HomeMatic User-Treffen Kassel 2017 6 Mai 2017

ioBroker-Adapter-Entwicklung

Schritt für Schritt

Ingo Fischer

ioBroker
Automate your life

- Ich möchte heute ein bisschen darüber informieren wie die Adapterentwicklung bei ioBroker so abläuft, was man braucht und ein paar best practices
- Leider wurde der Zeitslot für den Vortrag gekürzt, so dass ich auch den Vortrag kürzen musste. Ich stelle im Nachgang aber gern die lange Version zur Verfügung.
- Ich werde auch bei einigen Folien nur auf das wichtigste eingehen
- Fragen am besten am Ende oder danach oder natürlich heute Abend

Über Mich ...

- Seit ca. 20 Jahren in Software-Entwicklung tätig
  - ... Java, PHP und JavaScript/nodejs
  - ... als Entwickler und im Management
- 3 eigene ioBroker-Adapter
  - ... nut (Network UPS Tools/USV Daten bereitstellen)
  - ... smartmeter (Stromzähler über verschiedene Protokolle auslesen und Daten bereitstellen)
  - ... Daikin (Daikin-Klimaanlagen mit WLAN-Adapter Daten bereitstellen und steuern)
- Aktueller Maintainer der 3 History-Adapter von ioBroker
  - ... history (JSON-Files)
  - ... sql (SQL-DBs: MySQL, MSSQL, PostgreSQL und SQLite)
  - ... influxdb (Time Series Database)
- „Master of Adapter-Testing“
  - ... Adapter-Testing in nahezu alle Adapter eingebaut bzw. unterstützt ☺
- Forum- und Trello-Administrator

Was man alles braucht ...

- Funktionierende ioBroker-Installation (idealerweise Dev-Installation, lokal vorteilhaft)
- ioBroker-Forum Account
- optional ioBroker-Trello Account

- Grundlagen Programmierung
- JavaScript / nodejs (Adapter-Hauptlogik)
  ... Asynchrone Ausführung(!), callbacks/Promises
- HTML + Browser-JavaScript

- GitHub Account
- Git Grundwissen (add, commit, pull, push)
- npm Account

- Grundlage ist natürlich ein laufendes ioBroker-System, idealerweise sogar ein eigenes Entwicklungssystem oder lokale Installation
- ioBroker ist in JavaScript, genauer auf Basis von nodejs geschrieben
- Erfahrung in JavaScript hilft daher natürlich.
- (Wenn man von einer anderen Programmiersprache kommt ist die größte Besonderheit die Eventbasierte und asynchrone Logik.)
- (Üblicherweise wird viel asynchrones per sog. Callback-Aufrufen umgesetzt. Das sind Funktionen die dann mit dem Ergebnis der Aktion zu einem späteren Zeitpunkt aufgerufen werden.)
- (In neueren nodejs-Versionen gibt es noch andere Möglichkeiten wie Promises und ähnliches.)
- Vor allem die asynchrone Ausführung von Code und was das für Nebeneffekte hat muss man einmal kapieren und lernen und den Knoten im Kopf rausbekommen, aber dann ist das alles gar nicht so schwer.

Am Anfang steht die Idee ...

- Was genau soll der Adapter tun? Welches Device bindet er an?
- Gibt es diese Funktionalität oder ähnlichen Adapter schon? → Forum

- Was könnten andere User brauchen? → Forum
- Sind alle eigenen Annahmen auch für andere User sinnvoll? → Forum
- Was sollte konfigurierbar sein? Sinnvolle Standardwerte?

- Geplante Grundfunktion als Prototyp umsetzen/testen
  - ... per ioBroker-JavaScript-Adapter
  - ... ohne ioBroker direkt mit eigenem nodejs Skript
  - ... npm Libraries prüfen, sonst ggf. eigene Library erstellen

- Wenn die Grundfunktionen laufen, dann Adapter drum herum bauen
  - ... Adapter-Template oder ähnlichen einfachen Adapter als Grundlage nutzen!

- Minimal notwendige nodejs-Version für Nutzung des Adapters (Achtung bei genutzten Libraries!):
  - ... 0.12? (unterstützt vor allem RasPi's mit Debian Wheezy)
  - ... mindestens 4.x (ES6) (sollte inzwischen meistens reichen)
  - ... mindestens 6.x schränkt den Nutzerkreis teilweise ein!

- Die meisten ioBroker-Adapter binden ein bestimmtes Gerät an oder stellen Daten zur Verfügung. Man kann aber auch per Adapter andere Funktionalitäten bereitstellen.
- Wenn man dann eine eigene Idee hat sollte man das was man machen will erstmal ohne ioBroker umsetzen. Direkt in nodejs als Prototyp oder als Script z.B. im ioBroker Javascript-Adapter.
- Aus meiner Erfahrung fliesst hier die meiste Zeit rein bis man alles so auslesen und steuern kann wie man will.
- Wenn es die Anbindung noch nicht auf npm als Library gibt dann kann man auch überlegen erstmal eine Library zu entwickeln. Das ist beim späteren Debugging ggf viel einfacher.
- Ich empfehle erst dann einen Adapter drum herum zu bauen wenn man dann weiss wie die Anbindung oder die Logik funktioniert.
- Als Grundlage gibt es mehrere Adapter-Templates die man als Code-Basis benutzen kann.
- Aktuell sollte man seinen Code so schreiben das er mindestens ab nodejs 4 läuft.
- Ich persönlich habe immer mehr Zeit für die Logik ohne ioBroker gebraucht, als dann daraus einen Adapter zu bauen.

ioBroker Begriffe

- **Adapter:**
  Software-Paket aus npm, das eine bestimmte Funktionalität implementiert.
  Wird über eine Admin-Seite konfiguriert

- **Instanz:**
  Eine bestimmte Konfiguration des Adapters der als eigener Prozess
  auf einem zugewiesenen Host ausgeführt wird
  ... Konfigurationsänderungen bedeuten Instanz-Restart

- **State:**
  Datenpunkt in ioBroker (z.B. Werte von Devices, dynamische Device-Einstellungen)
  ... ID generell: <Adapter>.<Instanz-Nr>.*
  ... ID Single-Device-Adapter: <Adapter>.<Instanz-Nr>.[<Channel>].<State>
  ... ID Multi-Device-Adapter: <Adapter>.<Instanz-Nr>.<Device>.[<Channel>].<State>

- **Objekt:**
  Beschreibung des States (Metadaten), u.a. Name, Datentyp, Read, Write, UI-Rolle ...

https://github.com/ioBroker/ioBroker/blob/master/doc/SCHEMA.md

- Bevor man startet gibt es ein paar Begriffe die man kennen sollte.
- Adapter und Instanz sind hier recht einfach: Adapter ist die Software, eine Instanz ist eine Konfiguration als laufender Prozess der ausgeführt wird
- States sind die Datenpunkte in denen alle Daten gespeichert werden. Mit Punkten kann man verschiedene Ebenen trennen und so zeigen welche States zusammengehören
- Jeder State hat üblicherweise auch eine Beschreibung – ein Objekt. Dieses enthält weitere Meta-Daten zum State die dann z.B. von Visualisierungstools verwendet werden

State-Properties

- **val** – Value
- **ts** – Value Timestamp
- **lc** – Value Last Changed Timestamp
- **from** – Source of Value
- **expire** – Expiry in Seconds
- **q** – Quality

- **ack** – acknowledge Flag
  ... true: bestätigter aktueller Wert, der nur vom Device/Adapter gesetzt wird
  ... false: unbestätigter, von extern zur Steuerung gesetzter Wert

  Grundsätzlicher Ablauf:
  ... setState mit ack=false und Zielwert
  ... Adapter reagiert auf State-Änderung und führt Aktion aus
  ... Adapter setzt neuen bestätigten Wert mit setState und ack=true

https://github.com/ioBroker/ioBroker/wiki/Adapter-Development-Documentation

- Ein State enthält einige Informationen.
- Neben dem reinen Wert des States ist der wichtigste für die Entwicklung das acknowledge-Flag
- Ist das Flag „wahr“, so ist der State-Wert der finale vom Gerät bzw. Adapter gesetzte offizielle Wert.
- Wenn jemand von aussen ein Gerät steuern will so wird der gewünschte neue Wert mit false als Acknowledge-Flag gesetzt.
- Der Adapter reagiert auf dieses Kommando und für die entsprechende Aktion aus oder setzt den gewünschten Wert.
- Danach stellt der Adapter aber sicher das es wieder einen offiziellen Wert gibt indem er diesen wieder mit ack=true selbst setzt.

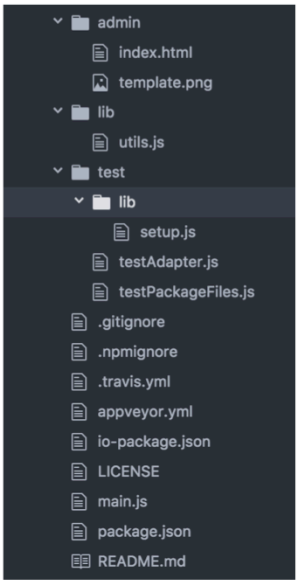
Adapter Modus

- „**schedule**“: Instanz wird basierend auf definiertem Zeitplan gestartet
- „**subscribe**“: Instanz wird bei Änderung eines definierten States gestartet
- „**deamon**“: Instanz läuft durchgehend inkl. Autostart

Funktionalität	schedule	subscribe	deamon
Regelmäßig geplante Aktionen ausführen, Daten abholen	JA	NEIN	JA (setTimeout)
Auf State-Änderungen reagieren/ Steuern	NEIN	JA (1 State!)	JA
Nachrichten (Funktionen mit JSON-Parameters) empfangen	NEIN	JA (exklusiv)	JA
RAM-Verbrauch laufender Prozess	Nur zur Laufzeit	Nur zur Laufzeit	Immer

- Wenn man jetzt mit dem Adapter startet ist eine der Entscheidungen die man treffen muss die in welchem Modus der Adapter läuft.
- Üblicherweise nutzen Adapter die nur Daten bereitstellen die regelmäßig gepollt werden den Modus schedule und die meisten anderen laufen als deamon.
- Vor allem wenn es States gibt mit denen man Geräte steuert kommt man um den Modus deamon nicht drumrum.

Adapter-Dateistruktur



index.html: Adapter-Konfiguration
.png/.jpg: Adapter-Icon

utils.js und eigene Libraries

Basis-Adapter-Testing
Package-File Checks

Travis-CI/Appveyor-Testing-Konfiguration

io-package.json: ioBroker-Package-File

main.js: Adapter-Code
package.json: npm-Package-File
README.md: GitHub-Readme für Adapter

- Ein Adapter besteht aus einigen Files, diese sind hier kurz dargestellt.

package.json – NPM Package Definition

- Die package.json wird für npm benötigt und enthält neben dem Namen vor allem Infos zu den Bibliotheken-Abhängigkeiten.

io-package.json – ioBroker Package Definition

```
{
  "common": {
    "name":  "adaptername",
    "version": "0.0.1",
    "news": {
      "0.0.1": {
        "en": "initial adapter",
        "de": "Initiale Version",
        "ru": "..."
      }
    },
    "title": "My Adapter",
    "desc": {
      "en": "ioBroker Adapter ...",
      "de": "ioBroker Adapter ...",
      "ru": "..."
    },
    "platform": "Javascript/Node.js",
    "mode": "daemon",
    "icon": "template.png",
    "enabled": true,
    "extIcon": "https://raw.githubusercontent.com/
                GitHubUser/ioBroker.adaptername/
                master/admin/template.png",
    "keywords": ["..."],
    "readme": "https://github.com/GitHubUser/
                ioBroker.adaptername/blob/master/
                README.md",
    "loglevel": "info",
    "type": "network",
    "restartAdapters": ["vis"]
  },
  "native": {
    "param1": "foo",
    "param2": 42
  },
  "objects": [

  ]
}
```

- Neben der package.json die für npm gebraucht wird holt sich ioBroker alle Meta-Informationen zum Adapter aus der io-package.json
- Wichtig sind hier vor allem die Versionsnummer und Versionshistory. Die Änderungen zwischen zwei Adapterversionen werden für die User im Admin-Bereich angezeigt und jede Version die hier drin ist kann im Admin-Bereich als für einen Rollback genutzt werden
- In der Native-Struktur sind alle Instanzparameter mit Ihren Standardwerten definiert, Diese Werte kann man in der Adapter-Konfiguration dann editieren.

Adapter Logik: Events

- **adapter.on(‘ready‘, ...) – PFLICHT**
  ... Ausgeführt nach Start der Instanz nach Connect zu Objects+States-DBs
  ... Initialisierung (alle Modi) und Haupt-Logik (schedule, subscribe)
  ... Subscribe für State-Änderungen (deamon)

- **adapter.on(‘stateChange‘, ...)**
  ... Ausgeführt wenn sich ein subscribed State ändert

- **adapter.on(‘objectChange‘, ...)**
  ... Ausgeführt wenn sich ein subscribed Object ändert

- **adapter.on(‘message‘, ...)**
  ... Ausgeführt wenn Nachricht empfangen wird

- **adapter.on(‘unload‘, ...)**
- **process.on(‘SIGINT‘, ...)**
- **process.on(‘uncaughtException‘, ...)**
  ... Aufräumen am Prozessende (Stop, Kill, Exception)

- Jetzt kommen wir endlich zum programmieren
- Ein Adapter muss mindestens auf ein Event reagieren, welches zum Start der Instanz nach Initialisierung der Datenbank-Verbindungen aufgerufen wird.
- Die restlichen Events sind je nach Bedarf nötig.
- Auf die ersten zwei gehe ich jetzt etwas detaillierter ein weil es die am meisten verwendeten sind. Der Rest muss jetzt hier aus Zeitgründen weichen, ist aber auch in den Adapter-Templates beschrieben

Event on(‘ready‘) / main()

```
adapter.on('ready', function () {
    main();
});

function main() {
    adapter.log.info('config param1: ' + parseInt(adapter.config.param1,10));

    adapter.setObjectNotExists('testSwitch', {
        type: 'state', // or channel or device
        common: {
            name: 'test-State',
            type: 'boolean', // or number or string ...
            role: 'switch‘ // see SCHEMA.md for allowed values
        },
        native: {}
    });

    adapter.subscribeStates('testSwitch');

    adapter.subscribeStates('*');
}
```

- Zum Adapterstart initiallisiert man üblicherweise alles was man braucht in der main Methode.
- Ein globales Objekt, was meistens adapter heisst, bietet hier Zugriff auf alle ioBroker-Funktionen die man als Adapter braucht.
- Man kann die Instanzparameter per adapter.config auslesen und zur Initiallisierung benutzen.
- Wenn man eigene Objekte und States anlegen will so nutzt man am besten setObjectNotExists, weil man das im Zweifel bei jedem Start so ausführen kann ohne Objekte zu überschreiben.
- Wenn der Adapter States hat die vom User oder anderen Adaptern zum Steuern genutzt werden, so muss man sich an diese States noch subscriben das der Adapter die Änderungen auch mitbekommt.

Read and write States

```
// the variable testVariable is set to true as command (ack=false)
adapter.setState('testSwitch', true);

// same thing, but the value is flagged "ack"
// ack should be always set to true if the value is received from or acknowledged from the target system
adapter.setState('testSwitch', {val: true, ack: true});

// same thing, but the state is deleted after 30s (getState will return null afterwards)
adapter.setState('testSwitch', {val: true, ack: true, expire: 30}, function (err, id) {
    if (err) adapter.log.error('Cannot set value for "' + id + '": ' + err);
});



adapter.getState('testSwitch', function (err, state) {
    if (err || !state) {
        // error or no state exists
    }
    adapter.log.info('testSwitch is ' + state.val);
});
```

- Nachdem die Objekte angelegt sind kann man die State-Werte auch setzen.
- Dazu nutzt man setState
- Beim lesen von State-Werten muss man beachten das das Ergebnis asynchron an eine callback-Funktion gemeldet wird.
- Das kann im Code zu callback-Ketten führen die sehr unschön sind.
- Ein Trick ist daher sich die States die man braucht auch in lokalen Skriptvariablen zu halten und bei jeder Änderung zu aktualisieren, aber dann kann man direkt die Variable nutzen wenn man den Wert braucht und weiss das Sie immer aktuell ist.

Event on('stateChange')

```
// is called if a subscribed state changes
adapter.on('stateChange', function (id, state) {
    adapter.log.info('stateChange ' + id + ' ' + JSON.stringify(state));
    // Warning, state can be null if it was deleted
    if (!state) return;

    // you can use the ack flag to detect if it is status (true) or command (false)
    if (!state.ack) {
        adapter.log.info('ack=false');
    }

    // we react on change of testSwitch with ack=false (=command) and val=true
    if (id === adapter.namespace + ".testSwitch" && !state.ack && state.val === true) {
        // Do what's needed here ...

        // and reset state to false
        adapter.setState('testSwitch', {val: false, ack: true});
    }
});
```

- Für alle State-Änderungen für die man sich subscribed hat wird bei Änderungen ein stateChange Event gefeuert und die hinterlegte Funktion ausgeführt.
- Wie vorhin schon gesagt sollten Steueraufgaben immer nur bei ack=false ablaufen, von daher sollte man das hier auch prüfen. Diese Funktion wäre auch die Stelle um lokale Skript-Variablen aktuell zu halten.

Event on(‘message‘)

```
// Some message was sent to adapter instance over message box.
adapter.on('message', function (obj) {
    if (typeof obj === 'object' && obj.message) {
        if (obj.command == 'doWatever') {
            // e.g. send email or pushover or whatever
            adapter.log.info('message received ' + obj.command);
            // do whatever needed here ...

            // Send response in callback if required
            if (obj.callback) adapter.sendTo(obj.from, obj.command, 'Wathever done', obj.callback);
            // or more complex return object
            if (obj.callback) adapter.sendTo(obj.from, obj.command, {
                response: 'Wathever done',
                value: 42,
                error: false
            }, obj.callback);
        }
    }
});
```

- Messages kann man nutzen um Kommandos oder Aktionen mit komplexen Daten umzusetzen.
- Solche Nachrichten kommen üblicherweise aus Skripten oder anderen Adaptern.
- Beipsiele sind die Kommunikationsadapter die über eine Nachricht „send“ dann alle Nachrichtendaten als Objekt bekommen

Adapter-Konfiguration (Admin)

```
<script type="text/javascript">
   systemDictionary = {
      "adapter settings": {"en": "My Adapter",  "de": "Mein Adapter", "ru": "..."},
      "param1":          {"en": "Text", "de": "Text", "ru": "..."},
      "param2":          {"en": "Number", "de": "Zahl", "ru": "..."},
      ...
   };
   // ... functions "load" and "save" can be used from template
</script>

<div id="adapter-container">
   <table>
      <tr><td><img src="template.png"/></td><td><h3 class="translate">adapter settings</h3></td></tr>
   </table>
   <p>
      <span class="translate">param1</span> <input class="value" id="param1"/><br>
      <span class="translate">param2</span> <input class="value number" id="param2"/><br>
   </p>
</div>
```

- Im ioBroker Forum haben einige gesagt das Sie mit der Logik kein Problem haben, aber vor der Konfiguration angst haben. Muss man nicht
- Für einfache Instanzparameter ist das ganze auf Basis der Adapter-Templates oder eines anderen Adapters der so ähnlich tickt meist gar nicht so schwierig
- Im einfachen Fall sind es zwei Stellen in der index.html im Verzeichnis admin die relevant sind
- Am Anfang das Übersetzungsobjekt wo zu texten die Übersetzungen in deutsch, englisch und teilweise russisch drin stehen. Fallback ist englisch ??
- Und weiter unten gibt es eine einfache HTML-Tabelle für die Formularfelder.
- Und das wars auch schon.
- Natürlich kann man das auch noch richtig fancy mit Tabellen und allen machen, ist aber meistens ja gar nicht nötig ☺
- Wichtig: nach änderungen an Admin-Files immer ein „iobroker upload <adaptername>“ ausführen, sonst sieht man diese Änderungen nicht!

Adapter verfügbar machen ...

- Neuer Adapter = Neues (Public) Projekt erzeugen
  ... Name: ioBroker.adaptername
  ... niemals als Fork eines anderen Adapters oder Templates!
- Lizenz auswählen und hinzufügen
- Alle Files reinpacken und committen
- Installation von GitHub in ioBroker
  ... iobroker url https://github.com/GitHubUser/ioBroker.adaptername
  ... iobroker upload adaptername
  ... iobroker add adaptername

- Wenn dann alles lokal klappt, dann wird es zeit den Adapter öffentlich Verfügbar zu machen um auch andere Nutzer teilhaben zu lassen.
- GitHub ist aktuell Quasi-Standard für Adapter-Code, am Ende aber egal, so lange es git ist
- Neues leeres Projekt z.B. auf Github anlegen, alles reinpacken und schon kann man es in ioBroker auch von dieser Quelle installieren

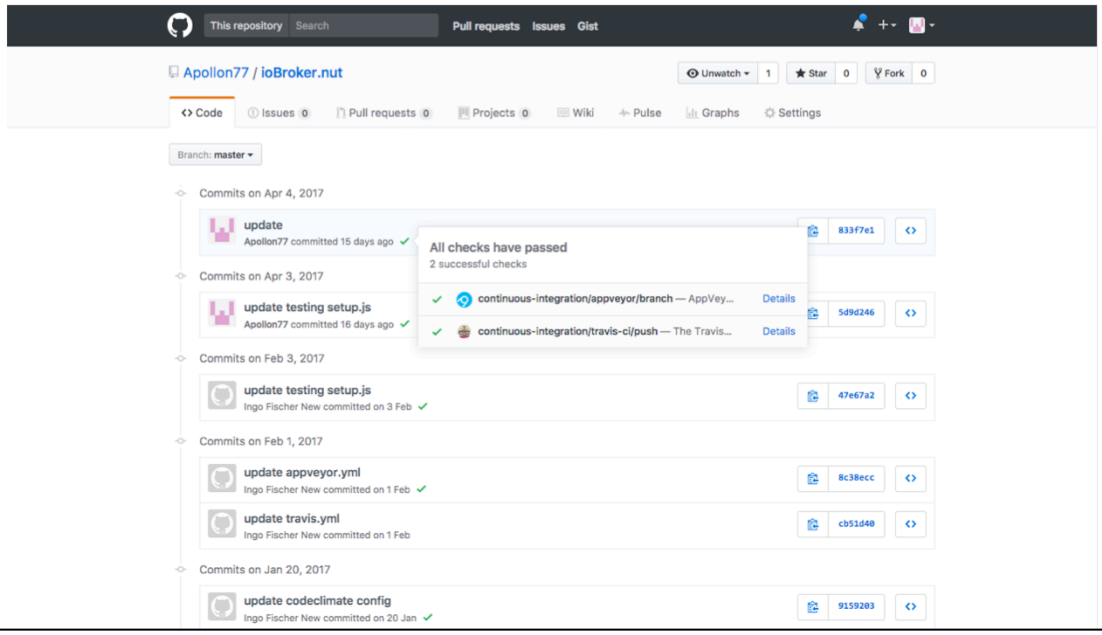
Adapter Testing

- Wozu Adapter Tests?
  - ... Sicherstellen das die Installation des Adapters und der Start klappen
  - ... Sicherstellen das alle nötigen npm-Abhängigkeiten aufgelöst werden können
  - ... Sicherstellen das alle unterstützen nodejs Versionen (+ ggf. zukünftige) funktionieren
    - ... aktuell 0.10 / 0.12 / 4.x / 6.x / 7.x
  - ... Sicherstellen das Adapter unter verschiedenen Betriebssystemen läuft
    - ... Linux, MacOS (Travis-CI)
    - ... Windows (Appveyor)
  - ... Sicherstellen das Package-Files valide und in sync (Versionsnummern) sind
  - ... Unterstützung bei Fehlersuche auf Kombinationen die man selbst nicht installiert hat
  - ... Erweiterbar um beliebige Adapterspezifische Tests,

- Was muss man tun?
  - ... Testing Konfiguration für CI-Systeme (.travis.yml / appveyor,yml) anlegen
  - ... Mit GitHub-Account bei https://travis-ci.org/ und https://ci.appveyor.com/ anmelden und GitHub-Projekte dort für automatische Tests aktivieren
  - ... Einen Commit durchführen
  - ... Testergebnisse ansehen (auch in GitHub beim Commit verlinkt!)

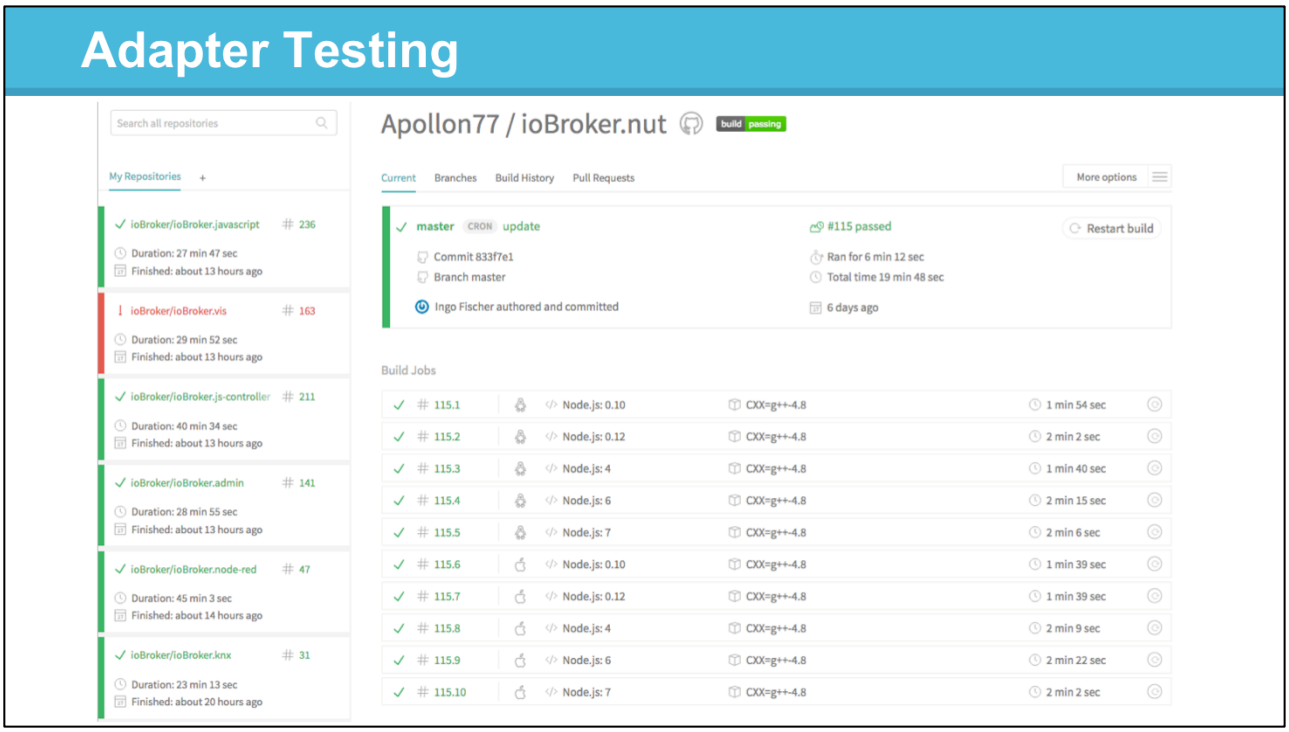
- Bevor ein Adapter aber wirklich fertig ist sollte er noch Basis-testing per Travis-CI und Appveyor haben.
- Das sind Continous Integration Systeme die mit GitHub verknüpft die Testa automatisch nach jedem commit starten.
- Appveyor installiert und startet den Adapter unter Windows und Travis unter Linux und MacOS und jeweils unter den nodejs Versionen die man angibt.
- Man sieht als Entwickler also sehr schnell ob man irgendwas grundlegendes bei einer Änderung kaputt gemacht hat.
- Eigene Adapterspezifische tests sind auch schnell erweitert.

Adapter Testing



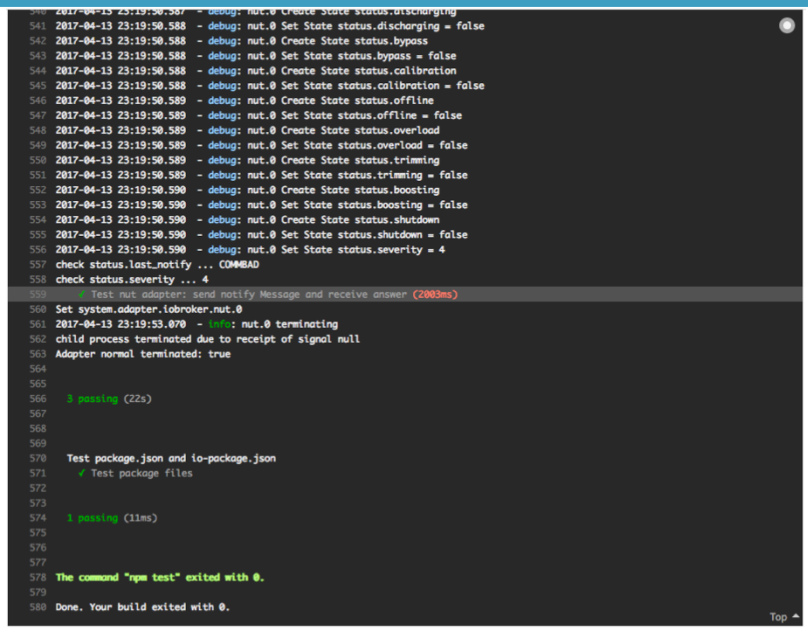
- In GitHub sind bei jedem commit die Testergebnisse verlinkt

Adapter Testing



- So sieht es in Travis-CI aus

Adapter Testing



- Und so ein Travis-Ausführungs-Shell-Log

Community Testing

- Erzähle der Community vom neuen Adapter und finde Tester
  - ... Forum-Thread unter „Tester“ und ggf. „Adapters - Allgemein“ eröffnen
  - ... Adapter vorstellen, was kann er, welches Problem löst er, welches Device er anbindet
  - ... ggf. in anderen Forum-Threads zu ähnlichen Themen verlinken
- Installation im ersten Schritt von GitHub reicht aus
- Im Forum präsent sein und Support geben

- Jetzt geht’s in die Community
- Erzählt was vom neuen Adapter im Forum und lasst andere Nutzer testen. Sammelt Feedback und verbessert den Adapter damit.
- Und vor allem nutzen diesen erweiterten Tester-Kreis um Fehler zu finden oder zu beheben
- Die Installation auf Basis der GitHub URL geht per Admin-Bereich für alle sehr einfach
- Wenn der Adapter dann auch bei anderen gut funktioniert und gutes Feedback bekommt ist es Zeit alles für die Aufnahme ins offizielle Adapter Repository zu tun

Adapter für ioBroker-Repository einreichen

- GitHub README mit Informationen (englisch oder deutsch), Doku und Versions-Historie erstellen
- Aktuellste Version auf NPM veröffentlichen
  ... npm Account einrichten (einmalig)
  https://docs.npmjs.com/getting-started/publishing-npm-packages
  ... npm publish
- Adapter Testing (mind. Basic-Testing per Travis-CI/Appveyor) eingebaut und „grün“
- Auf GitHub https://github.com/ioBroker/ioBroker.repositories forken
  ... eigenen Adapter in sources-dist.json (Latest Repository) einfügen
  ... Nach Rücksprache ggf. auch in sources-dist-stable.json (Stable Repository)
  ... Repo-Änderungen als Pull-Request senden
  ... Testing-Thread-Link aus Forum mit dazu schreiben!
  ... Adapter-Kategorie vorschlagen

- Spätestens jetzt muss der Adapter als NPM Modul verfügbar sein. Einmalig npm Account machen, lokal einrichten und dann per npm publish das Modul veröffentlichen
- Auch sollte das Adapter-Testing eingebaut sein und grün sein. Auf das Thema konnte ich heute leider nicht eingehen.
- Dann für die Eintragung ins Repository das GitHub-Projekt ioBroker.repositories forken und dort den eigenen Adapter im latest Repository eintragen
- Hier wird auch die Kategorie festgelegt in der der Adapter später aufgelistet wird.
- Dann diese Änderung als Pull-Request einreichen, am besten mit Link zum Testing-Thread aus dem Forum.
- Das wars auch schon ...
- Danach muss man seinen Adapter natürlich betreuen und auf Feedback und Fehlerberichte eingehen ...
- Ansonsten kann ich aber nur sagen ...

Betreue Deinen Adapter

- Auf User-Feedback reagieren
  - ... GitHub-Issues und –Pull-Requests
  - ... Forum-Threads in „Fehler“ bzw. „Adapter – Allgemein“
  - ... PNs im Forum
  - ... Trello-Tickets
  - ... Wenn man nicht überall präsent sein kann/will, klar sagen über welchen Kanal man Feedback annimmt!

- Änderungen Transparent machen
  - ... Semantic Versioning (major.minor.micro)
  - ... Versionsnummern je nach Typ der Änderung pflegen
  - ... Changelog in io-package.json (Anzeige+Rollback-Möglichkeiten im Admin) pflegen
  - ... Changelog in README.md pflegen

Naja, noch nicht ganz.
Das betreuen des Adapters danach, Bugfixes und Weiterentwicklung ist genauso wichtig

Was kann ein Adapter noch so?

- „Discovery“-Integration
  ... Finden der unterstützten Geräte zur automatischen Adapter-Installation & -Konfiguration
- Vis-Widgets für eigene Datenpunkte anbieten
- Adapter-Infos für die ioBroker-Webseite im Adapter hinterlegen

- Wenn man den Adapter dann veröffentlicht hat kann man sich noch ein paar Extras anschauen ... Aber das würde heute hier zu weit gehen.
- Und das allerwichtigste ist ...

Fragen ???

Habt Spass beim Programmieren !!

So schwierig ist es nicht ☺
(und falls doch hilft das Forum!)

- Habt Spass beim Programmieren und keine Angst.
- Lasst uns gemeinsam ioBroker erfolgreich machen.
- JavaScript ist nicht so anders und auch das asynchrone bekommt man in den Griff.
- Und jetzt haben wir hoffentlich noch ein par Minuten für Fragen ...

Thank you!

Ingo Fischer

Links

- Core Concepts
  https://github.com/ioBroker/ioBroker/blob/master/doc/SCHEMA.md
- Adapter Development Documentation
  https://github.com/ioBroker/ioBroker/wiki/Adapter-Development-Documentation
- Setup Dev Environment and IDE
  https://github.com/ioBroker/ioBroker/wiki/Installation,-setup-and-first-steps-with-an-ioBroker-Development-Environment
- Travis-CI/Appveyor
  http://forum.iobroker.net/viewtopic.php?f=24&t=4334

- Admin „How to show Help to users“
  https://github.com/ioBroker/ioBroker.admin/blob/master/DEVELOPER.md

- Adapter-Templates
  https://github.com/ioBroker/ioBroker.template
  https://github.com/ioBroker/ioBroker.template-rest
  https://github.com/ioBroker/ioBroker.vis-template