# BACnet Core - Dokumentation

**Ein Projekt der Vorlesung «Introduction to Internet and Security»**

Raphael Kreft, Tim Bachmann, Nico Aebischer

18. Juli 2021

## Inhaltsverzeichnis

In diesem Dokument wird zunächst einmal beschrieben, was das BACnet ist und in groben Zügen wie es funktioniert. Danach soll aber vor allem der BACnet Core beleuchtet werden: Inwiefern handelt es sich dabei um den "Kern"des BACnets? Wie ist die modulare Struktur aufgebaut? Wie kann bei der Implementierung anderer Komponenten des BACnets den Core nutzen und von ihm profitieren?
Solche Fragen sollte der Leser nach dieser Lektüre beantworten können. Das Dokument richtet sich also an Personen, die entweder am Prinzip des BACnets interessiert sind, oder daran weiterarbeiten und in Betracht ziehen, vom Core Gebrauch zu machen.

# 1 Das «Basel Citizen Net» - Kurz und knapp

Das Internet wird mittlerweile von Milliarden von Menschen gebraucht - jeden Tag. Viele vergessen dabei, dass die Benutzung des Internets alles andere als eine Selbstverständlichkeit ist. Dies erfahren vor allem unterdrückte Minderheiten: In den letzten Jahren kam es immer wieder zu Restriktionen, wobei das Internet ganz gekappt wurde oder gezielt politische Opposition targetiert wurde.
Dies motiviert viele Menschen, sich Gedanken über Alternativen zum Internet zu machen. Wie lässt sich Kommunikation über digitale Kanäle sicherstellen, ohne dabei dem Goodwill von Regierungen und anderen grossen Akteuren ausgesetzt zu sein?
Im Frühlingssemester 2020 begannen Bachelorstudenten der Universität Basel im Rahmen der Vorlesung «Introduction to Internet and Security» damit, ein dezentrales Netzwerk zu implementieren, in welchem Teilnehmer Daten lokal in einer Datenbank abspeichern können und in dem es verschiedene Kommunikationskanäle gibt, um Daten mit anderen Netzwerkteilnehmern zu teilen. Im Frühlingssemester 2021 wurde die Arbeit des Vorjahres aufgenommen und weitergeführt. Ziel war es, die Funktionalität des BACnets zu erweitern und die Nutzbarkeit bzw. die Qualität der vorangehenden Projekte zu verbessern.
Das BACnet basiert auf Konzepten, die in «Secure Scuttlebut» implementiert wurden. Nachfolgend wird näher darauf eingegangen.

## 1.1 Das Secure Scuttlebutt Prinzip

Das Projekt «Secure Scuttlebutt» ist ein Versuch eines dezentralen Netzwerks, bei dem verteilte Applikationen serverlos mittels abgeglichenen Append-Only-Logs funktionieren. Nachrichten werden nicht per Pakete übertragen und wieder vergessen, sondern in speziell dafür erstellten Logs - den sogenannten Feeds - organisiert.

Die Feeds werden von Netzwerkteilnehmern lokal erstellt. Jeder Feed hat eine feste Quelle und folgende Eigenschaften:

- Der Feed besteht aus Events, welche aneinander gehängt werden.
- Jeder Event enthält Metadaten, Nutzdaten und einen Verweis auf den vorherigen Event.
- Zu jedem Feed gehört ein Schlüsselpaar, welches genutzt wird, um Events zu signieren bzw. zu verifizieren.

Nutzer können Feeds anderer Nutzer abonnieren. Bei einem Datentransfer erhalten sie dann diejenigen Feeds lokal auf ihren Rechner, die sie abonniert haben und die vom Gegenüber zur Verfügung gestellt werden können. Somit entsteht ein dezentrales Netzwerk.
Die Idee des Initianten von Secure Scuttlebutt war es, eine Art Social Media zu machen, bei der man (unabhängig von einer Internetverbindung) News lesen kann. Man holt sich also beim Synchronisieren eine aktuelle Version seiner abonnierten Feeds auf den Rechner und kann diese dann zu einem beliebigen Zeitpunkt konsumieren. Eine Verbindung muss erst dann wieder hergestellt werden, wenn man wieder ein Update machen oder selbst etwas posten möchte.
Dabei spielt es keine Rolle, auf welchem Weg die Daten übertragen und zwischen den einzelnen Teilnehmern synchronisiert werden. Es kann neben einer normalen UDP/TCP Verbindung auch USB-Stick-Austausch (Sneakernet) oder Radiowellenübertragung genutzt werden. Durch die Freieheiten bei der Implementation der Replikation kann also sichergestellt werden, dass man die Kontrolle über das Netzwerk behält und man sich nicht von Drittparteien abhängig macht.

## 1.2 BACnet - aktueller Stand

Das Basel Citizen Net hat die Grundstruktur von Secure Scuttlebutt übernommen, arbeitet also auch mit Feeds und Events, welche Applikationsdaten verschiedener Netzwerkteilnehmer enthalten. Im Frühlingssemester 2020 wurde der Grundstein für die Entwicklung des BACnets gelegt. Die verschiedenen Gruppen haben an unterschiedlichen Bereichen gearbeitet, wobei die zu bewältigende Hürde dann darin lag, die Projekte am Schluss als funktionsfähige Teile eines dezentralen Netzwerks zusammenzufügen.

Die Projekte waren auf die verschiedenen Komponenten des BACnets verteilt. So gab es eine Storage Gruppe, welche für die datenbankspezifischen Fragen zuständig war und in enger Koordination mit der LogMerge Gruppe (Geregelter Abgleich verschiedener Versionen eines Feeds) und der LogSync Gruppe (Synchronisation zweier Datenbanken) gearbeitet hat. Dies führt, wie wir später sehen werden, auch zu unserem BACnet Core. Andere Gruppen entwickelten Applikationen (verschiedene Chats) oder implementierten verschiedene Kommunikationslinks zum Datenaustausch zwischen Nodes (z.B. Sneakernet, Bluetooth, LoraLink).
Im Frühlingssemester 2021 wurde die im Vorjahr gelegte Basis weiterentwickelt. Beispiele dafür sind die Weiterentwicklung einer Chat Applikation des Vorjahrs mit einer Sketch-Funktion, verbessertes LoraLink, Verschlüsselungsmechanismen für den Chat oder das Verpacken von News Artikeln (z.B. von SRF) in Events.

## 2 Grundstruktur des BACnets

Das BACnet hat einen relativ simplen modularen Aufbau. Grenzt man die einzelnen Funktionalitätsbereiche gut ab, so kann man gezielt an einem Modul arbeiten, ohne dabei die anderen in Betracht ziehen zu müssen. Folgend die groben Module: Storage, Replication, Application. Wie auch Secure Scuttlebutt kommt das BACnet mit wenigen grundlegenden Datenstrukturen aus. Jede lokale Datenbankinstanz stellt im BACnet einen Netzwerkteilnehmer (Node) dar. Sie besitzt genau einen Masterfeed und beliebig viele Feeds. Die Feeds sind eine Aneinanderkettung von Events, die ein bestimmtes Ereignis in der Applikation verkörpern und aus Metadaten sowie Nutzdaten (z.B. eine Chat-Nachricht) bestehen.

### 2.1 Das Storage-Modul

- kümmert sich um die Speicherarchitektur (Datenbankinstanz), die auf jeder Node vorhanden sein muss.
- bietet gegen aussen Methoden an, um eine Reihe von Datenbankabfragen zu machen sowie neue Events zu erstellen und abzuspeichern.

### 2.2 Das Replication-Modul

- kümmert sich um die Verbreitung von Einträgen (Feeds und Events) zwischen verschiedenen Nodes.
- kann mittels verschiedensten Technologien implementiert werden. Da die Unabhängigkeit vom gängigen Internet angestrebt wird, eignen sich besonders davon unangetastete Kommunikationsmittel.

### 2.3 Das Application-Modul

- besteht aus Applikationen, welche von Endnutzern eingesetzt werden und die auf das BACnet-Netzwerk zugreifen, um Nutzdaten zu speichern und an andere Teilnehmer weiterzugeben.
- dient als «Interface zum Menschen», bietet also GUIs oder andere Interfaces an, die für Dateninput genutzt werden können.

### 2.4 Der Masterfeed

- wird standardmässig immer zwischen den Nodes synchronisiert, wenn es über den Kommunikationslink des BACnet Cores zum Datenaustausch kommt.
- enthält Events, die auf Feeds verweisen. Die referenzierten Feeds können Feeds anderer Netzwerkteilnehmer sein, die man abonniert hat oder auch solche, die man selbst erstellt hat.
- wird beim Erstellen einer Datenbank immer als Erstes erzeugt.

### 2.5 Feeds

- können in beliebiger Anzahl für jede Applikation erstellt werden.
- besitzen einen eigenen Schlüssel bzw. ein Schlüsselpaar, das für das Signieren gebraucht wird.
- gehören zu genau einem Masterfeed. Der jeweils erste Event eines Feeds zeigt auf dessen Masterfeed.

- muss man vertrauen (wenn sie von anderen Nodes kommen), um von ihnen bei einer Synchronisation Daten abzugleichen.

# 3 BACnet Core

## 3.1 Die Problemstellung

Wie bereits erwähnt kann man das BACnet in verschiedene Module aufteilen. Dies war natürlich auch schon im Frühjahrssemester 2020 bekannt und die Gruppen wurden dementsprechend gebildet. Die Konsequenz der vielen kleineren Gruppen war dann allerdings, dass die Entwicklung des gesamten BACnets durch die Gruppierung dehomogenisiert wurde und daraus Ineffizienzen entstanden. Als Student im Frühjahrssemester 2021 war es dann zu Beginn sehr schwer, sich einen Überblick zu verschaffen. Was ist das BACnet überhaupt? Wie sind die Module strukturiert? Welche der Vorjahresgruppen liefern für mein eigenes Projekt verwertbare Vorarbeit?
Diese Fragen musste jede Gruppe für sich klären, bevor man sich an die Auswahl eines eigenen Projekts wagen konnte. Auch das Arbeiten mit dem bestehenden Code erwies sich als durchaus herausfordernd, da wenig Dokumentation vorhanden war, woran man sich orientieren könnte.
Uns ist das von Beginn weg aufgefallen und wir haben uns deshalb entschieden, eine API anzubieten, welche die zentralen Komponenten der letztjährigen Projekte auf sinnvolle Weise zusammenfasst und optimiert. Es soll einfacher werden, sich als Neuling im BACnet zurechtzufinden und mit dem Entwickeln von neuen Features zu starten.

## 3.2 Die Lösung

Auf der Suche nach dem «Kern» des BACnets kommt man unweigerlich zuerst auf das Storage-Modul. Es ist von äusserster Wichtigkeit, eine Speicherarchitektur zu schaffen, welche mit den dem BACnet zugrundeliegenden Datenstrukturen umgehen und diese effizient abspeichern kann. Es muss einfach sein, eine Anwendung basierend auf dem BACnet zu implementieren und ein geeignetes Speichermodul ist dafür unumgänglich. Der Grundstein des BACnets sind die einzelnen Nodes des dezentralen Netzwerks, sprich die Datenbankinstanzen. Daher besteht der neue «Core» zu einem grossen Teil aus speicherbezogenen Klassen.
Der Core muss dazu imstande sein, Events zu erstellen und effizient in Feeds zu organisieren. Diese müssen so verarbeitet werden können, dass man sie im permanenten Speicher (Datenbank) ablegen sowie aus Datenbankeinträgen wiederherstellen kann.
Schlussendlich muss er auch den geregelten Anschluss seinerselbst an das Netzwerk sicherstellen. Die Replikationsschicht des BACnets kann über verschiedene Channels via einen Communication-Link mit dem BACnet Core in Verbindung treten und Datenaustausch vornehmen.
All diese Funktionalitäten waren bislang auf verschiedene Projekte verteilt und der Überblick hat gefehlt. Mit dem Core ist es unser Ziel, das Erstellen, Synchronisieren und Verwalten von Nodes im Netzwerk einfach zu machen. Die Schwierigkeit für nachfolgende Gruppen soll nicht darin liegen, sich um das Erstellen von Events und um die Synchronisation zu kümmern, sondern sie sollen sich vollends auf die Implementation ihres eigenen Projekts fokussieren können.
Der Aufbau des Cores ist modularisiert. Dadurch wird es einfach, ihn zu modifizieren und an sich weiterentwickelnde Umstände anzupassen. Als Beispiel dafür könnte man auf einfache Art und Weise einen weiteren Channel für die Replikationsschicht implementieren, damit die Anzahl der unterstützten Übertragungskanäle vergrössert wird.

# 4 Aufbau des Cores

Um die Funktionsweise und die Möglichkeiten zur Nutzung des BACnet Cores zu verstehen, lohnt es sich, einen genaueren Blick auf den modularisierten Aufbau zu werfen. Wir unterteilen den Core in folgende Module und werden diese im Anschluss genauer untersuchen:

- Core Interface
- Storage
- Security
- Replication

## 4.1 Node

Die Node ist die grundlegendste Klasse des BACnet Cores. Sie beinhaltet die Datenbankinstanz, den Com-Link, einen Channel, Storage Controller und natürlich einen Masterfeed. Das Geschehen des Cores wird über die Node orchestriert, die als zentrale Komponente einer Blackboard Architektur aufgebaut ist.

## 4.2 Core Interface

Das Modul «Core Interface» dient der Organisation von Daten als Teil des BACnet Frameworks, strukturiert sie also in Events und Feeds. Das Modul ist folgendermassen aufgebaut:

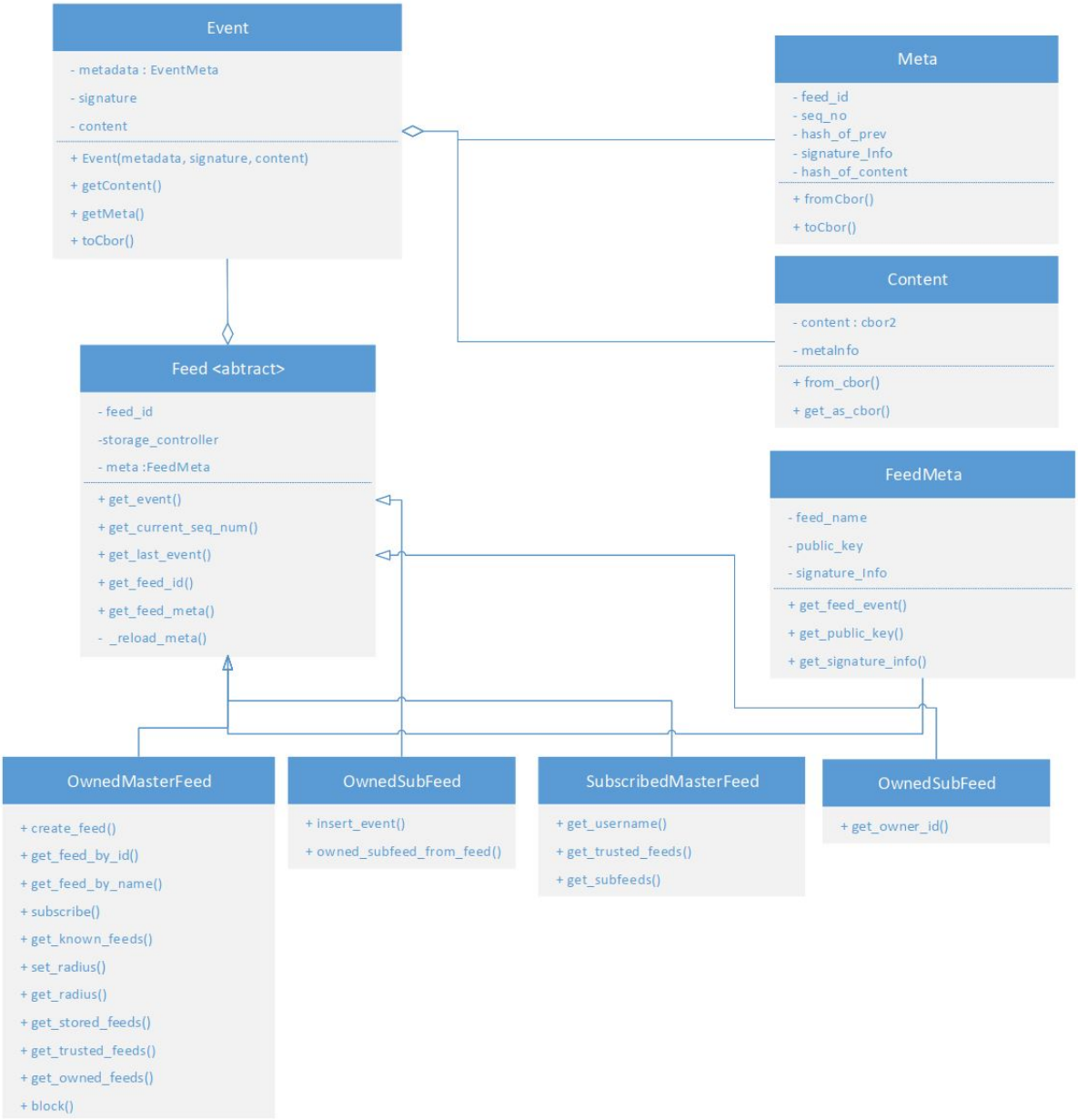


Abbildung 1: Aufbau des Core Interface Moduls.

### 4.2.1 Events

Events sind die grundlegendste Datenstruktur des BACnets. Alle Nutzdaten der Applikationen, die das BACnet Netzwerk brauchen, werden als Events auf der Node abgespeichert. Beispiele dafür sind Chat Nachrichten, Bilder

oder Wetterdaten einer Messstation. Events werden in der Datenbank abgespeichert und um sie zu versenden in ein Binärformat (CBOR) gebracht. CBOR orientiert sich am JSON-Format und eignet sich zur Serialisierung und damit zum Datentransfer zwischen den Nodes.
Events bestehen ihrerseits aus drei Hauptteilen:

- Metadaten
- Signatur
- Content

Die Metadaten sind essentiell für die Strukturierung der Feeds. Sie bestehen aus der Feed ID (Public Key) des Feeds, dem der Event angehört, einer Sequenznummer zur Einordnung des Events in den Feed und aus dem Hashwert des vorherigen Events im Feed. Ausserdem wird noch ein Feld für die Signature Info gebraucht, die angibt mit welchem Signaturalgorithmus gearbeitet wird und dem Hashwert der Nutzdaten, die der Event enthält. Die Signatur ermöglicht das Überprüfen der Authentizität eines Events.
Der Content eines Events beinhaltet die Nutzdaten der Applikation, die den Event erstellt hat. Es sind wiederum zwei Teile vorhanden: Ein Identifier sorgt dafür, dass man einen Event in Queries oder auf Applikationslevel identifizieren kann. Der Datenteil ist selbsterklärend. Hier können arbiträre Daten gespeichert werden.
Die wichtigsten Funktionen der Events sind jeweils getter-Methoden um die Meta- oder Nutzdaten zu bekommen sowie zwei Methoden, um Events in das CBOR-Format zu konvertieren bzw. von einem CBOR-Objekt einen Feed zu erstellen.

### 4.2.2 Feeds

Feeds sind eine Datenstruktur, bestehend aus einer Aneinanderkettung von Events, in der Events auf den jeweils vorangehenden Event referenzieren (via eines Hashwertes). Es sind Append-Only-Logs.Jeder Feed kann durch eine eindeutige Feed ID identifiziert werden. Wie auch Events enthält jeder Feed auch Metadaten. Diese sind zusammengesetzt aus dem Feed Name (ID), einem Public Key und der Signature Info, die besagt, mit welchem Signatur-Algorithmus auf dem Feed gearbeitet wird. Wir unterteilen folgende Klassen von Feeds:

- Owned Feed

  Master

  Content

- Subscribed Feed

  Master

  Content

Owned Feeds sind dadurch charakterisiert, dass sie sich auf der Node befinden, auf der sie erstellt wurden. Ein Owned Feed gehört also einer Node. Jeder Feed ist ursprünglich ein Owned Feed und wird zu einem Subscribed Feed, wenn er durch das Netzwerk auf andere Nodes gelangt. Jeder Netzwerkteilnehmer kann sich dazu entschliessen, gewisse Feeds von anderen Teilnehmern zu abonnieren, sie also nicht nur als Relais weiterzuleiten, sondern seinerseits in der eigenen Datenbank abzuspeichern.
Im Rahmen der Feeds wird auch folgendes Problem der Synchronisation zwischen Nodes gelöst: Als Node soll man zum Netzwerk beitragen, indem man Feeds und Events an andere Teilnehmer verschickt, welche noch nichts davon mitbekommen haben. Allerdings möchte ja nicht jeder Netzwerkteilnehmer alle Feeds bei sich lokal speichern. Man kann bestimmte Feeds abonnieren und ihnen vertrauen, oder aber bestimmten Feeds misstrauen und explizit nicht synchronisieren. Wie kann man nun schon vor der Synchronisation der eigentlichen Daten gewisse Feeds anfordern bzw. blocken?
Hierfür gibt es das Konzept der Masterfeeds. Zu jeder Node gehört gehört immer auch ein Masterfeed, der bei der Initialisierung der Datenbank erstellt wird. Für jeden neuen «normalen» Feed, der zum Beispiel einer bestimmten Applikation angehört, wird dann auf dem Masterfeed ein Event angehängt. Damit wird die Zugehörigkeit des Feeds zur Node (zum Masterfeed) angegeben. Der jeweils erste Event eines normalen Feeds verweist auf den Masterfeed, auf dem er erstellt wurde. Damit wird vermieden, dass ein Masterfeed einen Feed zu unrecht als den eigenen beansprucht. Bei der Synchronisation können die aus dieser Struktur auslesbaren Informationen verwendet werden, um nur die gewünschten Feeds zu importieren.
Der zentrale Unterschied zwischen Master-und Contentfeeds ist, dass den Masterfeeds standardmässig immmer vertraut wird. Damit entsteht im Verlauf der Zeit eine Sammlung vieler Masterfeeds im Netzwerk. Daraus kann dann eine Liste von Masterfeeds angelegt werden, denen man traut bzw. misstraut. Vor dem Datentransfer kann diese Liste abgeglichen und vorsortiert werden, welche Feeds man importiert und welche nicht in die eigene Datenbank aufgenommen werden sollen.
Die Content Feeds (Owned und Subscribed) dienen zur Eingliederung von Applikationsdaten in die Datenstruktur

des BACnets. Meist werden applikationsspezifisch Feeds angelegt, denn bei der Auswertung der Daten muss ja erkannt werden, wie die Nutzdaten interpretiert werden sollen: Repräsentiert diese Integer-Zahl einen Parameter in Wetterdaten oder ist sie Bestandteil einer Chat-Nachricht? Zum Beispiel kann dann bei einer Chat-App für jeden Chat-Partner ein Feed erstellt werden, der die einzelnen Nachrichten als Events verknüpft.

## 4.3 Storage

Das Modul «Storage» implementiert die Logik für das geregelte Abspeichern der Daten auf einer Node. Jede Node beinhaltet eine Datenbankinstanz. Da auch hier ein modularisiertes System erwünscht ist, soll der Zugriff der Node auf die Datenbankinstanz unabhängig von der gewählten Datenbanksprache sein. Es soll also möglich sein, die Sprache zu wechseln und dabei möglichst wenig im bestehenden Code anzupassen, sondern nur neuen Code passend auf das Interface, auf das die Node zugreift, zu schreiben. Daher ist die Architektur des Storage Moduls der Abbildung 2 entsprechend gewählt.

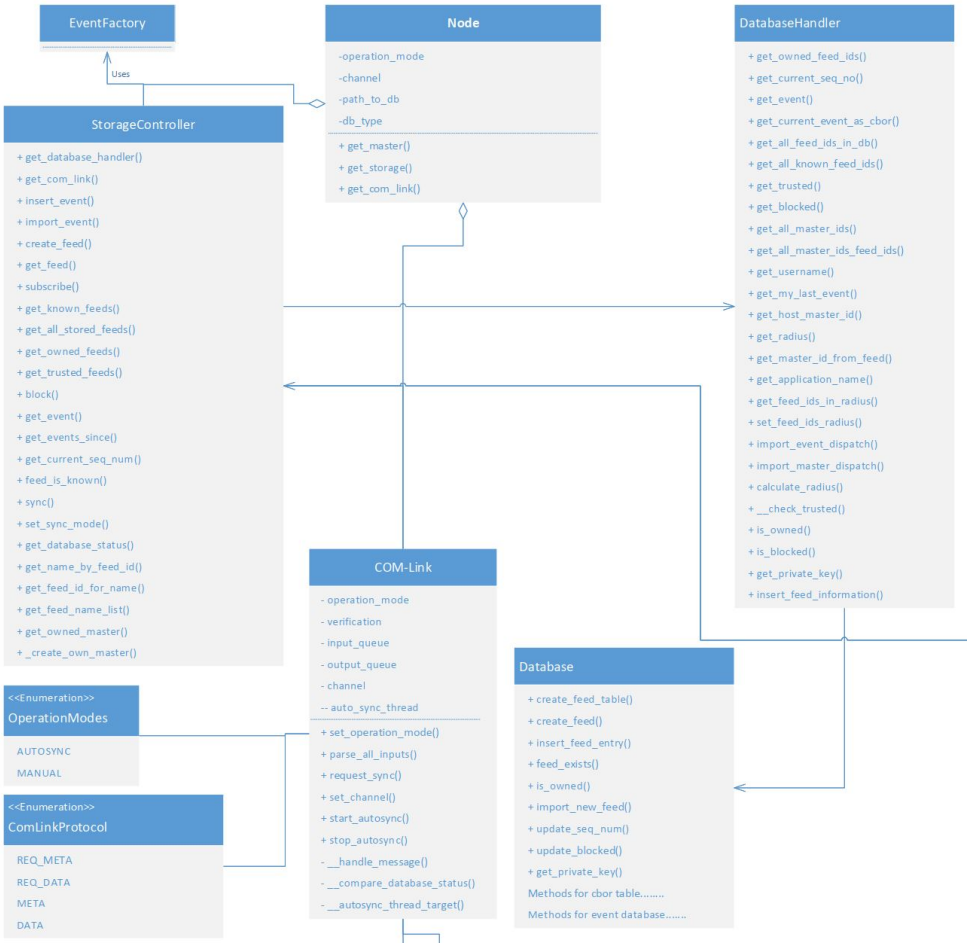


Abbildung 2: Aufbau des Storage Moduls.

### 4.3.1 Storage-Control

Die Storage-Control ist als abstrakte Klasse präsent und dient als Grundlage für die Implementierung verschiedener Möglichkeiten zur Datenbankanbindung. Sie ist die grundlegendste Komponente des BACnet Cores und wird von Feeds oder dem Com-Link getriggered. Die Aufgabe der Storage-Control ist es, die in allen Speichervarianten benötigten Funktionalitäten vorzugeben. Dazu gehören zum Beispiel Methoden um neue Feeds zu erstellen, Feeds zu blockieren, Änderungen zu pushen oder Events zu importieren. Diese High-Level-Funktionen müssen dann im Detail vom Database-Handler und vom SQLite Controller ermöglicht werden.

### 4.3.2 Database-Handler

Der Database-Handler wird von der Storage-Control aus «gesteuert» und nimmt sich der Low-Level Implementation der datenbankspezifischen Funktionen an. Der Code ist allerdings immer noch nicht auf eine Datenbankanbindung spezialisiert, dies wird im SQLite Controller gemacht.

### 4.3.3 SQLite Controller

Der SQLite Controller ist die vom BACnet Core spezifizierte Speichervariante. Es wird mit einer SQLite Datenbank gearbeitet. In den Versionen des Vorjahres wurden für verschiedene Apps verschiedene Storage-Anbindungen gemacht. Wir haben uns allerdings dazu entschlossen, dass wir nur eine allgemein gehaltene Version anbieten wollen und die Unterscheidung in die verschiedenen Apps dann mittels der Metadaten der Feeds gemacht wird.
Da die Storage-Control allgemein gehalten ist, wird es auch für zukünftige User des BACnet Cores möglich sein, auf einfache Art und Weise Storage-Anbindungen zu machen. Die Idee ist es, dass man nicht auf eine SQLite Anbindung limitiert ist, sondern je nach Bedarf z.B. auch noch eine MySQL Anbindung oder weitere Varianten hinzufügen kann.

### 4.3.4 Event Factory

Die Event Factory ist wie der Name schon sagt dazu da, Events (aber auch Feeds) zu erstellen. Sie wurde zu einem grossen Teil von der Vorjahresgruppe «LogMerge» übernommen. Beim Erstellen von Events und Feeds müssen zusätzlich zum Content noch einige weitere Informationen mitgegeben werden, wie zum Beispiel die Feed ID und die letzte Sequence Number, um den Event im Feed einzugliedern, die Signature Info oder einen Content Identifier, der für die Unterscheidung des Contents in die verschiedenen Apps zuständig ist.

## 4.4 Security

Das Modul «Security» behandelt die Sicherheit im BACnet. Ohne Sicherheitsmechanismen wäre es einfach, gefälschte oder sogar gefährliche Events in das System einzuspeisen. Es muss eine Kontrollstruktur vorhanden sein, die einerseits überprüft, ob eine Liste von Events als Kandidaten zum Import überhaupt in die Datenbank aufgenommen werden sollen und andererseits die Integrität von Feeds garantieren kann.
Dafür gibt es verschieden Kriterien. Für den Import gibt es eine Reihe von Regeln, anhand derer man Feeds überprüfen kann:

- Master Feeds werden immer importiert.
- Trusted und nicht blockierte Feeds werden importiert.
- Feeds müssen sich im vom User definierten Radius befinden. Der Radius gibt an, wie viele Hops zwischen der eigenen Node und der Herkunftsnode eines Feeds liegen. Feeds mit einem Radius von 1 sind die eigenen Feeds.

Für den Export existieren ebenfalls Regeln:

- Der eigene Masterfeed wird immer exportiert / angeboten.
- Trusted Feeds die nicht blockiert sind werden exportiert.

Dieses Set von Regeln zum Import respektive Export stellt schon mal sicher, dass keine Feeds ausserhalb des gewünschten Radius auf der eigenen Node zu liegen kommen und dass blockierte Feeds nicht importiert werden. Zusätzlich dazu gibt es verschiedene Funktionen, mit denen man die Hashwerte des Contents und Signaturen überprüfen und damit sicherstellen kann, dass die Integrität des Contents eines Events gegeben ist und dass die Reihenfolge der Events eines Feeds unangetastet geblieben ist.
Im BACnet Core werden die sicherheitsbezogenen Funktionalitäten in den Klassen *crypto.py* und *verification.py* implementiert.

## 4.5 Replication

Das Modul «Replication» nimmt sich der Verbreitung von Feeds und Events über verschiedene Netzwerkteilnehmer an. Jede Node ist Teil des dezentralisierten BACnets und muss über mindestens einen Kommunikationslink mit anderen Teilnehmern kommunizieren können, damit die Applikationsdaten über das Netz verteilt werden.

#### 4.5.1 COM-Link

Der Hauptbestandteil ist das Zusammenspiel des Communication-Links mit verschiedenen Channels. Der Communication-Link ist als Instanz auf der Node enthalten und agiert als Mediator, der Funktionen zum Datenaustausch gegen aussen anbietet. Beispiele dafür sind *importMasters()*, *exportAll()* oder *autoSync(timeInterval)*.
Der COM-Link wird von der Node aus angesteuert und kommuniziert dann mit der Storage-Control (SQLite Controller), wo die logisch «höheren» Anfragen zu einzelnen Datenbankabfragen verarbeitet werden.

#### 4.5.2 Channels

Die Channels sind das Verbindungsstück des BACnet Cores nach aussen. Die abstrakte Klasse «Channel» legt hierbei wie auch schon die Storage-Control fest, was bei einer spezifischen Implementierung sicher vorhanden sein muss. Hier ist ebenfalls der modulare Charakter der Architektur von grosser Bedeutung, damit weitere Channels einfach hinzugefügt werden können, ohne am Core selbst was ändern zu müssen.
Das Ziel ist auch hier wieder flexibel zu sein und verschiedenste Technologien zum Datentransfer unterstützen zu können. Der BACnet Core bietet eine vorgefertigte Implementaiton eines UDP-Channels an. Weitere Channels können dann auf einfache Weise selbst hinzugefügt werden.
Beispiele sind eine Sneakernet-Implementation, die es möglich macht, mittels USB-Sticks Daten zu übertragen. Dabei werden Einträge über den File Path abgefragt und synchronisiert.
Weiter kann man Peer to Peer Links implementieren. Diese stehen nicht für eine spezifische Implementierung des Datentransfers, sondern sie können unterschiedlicher Natur sein. Ein Beispiel für einen Peer to Peer Link könnte die Übertragung über eine Richtstrahlverbindung sein.

### 4.6 Samples

Das Modul «Samples» enthält Beispielcode, anhand dessen man sehen kann, wie der Core gebraucht werden kann. Die Beispiele waren erst primär für uns selbst gedacht, um unseren Code zu testen. Allerdings entschlossen wir uns dazu, die Samples im Projekt zu lassen, damit mögliche Nutzer des Cores einen einfacheren Einstieg haben

## 5 Ausblick

Rückblickend war die Arbeit am BACnet eine lehrreiche und unterhaltsame Aufgabe. Es war durchaus anspruchsvoll, sich in die Projekte des Vorjahres einzulesen und daraus Schlüsse für das eigene Projekt zu ziehen. Allerdings war das auch sicherlich eine gute Übung, da es sehr wahrscheinlich ist, dass wir wieder einmal auf ein Projekt stossen werden, bei dem erst einmal viel Recherche und Codebasis-Analyse ansteht. Wir haben das Gefühl, dass wir mit diesem Projekt sehr zielgerichtet gearbeitet haben, so wie wir es in «Software Engineering» gelernt haben. Zuerst kam eine lange und detailreiche Planungsphase. Danach haben wir das geplante vorerst als Prototyp mit Skeleton-Methoden implementiert und die Struktur nochmals angepasst, bevor wir dann die genaue Implementation begonnen haben.
Unser grosses Ziel mit dem BACnet Core war es, das Verständnis für das BACnet und die zentralen Aspekte davon zu verstärken. Es soll in Zukunft nicht mehr so schwierig sein, sich in die Funktionsweise des Ganzen einzulesen und sich die nötige Architektur für Erweiterungen zusammenzusuchen. Der Core sollte die nötigsten Funktionalitäten zum Start bieten und der modulare Charakter macht es möglich, in sinnvoller Frist Anpassungen für das eigene Projekt zu machen.
Wir hoffen, mit der Arbeit am BACnet Core den Weg für zukünftige Gruppen geebnet zu haben und damit unseren Beitrag für das Ergänzen des BACnets mit weiteren Funktionalitäten leisten konnten.

## 6 Vollständiges Klassendiagramm

Für eine grössere Version kann man sich das entsprechende PDF-File im Ordner «libStructure» bei den Dokumenten ansehen.

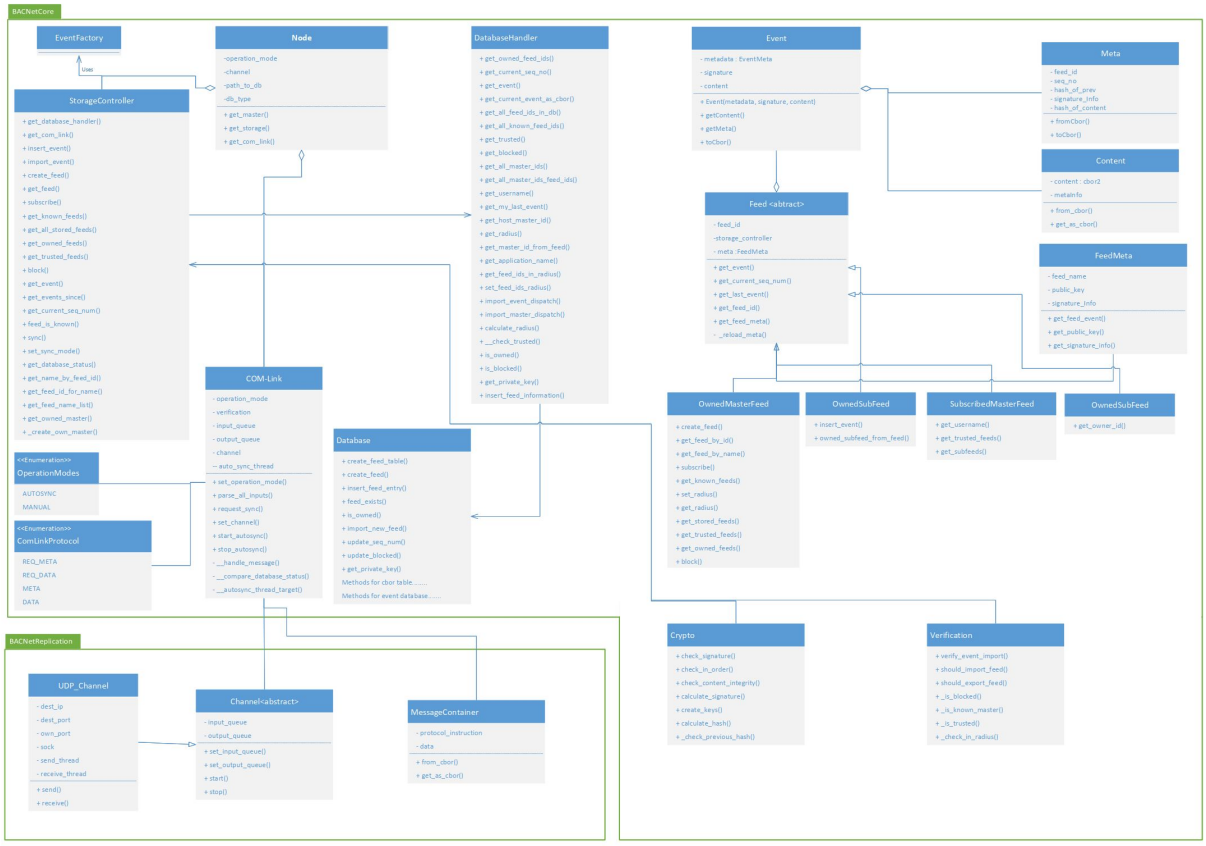


Abbildung 3: Klassendiagramm des BACnet Cores.