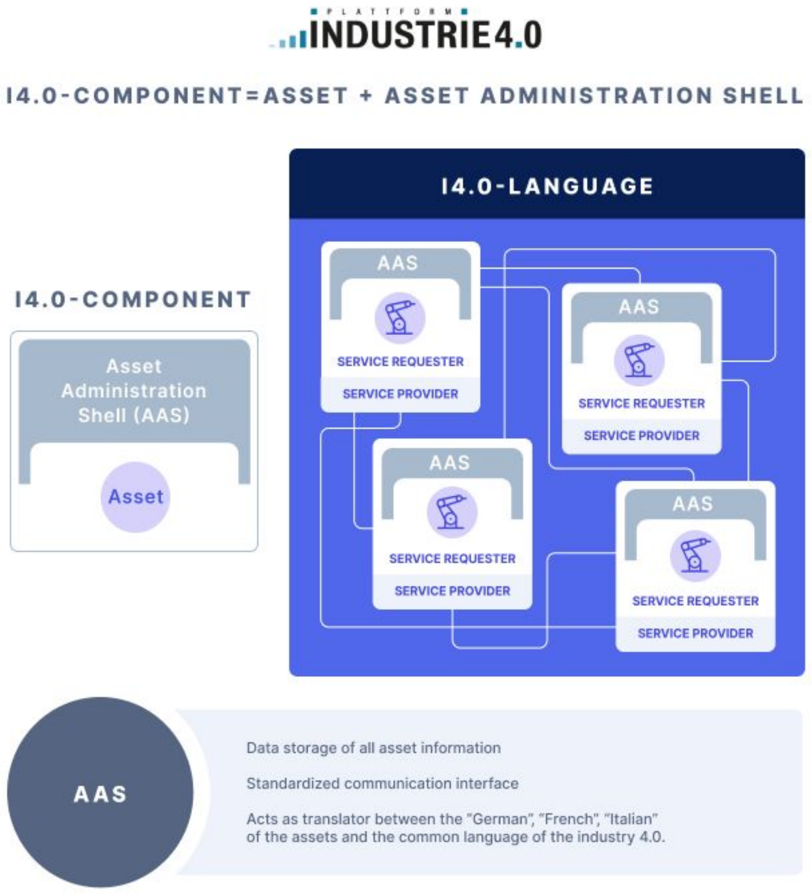
- Vendor and industry-neutral platform and communication
- Standardised communication for contracts, product data, purchasing, bids, orders, services
- Implementation of the I4.0 principles for driving forward digitalization and manufacturing
- Semantic language, based on open specifications, developed by Plattform Industrie 4.0 and academic institutions
- A decentralized and globally accessible protocol with paramount security
- Low system requirements
- Open source software
- Integrated, decentralized ID, to ensure the authenticity of all participants
- Integrated payment option for goods and services, without transaction fees
- Payment queues to execute outgoing payments in high frequency environments, e.g. buying many individual data sets, like weather data

- Immutable audit logs for every step (including payments) to be compliant with regulatory requirements
- Digital trust as a design principle throughout the IOTA Tangle


Platform Industrie 4.0
Semantic Contract language
Contract describing attributes
eCl@ss
Decentralized ID to ensure authenticity of participants
Vendor and Industry-neutral platform and communication
Autonomous Machine contracts
Integrated payment option without transaction fee
Real-time transactions
Autonomous Industry Marketplace
Developed on the IOTA Tangle, to provide a single source of truth
Immutable audit log of all activities to fulfill regulatory, legal and financial requirements
Based on open specifications, developed by Platform Industrie 4.0
Paramount post quantum security
Open source software

Industry 4.0

Industry 4.0 (I4.0) promises substantial changes in organizational and manufacturing management. Interaction between “intelligent” manufacturing components (i.e. cyber physical systems) will yield dynamic, self-organizing, self-optimizing, cross-company value chains.

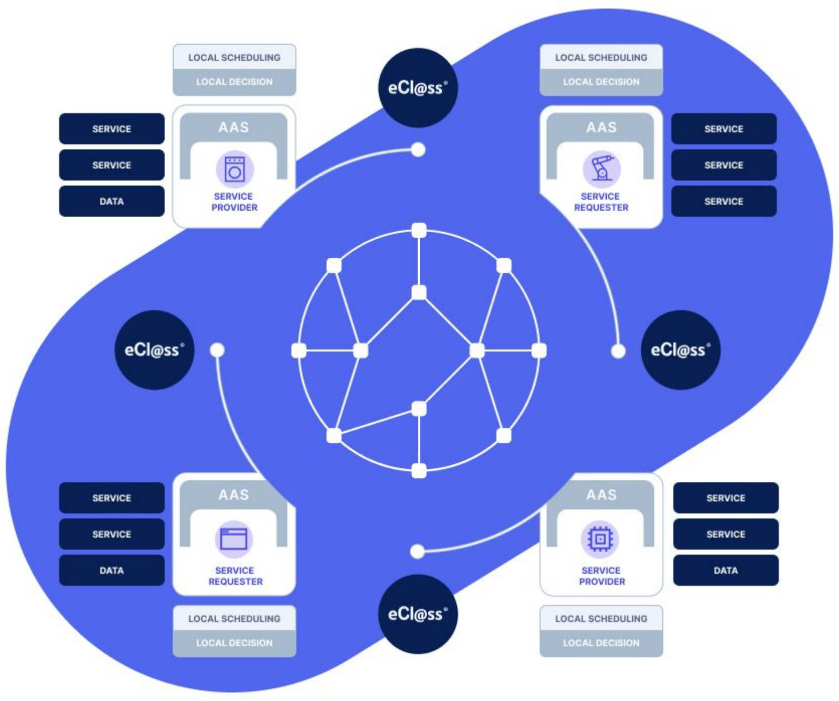

PLATTFORM INDUSTRIE4.0
I4.0-COMPONENT=ASSET + ASSET ADMINISTRATION SHELL
I4.0-COMPONENT
Asset Administration Shell (AAS)
Asset
I4.0-LANGUAGE
AAS
SERVICE REQUESTER
SERVICE PROVIDER
AAS
Data storage of all asset information
Standardized communication interface
Acts as translator between the "German", "French", "Italian" of the assets and the common language of the industry 4.0.

The Plattform Industrie 4.0 offers a concrete technical specification for I4.0. The Plattform Industrie 4.0 is the central network in Germany for advancing digital transformation in manufacturing. It was founded as a joint project of the German industrial associations BITKOM, VDMA, and ZVEI and parties across business, science, and research to further develop and implement the Federal Government's High-Tech Strategy.The focus of the Plattform Industrie 4.0's work lies in bundling I4.0 expertise in a way that makes it easily-accessible to companies.

One of Plattform Industrie 4.0's most important concepts for I4.0 is a so-called asset administration shell (AAS). An AAS is a form of digital twin that provides a standardized virtual representation of a tangible asset, such as a machine, equipment unit, or software. The basic idea is that an AAS contains an asset's description and provides a bridge between that asset and the Internet of Things (IoT), thereby making it available to interact in a machine-interpretable form. Together, an AAS and an asset form an I4.0 component, uniquely identifiable and able to communicate with other I4.0 components.

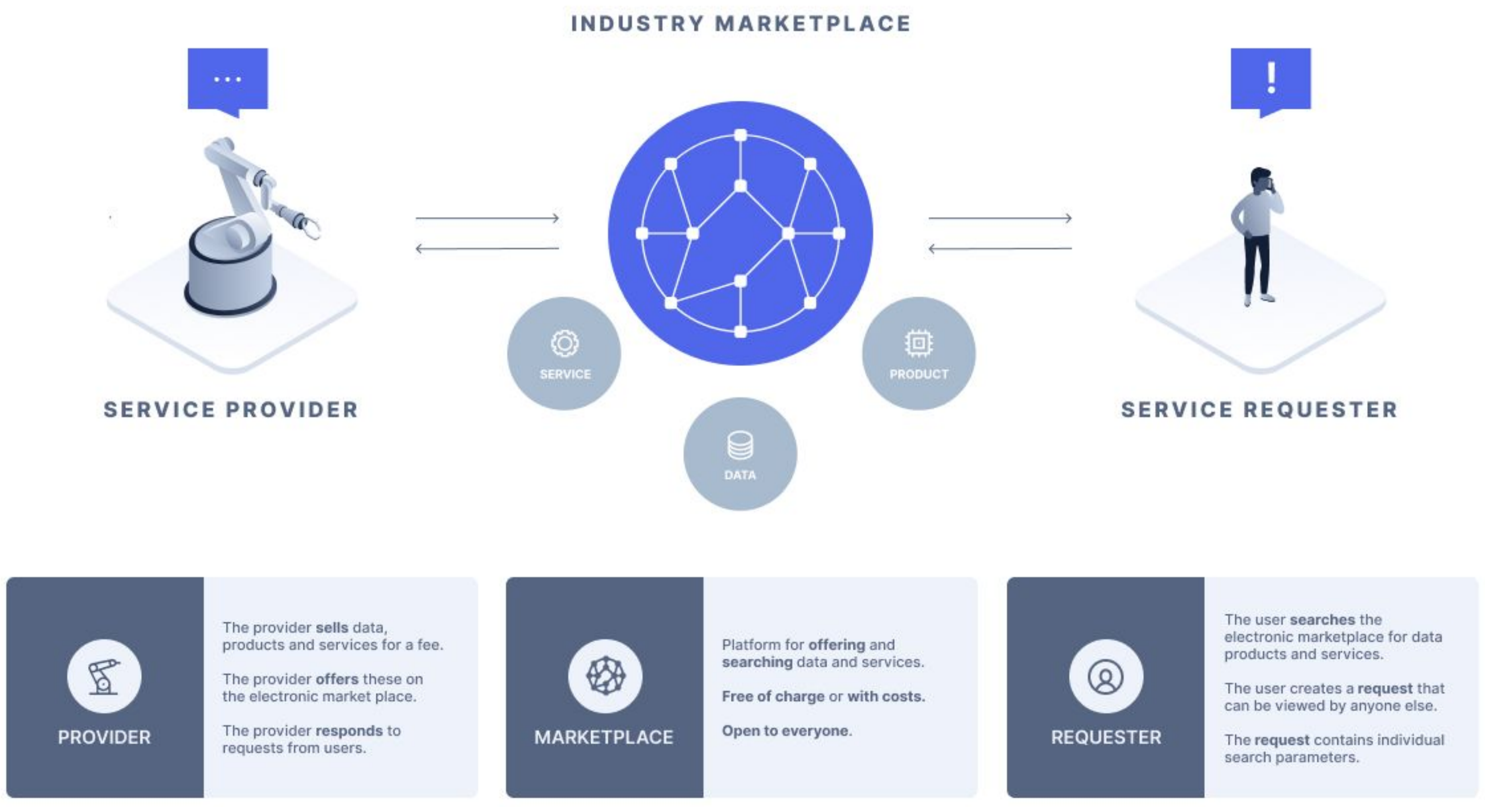
Using proactive decision and optimisation algorithms, I4.0 components can then engage in goal-oriented behaviours, as independent economic agents, together cooperating under market economy principles. Such interactions will lead to highly-flexible value creation networks, in which companies can cooperate in novel ways at both a national and global level.

At the core of I4.0, will be a global communication and computing infrastructure that facilitates economic relationships between machines. This is where IOTA comes in.



The Industry Marketplace

Intelligent production is a novel concept, where Industry 4.0 components negotiate with each other to fulfill tasks with predefined characteristics in a defined period of time, with a defined quality of service, and a defined cost range. In order for I4.0 components to cooperate in such a way, they must be able to speak a common language within a shared communication and exchange infrastructure.



The provider **sells** data, products and services for a fee.

The provider **offers** these on the electronic market place.

The provider **responds** to requests from users.

Platform for **offering** and **searching** data and services.

Free of charge or **with costs.**

Open to everyone.

The user **searches** the electronic marketplace for data products and services.

The user creates a **request** that can be viewed by anyone else.

The **request** contains individual search parameters.

The Industry Marketplace establishes a vendor-neutral marketplace for I4.0 components to buy and sell goods, data and services. Unlike conventional virtual marketplaces, the Industry Marketplace is an autonomous and decentralized platform for offering and searching for data and services, free of charge and open to everyone.

A participant in the Industry Marketplace can take one of two roles: the role of Service Requester or Service Provider.
The Service Requester searches for some data or services, e.g. charging, transporting, drilling etc., offered by a Service Provider. The Service Requester distributes a call for proposal in which they specify their requested service. The Service Providers can then generate proposals in response.

Both the call for proposal and the proposal itself contain a technical and commercial description of the service. This can include, for example, the price, time, quality and place for the provision of a service.

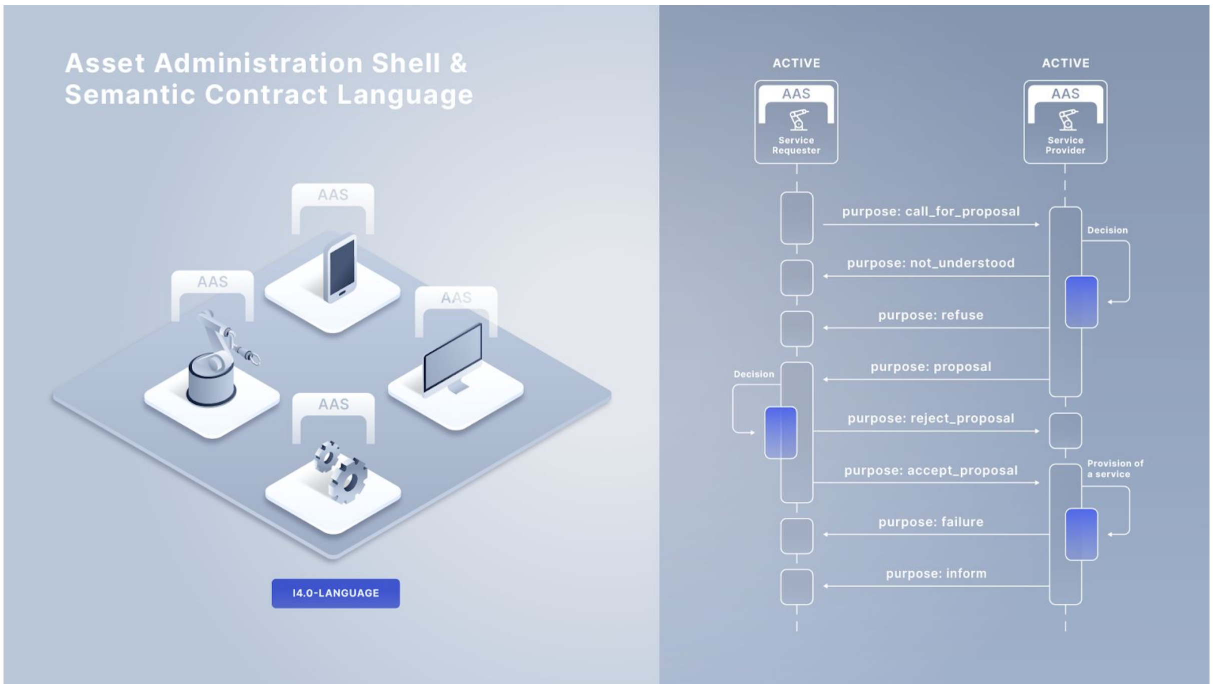
Payments in the Industry Marketplace utilize the integrated IOTA token. The token is a native part of the protocol, free to transfer without any fees. It can therefore be used in the purchase of larger goods or digital services, as well as in micropayments for small amounts of data (e.g. from weather stations).

The Market Manager

The Industry Marketplace results from the interactions between asset administration shells and their so-called Market Managers. The Market Manager plays the role of the middleman between AAS and the IOTA Tangle. Depending on the role of the AAS, the Market Manager will search for providers of required services or try to sell certain services.

The Market Manager manages IOTA transactions and performs the logical mapping between the Service Requester (SR) and Service Provider (SP). Its role includes:

- Translation of the I4.0 language messages from the SR and SP into IOTA transactions sent to a node.
- Translation of received IOTA transactions into I4.0 language messages and filtering those relevant for the connected SR and SP.
- Authentication of interacting parties by creating and verifying proof of decentralized authentication and identification (DID) ownership.



The figure above shows the service requester (SR) and service provider (SP) negotiating with each other through the IOTA Tangle. They are connected with an IOTA node. The description of services is serialized by JavaScript Object Notation (JSON). The logical mapping between the algorithms in the SR and SP and the IOTA transactions is deployed by the Market Manager. The Market Manager has the following tasks:

- translates the SR and SP message into the JSON transaction to be sent to the node
- translates the transaction messages into SR and SP messages
- creates a custom transaction tag for the related interaction pattern of the I4.0 system
- sends messages as transactions with custom tags to an IOTA node
- filters transactions from the IOTA Tangle that are relevant for the connected SR and SP
- controls the time span of one bidding process
- creates MAM channels to collect the outcomes of each bidding process for audit purposes

The interactions between AAS need an infrastructure that provides all means for registry, discovery, communication of messages, and trust. Each AAS is therefore connected to an IOTA node. Each message forms a transaction on the Tangle. The transaction's tag and data fields are specific to the related interaction pattern of the I4.0 system. The eCl@ss properties are part of the transaction data.

Messages of the Industrie 4.0 language supported by the Industry Marketplace

The technical and commercial description uses the vocabulary of I4.0 provided by **eCl@ss**. In this way, the machines can automatically interpret the distributed call for proposals with no need for human interaction.

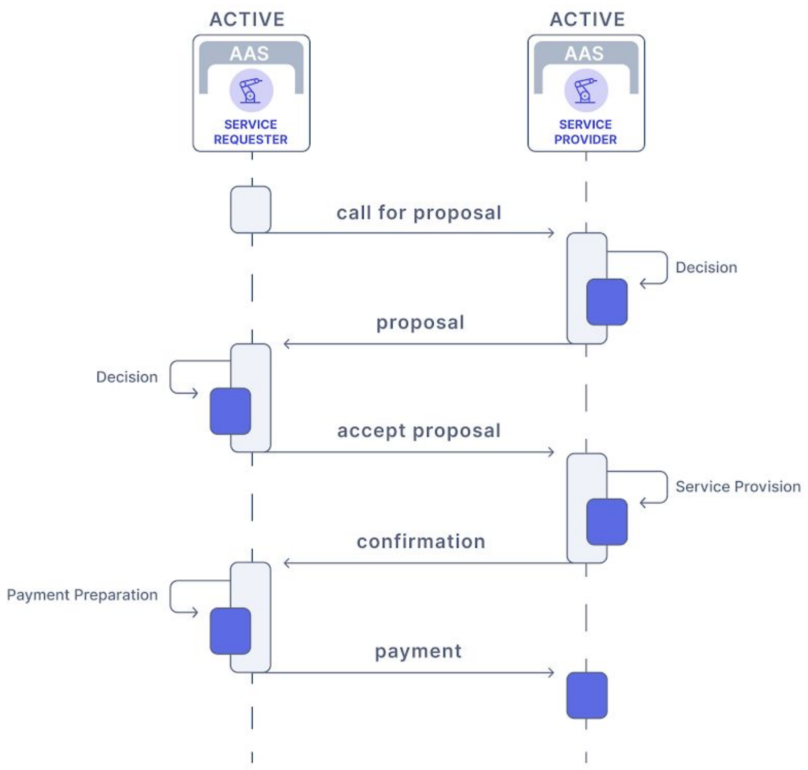
Supported Messages

The current version of Market Manager supports the following message types:

- callForProposal
- proposal
- acceptProposal
- rejectProposal
- informConfirm
- informPayment - new message type added by IOTA

Each message follows the specifications of the Industry 4.0 Language (VDI/VDE - 2193 - Part1 and Part 2) and the Specification "Details of the Asset Administration Shell Part 1 - The exchange of information between partners in the value chain of Industrie 4.0"

The sequence of the messages is the following:



Message Format

callForProposal

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "callForProposal",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "1",
    "sender": {
      "identification": {
        "id":
"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBBDRNJC9GCZGAUSNIRVT
PAYSRMQQA"
      }
    },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:45 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

proposal

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "proposal",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "2",
    "sender": {
      "identification": {
        "id":
"did:iota:YKQCDNLHAKDNSHQVYHVWCEOSSQZJSJTSLJJEVGSXDBIZVFCQPJUWHBEZWJEDMCHESWWBYSEZ
9C9SRQBOQ"
      }
    },
    "receiver":{
        "identification":{
             "id":"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBB
      DRNJC9GCZGAUSNIRVTPAYSRMQQA"
        }
     },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:46 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

acceptProposal

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "acceptProposal",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "3",
    "sender": {
      "identification": {
        "id":
"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBBDRNJC9GCZGAUSNIRVT
PAYSRMQQA"
      }
    },
          "receiver": {
      "identification": {
        "id":
"did:iota:YKQCDNLHAKDNSHQVYHVWCEOSSQZJSJTSLJJEVGSXDBIZVFCQPJUWHBEZWJEDMCHESWWBYSEZ
9C9SRQBOQ"
      }
    },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:48 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

rejectProposal

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "rejectProposal",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "3",
    "sender": {
      "identification": {
        "id":
"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBBDRNJC9GCZGAUSNIRVT
PAYSRMQQA"
      }
    },
         "receiver": {
      "identification": {
        "id":
"did:iota:YKQCDNLHAKDNSHQVYHVWCEOSSQZJSJTSLJJEVGSXDBIZVFCQPJUWHBEZWJEDMCHESWWBYSEZ
9C9SRQBOQ"
      }
    },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:48 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

informConfirm

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "informConfirm",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "4",
    "sender": {
      "identification": {
        "id":
"did:iota:YKQCDNLHAKDNSHQVYHVWCEOSSQZJSJTSLJJEVGSXDBIZVFCQPJUWHBEZWJEDMCHESWWBYSEZ
9C9SRQBOQ"
      }
    },
         "receiver": {
      "identification": {
        "id":
"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBBDRNJC9GCZGAUSNIRVT
PAYSRMQQA"
      }
    },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:55 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

informPayment

Please complete with submodelElements

```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "informPayment",
    "conversationId": "68175af8-8826-49f6-8953-5749fc0a45bd",
    "messageId": "5",
    "sender": {
      "identification": {
        "id":
"did:iota:UYARPMVEZTA9CZHITEWRWYZDFLWG9LCUH9CRHVLHRGGTQCJOYDJFBBDRNJC9GCZGAUSNIRVT
PAYSRMQQA"
```

```
      }
    },
         "receiver": {
      "identification": {

"id":"did:iota:YKQCDNLHAKDNSHQVYHVWCEOSSQZJSJTSLJJEVGSXDBIZVFCQPJUWHBEZWJEDMCHESWW
BYSEZ9C9SRQBOQ"
      }
    },
    "replyBy": 1704063600000,
    "location": "52.508,13.37789999999",
    "startTimestamp": 1564476317000,
    "endTimestamp": 1564562717000,
    "creationDate": "29 July, 2019 10:58 am "
  },
  "dataElements": {
    "submodels": [
      {
        "identification": {
          "id": "0173-1#02-BAF577#004",
          "submodelElements": ["INSERT SUBMODELELEMENTS HERE"]
        }
      }
    ]
  }
}
```

submodelElements

Set of the submodel elements (properties) is specific for each submodel. Submodels are standardised information models describing the different aspects of Assets in a form that can be leased and interpreted by machines.
Below is an exemplary description of a washing machine's capability to wash. Please find the additional examples in the document Asset Administration Shell in the praxis

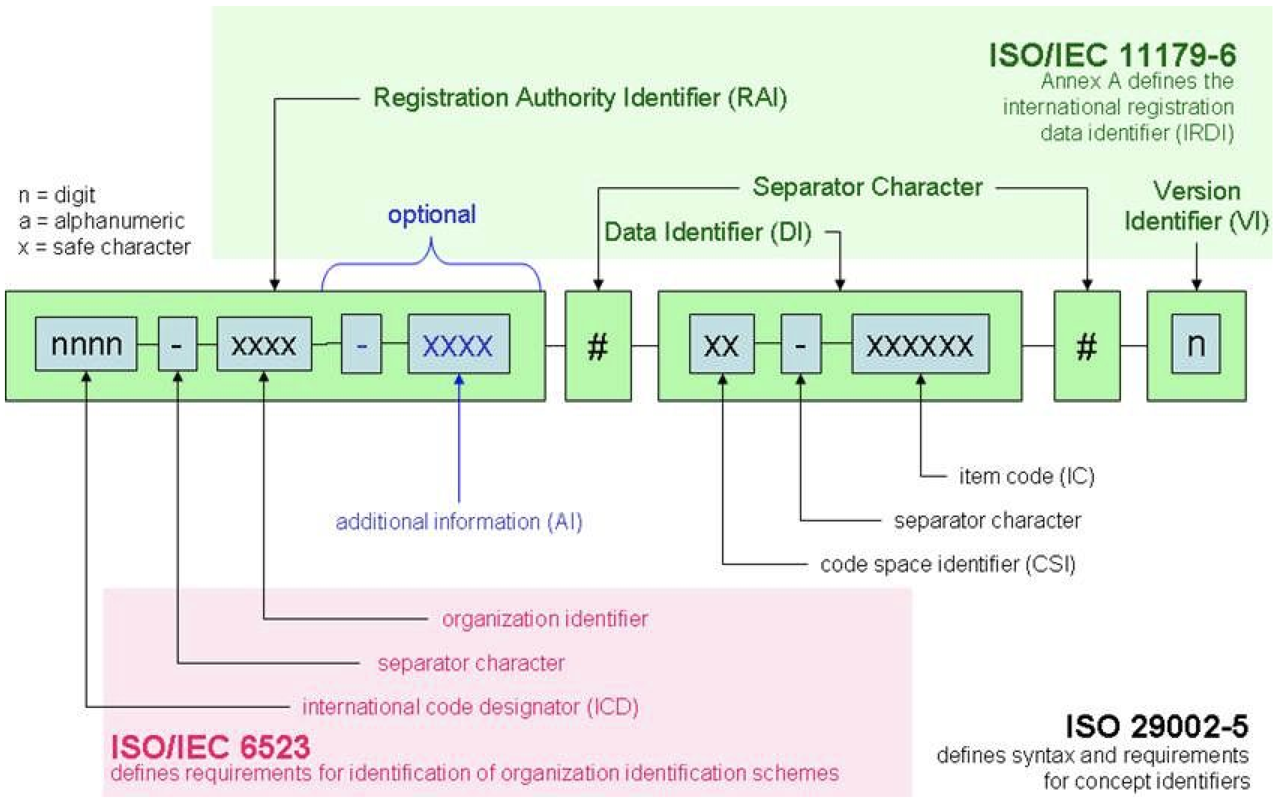
```
{
  "idShort": "gewicht",
  "modelType": "Property",
  "value": "5",
  "valueType": "string",
  "semanticId": "0173-1#02-AAB713#005"
},
{
  "idShort": "farbe",
  "modelType": "Property",
  "value": "schwarz",
  "valueType": "string",
  "semanticId": "0173-1#02-AAN521#005"
},
{
  "idShort": "material",
  "modelType": "Property",
  "value": "stahl",
  "valueType": "string",
  "semanticId": "0173-1#02-BAF634#008"
},
{
  "idShort": "ort",
  "modelType": "Property",
  "value": "berlin",
  "valueType": "string",
  "semanticId": "0173-1#02-BAF163#002"
},
{
  "idShort": "zeit",
  "modelType": "Property",
  "value": "1558461600",
  "valueType": "string",
  "semanticId": "0173-1#02-AAO738#001"
},
{
  "idShort": "preis",
  "modelType": "Property",
  "value": "5",
  "valueType": "string",
  "semanticId": "0173-1#02-AAO738#001"
}
```

International Registration Data Identifier (IRDI)

IRDI - International Registration Data Identifier, ISO29002-5, ISO IEC 6523 and ISO IEC 11179-6 [20] as an Identifier scheme for properties and classifications. They are created in a process of consortium-wise specification or international standardisation. To this end, users sit down together and feed their ideas into the consortia or standardisation bodies. Properties in ISO, IEC help to safeguard key commercial interests. Repositories like eCl@ss and others make it possible to standardise a relatively large number of Identifiers in an appropriately short time.

IRDIs are assumed to be already existing by an external specification and standardisation process, when it comes to the creation of a certain Administration Shell.

IRDI Structure


ISO/IEC 11179-6
Annex A defines the international registration data identifier (IRDI)
Registration Authority Identifier (RAI)
n = digit
a = alphanumeric
x = safe character
optional
Separator Character
Data Identifier (DI)
Version Identifier (VI)
nnnn
-
xxxx
-
xxxx
#
xx
-
xxxxxx
#
n
item code (IC)
separator character
code space identifier (CSI)
additional information (AI)
organization identifier
separator character
international code designator (ICD)
ISO/IEC 6523
defines requirements for identification of organization identification schemes
ISO 29002-5
defines syntax and requirements for concept identifiers

Unified example

The following example is used to demonstrate the main features of the data formats as explained in the following clauses for different data formats. Intention is to motivate the equivalency of information in different representations.
It shows an AAS with two properties: the actual rotation speed (idShort = “rotationSpeed”), a measurement value (category=VARIABLE) as well as the maximum rotation speed “NMax” (category=PARAMETER). The AAS represents a controller with short id “3S7PLFDRS35”.
Up to know there is no property defined within eCl@ss for the actual rotation speed. Therefore a corresponding concept description (with idShort=”N”) is added to the local dictionary of the AAS. It gets the global identifier “id=www.festo.com/dic/08111234” that is referenced via semanticId in the property “rotationSpeed”.
For the maximum rotation speed eCl@ss provides a semantic definition with global identifier “0173-1#02-BAA120#007”. A copy of the entry is created within the local dictionary. The id of the copy is the same as in eCl@ss.
The physical unit of the rotation speed properties and concept description is 1/min, denoted by a globally unique IRDI “0173-1#05-AAA650#002” for 1/min as defined by eCl@ss.

Demo Catalog

As long as the original eCl@ss catalog is not accessible to fetch by the API, we maintain a demo catalog with a limited number of pre-configured operations.

Our demo catalog includes operations from transportation and mobility sector, unmanned vehicle services, electrical charging, digital equipment, cartography and navigation, telecommunication services, purchase of goods.

AssetAdministration Shell (german: Verwaltungsschale)

You can find a very detailed specification of the AssetAdministration Shell here.

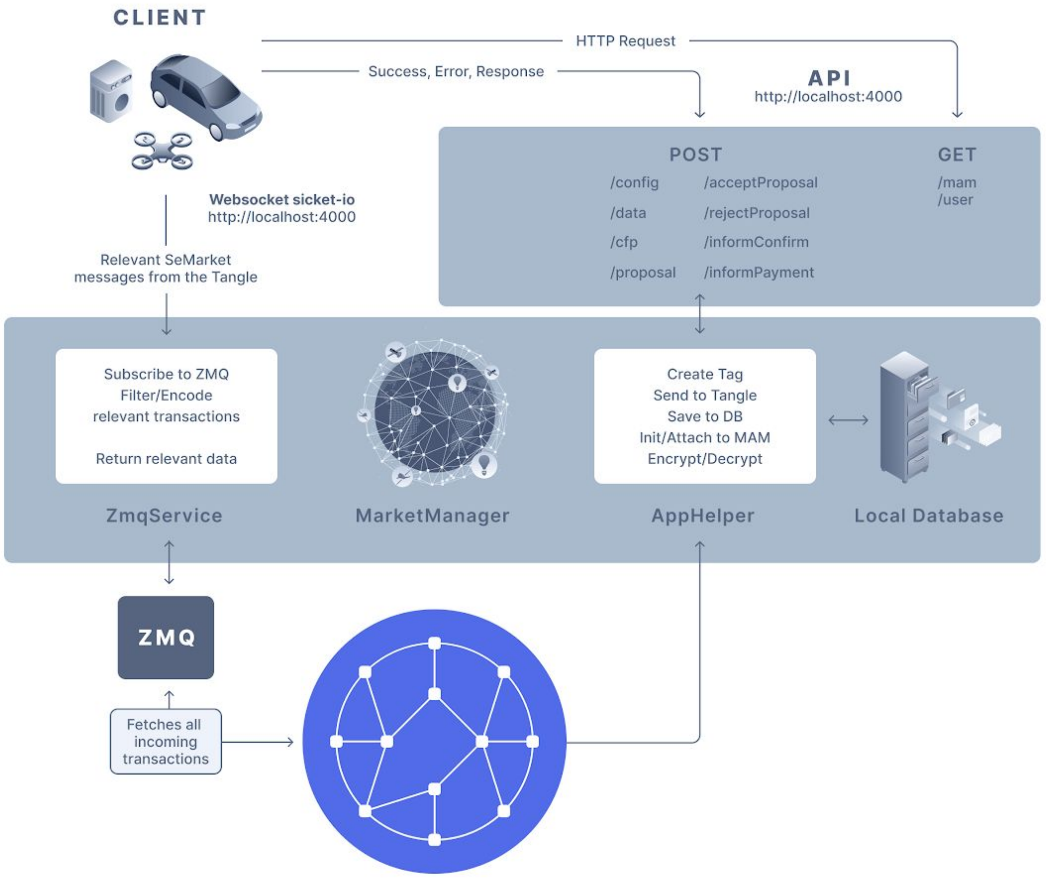
Architecture

Running an open source project, like any human endeavor, involves uncertainty and trade-offs. We hope the architecture described below helps you to deploy similar systems, but it may include mistakes, and can't address every situation. If you have any questions about how to use this architecture for your project, we encourage you to do your own research, seek out experts, and discuss with IOTA community.

The application consists of the backend NodeJS script and local SQLite3 database. The application can be configured and managed using built-in scripts and API endpoints. There is an optional UI application included to simplify operations and enhance customer experience.

The entry point to the Industry Marketplace is the MarketManager. This is a NodeJS application that can run locally or be deployed to a cloud instance. To deploy and run it as a cloud instance please use the provided Dockerfile from the root folder.

One of the major tasks of the MarketManager is to transmit relevant messages from the Tangle to the receiver. Therefore, the MarketManager needs to build up a persistent connection to the Zero Message Queue, which fetches all incoming transactions from the Tangle. Since the REST API is in first place not suitable for a persistent connection, websockets are used to tackle this task. The MarketManager application receives all incoming transactions from the ZMQ, it filters only relevant ones for the client by matching its configuration with the content of the messages provided with the Semantic I4.0 Language.

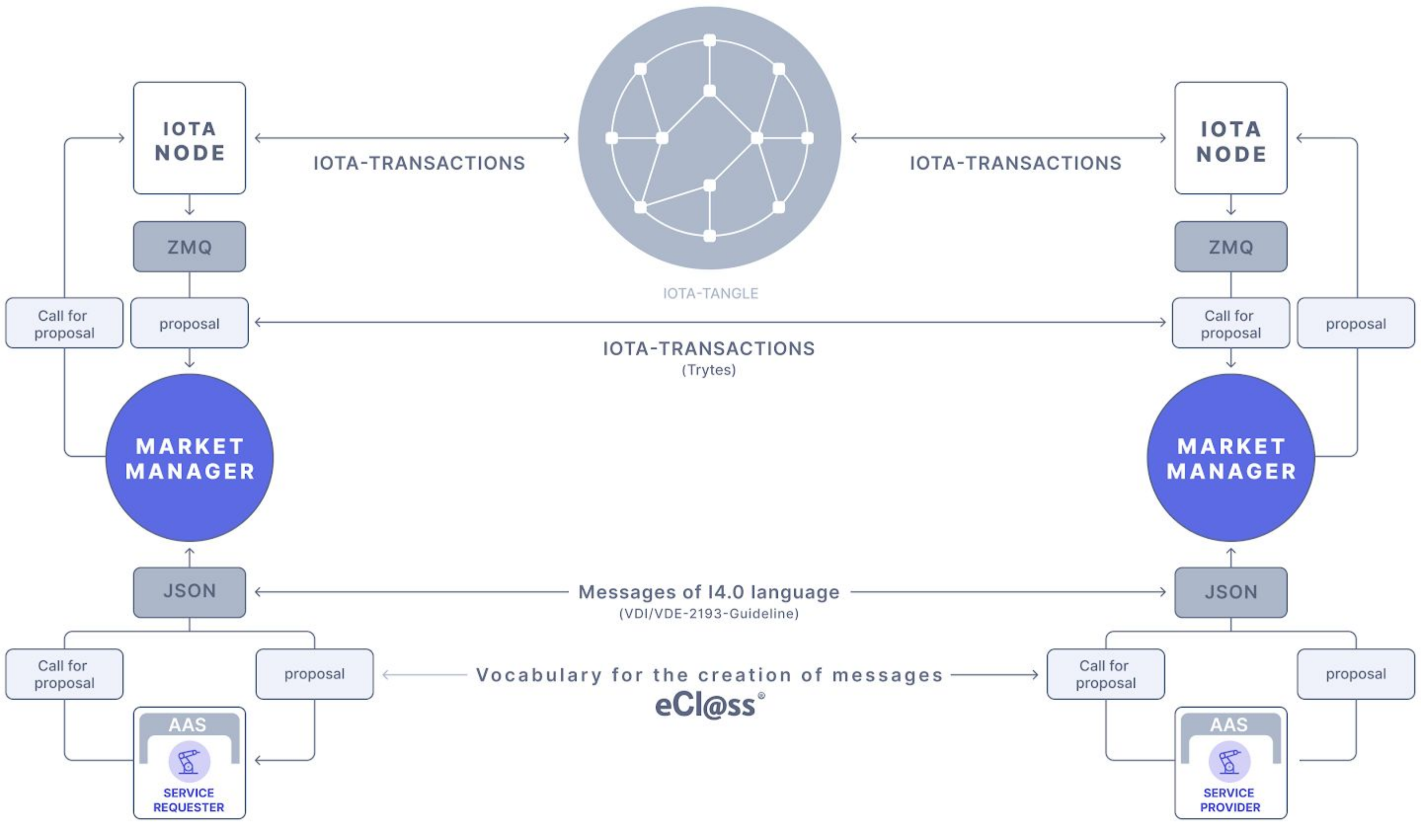


AppHelper script contains a collection of the API endpoints as well as the headers, CORS and port configuration for the WebServer. By default it is configured to allow all incoming connections and runs on port 4000 for a local usage.

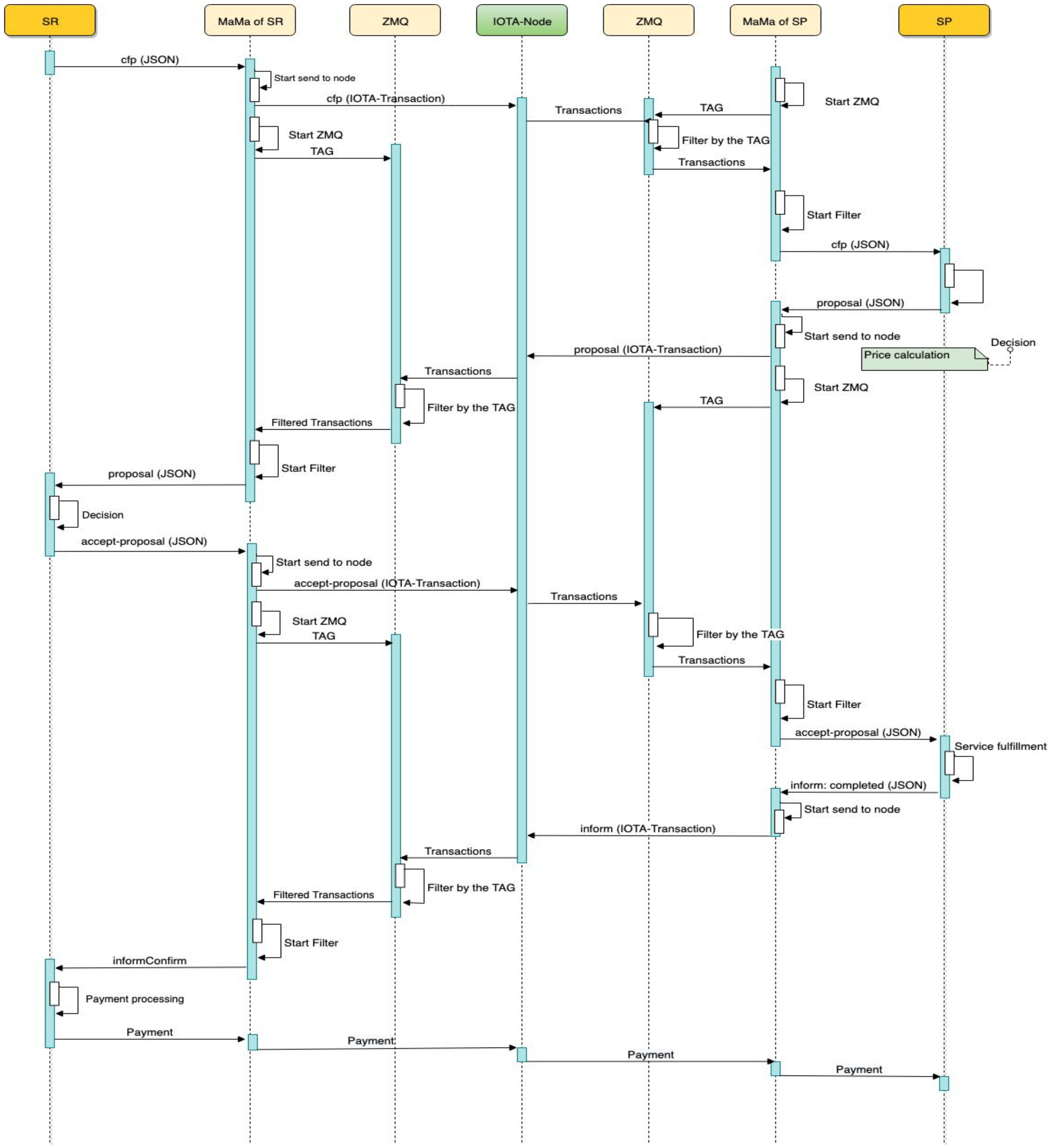
Local SQLite3 database holds user data, access credentials for MAM channels and wallet data.

When MarketManager is running, an internal payment queue script is collecting information about outstanding payments. In specified time intervals the payment queue attempts to transfer tokens to all recipients in one payment transaction. If the internal wallet is empty or in a pending state, new wallet is initialized and funded using the free Devnet IOTA tokens. Payment operation continues in the next iteration.

An essential dependency of the MarketManager is the Industry 4.0 Language library, developed by IOTA and provided as NPM package. This library assembles the semantically correct message of the specified type according to the Semantic I4.0 Language specification from the parameters provided within POST request body of the API calls.



Sequence Diagrams (next page)


SR
MaMa of SR
ZMQ
IOTA-Node
ZMQ
MaMa of SP
SP
cfp (JSON)
Start send to node
cfp (IOTA-Transaction)
Transactions
TAG
Start ZMQ
Start ZMQ
TAG
Filter by the TAG
Transactions
Start Filter
cfp (JSON)
proposal (JSON)
Start send to node
Decision
Price calculation
proposal (IOTA-Transaction)
Transactions
Start ZMQ
Filter by the TAG
TAG
Filtered Transactions
Start Filter
proposal (JSON)
Decision
accept-proposal (JSON)
Start send to node
accept-proposal (IOTA-Transaction)
Transactions
Start ZMQ
TAG
Filter by the TAG
Transactions
Start Filter
accept-proposal (JSON)
Service fulfillment
inform: completed (JSON)
Start send to node
inform (IOTA-Transaction)
Transactions
Filter by the TAG
Filtered Transactions
Start Filter
informConfirm
Payment processing
Payment
Payment
Payment
Payment

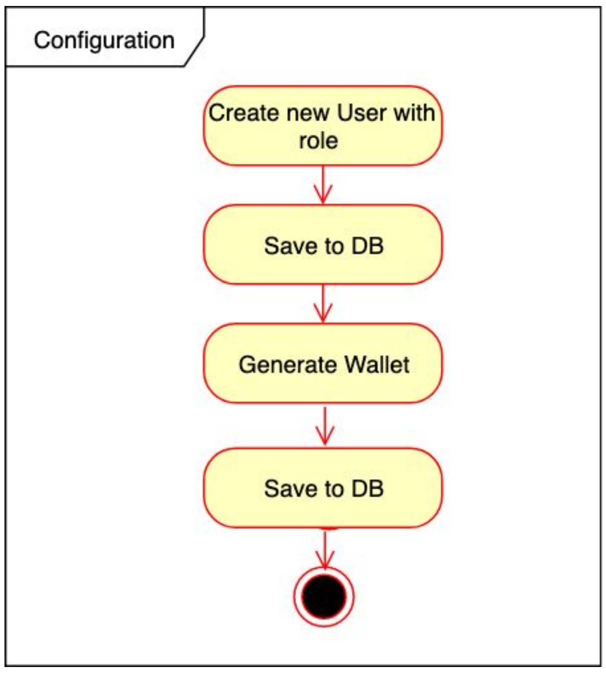
Workflows

The logical mapping between the algorithms in the SR and SP and the IOTA transactions is deployed by the Market Manager. The Market Manager has the following tasks:

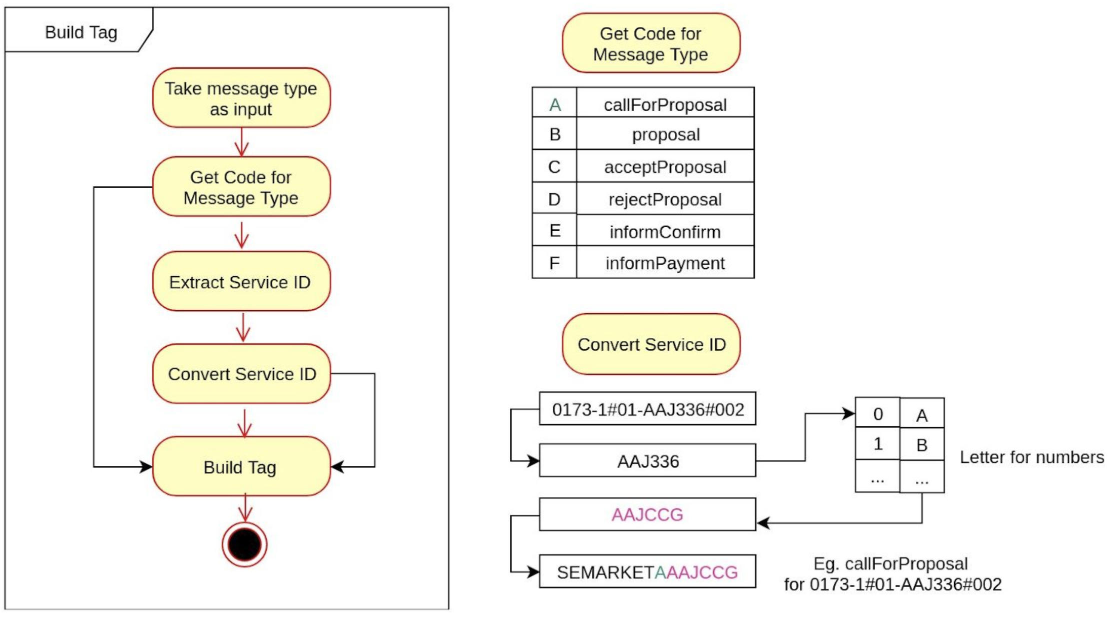
- translates the SR and SP message into the JSON transaction to be sent to the node
- translates the transaction messages into SR and SP messages
- creates a custom transaction tag for the related interaction pattern of the I4.0 system
- sends messages as transactions with custom tags to an IOTA node
- filters transactions from the IOTA Tangle that are relevant for the connected SR and SP
- controls the time span of one bidding process
- creates MAM channels to collect the outcomes of each bidding process for audit purposes

The activity diagrams below describe the functionality and the internal logic of the Market Manager.

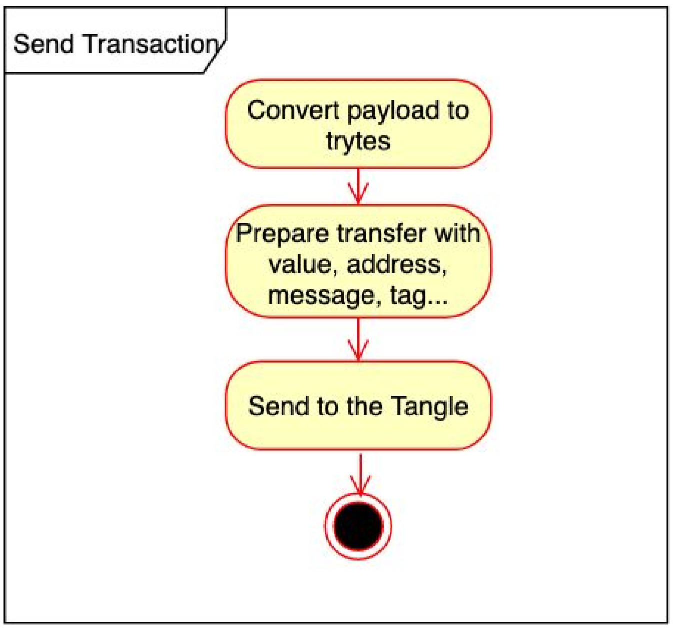
Configure Entity



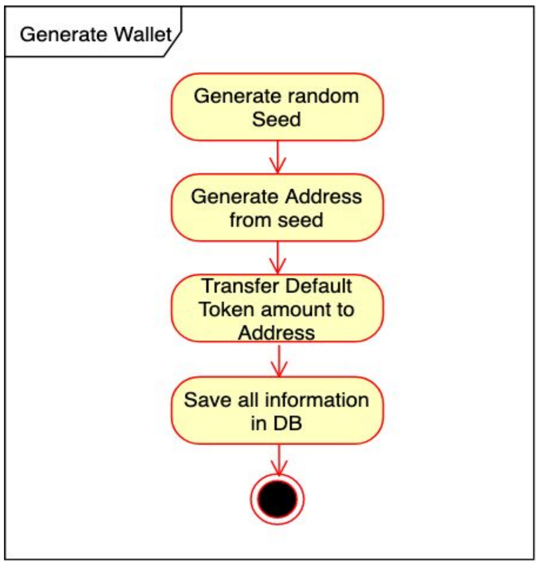
Build Tag



Send Transaction

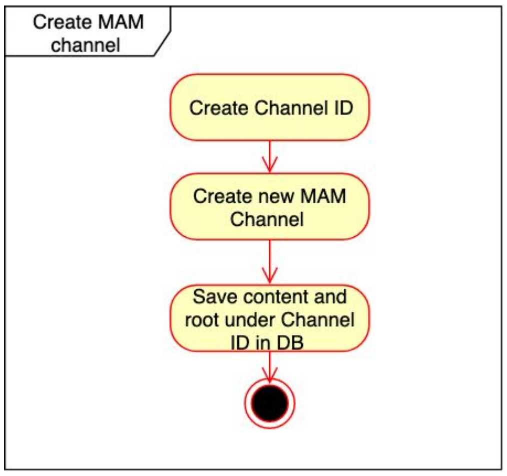

Send Transaction
Convert payload to trytes
Prepare transfer with value, address, message, tag...
Send to the Tangle

Generate and Fund Wallet

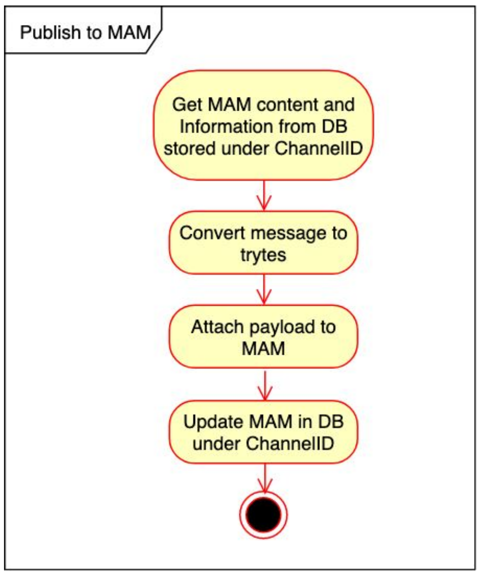


Generate Wallet
Generate random Seed
Generate Address from seed
Transfer Default Token amount to Address
Save all information in DB

Create MAM

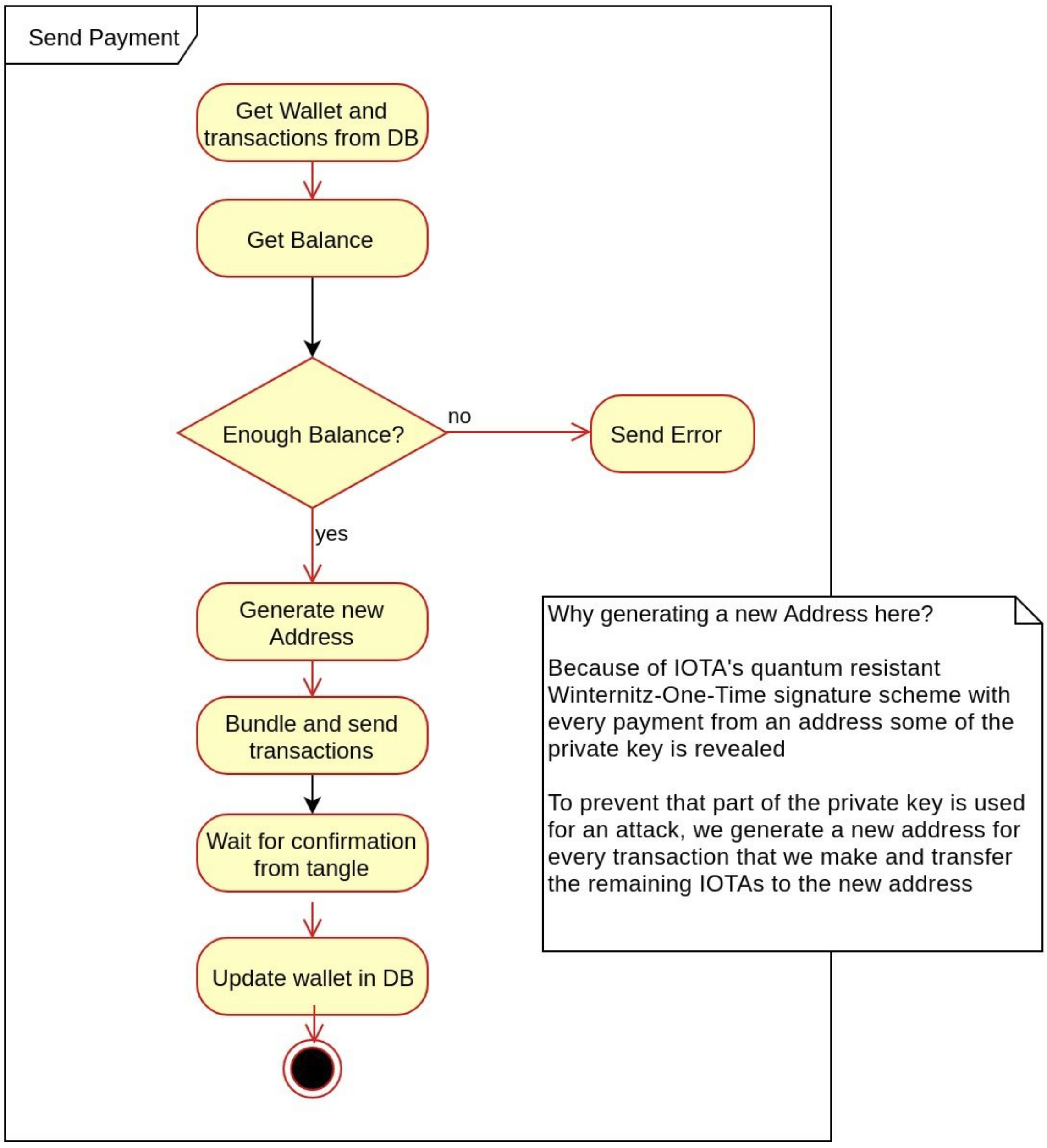


Publish MAM

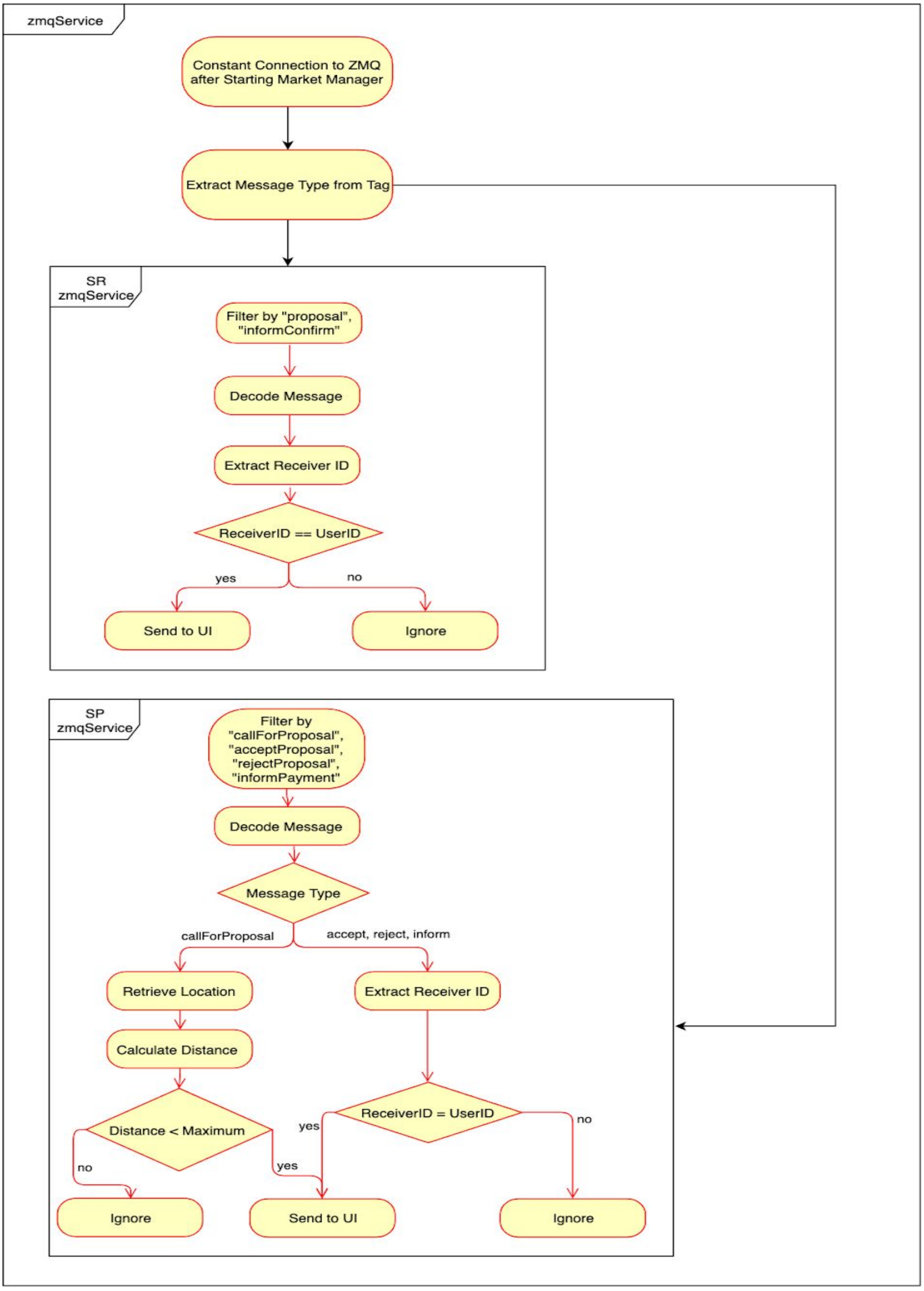


Publish to MAM
Get MAM content and Information from DB stored under ChannelID
Convert message to trytes
Attach payload to MAM
Update MAM in DB under ChannelID

Send Payment

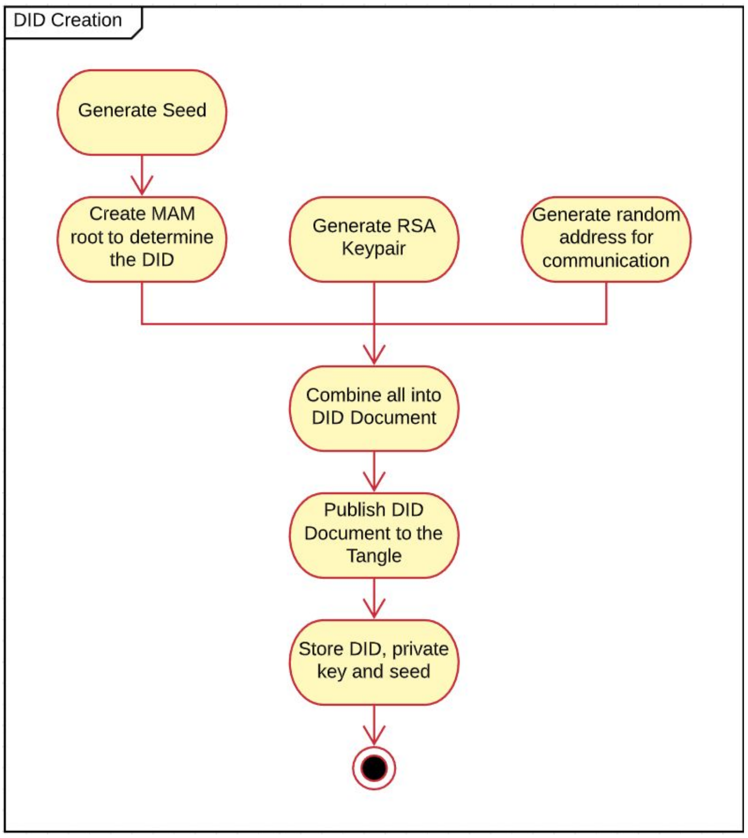

Send Payment
Get Wallet and transactions from DB
Get Balance
Enough Balance?
no
Send Error
yes
Generate new Address
Bundle and send transactions
Wait for confirmation from tangle
Update wallet in DB
Why generating a new Address here?
Because of IOTA's quantum resistant Winternitz-One-Time signature scheme with every payment from an address some of the private key is revealed
To prevent that part of the private key is used for an attack, we generate a new address for every transaction that we make and transfer the remaining IOTAs to the new address

ZQM Service

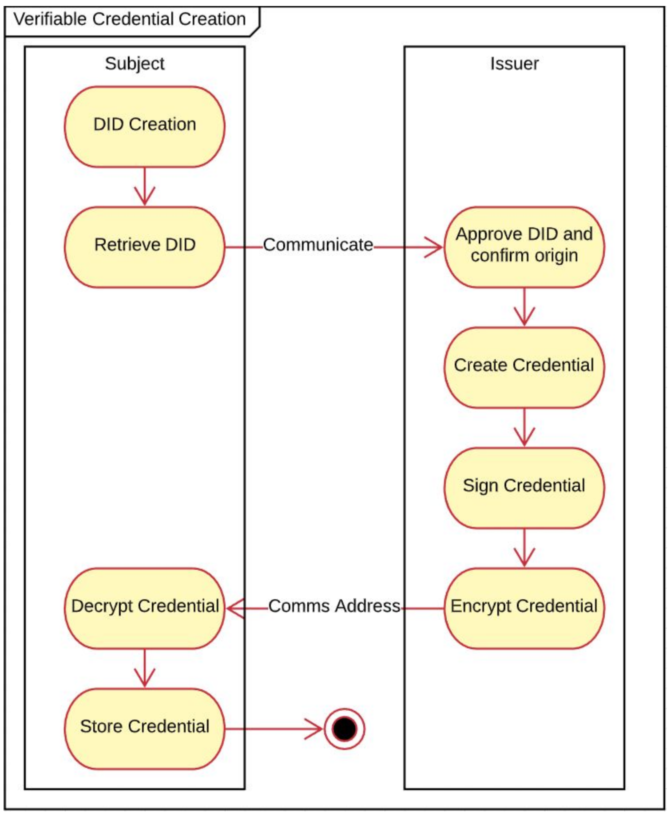


zmqService
Constant Connection to ZMQ after Starting Market Manager
Extract Message Type from Tag
SR zmqService
Filter by "proposal", "informConfirm"
Decode Message
Extract Receiver ID
ReceiverID == UserID
yes
no
Send to UI
Ignore
SP zmqService
Filter by "callForProposal", "acceptProposal", "rejectProposal", "informPayment"
Decode Message
Message Type
callForProposal
accept, reject, inform
Retrieve Location
Extract Receiver ID
Calculate Distance
Distance < Maximum
ReceiverID = UserID
yes
no
no
yes
Ignore
Send to UI
Ignore

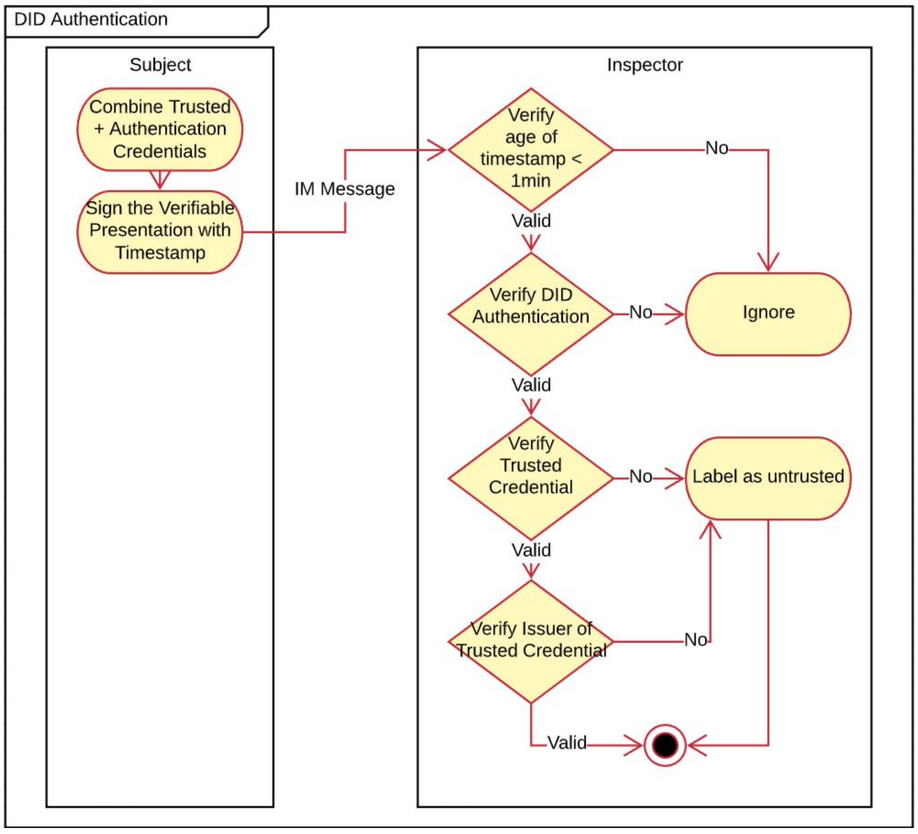
DID Creation


DID Creation
Generate Seed
Create MAM root to determine the DID
Generate RSA Keypair
Generate random address for communication
Combine all into DID Document
Publish DID Document to the Tangle
Store DID, private key and seed

Verifiable Credential Creation



DID Authentication


DID Authentication
Subject
Combine Trusted + Authentication Credentials
Sign the Verifiable Presentation with Timestamp
IM Message
Inspector
Verify age of timestamp < 1min
No
Valid
Verify DID Authentication
No
Ignore
Valid
Verify Trusted Credential
No
Label as untrusted
Valid
Verify Issuer of Trusted Credential
No
Valid

Industry Marketplace Entity

To use the Industry Marketplace as either Service Requester or Service Provider, an entity should be created and configured.
Configuration includes specifying of the role, organisation or customer display name, location (address or GPS coordinates). In addition an internal IOTA Wallet can be created and funded within the same configuration process.

Entity ID is generated automatically and includes a fixed prefix “did:iota:” followed by the root address of a public MAM channel which holds public key and additional information related to digital identity.
Entity ID is used to populate the “`frame : sender : identification : id`” field of the standard Industry 4.0 Semantic

Entity API

Entity information can be set and modified using the POST API endpoint `/config`
Please see the API endpoint description section for details.

Configuration Dialog

Entity information can be set and modified using the “Modify Configuration” dialog available on the UI.

Entity Data

Entity data can be retrieved from the internal local database using the GET API endpoint `/user`. Please see the API endpoint description section for details.
On the UI this information is fetched in regular time intervals to ensure the wallet balance display is up to date.

Transaction Audit

Once the transaction with a valid eCl@ss message is sent to the Tangle, its content also securely stored for audit reasons on the Tangle. The underlying technology to store the content of the message is called Masked Authenticated Messaging (MAM).

The backend MarketManager script only need to conduct a small amount of “Proof of Work”* to allow the data to propagate through the network.

*** Proof of Work (PoW)** - An algorithm which prevents Denial of Service and spam attacks on a network. a computationally hard puzzle to solve, but easy to verify. IOTA uses a Hashcash based puzzle.

Since these messages are part of the distributed ledger, they both contribute to the security of the network by increasing total hashing power and benefit from the data integrity properties of the network as other transactions continue to indirectly reference them.

What is MAM

Masked Authenticated Messaging (MAM) is a second layer data communication protocol which adds functionality to emit and access encrypted data stream, like RSS, over the Tangle (IOTA's distributed ledger) regardless of the size or cost of the device. IOTA's consensus protocol adds integrity to these message streams. Given these properties, MAM fulfills an important need in industries where integrity and privacy are required.

MAM uses Encrypted Messaging Streams, to emit and access a forward-secret and encrypted data stream over the Tangle.

Masked Authenticated Messaging allow for public or encrypted messaging streams to be published to the Tangle.
These streams can be read back by anyone who has access to the Ledger, and anyone who has the encryption key. These message streams are signed so you are also able to attest to the authenticity of the data.
Current MAM is written in RUST with bindings for Javascript and Java.

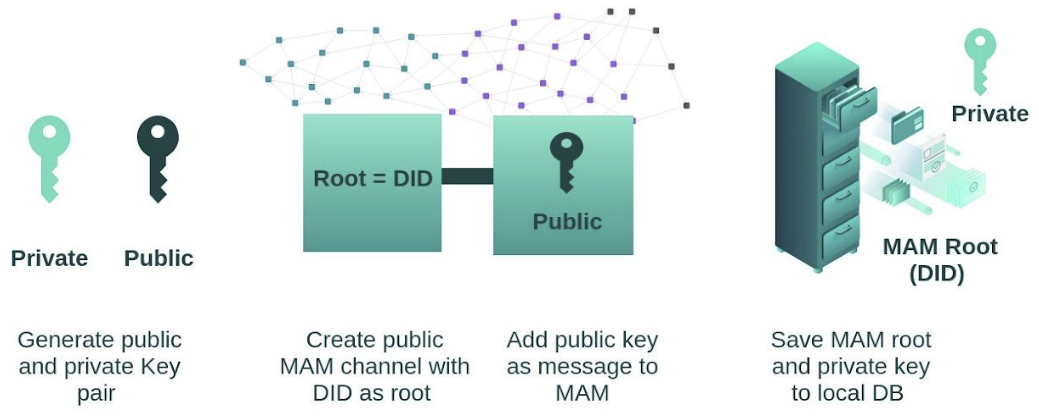
Read more about the MAM here.

How MAM is used to store audit logs

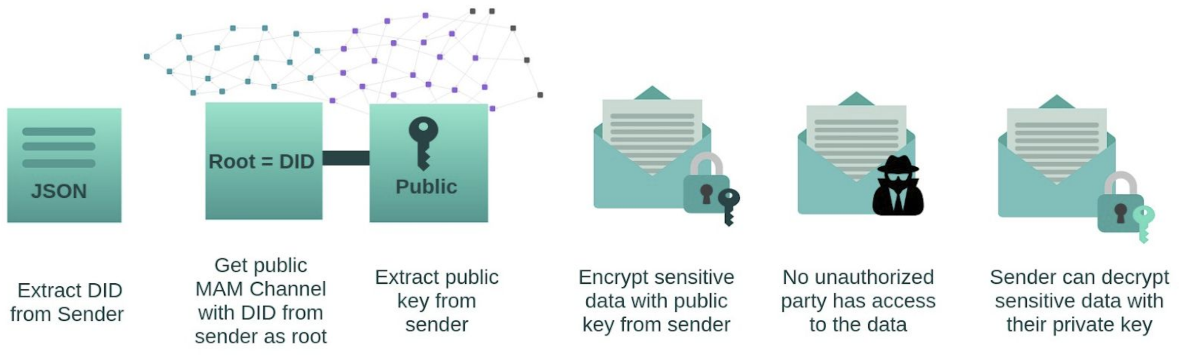
In the Industry Marketplace implementation, every message in communication between Service Requester and selected Service Provider is encrypted with a randomly generated encryption key and stored on the Tangle inside an MAM-channel, that is created for every new request issued by a Service Requester. After a Service Provider is selected, its public key is used to encrypt and securely send the key for the MAM channel of the request. This ensures that only Service Requester and selected Service Provider are in possession of a key to decrypt and read the content of the MAM channel and therefore retrieve transaction data for audit.

Digital Identity and Encryption

Within the Industry Marketplace, every entity generates a public/private key pair. Public MAM channel is created and its root address becomes part of the user ID, which at the same time serves as Digital Identified (DID) with the purpose to publish the public key of the entity. Private key along with the channels seed is locally stored in the local database of the entity.


Private
Public
Root = DID
Public
Private
MAM Root (DID)
Generate public and private Key pair
Create public MAM channel with DID as root
Add public key as message to MAM
Save MAM root and private key to local DB

The public key of an entity can be used by others to encrypt sensitive data, that should only be accessible only by the entity. This is enabled via asynchronous encryption, where messages that are encrypted with a public key can only be decrypted with the matching private key.


JSON
Root = DID
Public
Extract DID from Sender
Get public MAM Channel with DID from sender as root
Extract public key from sender
Encrypt sensitive data with public key from sender
No unauthorized party has access to the data
Sender can decrypt sensitive data with their private key

In the Industry Marketplace the transaction information used for audit is stored in a restricted MAM channel, and the key to access the channel is encrypted using the public key of the recipient, so only both communication partners can access the information stored in the channel.

Authentication

The digital identities are also used to establish trust between the communication partners. This is done on two levels, proof that the communication partner owns the DID and the partner can, optionally, proof that it is a trusted participant. This authentication process happens during the first messages of interaction. In order for one partner to listen to another, it authenticates the messages.

The DID ownership authentication is done by proving a digitally signed copy of the DID document. This DID document is signed with a timestamp. The receiving party will not accept the authentication if the timestamp is older than 1 minute. If the authentication fails, the message will be ignored.

The authentication of trust between the two parties is based on a Verifiable Credential, given out by a trusted issuer for the receiving party. If the receiving party doesn’t trust the issuer, the credential is deemed worthless. If the party is recognized, the credential is verified and the messages will be labelled as trusted.

In order the acquire this credential, the party needs to contact the issuer and ask for the credential. The issuer will need to know the DID of the party. If they decide to give out the credential, the issuer will sign the credential and send it encrypted over the Tangle to the requesting party’s Tangle communication address. This is listed as a Service Endpoint (DID standard) in the DID Document of the requesting party.

Payment

In the Industry Marketplace ecosystem each entity (user/organisation) needs to have an IOTA Wallet funded with IOTA Tokens.
Existence of the wallet and positive balance sufficient to cover the cost of the service or data are prerequisites to accept proposals received by Service Providers (SP).

When Service Requester (SR) sends a request, multiple Service Providers (SP) can react on it by sending their proposals along with the **price for their services**. The price becomes a part of the submodel values, and therefore included into the “proposal” message.

Service Requester (SR) can then review received proposals and accept one of them. By doing so, Service Requester accepts the price of the service. An “acceptProposal” message is sent to the SP.
Acceptance of the proposal creates a contract between SR and SP, which is now signed by one part. SR has to pay for services or data provided by the selected SP once the request is fulfilled.
The fulfillment of the request is indicated by sending the “informConfirm” message from the SP to SR. SP also adds its wallet address to the message, to indicate where IOTA Tokens should be transferred.

As a reaction to this message, SR has to transfer the amount of IOTA Tokens that was agreed during the proposal selection step, to the known wallet address of the Service Provider. Payment transfer is confirmed by the “informPayment” message.

Wallet, Faucet

For the sake of demonstration we will be distributing small portions of the free IOTA Devnet Tokens for every new entity which joins the initiative in 2019. We are using a tool called Faucet, which transfers a specified amount of IOTA Devnet Tokens to the internal entity’s wallet address when the wallet is created for the first time.

Entity can choose to re-generate the internal wallet. In this case, the balance of the previous wallet will be lost, as the seed will be replaced. During re-generation the Faucet tool will become active again, providing the same initial small amount of Devnet Tokens.

On the later stage, when the Industry Marketplace will run on the Mainnet, we will eliminate the Faucet and provide User Interface and API endpoint specification to transfer IOTA Tokens between the internal wallet of the entity and the Trinity Wallet Application that holds the main wallet.

Seed storage

When the internal IOTA wallet is generated for the first time, its **seed** is stored in a local database.
Please note that the wallet seed should be stored securely at all times and **should not** be shared, distributed or made accessible by any unauthorized person.

Wallet API

We provide API endpoints to retrieve the wallet balance and to [re-]generate a wallet.
To retrieve the balance, call the user API endpoint

Endpoint
GET https://[your_URL]/user

Response

```
{
   "balance": 5000,
   ...
}
```

Endpoint
POST https://[your_URL]/config

Request

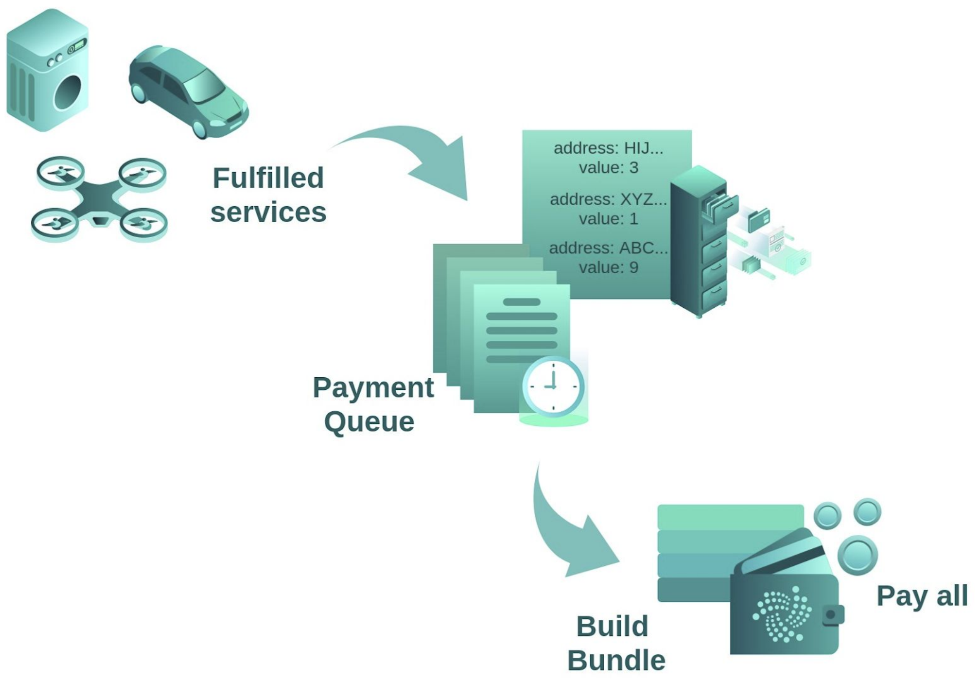
```
{
   "wallet": true                    /* true or false */
}
```

Response

```
{
   "success": true
}
```

Payment Queue

All payment requests of the given entity are sent to the payment queue and kept there for up to 5 minutes. Every 5 minutes the queue is read and all payment requests are bundled into one payment transaction and send to the Tangle.



This payment type is non-blocking, it runs on the background and updates the internal wallet balance once the transaction confirmation is received.

Sensor Data Provision

In some cases no actual service should be provided by the Service Provider, but instead certain data needs to be securely delivered to the Service Requester.
To indicate which operations are expected to deliver data, we maintain configuration property called “`dataRequest`” which can be configured in `ServiceApp/server/src/config.json`. Messages with IRDIs that listed in this configuration property are expected to contain information to retrieve data in the additional “`sensorData`” object added to the message. Service Provider entity can use POST API Endpoint `/data` to save this information in a local database and later submit it to the Service Requester.

We use sensor data format and delivery mechanisms implemented in another IOTA project called Data Marketplace. You can read more about how to publish your data to the Tangle here and here.

When your data is ready, you will need your Data Marketplace user ID (available from the Dashboard), device ID and data schema, which corresponds to the schema you used to generate and publish your data. Typical data schema is an array of objects, where each object consists of 3 key/value pairs to specify the ID of the property, its display name and unit of measure. You can add as many of the objects as needed.
Below is an example of a valid data schema

```
[
  {
    "id": "temp",
    "name": "Temperature",
    "unit": "C"
  },
  {
    "id": "hum",
    "name": "Humidity",
    "unit": "g/m3"
  },
  {
    "id": "press",
    "name": "Pressure",
    "unit": "bar"
  }
]
```

Submit information about the sensor data as payload of the POST request sent to the /data API endpoint. See API description section for details.

MarketManager UI

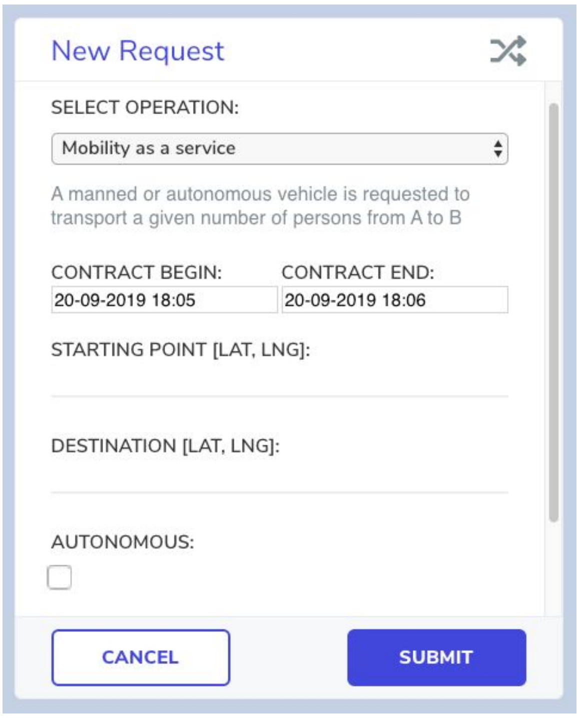
General Information

The UI application is developed using JavaScript, NodeJS v10 and React Framework.
In is connected to a local or remote instance of a MarketManager application.
Connection is established by specifying the URL of the MarketManager instance.
To change the default link to a local instance, you need to modify `proxy` setting in the `ServiceApp/package.json` as well as the domain setting in the `ServiceApp/src/config.json` file.

The UI application subscribes with the MarketManager to receive Socket.io messages addressed to it. Communication to the MarketManager is done using REST interface.

Randomizer

If you want to try the demo instance of the Service Requester, without a need to enter real values into the required input fields, you can use a built-in randomizer to generate valid values for each field. The generated values can be adjusted according to your preferences. Press on the crossed arrows on the top of the dialog to generate random values.



Mobility as a service

A manned or autonomous vehicle is requested to transport a given number of persons from A to B

CONTRACT BEGIN: 20-09-2019 18:05

CONTRACT END: 20-09-2019 18:06

STARTING POINT [LAT, LNG]:

DESTINATION [LAT, LNG]:

AUTONOMOUS: ☐

Request Expiration

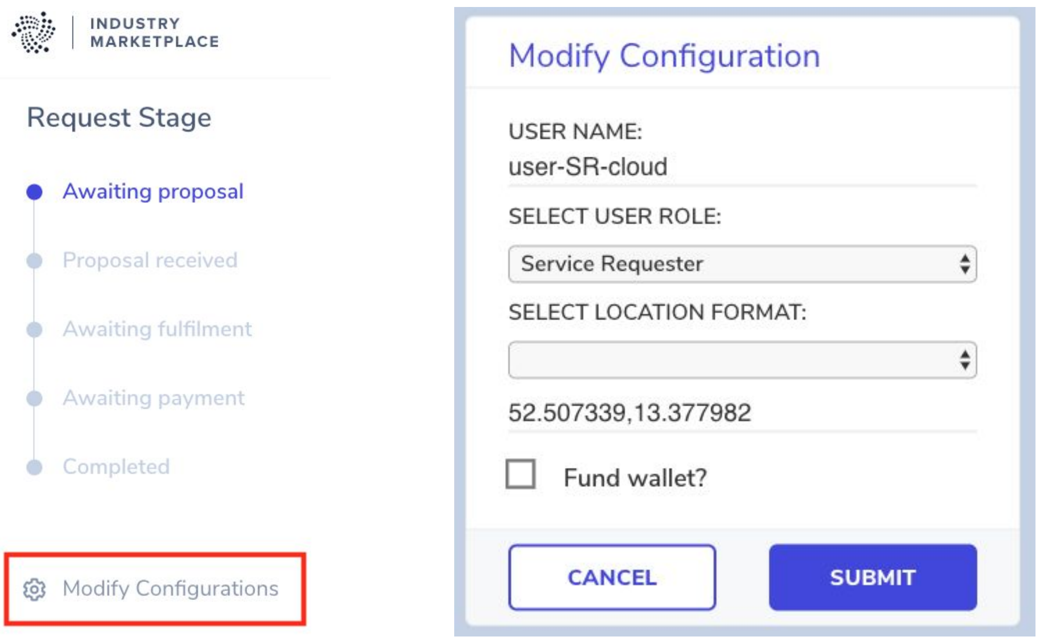
All created requests are expire and then removed from the view. Default expiration time is set to 10 minutes.
Request expiration time can be configured in the `config.json` file under `ServiceApp/src` folder. Variable name is `waitingTime`

Automatic Request Rejection

For one request there might be multiple proposals from different Service Providers. The Service Requested can accept only one proposal for the request. All other proposals are automatically rejected. The respective rejectProposal messages are sent to the corresponding recipients.

Configuration Dialog

From the configuration dialog you can specify the entity name, location (as address or GPS coordinates), select one of the available types (Requester or Provider).
You can also fund an existing internal wallet using the free Devnet IOTA Tokens

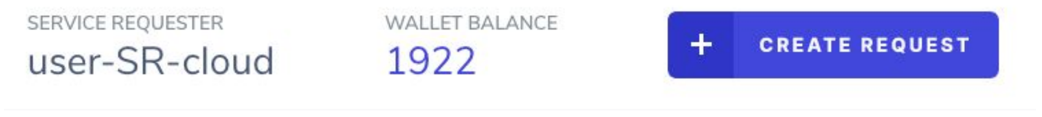


USER NAME:
user-SR-cloud
SELECT USER ROLE:
Service Requester
SELECT LOCATION FORMAT:
52.507339,13.377982
Fund wallet?

Configuration dialog is available for the instances that are operated locally. It is disabled for the Demo instance in order to prevent unwanted modifications.

Accepting of Proposals and issuing Payments

A prerequisite to accept a proposal and to issue a payment for the service is a wallet with enough balance. You can find the actual wallet balance on the top right side of the page.



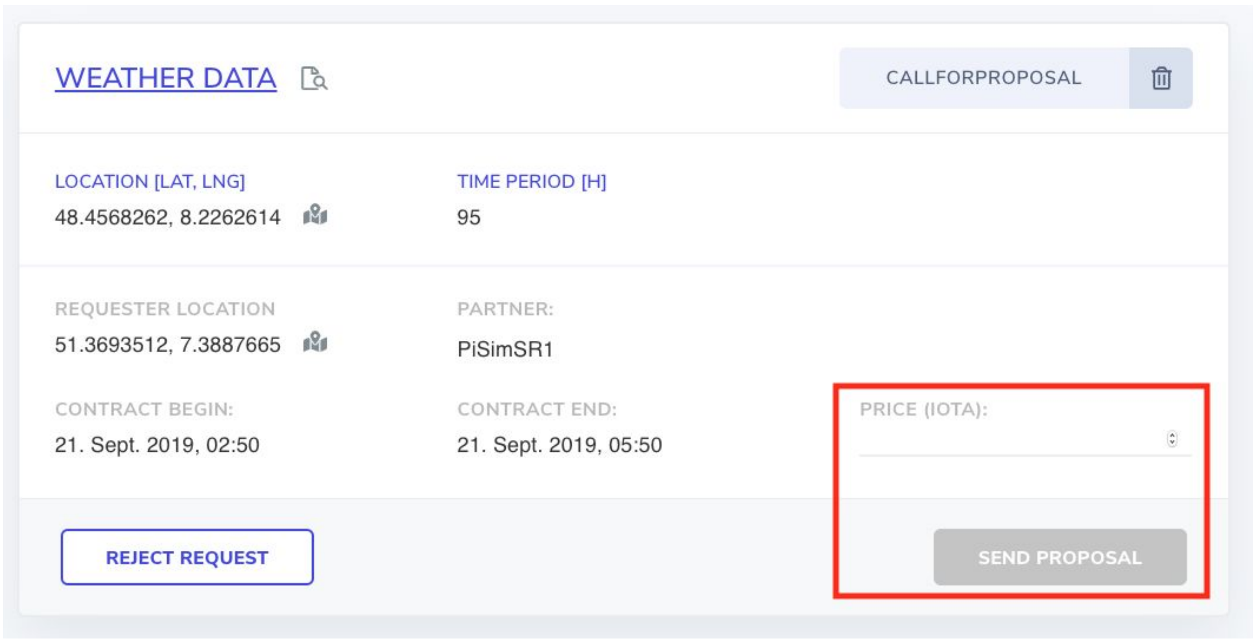
SERVICE REQUESTER
user-SR-cloud
WALLET BALANCE
1922

During the payment operation the wallet balance is temporary set to 0 to prevent double-spending.

Wallet balance is automatically fetched every 2 minutes.

Expired Requests can not be accepted.

To accept the proposal Service Requester is required to enter the price for the service. The price needs to be entered in IOTA tokens.
Until the price is entered, the button to send out a proposal remains disabled.



WEATHER DATA

CALLFORPROPOSAL

LOCATION [LAT, LNG]
48.4568262, 8.2262614

TIME PERIOD [H]
95

REQUESTER LOCATION
51.3693512, 7.3887665

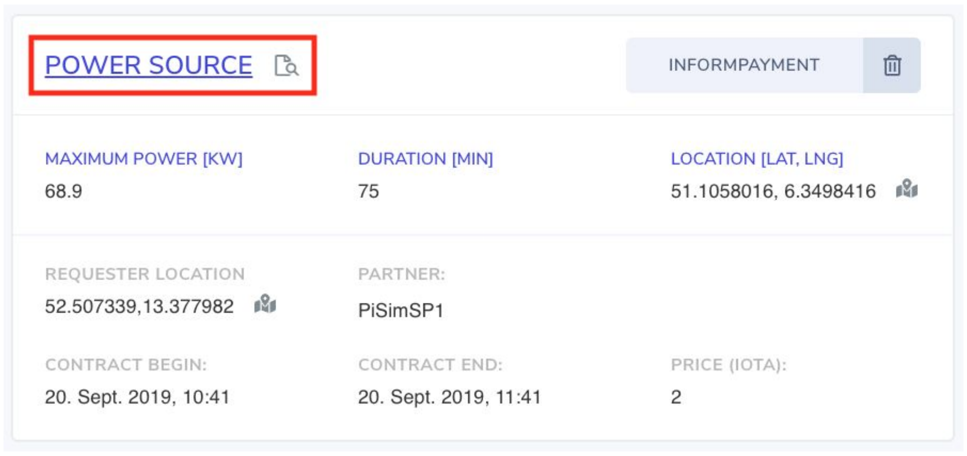
PARTNER:
PiSimSR1

CONTRACT BEGIN:
21. Sept. 2019, 02:50

CONTRACT END:
21. Sept. 2019, 05:50

PRICE (IOTA):

Audit Log / Transaction History



POWER SOURCE

INFORMPAYMENT

MAXIMUM POWER [KW]
68.9

DURATION [MIN]
75

LOCATION [LAT, LNG]
51.1058016, 6.3498416

REQUESTER LOCATION
52.507339,13.377982

PARTNER:
PiSimSP1

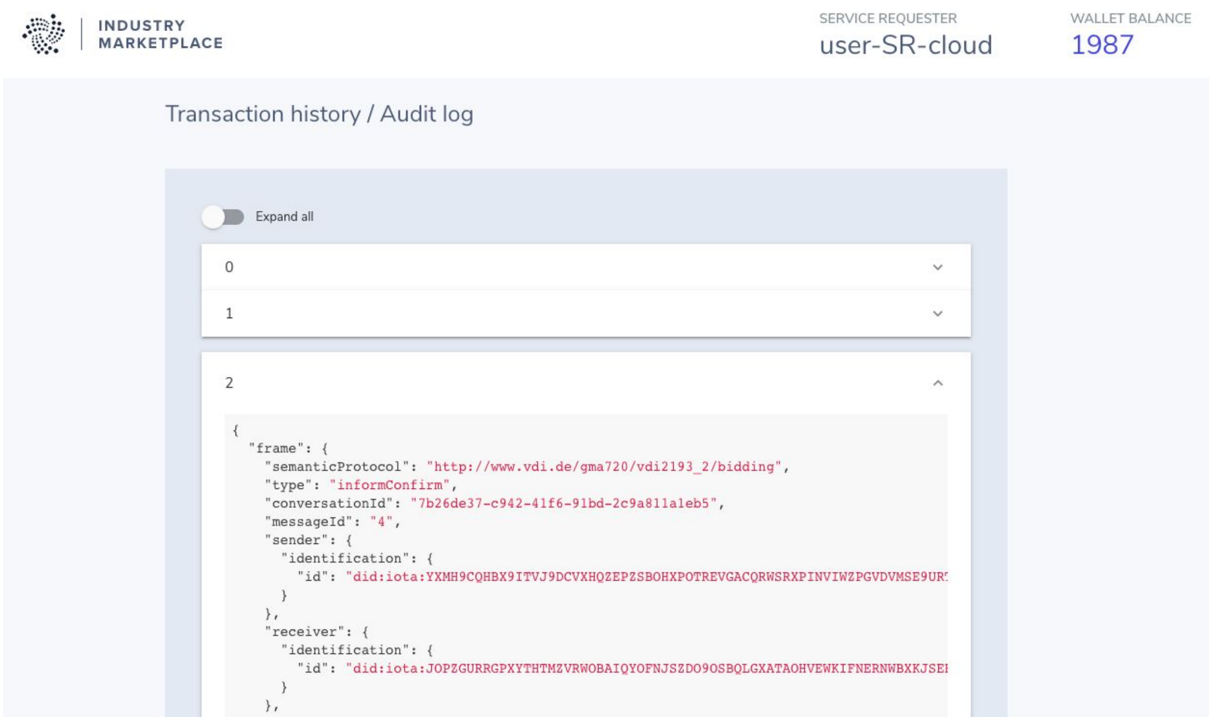
CONTRACT BEGIN:
20. Sept. 2019, 10:41

CONTRACT END:
20. Sept. 2019, 11:41

PRICE (IOTA):
2

By clicking on the request card header you will be redirected to another page where transaction history is shown. This data is retrieved directly from the encrypted MAM channel

on the Tangle, and can only be accessed by Service Requester and Service Provider entities participating in the transaction.
The data can be used as audit log.

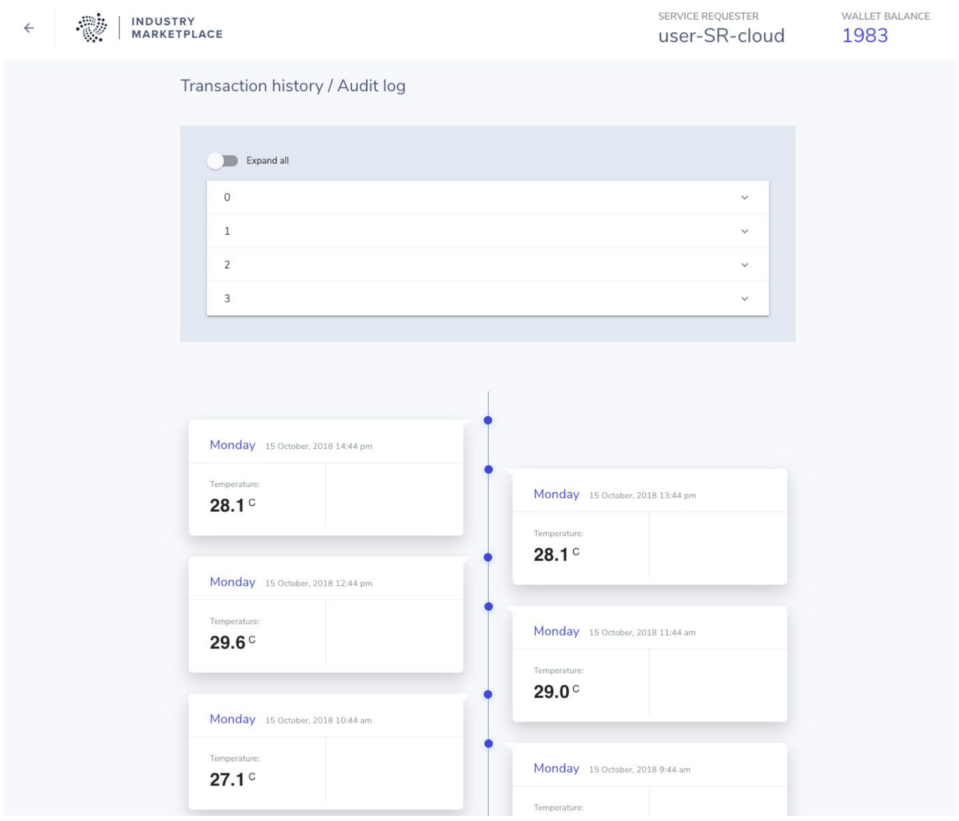


```
{
  "frame": {
    "semanticProtocol": "http://www.vdi.de/gma720/vdi2193_2/bidding",
    "type": "informConfirm",
    "conversationId": "7b26de37-c942-41f6-91bd-2c9a811a1eb5",
    "messageId": "4",
    "sender": {
      "identification": {
        "id": "did:iota:YXMH9CQHBX9ITVJ9DCVXHQZEPZSBOHXPOTREVGACQRWSRXPINVIWZPGVDVMSE9UR
      }
    },
    "receiver": {
      "identification": {
        "id": "did:iota:JOPZGURRGPXYTHTMZVRWOBAIQYOFNJSZDO9OSBQLGXATAOHVEWKIFNERNWBXKJSE
      }
    },
```

Sensor Data

If the requested service is for providing data from a device or sensor, the data can be accessed on the same page where the transaction log is available.
Click on the request card header, wait until data is fetched from the encrypted MAM channel using credentials provided by the Service Provider.
The data is displayed as cards in chronological order.
This information can also be retrieved using a script and therefore can be exported into another application.

Please note that the data is only available after the request was marked as fulfilled.
Service Provider can determine how long the data will remain available.



INDUSTRY MARKETPLACE

SERVICE REQUESTER
user-SR-cloud

WALLET BALANCE
1983

Transaction history / Audit log

Expand all

0

1

2

3

Monday 15 October, 2018 14:44 pm
Temperature:
28.1 C

Monday 15 October, 2018 13:44 pm
Temperature:
28.1 C

Monday 15 October, 2018 12:44 pm
Temperature:
29.6 C

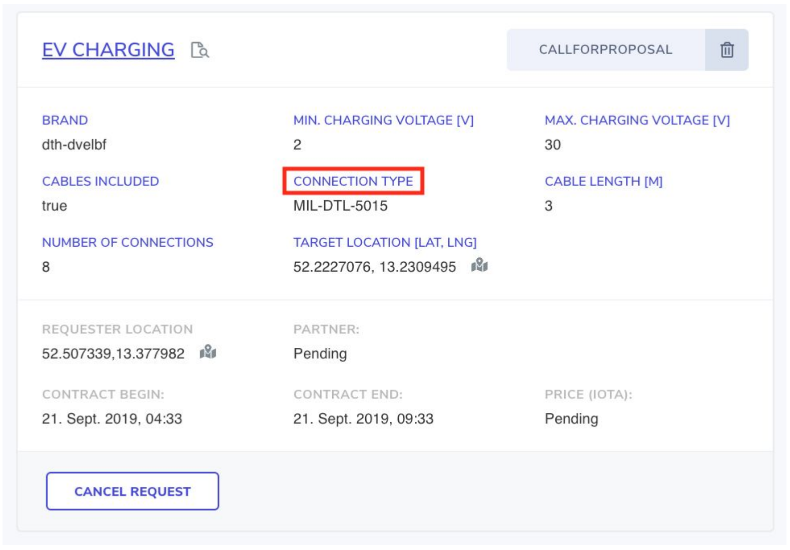
Monday 15 October, 2018 11:44 am
Temperature:
29.0 C

Monday 15 October, 2018 10:44 am
Temperature:
27.1 C

Monday 15 October, 2018 9:44 am
Temperature:

External Links

All blue-colored captions on the upper part of the request card are clickable links.



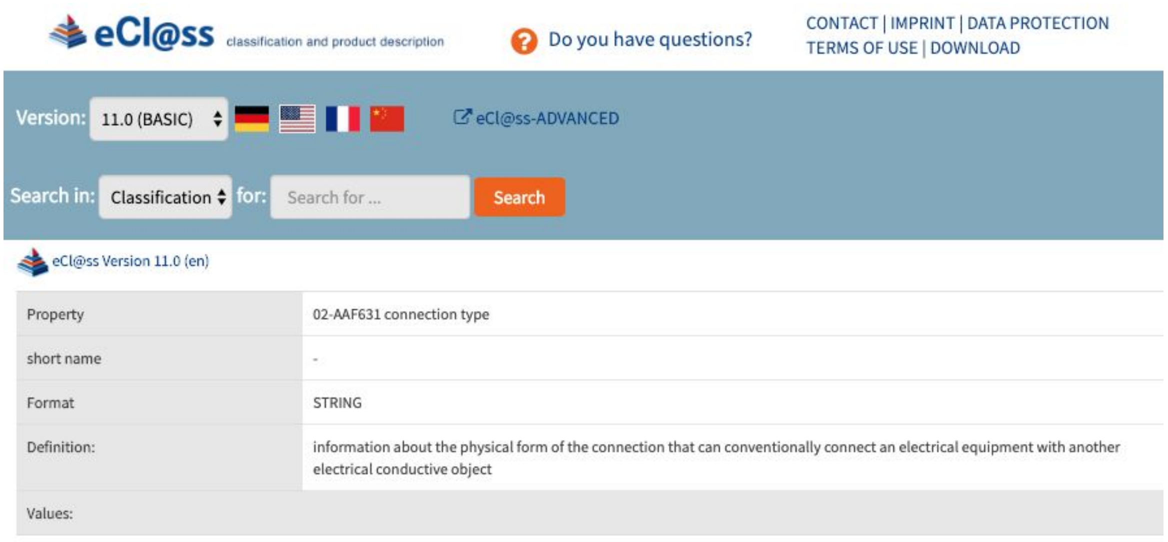
EV CHARGING

CALLFORPROPOSAL

BRAND dth-dvelbf	MIN. CHARGING VOLTAGE [V] 2	MAX. CHARGING VOLTAGE [V] 30
CABLES INCLUDED true	CONNECTION TYPE MIL-DTL-5015	CABLE LENGTH [M] 3
NUMBER OF CONNECTIONS 8	TARGET LOCATION [LAT, LNG] 52.2227076, 13.2309495	
REQUESTER LOCATION 52.507339,13.377982	PARTNER: Pending	
CONTRACT BEGIN: 21. Sept. 2019, 04:33	CONTRACT END: 21. Sept. 2019, 09:33	PRICE (IOTA): Pending

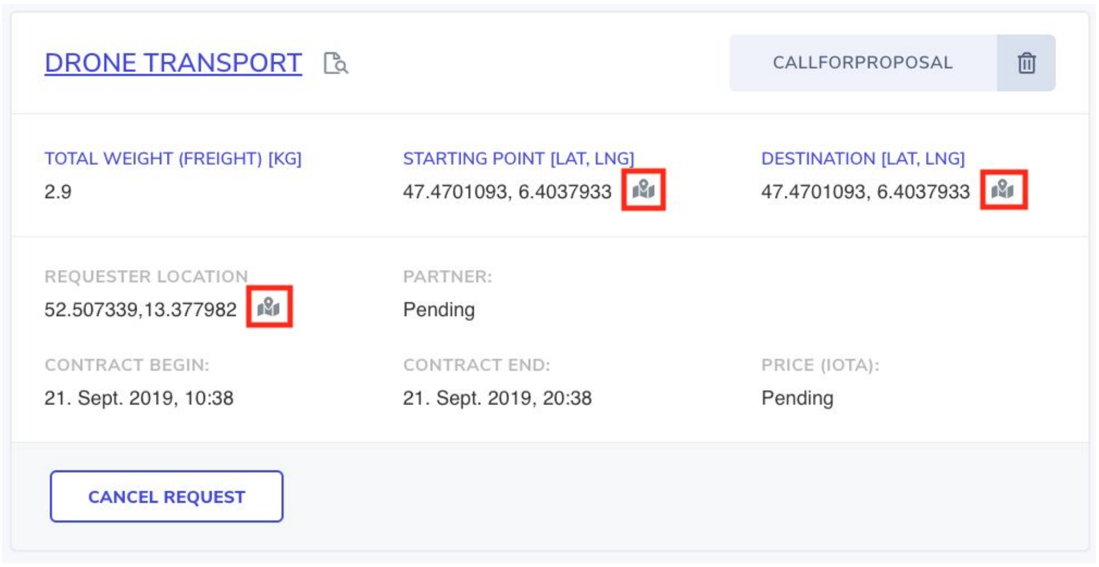
CANCEL REQUEST

The link redirects to the respective parameter description in the eCl@ss catalog


eCl@ss classification and product description
Do you have questions?
CONTACT | IMPRINT | DATA PROTECTION
TERMS OF USE | DOWNLOAD
Version: 11.0 (BASIC)
eCl@ss-ADVANCED
Search in: Classification for: Search for ...
Search
eCl@ss Version 11.0 (en)

Property	02-AAF631 connection type
short name	-
Format	STRING
Definition:	information about the physical form of the connection that can conventionally connect an electrical equipment with another electrical conductive object
Values:	

Another type of external links redirects to a Google Maps service with the location specified by GPS coordinates



TOTAL WEIGHT (FREIGHT) [KG]	STARTING POINT [LAT, LNG]	DESTINATION [LAT, LNG]
2.9	47.4701093, 6.4037933	47.4701093, 6.4037933
REQUESTER LOCATION	PARTNER:	
52.507339,13.377982	Pending	
CONTRACT BEGIN:	CONTRACT END:	PRICE (IOTA):
21. Sept. 2019, 10:38	21. Sept. 2019, 20:38	Pending

ZMQ Service

Every IOTA reference implementation (IRI) comes with a Zero Message Queue attached, which allows streaming of the near real-time Tangle data from a node. To process this data, we use a stream processing library called ZeroMQ to parse the raw event messages into class instances. All ZMQ events are available for processing. You can read more about the ZMQ implementation for IRI here.
In this project we only consider events of type **tx**, which represent transactions.

Functionality Description

In this project we use WebSockets technology, implemented by the library called socket.io. Industry Marketplace UI subscribes to receive WebSocket notifications about relevant messages.
ZMQ library processes all events that happens on the Tangle nearly in real-time, filters out event of type **tx** and starts with the decoding and filtering routine which is described in detail in the next subtopics.
Once the event payload was decoded and the bundle hash was retrieved, we perform lookup on the Tangle in order to find the transactions of the bundle. After this we extract signature message fragments from these transactions, concat them together and decode payload from trytes.

Applied Filters

In order to correctly route the received eCl@ss message to the respective receiver, we perform multi-step filtering routine.

Filters can be configured using the `config.json` file from the `ServiceApp/server/src` folder.

Filter by Operation Type. This filter applies for both Service Requester (SR) and Service Provider (SP) roles. An entity can specify a list of supported operations under the attribute "`operations`": [...] within the configuration script (`config.json`) of the Market Manager. The operation identifier is added to the transaction tag for faster filtering. Only messages related to operations from the list are allowed to pass through.
Configuration parameter "`operations`" from the `config.json` file

Filter by Entity Role and Message Type. We differentiate between Service Requester (SR) and Service Provider (SP) roles. Each role has a specific set of messages that it can accept. For SR the valid messages are "proposal" and "informConfirm". For SP the valid messages are "callForProposal", "acceptProposal", "rejectProposal", and "informPayment". For the YellowPages Application accepted messages are "callForProposal", "proposal", and "acceptProposal".

Filter by Location. This filter only applies for Service Provider. If entity has defined its location and maximum range within requests should be accepted , the filter calculates distance between SR and SP. This distance is then compared with the maximum range value defined in the configuration script. Only requests from Service Requesters located within this range are allowed to pass through.
Configuration parameter "`maxDistance`" from the `config.json` file, along with own location, configured via API or MarketManager UI configuration dialog.

Filter by Recipient (user DID). This last filter applies for message types different from "`callForProposal`", which is sent to every Service Provider. All other messages already specify the "`receiver`" ID field, which is retrieved and compared to the internal DID of the entity. Only in case of a match the message is allowed to pass through.

Local Database

For this project it is required to store certain data locally. We are using an sqlite3 library to utilize database functionality.

Storage for MAM Data

For every new request (call for proposal) a new restricted MAM channel is created. Its root address, randomly generated encryption key and other meta-data is stored in the table called “mam”. This information is required in order to append new transaction information about the request to the MAM channel.

Storage for Entity Data

For every new entity (organisation or customer) its basic information like role, DID, display name, GPS coordinates are stored in the table called “user”.

Storage for Wallet Data

For every new entity (organisation or customer) its wallet information like seed, address and other meta-data are stored in the table called “wallet”.

Storage for Digital Identity

For every new entity (organisation or customer) a private/public key pair is generated. Public key is then published in a public MAM channel, root address of this channel becomes a part of the entity’s DID. Private key along with the root address and seed of the public MAM channel and other meta-data are stored in the table called “did”.

Deployment and Hosting

The project consists of 2 main parts. The backend part, which is called **MarketManager**, can be used without the User Interface part, if your entity prefer to operate using API endpoints and MQTT streams.
Frontend part provides User Interface and should be connected to the respective **MarketManager** instance using proxy, where the URL and port of the **MarketManager** are specified.

The YellowPages Application is built using the similar architecture, where you also need to operate a simplified instance of the **MarketManager** and the User Interface application.

Backend Deployment

This application is designed to run on small IoT devices like Raspberry Pi, inside a Docker container, as a cloud instance or locally.
We support Raspberry Pi 3 Model 3+ and newer models. Please note that NodeJS v.10 and above is required to run the application.
For Docker container we suggest to use `10.16.2-alpine` image. We also ship a pre-configured Dockerfile with the required configuration.

In order to run the instance locally or on the Raspberry Pi 3/4, check out the source code of the ServiceApp, proceed to the server folder, ensure the configuration settings in src/config.json are correct for your entity. You can start the application by executing the “`yarn start-dev`” command from the “`server`” root folder. The backend will run on http://localhost:4000

If you want to deploy an instance of the MarketManager to the cloud, you can prepare a distributive version by executing the “`yarn build`” command from the “`server`” root folder. It will create a folder called “`build`” that you can deploy to the cloud provider.

Our demo instances are deployed to the **`Zeit.co`** cloud provider, and the project is shipped with configuration files to simplify deployment process. If you decide to use `Zeit.co` for deployment, please ensure that v1 deployments are enabled for your account. then run “`yarn deploy`” command to proceed with deployment. No other actions needed in this case.

Additional information (Preparations for the start)

To use the MarketManager, the following steps are to be done:

1. Make sure you have installed git, yarn and node

2. Clone or download repository from https://github.com/iotaledger/industry-marketplace.git

3. In folder *industry-marketplace/ServiceApp* run command `yarn dev` to install required packages, build and launch the application.

User Interface Deployment

User Interface Application is implemented using React Framework. It requires NodeJS v.10 or newer. The frontend application can be started locally by executing "`yarn start`" command from the root folder. The frontend will run on http://localhost:3000

You can also create a set of deployable files by executing "`yarn build`" command. It will produce a "`build`" folder that can be uploaded to the cloud provider.

Our demo instances are deployed to the **`Zeit.co`** cloud provider, and the project is shipped with configuration files to simplify deployment process. If you decide to use `Zeit.co` for deployment, run "`now`" command to proceed with deployment. No other actions needed in this case.

YellowPages Application Deployment

YellowPages application has a similar structure and deployment instruction as the main **ServiceApp** project folder. Please deploy or start locally the server instance first, then deploy or start the User Interface Application.

Whitelisting your Requesters and Providers

In order to get the proposals from your Requesters and/or Providers labelled as trusted, the MarketManager need to receive a Verifiable Credential from the IOTA Foundation. This can be done by emailing the DID of the MarketManager instance (Requester or Provider) to industry@iota.org.

The DID can be retrieved by running the didHelper.js script in the server/build/src/utils folder. Run the commands: "cd server/build/src/utils" and "node didHelper.js" in succession. Please note that the didHelper.js only runs correctly after the cd command. This outputs the DID of the instance in the command line. Copy this into the email with your company name. Keep the marketmanager online at least until a confirmation email has been sent.

This process will be improved in the next set of updates. It will later be possible to whitelabel your own DID's directly. Any party that trusts the organisation, will then trust every DID whitelabelled by that organisation.

Additional Services

Wallet Faucet

In the Industry Marketplace ecosystem each entity (user/organisation) needs to have an IOTA Wallet funded with IOTA Tokens.
For the sake of demonstration we will be distributing small portions of the free IOTA Devnet Tokens for every new entity which joins the initiative in 2019. We are using a tool called Faucet, which transfers a specified amount of IOTA Devnet Tokens to the internal entity's wallet address when the wallet is created for the first time.

Address Converter

An entity can specify its location (optional) as GPS coordinates in the format of two comma separated numbers for latitude and longitude ["51.507, 12.377"] or as a valid postal address. If you specify an address, please use only latin letters and no german umlaut letters. The address will be converted to GPS coordinates using a cloud based Google Maps converter. Address conversion is run on a microservice operated by IOTA, which stores the Google API key.

Mapbox (YellowPages Application)

Mapbox service is used by the YellowPages Application to display request locations on the map. Mapbox API key is managed by a microservice operated by IOTA.

Industry Marketplace Push Interfaces

WebSockets

One of the major tasks of the Market Manager is to transmit relevant messages from the Tangle to the receiver. Therefore, the Market Manager needs to build up a persistent connection to the Zero Message Queue, which fetches all incoming transactions from the Tangle. Since the REST API is in first place not suitable for a persistent connection, websockets are used to tackle this task. The Market Manager application receives all incoming transactions from the ZMQ, it filters only relevant ones for the client by matching its configuration with the content of the messages provided with the Semantic I4.0 Language.

The implemented WebSockets are based on Socket.io and therefore a socket-io-client is required on the client side.

Connect to server

```
const socket = require('socket.io-client')('http://localhost:4000');

socket.on('connect', () => {
    console.log("Connected")
});
```

Subscribe to messages from Market Manager

```
socket.emit('subscribe', { events: ['tx'] })
```

Receive Data from the Market Manager

```
socket.on('zmq', (data) => {
    console.log('Received message from Market Manager:', data)
});
```

Unsubscribe to messages from Market Manager

```
socket.emit('unsubscribe', { subscriptionIds: ['subscriptionId'] } )
```

MQTT

As an alternative to the websocket connection, the Market Manager also offers an MQTT Interface. A helper client is created, which connects to the websockets and publishes the messages via MQTT. The MQTT Interface has to be activated via an API call.

Market Manager can transmit incoming messages that are relevant to the user via MQTT.

The Market Manager does not offer an own MQTT Broker, it is suggested to either use an open source MQTT broker such as 'test.mosquitto.org' or implement an own MQTT Broker.

Market Manager API

The client transmits messages to the Market Manager via a REST API. Depending on the type of message, the Market Manager then executes different tasks. One major task that the Market Manager executes for every incoming message is wrapping it up into a transaction and sending it to the Tangle.

Entity configuration

Receives display name, role, location as GPS coordinates and flag for new wallet generation.

- Writes configuration details to database

Returns success or failure notification

Endpoint
https://[your_URL]/config

POST

Request
```
{
   "name":   "service-requester-1",         /* text */
   "role":   "SR",                          /* "SR" or "SP" */
   "gps":    "50.34556, 12.85844",          /* text */
   "wallet": true                           /* true or false */
}
```

Response
```
{
   "success": true
}

or

{
   "success": false,
   "error": [error message]
}
```

Sensor Data provision

Receives conversationId, access credentials for sensor data and schema of sensor data

- Writes access details to local database

Returns success or failure notification

Endpoint
https://[your_URL]/data

POST

Request

```
{
   "conversationId": "8e585f93-aea2-41b4-4abe", /* text */
   "deviceId":       "WeatherStation",          /* text */
   "userId":         "jkldnkfvnksd",            /* text */
   "schema": [                                  /* array of JSON objects */
      { id: 'temp', name: 'Temperature', unit: 'C' },
      { id: 'hum',  name: 'Humidity',    unit: 'g/m3' },
      { id: 'temp', name: 'Pressure',    unit: 'bar' }
   ]
}
```

Response

```
{
   "success": true
}
```

or

```
{
   "success": false,
   "error": [error message]
}
```

Entity information

- Generates public/private key pair if not in local database
- Creates public MAM channel and uses root address as DID
- Publishes public key as message to public MAM channel
- Saves private key under DID in local database

Returns display name, role, location, wallet address and wallet balance

Endpoint
https://[your_URL]/user

GET

Request

```
https://[your_URL]/user
```

Response

```
{                                            /* JSON object */
   "name": "service-requester-1",
   "role": "SR",
   "id": "did:iota:KFYZBQEETPCNFZOFQGVEJTECNRGYMJJZTLUGQLXJC",
   "location": "51.507339, 12.377985",
   "balance": 5000,
   "wallet": "EKT9QZIW9NDS9XKBDMGAKNUVSYKFUYPGPCWWVGAXWFNCTWTACB",
}
```

MAM Channel information

Returns MAM channel access details

Endpoint
https://[your_URL]/mam

GET

Request

```
https://[your_URL]/mam
```

Response

```
{                                            /* JSON object */
   "id": "8ee11981-3aa9-4873-b610-0293e3098d15",
   "root": "S9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHLUYHUIMMGAKN",
   "seed": "FZOFQGVEJTECNRGYMJJZTLUGQLXJCUQEDOPPAJURPWBNEGKDMAL",
   "next_root": "KFYZBQEETPCNFZOFQGVEJTECNRGYMJJZTLUGQLXJCKFUYPGPCWWV",
   "start": 5,
   "side_key": "EKT9QZIW9NDS9XKBDMGAKNUVSYKFUYPGPCWWVGAXWFNKFCTWTACB",
}
```

MQTT Interface

As an alternative to the websocket connection, the Market Manager also offers a MQTT Interface. For this, a HelperClient is created, which connects to the websockets and publishes the messages via MQTT. The MQTT Interface has to be activated via an API call.

To unsubscribe to the messages another API call is required.

Returns success or failure notification and a subscriptionID. The subscriptionID is used as a topic to publish all messages that belong to the client that received the subscriptionID.

Endpoint
POST https://[your_URL]/mqtt

Subscribe Request

```
{
   "message": "subscribe"
}
```

Creates helper client that connects to websockets

Helper client subscribes to messages from Market Manager and publishes them under the subscriptionID as topic.

Unsubscribe Request

```
{
   "message": "unsubscribe",
   "subscriptionId": "5742a685-657b-4b94-a704-36e00bc46a5a"
}
```

Unsubscribes to messages from Market Manager

Response

```
{
   "success": true,
   "id": "5742a685-657b-4b94-a704",
}

or

{
   "success": false,
   "error": [error message]
}
```

Call For Proposal

Receives ‘Call for Proposal’ according to Industry 4.0 Semantic

- Creates a custom tag from Prefix, type of message and operationID
- Sends transaction with custom tag to the Tangle
- Creates MAM-channel and publishes call for proposal content to MAM channel
- stores MAM-channel in database under conversation-ID

Returns success or failure notification, tag, transaction hash and MAM information.

Endpoint
https://[your_URL]/cfp

POST

Request

```
{
   "messageType":    "callForProposal",               /* text */
   "irdi":           "0173-1#01-AAJ336#002",          /* text */
   "userId":         "did:IOTA:DFGDFKJSKSJGKJSGL",    /* text */
   "replyTime":      10,                              /* number */
   "location":       "52.507339, 13.377982",          /* text */
   "startTimestamp": 1568955521151,                   /* number */
   "endTimestamp":   1568959121151,                   /* number */
   "creationDate":   "19 September, 2019 23:58 pm",   /* text */
   "userName":       "serviceRequester11",            /* text */
   "submodelValues": { …                             /* JSON object */
       '0173-1#02-AAI711#001': 45.5,
       '0173-1#02-BAF464#008': true,
       '0173-1#02-BAF163#002': '52.507339, 13.377982'
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL",
   "mam":     "EXTGKJG99VVDUWJJKMLISHLUYHUIMMGAKNGPCWVGAXW"
}
```

or

```
{
   "success": false,
   "error":   [error message]
}
```

Proposal

Receives ‘proposal’ according to Industry 4.0 Semantic

- Creates a custom tag from Prefix, type of message and operationID

- Sends transaction with custom tag to the Tangle

Returns success or failure notification, tag and transaction hash.

Endpoint
https://[your_URL]/proposal

POST

Request

```
{
   "price":           10,                              /* number */
   "messageType":     "proposal",                      /* text */
   "irdi":            "0173-1#01-AAJ336#002",          /* text */
   "userId":          "did:IOTA:SFLDFBNMASDGKSADG",    /* text */
   "replyTime":       10,                              /* number */
   "userName":        "serviceProvider11",             /* text */
   "originalMessage": { …                             /* JSON object */
      "frame": { … },                                 /* JSON object */
      "dataElements": { … }                           /* JSON object */
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL"
}

or

{
   "success": false,
   "error":   [error message]
}
```

Accept Proposal

Receives ‘acceptProposal’ according to Industry 4.0 Semantic

- Creates a custom tag from Prefix, type of message and operationID
- Publishes acceptProposal to MAM-channel
- Sends transaction with custom tag to the Tangle

Returns success or failure notification, tag, transaction hash and MAM information.

Endpoint
https://[your_URL]/acceptProposal

POST

Request

```
{
   "price":           10,                              /* number */
   "messageType":     "acceptProposal",                /* text */
   "irdi":            "0173-1#01-AAJ336#002",          /* text */
   "userId":          "did:IOTA:DFGDFKJSKSJGKJSGL",    /* text */
   "replyTime":       10,                              /* number */
   "userName":        "serviceRequester11",            /* text */
   "originalMessage": { …                             /* JSON object */
      "frame": { … },                                 /* JSON object */
      "dataElements": { … }                           /* JSON object */
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL",
   "mam":     "EXTGKJG99VVDUWJJKMLISHLUYHUIMMGAKNGPCWVGAXW"
}
```

or

```
{
   "success": false,
   "error":   [error message]
}
```

Reject Proposal

Receives ‘rejectProposal’ according to Industry 4.0 Semantic

- Creates a custom tag from Prefix, type of message and operationID
- Sends transaction with custom tag to Tangle

Returns success or failure notification, tag and transaction hash.

Endpoint
https://[your_URL]/rejectProposal

POST

Request

```
{
   "price":          10,                              /* number */
   "messageType":    "rejectProposal",                /* text */
   "irdi":           "0173-1#01-AAJ336#002",          /* text */
   "userId":         "did:IOTA:DFGDFKJSKSJGKJSGL",    /* text */
   "replyTime":      10,                              /* number */
   "userName":       "serviceRequester11",            /* text */
   "originalMessage": { …                            /* JSON object */
      "frame": { … },                                /* JSON object */
      "dataElements": { … }                          /* JSON object */
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL"
}
```

or

```
{
   "success": false,
   "error":   [error message]
}
```

Inform Confirm

Receives ‘informConfirm’ according to Industry 4.0 Semantic

- Creates a custom tag from Prefix, type of message and operationID
- Publishes informConfirm to MAM-channel
- Retrieves wallet address from database
- Adds wallet address to payload of transaction
- Sends transaction with custom tag to Tangle
- In case of sensor data request, retrieves access credentials and appends them to the payload before sending the transaction

Returns success or failure notification, tag and transaction hash.

Endpoint
https://[your_URL]/informConfirm

POST

Request

```
{
   "price":           10,                              /* number */
   "messageType":     "informConfirm",                 /* text */
   "irdi":            "0173-1#01-AAJ336#002",          /* text */
   "userId":          "did:IOTA:SFLDFBNMASDGKSADG",    /* text */
   "replyTime":       10,                              /* number */
   "userName":        "serviceProvider11",             /* text */
   "originalMessage": { …                             /* JSON object */
      "frame": { … },                                 /* JSON object */
      "dataElements": { … }                           /* JSON object */
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL"
}

or

{
   "success": false,
   "error":   [error message]
}
```

Inform Payment

Receives 'informPayment' according to Industry 4.0 Semantic

- processes payment
- Creates a custom tag from Prefix, type of message and operationID
- Publishes informPayment to MAM-channel
- Sends transaction with custom tag to Tangle

Returns success or failure notification, tag and transaction hash and MAM information.

Endpoint
https://[your_URL]/informPayment

POST

Request

```
{
   "price":          10,                              /* number */
   "messageType":    "rejectProposal",                /* text */
   "irdi":           "0173-1#01-AAJ336#002",          /* text */
   "userId":         "did:IOTA:DFGDFKJSKSJGKJSGL",    /* text */
   "replyTime":      10,                              /* number */
   "userName":       "serviceRequester11",            /* text */
   "originalMessage": { …                            /* JSON object */
      "frame": { … },                                /* JSON object */
      "dataElements": { … },                         /* JSON object */
      "sensorData": { … },                           /* JSON object */
      "walletAddress": "KGKSNGKNSVNKNSVKNAWEI...",    /* text */
   },
}
```

Response

```
{
   "success": true,
   "tag":     "SEMARKETEAAJDDG999999999999",
   "hash":    "9NDS9XKBDYQSNBJXDGXYEXTGKJG99VVDUWJJKMLISHL"
}
```

or

```
{
   "success": false,
   "error":   [error message]
}
```

Glossary Of Terms

- **IRDI -** International Registration Data Identifier**,** ISO29002-5, ISO IEC 6523 and ISO IEC 11179-6 [20] as an Identifier scheme for properties and classifications. They are created in a process of consortium-wise specification or international standardisation. To this end, users sit down together and feed their ideas into the consortia or standardisation bodies. Properties in ISO, IEC help to safeguard key commercial interests. Repositories like eCl@ss and others make it possible to standardise a relatively large number of Identifiers in an appropriately short time.
- **MAM** - Masked Authentication Messaging is a second layer data communication protocol which adds functionality to emit and access an encrypted data stream, like RSS, over the Tangle https://blog.iota.org/introducing-masked-authenticated-messaging-e55c1822d50e
- **Proof of Work (PoW)** - An algorithm which prevents Denial of Service and spam attacks on a network. a computationally hard puzzle to solve, but easy to verify. IOTA uses a Hashcash based puzzle.

Additional Resources

- Project main page - https://industry.iota.org/
- YellowPages - https://industry.iota.org/demo
- Service Requester (demo instance) - https://service-requester.iota-dev1.now.sh/
- Service Provider (demo instance) - https://service-provider.iota-dev1.now.sh/
- Project source code - https://github.com/iotaledger/industry-marketplace
- Project documentation - https://industry.iota.org/files/Industry_Marketplace_Technical_Documentation.pdf
- Project press release - https://industry.iota.org/files/IOTA_Industry_Marketplace.pdf
- IOTA documentation - https://docs.iota.org