|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Description** | **Reviser** | **Date** |
| A | Initial Draft | Axel Meyer | 24.02.2011 |
| B | Specification Version 2 | Axel Meyer | 17.05.2011 |
| C | Review with Christian Rusch – Claas | Axel Meyer | 26.05.2011 |
| D | Review with Jan Lehnardt – CouchBase | Axel Meyer | 15.06.2011 |
| E | 2nd Review with Jan Lehnardt – CouchBase | Axel Meyer | 24.06.2011 |
| F | Modifications due to reference implementation and first test | Axel Meyer | 01.07.2011 |
| G | Spell-Check and comments by DFKI / Field-test experiences | Axel Meyer | 08.11.2011 |
| H | Detailed specification of URL’s | Axel Meyer | 18.01.2012 |

System Specification

For

iGreen MachineConnector V2

Contents

[1.1. Motivation 4](#_Toc308524363)

[1.2. Overview 4](#_Toc308524364)

[1.3. Scope 6](#_Toc308524365)

[1.4. Reference Documents 7](#_Toc308524366)

[1.5. Definitions 7](#_Toc308524367)

[1.6. Acronyms 8](#_Toc308524368)

[1.7. History 8](#_Toc308524369)

[2.1. General 9](#_Toc308524370)

[2.2. Addressing Device Nodes 9](#_Toc308524371)

[2.3. System Architecture 10](#_Toc308524372)

[2.4. Software Architecture 11](#_Toc308524373)

[2.4.1. Apache CouchDB 11](#_Toc308524374)

[2.4.2. Security Architecture 12](#_Toc308524375)

[2.4.2.1. ACLs / Certification and Validation 12](#_Toc308524376)

[2.4.2.2. SSL Encryption 12](#_Toc308524377)

[2.4.2.3. Basic Authentication 12](#_Toc308524378)

[2.4.2.4. Encryption of single documents 13](#_Toc308524379)

[2.4.3. OnlineBox connection 13](#_Toc308524380)

[2.5. Data types and structures 13](#_Toc308524381)

[2.5.1. Real time data 14](#_Toc308524382)

[2.5.1.1. History of data 15](#_Toc308524383)

[2.5.2. Documentation data / Task data 15](#_Toc308524384)

[2.5.3. Configuration data 16](#_Toc308524385)

[2.6. Machine Connector algorithms 18](#_Toc308524386)

[2.6.1. Algorithm for Machine Connector Client 19](#_Toc308524387)

[2.6.1.1. Initialization 19](#_Toc308524388)

[2.6.1.2. Replication 19](#_Toc308524389)

[2.6.1.3. Updating address list 21](#_Toc308524390)

[2.6.1.4. Compaction 22](#_Toc308524391)

[2.6.1.5. Thin Clients without CouchDB 22](#_Toc308524392)

[2.6.1.6. Maintaining the history of real-time status information 22](#_Toc308524393)

[2.6.2. Algorithm for Machine Connector Server 22](#_Toc308524394)

[2.6.2.1. Configuration User Interface 23](#_Toc308524395)

[2.6.2.2. OnlineBox interface 23](#_Toc308524396)

[2.7. Outlook to future revisions 23](#_Toc308524397)

[5.1. Use Case 1 27](#_Toc308524398)

[5.2. Use Case 2 27](#_Toc308524399)

[5.3. Use Case 3 28](#_Toc308524400)

[5.4. Use Case 4 28](#_Toc308524401)

Tables

[Table 1.1 – Definitions 7](#_Toc308524402)

[Table 1.2 – Acronyms 8](#_Toc308524403)

1. General

# Motivation

The global demand for agricultural products is growing rapidly. The two main drivers for this fact are the steadily growing demand for food due to the growing and developing world population on the one side and the increasing importance of biomass as an energy source on the other. In order to keep pace with this development in a sustainable and economic manner, improvements in agricultural productivity are required. Hence, the research project iGreen funded by the German Federal Ministry of Education and Research (BMBF) was started in 2009. Its purpose is to investigate how web-based technologies can be utilized to meet the described challenge. Therefore, an infrastructure is set up that enables efficient data exchange by providing innovative online services. Today, manufacturers of agricultural machines offer basic telematics solutions that allow to monitor machines remotely by providing information on its position, fuel consumption and workload in real-time. Even though such systems help to increase machine productivity, they mainly focus on optimizing single machine’s productivity only and are not compatible to each other. In order to overcome this productivity gap, an infrastructure enabling online data exchange between machines of iGreen-compatible manufacturers is one of the core objectives of the project. Besides basic telematics functionalities it enables advanced logistics and dispatching to meet the needs of mixed fleet operations.

# Overview

The vision of iGreen is to establish the methodologies and an according infrastructure to allow agricultural equipment to be included into a network in order to profit from services provided by various stakeholders participating in it as depicted in figure 1.

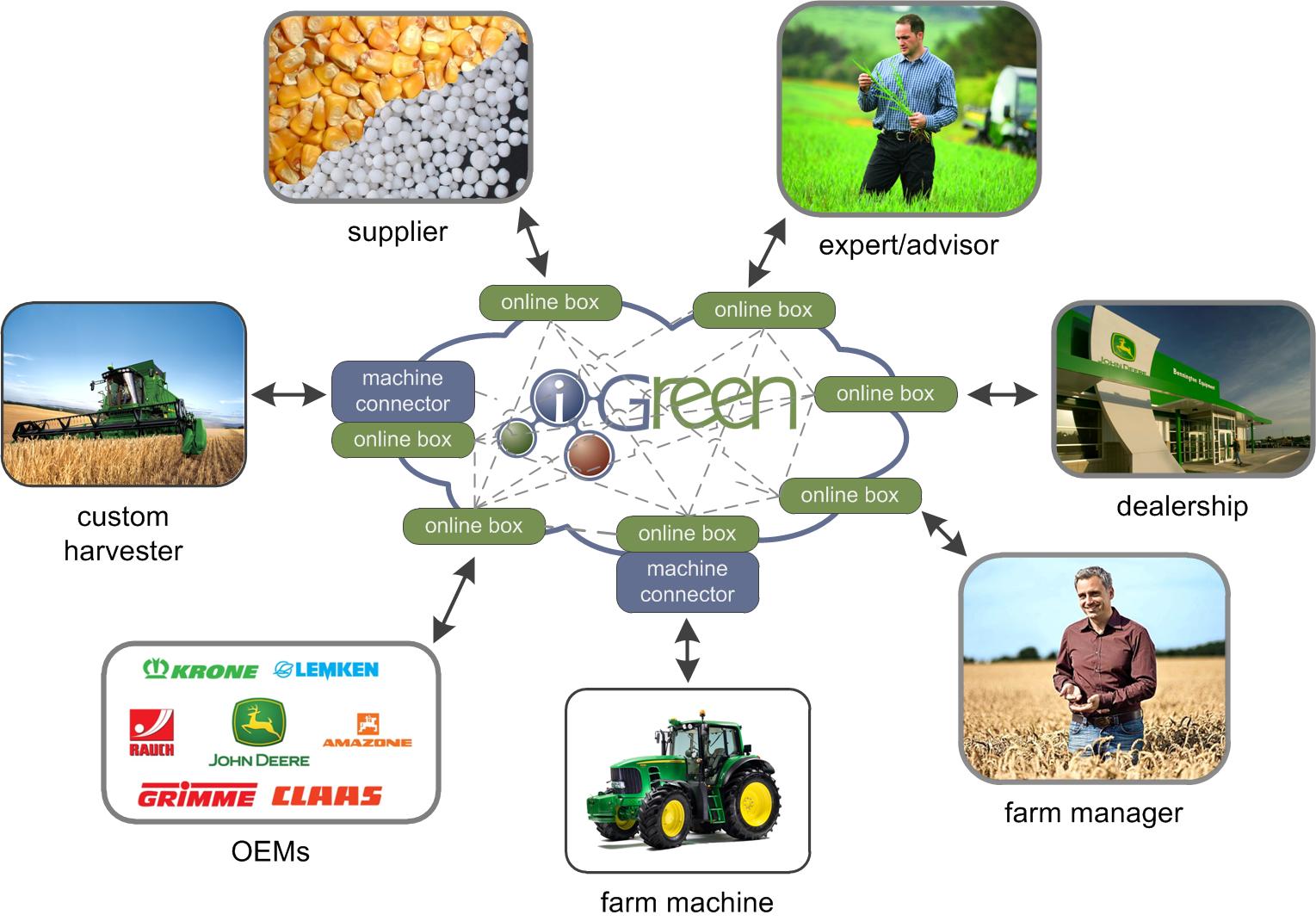


Figure : Data exchange concept via the iGreen network

Here, data of different physical origins is aggregated and propagated using semantic web technology. This is achieved via a distributed structure for data storage (spanned by the so called “Online Boxes”) that contains mechanisms for mediating content between potentially collaborating entities. Within this setup, there exist two main groups that contribute information. The first group is formed by agricultural machines that capture data during operation with their sensors. The second one consists of stakeholders such as OEMs, suppliers, public sources, and companies offering proprietary data and services that share their knowledge via the structure. The technical challenges that need to be overcome in order to be successful span from proper ways for data aggregation on an intra-machine level (such as sensor fusion in modular vehicle & implement combinations) all the way to the realization of distributed data storage/sharing and authentication or authorization functionalities to ensure data privacy.

Needless to say this task does not get easier if one considers the heterogeneity of the types of data that ranges from machine data to hardware information, task management, documentation/reporting data, prescriptions, or environmental information like the weather forecast or prognosis for pest infestation. Thus, mechanisms need to be established that allow the infrastructure to manage this vast load of data in an efficient way. Two key aspects that will be discussed in following sections in more detail are the Machine Connector and the Intelligent Vehicle Data Management system that both reside on the machines themselves. The former is responsible for managing data consistency and exchange on a technical level, while the latter ensures proper signal-level data processing and fusion within a single machine or machine combinations.

The iGreen Machine Connector (MC) is responsible for linking machines to the iGreen infrastructure (M2I) as well as to other machines (M2M). The basic requirement for this component is to enable make-independent data exchange. As no permanent data uplink can be assumed, mechanisms for buffering data during connection losses and successive synchronization are mandatory. According to these requirements the MC was designed as shown in figure 2. It comprises an internal database system, the machine data interfaces, and the so called MC daemon responsible for connecting the vehicle to the overall iGreen network.

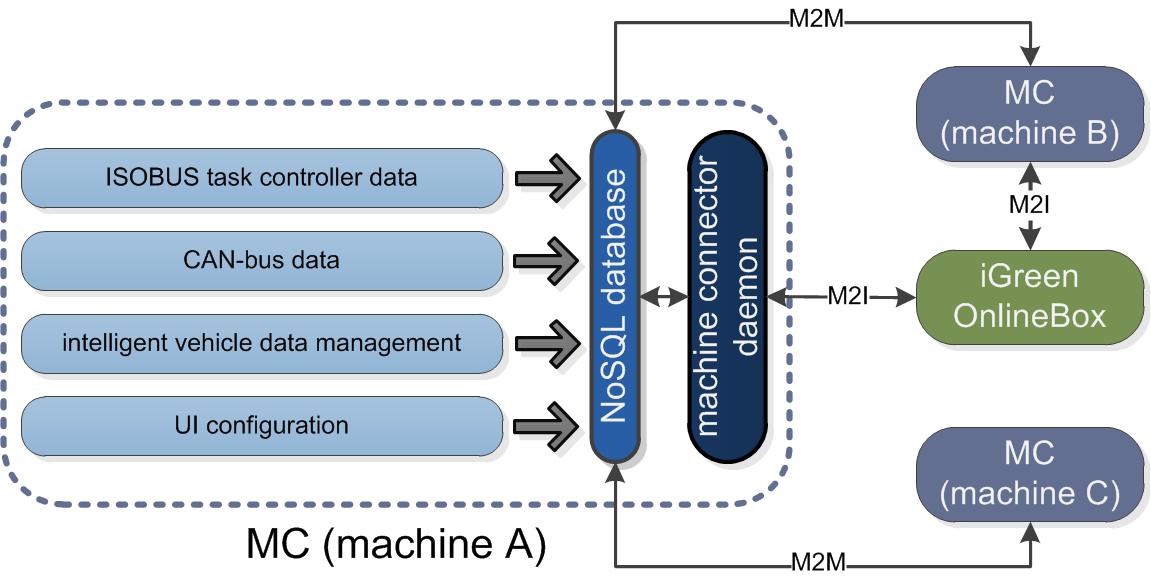


Figure : Schematic drawing illustrating the components of the machine connector (MC) as well as both modes of interaction between machines (M2M) and one machine with the iGreen infrastructure (M2I)

As storage entity a NoSQL (Not only SQL) type database is used. In contrast to classical SQL systems this allows for a greater flexibility regarding the addition of new vehicle types and data entries later on. Via the specified interfaces all relevant machine information that is provided by the ISOBUS Task Controller, CAN-Bus or Intelligent Vehicle Data Management is delivered to the database. The MC daemon is mainly responsible for enabling data exchange between the MC database and the overall iGreen infrastructure via the online box it is associated with. Furthermore, M2M data sharing is realized via the built-in replication and synchronization mechanism of the respective NoSQL databases.

For the MC three types of data are defined: Real-time, documentation and configuration information. The real-time data is used to share machine status information like the consolidated machine position. Real-time in this case refers to the fact that data is updated on change (threshold exceeded) or after a certain period of time. Documentation data contains totals and log files in ISOXML or any other proprietary data format. In contrast to real-time data not only the latest value is stored but the history of the data is maintained. Finally, the configuration information defines the settings for each individual MC, like the addresses and names of all the other MC nodes that data is to be replicated with.

In addition to the mentioned basic setup, it is desirable to support a store and forward data transfer[[1]](#footnote-1) which means that data is transferred wirelessly from e.g. a combine to the grain cart tractor during overloading. When the tractor then reaches the farm yard these entries are synchronized with the farm server. This mechanism would help to save provider costs and to enable data transfer in regions without cellular network coverage.

# Scope

This specification defines how the communication between a machine terminal (Embedded Display, Mobile Device and Automotive Computer) and the iGreen OnlineBox is working. Also, the iGreen MachineConnector should be able to allow Machine to Machine (M2M) communication. This specification must be in a detailed level which allows independent implementations of the machine connector communicating with each other. One implementation of this specification is done by DFKI as a part of the iGreen project.

Not in scope of this specification is the analysis of the data which is stored and exchanged with the machine connector or how to generate additional value out of this architecture.

# Reference Documents

For details on J1939 communication see the common available specification for J1939. The ISOBUS and ISOXML standard defined in ISO11783 are as well referenced in this specification.

# Definitions

These are the definitions that are common to all documents in the document. Despite other definitions that may exist for these terms, they are always to be used in this document as defined below to avoid confusion.

| **Term** | **Definition** |
| --- | --- |
| **Terminal** | Embedded Machine Terminal, Cell phone device, Automotive Computer |
| **iGreenID** | This identification number identifies a unique node in the iGreen network. For further details, please see the header: Addressing |
| **iGreen node** | (deprecated) This is a data end point in the iGreen Machine Connector network or cloud. |
| **Device Node** | New name for iGreen Node to prevent a confusion of terms in the iGreen project. |
| **Replication** | In CouchDB terms, for *synchronization* the word *replication* is used. |

Table 1.1 – Definitions

# Acronyms

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| REST | Representational State Transfer |
| CouchDB | Apache CouchDB |
| ACL | Access control list |
| M2M | Machine to machine communication |
| SSL | Security Service Layer |

Table 1.2 – Acronyms

# History

* 24.02.2011 – Initial release
* 17.05.2011 – Version 2 with replication mechanism and CouchDB integration
* 26.05.2011 – Review with Christian Rusch (Claas)
* 16.06.2011 – Review with Jan Lehnardt (CouchBase)
* 24.06.2011 – 2nd Review with Jan Lehnardt (CouchBase)
* Changed iGreenID naming using / (%2f) instead of \_ to support also minimalistic file systems which have limitations with the number of files in one directory. The old specification would cuase that there could be 1000 of files in one directory. The slash creates for each domain a sub directory which allows also easier service of the file system.
* Changed the automatic compression configuration to save CPU performance.
* Changed the data structure from the addresstable in the configuration data to a key value pair because this allows an easier implementation with the standard API.
* 01.07.2011 – Modifications due to reference implementation and first lab-test
* Comments and spell-check from Grimme (Hans van Zadelhoff / Dr. Johannes Sonnen) is integrated
* Changed CouchDB configuration: require\_valid\_user=true
* Changed section 2.5.3. to work with reference implementation
* Changed section 2.6.1.1. and 2.6.1.2. according to the reference implementation
* Added reference implementation
* Added sample use cases
* 20.10.2011 – Spell-check and comments from DFKI and field-test experiences
* Renaming of “iGreen Node” to “Device Node” as a outcome of conference from 21.07.2011
* Changed some details in the replication section as a result from the first implementations and field tests. The /status db is replicated now only one way and there is always a replication to the root server.
* Added a note to the security section to add manual the \_security document to each database.
* Real-time status information is not using PGN’s anymore, now the SPN’s are used in decimal format.
* The data type of real-time status information is now defined more precisely to use the raw integer value.
* The specification defines now where to use upper and lower case strings.
* The specification defines now more precise how to store timestamps in the UTC format.
* New configuration parameters for CouchDB are mandatory.
* Added a motivation and overview section
* Moved sample configurations and DDI list to SVN, updated links
* 20.10.2011 – Spell-check and comments
* 18.01.2012 – Detailed specification of URL usage
* Updated section 2.5.3. to specify the declaration of URL’s

1. Specification

# General

The iGreen Machine Connector team has decided in the telephone conference at the 24.02.2011 to use a REST interface based communication interface. This technology allows to communicate with an OnlineBox directly as well as with an Apache CouchDB installed on the terminal. CouchDB allows having an offline cache which is synchronized bidirectionally with other terminals and the OnlineBox.

The Machine Connector is based heavily on CouchDB. Identifiers (such as database names) explained in the following are, thus, subject to CouchDB constraints with regard to charset support and length limitations.

# Addressing Device Nodes

For the transfer from machine data from one machine to the other and to store files in a global data store it is required to have an addressing and security schema defined. Every Device Node has a unique identifier in the network. The addressing scheme allows to create access control lists and configuration files which include a filtering (semi security layer), by skipping the last characters.

The basic addressing is given by the iGreenID: **[TLD]/[Customer Name]/[Node Name]**  
The TLD represents the top level domain name from the country where the customer is located. The Customer Name needs to be a unique name in the TLD area. To get a unique Customer Name, it is recommended to use a concatenation of ZIP code and farm name. Those strings must not include a slash “/”, so only three slashes allowed in the iGreenID. It is also possible to use a registered domain name which is unique any way. In the iGreen community, a name validation service should be established.

The Node Name represents a unique identifier in the scope of [TLD]/[Customer Name]. This name starts with the manufacturer name. Then it is followed by the serial number or the license plate or any other unique identifier. It is possible to represent with the Node Name an identifier for any other Device Node than a vehicle - e.g., a PC with a Desktop Software, or a web portal.

Every iGreen Machine Connector node has to provide a configuration set which is stored in the /config CouchDB database related to each iGreenID: **[TLD]/[Customer Name]/[Node Name]/config**

Here are some examples for the addressing schema:

iGreenID of a John Deere tractor at a customer: de/66482sonnenhof/jd6930  
its configuration file will be stored in: de/66482sonnenhof/jd6930/config

This customer runs a self hosted farm server with an OnlineBox and a Machine Connector installed on it, which is reachable through the following iGreenID: de/66482sonnenhof /server1

To enable an easy management of access control lists, the addresses could be filtered like in the following example which means that all Device Nodes from the de/66482sonnenhof are covered: de/66482sonnenhof~~/jd6930/config~~

# System Architecture

The system architecture defines the communication components and connections between instances. The following drawing shows how a machine connector is designed internally as well as how a node can interact with other Machine Connector Nodes.

iGreen_MachineConnector_Spec_JD_V2.emf

Figure Components of a Machine Connector

The Machine Connector Client is integrated into the Machine Terminal or implemented as a separate hardware box, depending on the manufacturers. This Machine Connector has an interface to the ISOBUS Task Controller which is running on the vehicle, for example to provide the total data which is processed out of the ISOBUS communication from the vehicle. In addition to that interface, there is a CAN-Bus connector which is directly connected with the ISOBUS and optional with additional proprietary vehicle bus systems. This interface provides real-time status information which does not need to be processed by the ISOBUS Task Controller (CAN messages raw data.)

The Machine Connector Daemon is a piece of software with some algorithms to connect the interfaces to the local data store and to deploy the configuration of the Device Node. The functionality is described in further detail in the section “Machine Connector algorithms”.

If the hardware or architecture does not allow a local instance of a CouchDB, it is also possible to connect with the Machine Connector Client directly with a HTTP/REST connection to a remote CouchDB. This requires a permanent connection to the remote server which has the following limitations: It is not possible to work offline if the wireless connection is broken, and it is not possible to access or store data from other machines.

This specification does not claim to define the wireless communication links because there are manufacturer specific implementations and limitations which cannot be solved in the iGreen project scope. This specification requires a TCP/IP connection link and requires at least a reachable IP address or network name of the Machine Connector Server node. The assignment of IP addresses, SSIDs, and encryption keys is implemented proprietarily by manufactures.

The Machine Connector Server has an interface to an OnlineBox which could be installed local on the same hardware or in a remote instance. The OnlineBox interface is used to enable data exchange to Device Nodes which are not a part of the Machine Connector network. For example this is a service of an agriculture consulting company.

*Note: Not only vehicles could implement a Machine Connector setup or connect directly to the Machine Connector Server. It is also possible and appreciated if mobile devices, smart phones or desktop software implement this specification and use the decentralized data store.*

# Software Architecture

This section describes which software is used and how it is configured to implement the above described system architecture.

## Apache CouchDB

The concept uses as a local and remote data store the Apache CouchDB. It allows creating a decentralized and synchronized data store with mobile devices as well as with scalable server infrastructures. The internal data format is the JSON format, which allows to serialize object structures and to store and convert XML data structures, which are used as the communication standard between machine terminals. The advantage of this type of database system is that it does not require defining the data structure from the beginning. With the NoSQL type of databases it is possible to extend the data store with new requirements out of the agriculture development.

This specification is based on Apache CouchDB version 1.1.0. The hosting operating system is not specified. The CouchDB must be reachable on port 5984 or 6984 (when SSL is used) without a proxy configuration. The CouchDB must be configured as followed:

The value in the CouchDB configuration “attachments🡪compressible\_types” must set to the following value: “text/\*, application/javascript, application/json, application/xml, application/octet-stream“. This enables automatic compression of XML and BIN attachments.

The value in the CouchDB configuration “couch\_httpd\_auth🡪 require\_valid\_user” must set to “true”. This prevents non iGreen users from accessing the database from the public Internet. Please follow also section 2.4.2.3. to setup basic authentication.

The value in the CouchDB configuration “httpd🡪bind\_address” must set to the following value: “0.0.0.0”. This configures CouchDB to bind to all network interfaces (in most cases the IP address is not known because of DHCP).

The value in the CouchDB configuration “couchdb🡪delayed\_commits” has to be set as “false” on all machine connectors which are used as mobile versions (e.g. on a vehicle, smart phone, etc.). Only servers where a constant power supply is ensured could use the option “delayed\_commits=true” to increase the performance with the risk of lost data and inconsistency when the server or CouchDB is crashing.

## Security Architecture

In the iGreen Machine Connector setup, several security aspects are relevant. This is necessary in terms of data ownership, data privacy, protect against hackers and to ensure that manufacture specific features are protected to create additional value on the iGreen network.

The following aspects need to be addressed:

* Protection from external malicious access. Any party that cannot authenticate should see as little of the Machine Connector interface as possible in order to minimize potential attack surface.
* Prevention of data leaks. Especially unauthorized copying of data must be prevented.
* Protection against white box attacks: Even if a malicious party gains knowledge of the Machine Connector design, it must not be possible to attack existing Machine Connector installations.

The current version of the Machine Connector specification addresses these concerns only partially and will be extended in the future.

### ACLs / Certification and Validation

There are several security aspects in the setup of a Machine Connector. The relation of an iGreenNode to another iGreenNode is described in the section “Data types and structures – Configuration data”. It is possible to configure synchronization (replication) lists and Access Control Lists (ACLs). Enforcement of ACLs is implemented on the Machine Connector level, not on the CouchDB level. To ensure that the implementation of the Machine Connector specification does not ignore access control lists, it is proposed that an independent organization gives a “iGreen Machine Connector” certificate like it is done with ISOBUS controllers of the DLG today.

### SSL Encryption

To ensure that the TCP/IP connection over the public internet is secured and no data is logged by attackers, the communication link HTTP/REST allows SSL encryption in the communication. The communication over the public Internet (especially to the root nodes) has to use SSL encryption which is supported by CouchDB. For communication between local wireless networks (which feature separate encryption), standard HTTP communication is allowed.

Possible future extension: Using SSL encryption allows using certificates for authentication of peers. This way, only peers that can present a trusted certificate can be allowed to connect to CouchDB/the Machine Connector which will give better security than the approach currently in use (see below).

### Basic Authentication

*Note: The default password approach outlined below is a simplification for the development phase only and not intended for productive use. The default password is not to be published more than needed. In future, this password will be replaced by GUID, a more complex passphrase, or SSL certificates.*

To protect the network and the Machine Connector against simple hacking attacks and against users with medium experience in CouchDB a default administrator account is set. This mechanism is not used to replace a SSL encryption or to protect single databases or documents in the iGreen network. It is also not used to control or replace the addressing mechanism which is used for the iGreen Machine Connector.

The default root user has to be configured as:  
username: admin  
password: iGreenMC

For every database the security options has to be configured like this:  
Admins 🡪 Names: ["admin"]  
Admins 🡪 Roles: ["admin"]  
Readers 🡪 Names: ["admin"]  
Readers 🡪 Roles: ["admin"]

*Attention: Depending on the used API CouchDB is not generating the \_security document for a new database. Also the replication mechanism is not replicating the \_security document. There is no inheritance of security settings and by default a new database is not protected. Due to this fact the iGreen Machine Connector implementation needs to take care on generating proper \_security documents for each database.*

### Encryption of single documents

CouchDB features only very limited built-in access control. In particular, all data stored in one CouchDB instance can be read by any peer authenticated with it. However, in some use cases the machine manufactures would like to store and exchange machine status information which are proprietary and should not be public or accessed by other machines for several reasons. Because the iGreen Machine Connector allows configuring mixed fleets where the data is available for every default user or when a machine is carrying only the data from one point to another, it might be necessary to have data encryption for every document in the database.

To enable this feature, an asymmetric encryption mechanism is used where the public key allows every Machine Connector to encrypt specific data for a certain group of readers (owners of the private key). The owners of the private key are for example each machine manufacturer or a consultant. With a mix of public keys, multiple readers could be enabled to read the data.

Currently, the best document encryption mechanism is discussed with some developers to identify if this feature becomes a basic feature of CouchDB or if it is implemented on the application side which means that this feature will be a part of the Machine Connector specification. This will be specified in more detail in one of the next revisions of this specification.

## OnlineBox connection

The Machine Connector network is able to run without an OnlineBox, but to enable the data exchange with other nodes in the iGreen cloud the Machine Connector features an interface which synchronizes the data into the OnlineBox. The algorithm to do this is described in the section “Machine Connector Algorithms”. The Machine Connector and the OnlineBox could be hosted on the same server to reduce the complexity of the setup. Where this “Machine Connector server” is hosted is up to the customer or the machine manufacturer that might provide an integrated solution. It is also depending on the legal aspects of data ownership which is also related to the country where this setup is used.

# Data types and structures

The following sections defining which data types are available. For each data type and iGreenID, a single database exists which allows having a dedicated replication of these data types.

There is a database for each Device Node with the name: iGreenID/proprietary which allows storing and replicating manufacture specific data. The format of this database is not defined more detailed.

All timestamps which are stored or being part of a document ID are in the UTC-0 time format which standardized. This means the time zone of a device needs to calculated local on the stored values. There are two ways defined for a UTC string “1985-04-12T23:20:50” or “1985-04-12 23:20:50”. All time stamps which are stored in the JSON format must use the version with the space “ “. All timestamps which are part of a document ID must use the version with the “T” separator in upper case, because CouchDB don’t support spaces in document IDs.

## Real time data

The real-time data is used to share machine status information and messages between the nodes. For example, the current machine position, the current fuel consumption or the current moisture. Real-time means in this case that the data is updated on change, with a threshold or after a certain time to reduce the update and replication cycles. For example in the case of the current machine position it would make sense to write a new machine position into CouchDB when the GPS course has changed more than 15° and every minute.

In future there might be some additional data structures for real-time status information like a messaging protocol, this is currently not specified.

The real-time data is stored in a separate database for each iGreenID in the following format: [iGreenID]/status for example: de/jdcustomer/jd6920/status

The real-time data is stored in a format according to the ISOBUS and J1939 specification. If the status data is available as a CAN message a new document with the standardized SPN (decimal number) is created in the format: can\_spn\_[SPN] for example: can\_spn\_584. The interpretation in terms of offset, scaling and range of the data is defined in the above referenced standard documents. In the document is a field called “value” where the INTEGER value as decimal number (without scaling and offset) of the according SPN is stored, like the following example shows:

{

"\_id": "can\_spn\_584",

"\_rev": "5389-979846b8fedac4538032f4a1c34c43b4",

"value": 2595016910,

"time": "2011-05-26 23:20:50", /\* UTC-0 Date and Time from GPS \*/

"history\_pointer": 34, /\* Number of the document with the last value \*/

"confidence": 90, /\* If available: confidence level from sensor in % \*/

"accuracy": [ /\* If available: accuracy from sensor in percent % \*/

-3,

5

]

}

If the real-time status data is defined in the ISOBUS part 11 as DDI it is stored in the following structure in the /status database. There is a document created with the id: ddi\_x\_[ISONAME]\_[DET]\_[DDI] (the x indicates that the numbers are in hexadecimal format) for example: ddi\_x\_A00484000430F447\_2\_e1ff in this document is a filed named “value” which contains the INTEGER value as decimal number (without scaling and offset) like defined for the DDI in ISO 11783-11. The concatenation of the working set master with the device element numbers from the object pool ensures that the DDI is unique also if there are e.g. multiple implements connected. In addition to that there is a DDI list in the appendix of this specification available with proprietary values which are currently not supported by the ISO standard. Here is an example JSON file to store a DDI value with the machine connector:

{

"\_id": "ddi\_x\_A00484000430F447\_2\_e1ff",

"\_rev": "2-c4d9cdebb2fed10b516a0c01da5dfc60",

"value": 345,

"time": "2011-05-26 23:20:50", /\* UTC-0 Date and Time from GPS \*/

"history\_pointer": 34, /\* Number of the document with the last value \*/

"confidence": 90, /\* If available: confidence level from sensor in % \*/

"accuracy": [ /\* If available: accuracy from sensor in percent % \*/

-3,

5

]

}

### History of data

To store a history about the real-time data it is not possible to use the internal revisions of CouchDB, because they are used only for the conflict-free concurrent updates of documents and are not available after a successful replication and compaction of a database. Therefore it is necessary to have a history implementation specified in the application:

*Note: To store a history and how long a history is stored is up to the implementation of each manufacture. Due to resource limitations it is not possible to ensure that every machine connector stores the last 100 values for every machine in the fleet.*

If the history should be stored by the application, the document for that value has a filed called: “history\_pointer” with an integer value with the number of the document with the last value. This means that in the document with the id “ddi\_x\_” or “can\_x\_” always the latest value is stored. If a value update occurs the Machine Connector client creates a new document with an id ending with the increment of the history\_pointer and modifies the history\_Pointer in the main document. The application needs to ensure that the following operations are not running in parallel to ensure that the data is always consistent: reading the last history\_pointer, adding a new history document and the update of a master document. Because this data is not modified on other nodes than the node where the data belongs to it is not necessary to prevent atomic operations over the decentralized network and the application itself needs to take care of the protection of these functions.

This is similar to a liked list structure and allows iterating with the application over all versions. The application has to do a clean up with a manufacture specific parameter that e.g. only the last 50 values are stored by deleting the first documents.

## Documentation data / Task data

This data type contains documentation data or in ISOBUS terms Task data. All data which belongs to the processing and documentation of tasks like defined in the ISO 11783-10 Standard will be stored in this kind of database for each iGreenID. All ISOXML files which are stored in the Machine Connector need to be validated with the .XSD schema of the VDMA or AEF from 2009. The standard defines that the ISOXML files have the following states: initial, in process, paused and finished, the status is stored in the taskdata.xml file. All states are supported in this database structure. This database is not used to store and distribute status information there for is the database /status used.

*Note: CouchDB is case sensitive, also when the algorithm is running under windows operating system. Due to that all files which are stored as attachment to a document has to be in lower case and the name of the file in the document ID as well, also when the “T” of the UTC timestamp is in uppercase.*

Each iGreen Machine Connector stores the task data in the database with the name of the iGreenID/isoxml.

The standard defines as well that an ISOXML file might have sub files which are referenced by the taskdata.xml file. Because the replication algorithm of the CouchDB does not merge data in a document for each file a separate document is created in the database. The document name (id) defines which files belong together.

The document id has the following format: tc[YYYY-mm-ddThh:mm:ss]\_[filename] the time stamp is in the UTC-0 format to ensure that the document is unique for each iGreenID. For the UTC time format are two versions available, one which fills the space with a “T”; in this case the “T” is used instead of the “ “ because CouchDB doen’t allow document IDs with a space.

For example tc2011-10-20T19:24:31\_taskdata.xml.

To search for files which are belong together the machine connector software has to look only on the start sequence of an id.

Documentation and task data is only deleted by the FIMS or the Machine Connector Server user interface when they have ensured that the task data is stored to the long term documentation and not required anymore on the vehicle for the field work.

This is an example JSON document how to store the taskdata.xml:

{

"\_id": " tc2011-10-20T19:24:31\_taskdata.xml",

"\_rev": "3-ec53dedfce34acfb6d5d7eefbc10d99c",

"\_attachments": {

"taskdata.xml": {

"content\_type": "text/xml",

"revpos": 2,

"length": 661,

"stub": true

}

}

}

This is an example JSON document how to store the TLG00000.XML:

{

"\_id": " tc2011-10-20T19:24:31\_tlg00000.xml",

"\_rev": "3-38b77836a71912cd2cfaef3cef589545",

"\_attachments": {

"tlg00000.xml": {

"content\_type": "text/xml",

"revpos": 2,

"length": 80,

"stub": true

}

}

}

This is an example JSON document how to store the TLG00000.BIN:

{

"\_id": " tc2011-10-20T19:24:31\_tlg00000.bin",

"\_rev": "4-8816750abdff8d34164f831fdcecde5e",

"\_attachments": {

"tlg00000.bin": {

"content\_type": "application/octet-stream",

"revpos": 2,

"length": 6900,

"stub": true

}

}

}

## Configuration data

The configuration data defines the settings for the iGreen Machine Connector, like the username, password and URL to the Device Node. In this configuration document a list with the addresses and names of the other iGreen Machine Connector nodes is stored where is defined with which other nodes they are synchronized with the CouchDB replication mechanism. An advantage is that the network configuration (fleet configuration) is automatically deployed over all connected CouchDB which requires that the iGreen Machine Connector software starts the replication of the configuration database by default when any node is available.

Every iGreen Machine Connector node has to provide a configuration set which is stored in the /config database related to each iGreenID: [TLD]/[Customer Name]/[Node Name]/config

This database is replicated by default with every node which is in that domain. Further details to that algorithm are described in the section of “Machine Connector Algorithms”.

In this database a document with the id and name “igreen\_machineconnector” has to exist which stores the configuration. It is allowed to add more documents in this database for proprietary configuration data. Please keep in mind that all the date in this database is automatically deployed in the network. The document igreen\_machineconnector a structure which is defined by the following example JSON document and its notes (please remember that comments are not allowed in JSON, which not allows to copy and paste the code shown here).

All stored addresses to nodes need to be declared as full qualified URL’s (for reference please see: RFC 2396, RFC 2732) beginning with the protocol, including the port number and ending with a “/” splash. This is necessary to be compatible with several server configurations, SSL encryption and servers behind a proxy server. These are examples for allowed URL’s:

<http://igreen.couchone.com:5984/>

<http://demonstrator.igreen-projekt.de:5984/couchdb/>

<https://igreen.deere.com:6984/>

{

"\_id": "igreen\_machineconnector",

"\_rev": "19-39db0290779b880616dd08a7d2517dad", /\* Revision is auto generated by CouchDB \*/

"readaccess": [

/\* List with iGreenID’s which are allowed to read data from the nodes. Do not duplicate them on the write access list which includes a read access.

You can also cut of the last part of the iGreenID to grant the complete domain area read access to it.

\*/

"de/kronecustomer/kr3442",

"de/consulting23"

],

"writeaccess": [

/\* List with iGreenID’s which are allowed to write and read data from the nodes.

Do not duplicate them on the read access list which includes a write access.

You can also cut of the last part of the iGreenID to grant the complete domain area write access to it.

\*/

"de/kronecustomer/kr3444",

"de/consultingdlr"

],

"onlinebox": {

/\* This section is optional if in this implementation of the machine connector the interface module for the OnlineBox is existing. \*/

"username": "meyeraxel@johndeere.com",

"password": "test16",

"url": "http://onlinebox15.dfki.uni-kl.de:80/"

},

"syncnodes": [

/\* List with iGreenID’s with which other MachineConnector CouchDB’s the algorithms should try to start the replication of the data. This list has no impact on read and write access which allows to use a machine connector as a transportation data store (e.g. a machine connector on a tractor with Bluetooth connection to a combine and the server at the farm yard to allow a store and forward data transport.) \*/

"de/kronecustomer/kr3442",

"de/jdtestcustomer/rootnode", /\* The root node has to be in the sync list \*/

"de/kronecustomer/kr3444",

"de/jdtestcustomer/jd6920s" /\* The iGreenID of this node has to be in the synclist as well to enable a remote configuration and replication to the root node \*/

],

"machineconnector": {

"igreenid": "de/jdtestcustomer/jd6920s",//iGreenID of this node

"rootnode": "de/jdtestcustomer/server1"

/\* Next reachable root in this domain, this node needs to be direct reachable (e.g. fixed IP which has to be in the addresstable). This node deploys the initial configuration data of the network and the application data from the other nodes. \*/

},

"version": 2, // Version number for later compatibility check

"addresstable": [

/\* This array is the address translation table for iGreenID’s which are directly reachable from this node. The rootnode address is mandatory. This address list needs to be updated by the manufacture specific implementation of the machine connector algorithms. \*/

{

"igreennode": "de/jdtestcustomer/rootnode",

"address": "http://jdtestcustomer.dyndns.org:5984/"

},

{

"igreennode": "de/kronecustomer/kr3442",

"address": "http://kr3442\_vehicle.dyndns.org:5984/"

},

{

"igreennode": "de/kronecustomer/kr3444",

"address": "http://192.168.100.23:5984/"

}

]

}

The Machine Connector algorithm needs to store local only the iGreenID of this machine connector, the URL to the root node and the iGreenID of the root node to be configured automatically. For an easier administration and support these three values should be stored in addition to the local file system into the configuration document like shown above.

# Machine Connector algorithms

The specification describes a client and a server algorithm which are very similar form the implementation. The difference is that the Client is only able to communicate to another client or a server which means which allows only a communication between CouchDB servers. The server version is the same like the client but it has an optional module which is able to synchronize the CouchDB database with the DFKI OnlineBox.

It is up to the implementation which programming language is used for the machine connector. For the CouchDB are several API’s for different programming languages available. A selection criterion for an API is that they need to support the following features:

* Changes interface
* Support of attachments
* Support and control of the replication interface
* Compaction interface

*Note: All CouchDB functions could be replaced or implemented with manual REST commands, but it makes the implementation of the application easier to use an API for it.*

## Algorithm for Machine Connector Client

This section describes the algorithms and functions which are required on every machine connector. This piece of software can be implemented as a windows service or a demon or just as a piece of another software which is running on the machine connector operating system. Many functions are related to the CouchDB API.

### Initialization

If the software starts the first time it is necessary to enter the name of the iGreenID for this machine connector. The second value which needs to be stored local is the TCP/IP address of the root node of this iGreen Machine Connector. When the values are stored in the local file system the algorithm first tries to connect to the root node and replicate a configuration from this node to the local database. If there is no pre-configured configuration available the algorithm creates an empty iGreenID/config database.

The initialization function checks as well if the two other databases iGreenID/isoxml, iGreenID/ proprietary and iGreenID/status are existing, if not they are created.

After the initialization it should start the replication functions.

### Replication

This function needs to ensure that the internal replication mechanism of the CouchDB is configured and started properly. Since version 1.1.0 of CouchDB the databases which have to be replicated are configured with the system database “\_replicator”.

The Machine Connector application has to setup continues replications for each entry in the “syncnodes” list. The algorithm checks if there is a direct connection to the specific node in the “addresstable” available. If there is a direct link the replication configuration is set additional to the direct link (IP address or DNS name) and to the rootnode. The replication is set always in both directions (switching source and target), excepting the /status database of the local Device Node which is only replicated from the source to the target. This allows a replication via push mechanism to nodes which are not directly reachable (behind a NAT). This is for example the case if the rootnode like to send data to the via cellphone connected tractor which is behind a NAT, but it is possible because the tractor keeps the connection open and not the rootnode. For each iGreenID there are four replications installed: /config, /status, /isoxml, /proprietary. The following example shows how this looks:

Replication from the rootnode to the Device Node:

{

"\_id": "de/jdtest/fzxsrv09-de/jdtest/fzxsrv09/config",

/\* The name of the replication document in this direction is set as [nodename]-[nodename]/[databasetyp] \*/

"\_rev": "21-bb4ad841d7c4c4cb2c7a3b5cb1e59f95",

"source": "http://admin:iGreenMC@gfzweapp1.dpn.r2.deere.com:5984/de%2Fjdtest%2Ffzxsrv09%2Fconfig",

/\* The URL needs to replace the slash by %2F and include the username and password to get the basic auth. \*/

"continuous": true,

"create\_target": true,

"target": "de/jdtest/fzxsrv09/config", // This is the local target

"user\_ctx": { // This object is required to get local permissions

"roles": [

"\_admin"

]

}

}

Replication from the Device Node to the rootnode:

{

"\_id": "de/jdtest/fzxsrv09/config-de/jdtest/fzxsrv09",

/\* The name of the replication document in this direction is set as [nodename]/[databasetyp]-[nodename] \*/

"\_rev": "20-426068805e2e5832a667f8190fdb3c5c",

"source": "de/jdtest/fzxsrv09/config", // This is the local source

"continuous": true,

"create\_target": true,

"target": "http://admin:iGreenMC@gfzweapp1.dpn.r2.deere.com:5984/de%2Fjdtest%2Ffzxsrv09%2Fconfig",

/\* The URL needs to replace the slash by %2F and include the username and password to get the basic auth. \*/

"user\_ctx": { // This object is required to get local permissions

"roles": [

"\_admin"

]

}

}

It is important that always the /config database of all nodes in “syncnode” table are replicated and then in a second iteration the other databases. This ensures that with instable wireless connections always the latest IP addresses of mobile devices are deployed.

*Note: In upcoming versions there might be a mechanism which allows configuring a dependency of the replication of databases. Currently the only solution to trigger a preferred replication is to remove and insert again the field for the replication of the /config database. If this is necessary needs to be confirmed during filed testing.*

Normally the replicator of CouchDB takes automatically care on the continues replication and to resume after a restart of the CouchDB or after a broken network connection. The default algorithm does a retry after an always doubled timeout and has a maximum retry threshold. This works for normal short network breakdowns well, but if a M2M communication is not available for several hours it will not recognize when a new connection is available. The network protocol stack of the Machine Connector needs to implement a function like IP-up and needs to remove and insert this specific replication again to start an immediate replication.

After the initial replication is configured like described above, the algorithm needs to watch for changes in the iGreenID/config database to ensure that if the configuration has changed (new IP address of a directly reachable node or a new syncnode). If the config has changed the \_replicator database needs to be updated.

A challenge is to keep the \_replicator database clean, there for replication documents of deleted Device Nodes need to be removed as well as replication to nodes which are not available any more (IP address has changed). Also it is possible that other applications are using the local CouchDB instance, where it is unknown if the replication document was created by the Machine Connector algorithm. The cleanup function is highly dependent on the individual Machine Connector implementation and the API which is used; this is the reason why in this document no algorithm is specified for this purpose.

#### Replication conflicts

The CouchDB replication algorithm tries automatically to solve conflicts which might exist due to the decentralized databases. A conflict is exists when two databases which are in replication losing their TCP/IP connection and on both databases are changes made in the same document. Another scenario is that the Machine Connector software has a bug or is shutting down and the replication is not triggered anymore; if there is made a change in both documents after that the documents are in a conflicting state. The conflict solving algorithm is deterministic which ensures that independent (not connected) databases getting the same result by choosing the version with the longest history. For the most applications this algorithm might be the best, but in this use case for the machine connector the conflict solving need to be changed a bit, because in this specific use case the assumption is made, that the database on the machine knows the latest changes.

If CouchDB has solved a conflict, the document is marked with the entry: "\_conflicts":true. The application is informed about that with the \_changes feed. When such automatic solved conflict is reported, the Machine Connector algorithm, must ensure that all changes from its own iGreenID (this, localhost) getting the winning version, which does not include that it has the most revisions. To know which one is the local revision the algorithm needs to step backwards through the last revisions and check which is the latest document without the \_conflicts tag. This revision is then manually set as the winning revision by writing the document without a conflict to the latest version and deleting the \_conflicts tag. Then the revisions with the conflicts tag are deleted and the compaction function needs to be called.

The other nodes for example the root server is informed about the solved conflict over the changes interface. This is useful in the following example: On the farm server the manager as modified a task which is already processed in the field and the Machine Connector is offline during this time. When the Machine Connector solves this conflict the farm manger would appreciate to be informed that his change request was not delivered and that the task was processed with the old description.

*Note: A synchronized timestamp would be available because the machine connector usually has GPS or an internet connection. But to use the timestamp in the algorithm would increase the complexity and data volume and limit the usage to vehicles which ensures a valid timestamp. The main reason to not use the latest change is that the use case could cause that another node than the this-iGreenID has made the latest change. Assuming that the this-iGreenID vehicle knows always the own status at best, let it win.*

### Updating address list

Like specified in the main configuration there is an ”addresstable” available which stores the direct TCP/IP connection to the iGreenID’s. If there is a direct connection available the replication algorithm like described in the above section is synchronizing the databases directly. It is important that the Machine Connector algorithm is updating always the latest IP addresses to the addresstable. This requires that the software has interfaces to the wireless communication stacks of the operating system.

Please note that if the IP address configuration has changed and the host becomes available again the replication has to be triggered manually like described in section 2.6.1.2.

For direct connections which are always reachable (e.g. a CouchDB Server from a service contractor with dyndns address) it is useful if the Machine Connector provides a user interface on the machine or on the farm server to allow the user to configure the address table manually. This allows connections bypassing the tree structure which requires a reachable root node.

### Compaction

The CouchDB stores by default old revisions of documents which is required for the replication of offline databases. This causes that the data volume of the database is increasing by the time. Assuming that the Machine Connector nodes are synchronized by the time it would make sense to have a garbage collector running. CouchDB provides the integrated function \_compact to delete unused sections and revisions which have been already synchronized.

It is recommended that the CouchDB algorithm calls the \_compact function 10 minutes after the CouchDB was started. Then the system performance is better than after an immediate compaction after the startup. But it is up to the manufacture implementation how often this is done or if it is done periodically because it is depending on the CPU power and the available disk space of the client device how often the compaction is required and useful.

### Thin Clients without CouchDB

There might be some thin clients who do not allow installing a CouchDB on the device, but they are fitting as well in the Machine Connector architecture. In this case it is required to have a permanent internet connection available, if this connection is broken down there are some limitations like no offline availability of the data.

In the case of a thin client without the CouchDB the architecture is very similar to that what is defined in this document but the client do not need to take care of the replication, compaction and conflict solving. The client software needs to know like with the standard initialization function the iGreenID, the password to the root CouchDB and the iGreenID and URL to the root node. Then the software is connecting and working directly with the root node which takes care of all the other algorithms related to replication. Data is read and written directly from and to the remote CouchDB on the root node.

### Maintaining the history of real-time status information

Maintaining the history of real-time status information means that the Machine Connector does a cleanup of the history documents depending on the resources of the Machine Connector and the application needs. In some cases the last 50 values are required in other cases it make sense to ensure that the data from the last hour is available. Also this function needs to ensure that if the history is enabled and a value update occurs a document with the last history is stored and the history\_pointer is set right. How the linked list of history versions is working is specified in section 2.5.1.1.

This function also needs to provide the history data to the application e.g. as a view or design document. Because this is really depending on the application what is done with the data this is not described more detailed in this specification.

## Algorithm for Machine Connector Server

The algorithms on the Machine Connector Server are similar to the functions on the client because the server has the same functionality of a client. But the server has some further features which might be required depending on the usage of the server. Usually the server has more disk space and CPU power which allows the server to store and provide the data of the whole fleet or domain or of multiple domains.

The Server has its own iGreenID in the domain and also the /config database which tells the algorithms (which are specified in the client section) with which nodes it should replicate. Other databases as an extension of the iGreenID are allowed to provide advanced features on the server. These might be CouchApps with a configuration user interface or some proprietary databases.

### Configuration User Interface

This is an optional component which runs on the Machine Connector Server which allows the user to have a centralized configuration tool for each Device Node and the fleet configuration. This central configuration is possible because the Machine Connector Server has a replication of all /config databases in his network which is automatically deployed.

Because this function is depending on the product portfolio and strategic planning of each manufacture the implementation and integration into existing platforms is not specified in this document.

### OnlineBox interface

The Machine Connector architecture allows running the network independent from the OnlineBox. To connect to iGreen partners who are not connect to the Machine Connector network there is an optional interface which synchronizes the task data into the OnlineBox.

Currently the OnlineBox does not allow to store fleet configurations and to associate data files to a specific machine. The association of ISOXML files to a machine is stored in the ISOXML file which requires changing the ISOXML file when the association of this task is changed. Also the identifiers in the ISOXML document are only unique within one task set and not over the complete network. This algorithm needs to poll for changes in the OnlineBox and synchronize them with the CouchDB data structure.

Because the interfaces might change in the next time this feature is not specified more detailed.

# Outlook to future revisions

Currently not covered by this specification are the following topics. They are not mandatory for the basic functionality of a machine connector and a proof of concept for this setup:

* Encryption for single documents with several key combinations. This allows to share data between different ownership spheres (e.g. Krone and John Deere data or between customers)
* Sharing and connecting of wireless networks (they are given as connected)
* Deployment of software updates and version control (currently solved proprietary)
* Converting ISOXML files into JSON format to allow to query to the file and to connect ontology directly to CouchDB. This requires a CouchDB version which supports direct compression of JSON objects.
* iGreenID verification service to ensure that the ID is unique in the network.
* Messaging service for Machine Connector Clients.

1. Appendix

The following Excel Spreadsheet is a list with new DDI’s which are not covered by the current ISO11783-11 standard for “Data Dictionary Identifiers”. In the iGreen project these values are specified in this document, and it need to be discussed in the consortium if they are proposed to the VDMA / AEF standardization committee. In this document only additional DDI’s are defined because the ISO 11783-11 is a living standard.

<https://igreen.opendfki.de/browser/trunk/AP-Material/AP6000/AG_Machine_Connector/ISO11783-11_iGreen_Proposal.xlsx>

1. Sample Configuration

The following drawing shows a sample configuration of the iGreen Machine Connector like specified in this document. The following zip file contains the var\lib\ folder of the CouchDB with the sample data to allow developers a quick start. This database is created with version 1.1.0 of Apache CouchDB.

<https://igreen.opendfki.de/browser/trunk/AP-Material/AP6000/AG_Machine_Connector/samples/couchdb_var_lib_v2f.zip>

This sample configuration is also online available, please do not modify this database or use it for production systems because this server is not supported official.

<http://igreenmc.couchone.com:5984/>

<http://igreenmc.couchone.com:5984/_utils/>

1. Use Cases

This chapter describes some example use cases which are related to the sample configuration in this specification.

# Use Case 1

iGreen_MachineConnector_Spec_JD_V2_case12.emf

The manager of the service contractor generates in the field with the customer a new task description with his mobile device. The task is assigned to the tractor cl7562 from the contractor. At this time when he creates the task he has no cell phone connection because the file is off site. During the drive to the next customer the mobile device replicates the task to the farm server of the service contractor. The task is transferred at the farm yard with WiFi on the tractor because the tractor doese not have a cell phone connection due to cost savings. In the evening when the tractor is on the farm yard again the documentation data is transferred on the farm server.

# Use Case 2

iGreen_MachineConnector_Spec_JD_V2_case12.emf

This use case is similar to the first one. When the manager has created the job with his mobile device he meets the tractor driver on the way to the next customer. Both don’t have cell phone cenectivity but the mobile device send via Bluetooth the task description to the tractor. The tractor is operating the complete day the task without a wireless connectivity. In this time the mobile device was able to send the task also to the farm server. In the evening the tractor replicates the documentation data of this job to the farm server where the collected data is merged.

# Use Case 3

iGreen_MachineConnector_Spec_JD_V2_case3.emf

The forage harvester has no cell phone connection at all, only WiFi connectivity on the farm yard and other fleet members. In the morning the SPFH gets an initial set of tasks for the day via WiFi on the farm yard. During the day the trailer tractors and the SPFH having a replication via WiFi. This enables the tractors to carry the data from the field to the farm server and the other way to send new tasks to the SPFH. On the farm server and the tractors the GPS position and machine status (fuel level) is shown. This enables the trailer tractor drivers to find the SPFH in the field.

# Use Case 4

iGreen_MachineConnector_Spec_JD_V2_case4.emf

This use case is a more simple one but similar to 3. All vehicles have a direct cell phone connection to the farm server which allows delivering the documentation and machine data much faster. When the cell phone connection breaks down it is possible to continue working because everything is replicated to the local CouchDB.

1. Reference Implementation

According to this specification the iGreen project member John Deere provides a reference implementation to get a better understanding of the Machine Connector, to enable a proof of concept and to enable other partners a fast entry in this technology.

John Deere allows using and modifying the source code and binaries but is not responsible for this software with all rights are reserved.

***Es entstehen keine Rechte/Ansprüche gegenüber den Projektpartnern oder Dritten, die Partner dürfen diese Software nicht als Ihre Arbeit darstellen oder anbieten.***

This disclaimer and legal regulation is valid for all software and versions which are provided by John Deere. The current version is located on the iGreen DFKI SVN: <https://igreen.opendfki.de/browser/trunk/AP-Material/AP6000/AG_Machine_Connector/implementations/iGreenMachineConnector_JD_referenceImplementation>.

The reference implementation is written with Java using the Ektorp API ([www.ektorp.org](http://www.ektorp.org)) to access the CouchDB. The project is created with Netbeans 7.0 ([www.netbeans.org](http://www.netbeans.org)).

In the implementation a minimal set of functions for the server and client are implemented. The main()-Function requires the following three parameters:

1. iGreenID of this instance (e.g. de/jdtest/fzxsrv09)
2. The CouchDB URL of this instance (in most cases localhost)
3. The URL to the next root node which is reachable (e.g. igreen.deere.com)

The main()-Function creates a thread for the core services like replication and initialization and a second thread for the compaction. Other threads may follow when the history maintenance or the OnlineBox connector is developed.

The representation of the JSON documents is done in the models package. For each JSON representation a Support Repository needs to be created.

The core thread implements in the initConfigDB()-Function the section 2.6.1.1. of this specification. The updateReplicator()-Function implements the section 2.6.1.2.

Sample databases are located in section 4 of this specification.

1. Rusch,C. Meyer,H.J. Geimer,M. Scherer,M. Herlitzius,T. Engel,T. Ostermeier,R. Fechteler,T. Kritzner,A.: Autonomous Agriculture Documentation System for the Retracement of Harvested Crops, 68th International Conference on Agriculture Engineering LAND.TECHNIK AgEng, 2009. [↑](#footnote-ref-1)