# StEmp-MV Documentation

**Reiner Lemoine Institut**

**20.05.2020**

# Inhalt

StEmp-MV ist ein Stakeholder-Empowerment-Tool für die Gemeinde Groß-Trebbow. Es wurde vom Reiner Lemoine Institut (RLI) im Rahmen des Kopernikus-Projekts „ENavi“ entwickelt. Sie finden das Tool auf den WAM-Seiten des RLI.

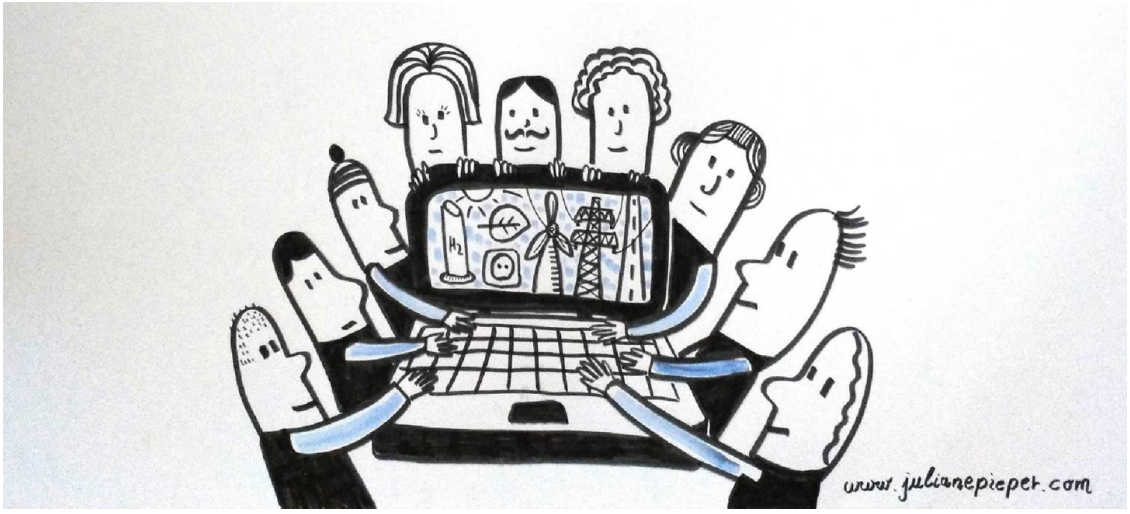



Abb. 1: © Juliane Pieper | Reiner Lemoine Institut

In dieser Dokumentation finden Sie methodische und technische Hintergrundinformationen und Anleitungen.

KAPITEL 1

# Über dieses Tool

Die Energiewende kann nur gemeinsam erreicht werden. Wie aber können alle Interessensgruppen ihre Sichtweisen und Bedürfnisse in die Energiewendeplanung einbringen? Das Reiner Lemoine Institut (RLI) hat dieses Stakeholder-Empowerment-Tool entwickelt, das es Hausbesitzerinnen und Hausbesitzern ermöglicht, in ihrem Wirkungsraum einen Betrag zur Energiewende zu leisten, indem sie bei Neuinvestitionen in Heizungsanlagen Technologieabwägungen auf wissenschaftlicher Grundlage treffen können.


Menu
Welches Szenario möchten Sie erstellen?
Einzelhaus
Viertel
Abbrechen


Menu
Viertel erstellen
1
2
3
4
HEIZUNGSOPTIONEN
Lützow
BHKW
PV + Wärmepumpe
Ölheizung
Gasheizung
Zurück
Weiter
Abbrechen


Über das Menü links gelangen Sie auf die entsprechenden Seiten.

## 1.1 Motivation

Mecklenburg nimmt als Modellregion im Projekt ENavi teil. Im Landkreis Mecklenburg gibt es bereits verschiedene Aktivitäten im Bereich Energiewende und Daseinsvorsorge. Im Frühjahr 2018 mit der Erstellung eines Klimaschutzkonzeptes begonnen, durch den Komob e.V. wird ländliche Mobilität neu gedacht und es gibt einen Amtsentwicklungsausschuss AG Daseinsvorsorge die sich mit der Zukunft der Region beschäftigt. In diesen Kontext reiht sich das Stemp Tool ein. In Zusammenarbeit mit dem LK Mecklenburg und KOMOB e.V. wurden Quartiere in Nordwest-Mecklenburg identifiziert, die sich für die gemeinsame Entwicklung des Tools eignen.

## 1.2 Hintergrund

Die ländliche, strukturschwache Region mit abnehmender Bevölkerungszahl, hat die Energiewende in der Region bisher so erlebt: viele Windparks, wenig regionale Wertschöpfung, wenig Partizipationsmöglichkeiten. Im Bereich Wärmeversorgung wurde Anfang der 1990er Jahre in vielen Gebäuden in der Region die Wärmeversorgung erneuert, meist durch dezentrale Öl- oder Gasthermen. Nach fast 30 Jahren Nutzung steht in den kommenden Jahren eine Erneuerung an. An dieser Stelle soll das Stemp-Tool eingesetzt werden und den Entscheidungsprozess zu einer neuen Wärmeversorgung unterstützen. Hier ist jetzt der günstige Moment, eine Investitionsentscheidung die ansteht zu Nutzen um die Wärmewende ein Stück weiter voranzubringen.

## 1.3 Ziele

Mit dem Stemp - Tool für die Region Nordwest-Mecklenburg verfolgen das Reiner Lemoine Institut und der Landkreis folgende Ziele:

- Identifikation der Werte und Interessen der regionalen Akteure
- Verbesserung des Verständnisses von Zusammenhängen zwischen technischen, energiewirtschaftlichen und ökologischen Eigenschaften von Heizungssystemen

- Transparente und interessensneutrale Bereitstellung von Informationen zu verschiedenen Heizungstechnologien auf wissenschaftlicher Grundlage aber in der Darstellung an die Zivilgesellschaft angepasst.
- Befähigung, bessere Entscheidungen treffen zu können
- Beitrag zu ‚Daseinsvorsorge' damit die Region durch die Energiewende nachhaltig profitieren kann anstatt abgehängt zu werden
- Bereitstellung von Daten und Quellcode, um die Umsetzung ähnlicher Tools zu erleichtern.

## 1.4 ENavi

Das Tool wurde im Kopernikus-Projekt „ENavi" entwickelt, einem von vier Projekten der Förderinitiative Kopernikus des Bundesministeriums für Bildung und Forschung (BMBF).

Förderkennzeichen: 03SFK4E1

```
„Mit der Energiewende hat sich Deutschland zum Ziel gesetzt, das gegenwärtige
Energiesystem in ein weitgehend CO2-freies und auf erneuerbaren Energien
basierendes System zu transformieren. Ein wirtschaftliches,
umweltverträgliches, verlässliches und sozialverträgliches Energiesystem
benötigt eine ganzheitliche Betrachtung auf Systemebene. ENavi sieht die
Energiewende daher als einen gesamtgesellschaftlichen Transformationsprozess
und verknüpft wissenschaftliche Analysen mit politisch-gesellschaftlichen
Anforderungen."
```

## 1.5 Lizenz

KAPITEL 2

# Was ist ein StEmp-Tool?

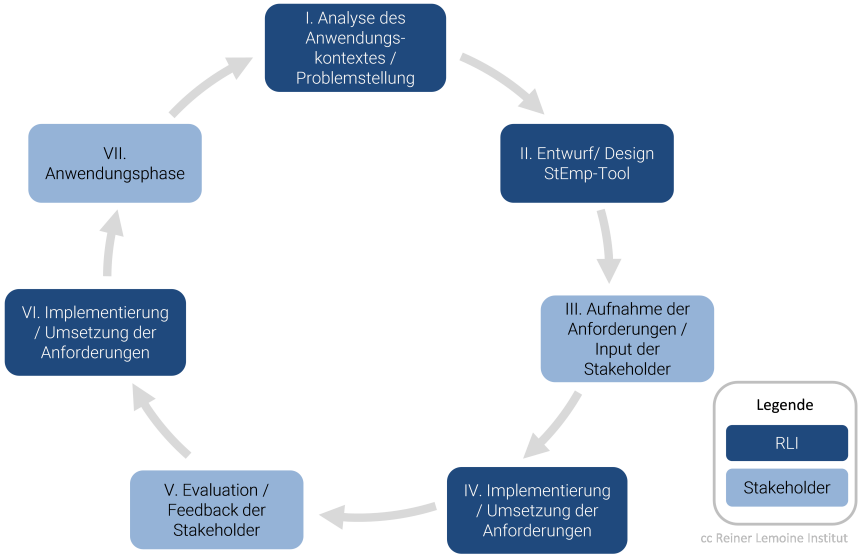
Ein Stakeholder-Empowerment-Tool - kurz StEmp-Tool - ist ein fachliches Berechnungsprogramm, das verschiedene Akteure auf verständlicher Basis unterstützen soll. Ziel ist es dabei, eine Grundlage für die Partizipation unterschiedlicher Interessensgruppen zu schaffen, indem relevante Daten so aufbereitet und dargestellt werden, dass sie einfach zugänglich und verständlich sind. Dies ermöglicht eine fundierte Diskussion auf Augenhöhe, die auf Daten und Fakten basiert.

## 2.1 Hintergrund

Für eine erfolgreiche Energiewende ist die Partizipation der Bürger und Einbeziehung unterschiedlichster Interessensgruppen von zentraler Bedeutung. Leider werden Diskussionen zum Teil auf sehr emotionaler Ebene geführt und führen so zu scheinbar unlösbaren Konflikten. Um einen Diskurs möglichst frei von Interpretationen zu ermöglichen, ist es daher wichtig, allen Beteiligten die zugrundeliegenden Daten und Verordnungen zugänglich zu machen und verständlich aufzubereiten. Diesem Zweck dienen die StEmp-Tools, die informieren und so dazu befähigen sollen, einen fundierten Diskurs zu führen.

## 2.2 Entwicklungsprozess

Die Tools wurden iterativ und in enger Zusammenarbeit mit der entsprechenden Stakeholder-Gruppe konzipiert und weiterentwickelt. Wie in der Abbildung dargestellt, wurde das Tools nach dem ersten Entwurf wiederholt von den Stakeholdern getestet und Verbesserungsvorschläge angebracht. Diese wurden seitens des RLI umgesetzt, um dann einer erneuten Testphase unterzogen zu werden. Durch diese enge Einbindung der Zielgruppe ist gewährleistet, dass die erstellte Software dem Bedarf angepasst und benutzerfreundlich gestaltet ist.


I. Analyse des Anwendungs-kontextes / Problemstellung
II. Entwurf/ Design StEmp-Tool
III. Aufnahme der Anforderungen / Input der Stakeholder
IV. Implementierung / Umsetzung der Anforderungen
V. Evaluation / Feedback der Stakeholder
VI. Implementierung / Umsetzung der Anforderungen
VII. Anwendungsphase
Legende
RLI
Stakeholder
cc Reiner Lemoine Institut

KAPITEL 3

# Verwendung

Eine kurze Einführung in das Tool ist bereits auf der Startseite gegeben. Im Folgenden werden die wichtigsten Elemente und Funktionen der Benutzeroberfläche beschrieben.

## 3.1 Aufbau des Tools

Das Tool führt schrittweise von der Bestimmung des Wärmebedarfs eines Haushalts oder Viertels hin zum Vergleich verschiedener Heizungsoptionen. Die Schritte in denen durch das Tool geführt wird sind Folgende:

1. Erstellung eines Haushalts **oder**
2. Erstellung eines Viertels (bestehend aus mehreren Haushalten)
3. Auswahl der zu vergleichenden Heizungstechnologien
4. Anpassung der Parameter der verwendetenen Heizungstechnologien
5. Zusammenfassung der vorherigen Schritte
6. Berechnung der Ergebnisse und Vergleich der Szenarien

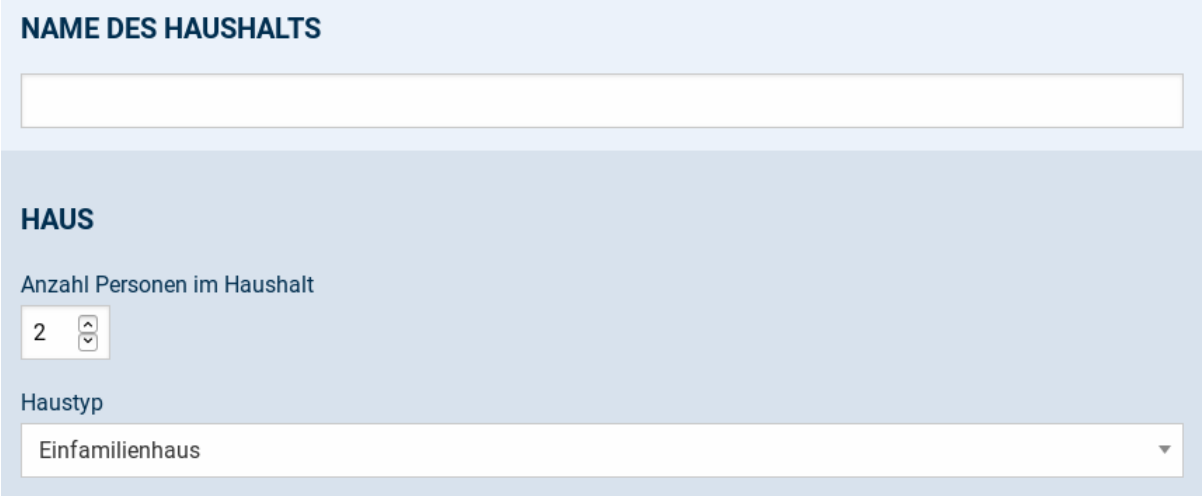
Die Schritte werden im folgenden genauer erläutert.

## 3.2 Erstellung eines Haushalts

Ein einzelner Haushalt kann entweder mit individuellen Angaben erstellt werden, oder als vordefinierter bzw. gespeicherter Haushalt aus einer Liste gewählt werden:

Soll ein Haushalt aus einer **Liste** gewählt werden, erfolgt dies über den Namen des Haushalts. Nach Auswahl eines Haushaltes werden zusätzlich die Angaben zur Anzahl der Personen, verfügbarer Dachfläche und Energiebedarf (Heizung und Warmwasser getrennt) eingeblendet:

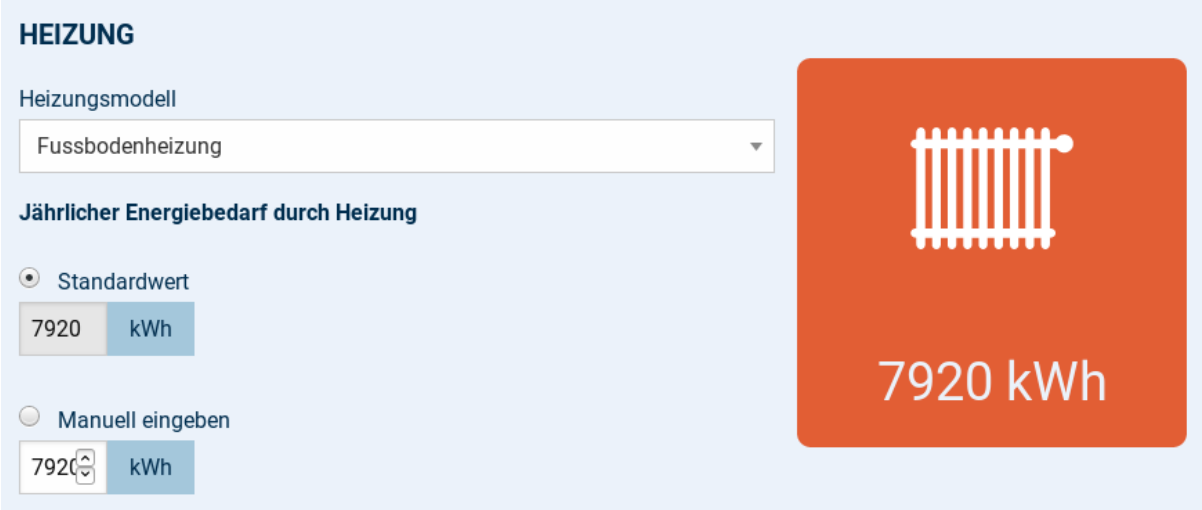

Bei der **benutzerdefinierten** Erstellung eines Haushalts, wird zuerst ein eindeutiger Name vergeben. Anschließend werden Angaben zur Anzahl der Personen und zum Haustyp (Einfamilienhaus/Mehrfamilienhaus) beantwortet:




Basierend auf der Anzahl der Personen werden automatisch standardisierte Werte für die folgenden Angaben (Grundfläche, Energiebedarf, usw.) ermittelt und zur Verfügung gestellt. Grundsätzlich ist der Standardwert für alle Angaben ausgewählt. Abweichend vom Standardwert können benutzerdefinierte Werte eingegeben werden, indem das Feld „Manuell eingeben“ ausgewählt wird und der dortige Wert entsprechend angepasst wird (hier am Beispiel der Grundfläche):

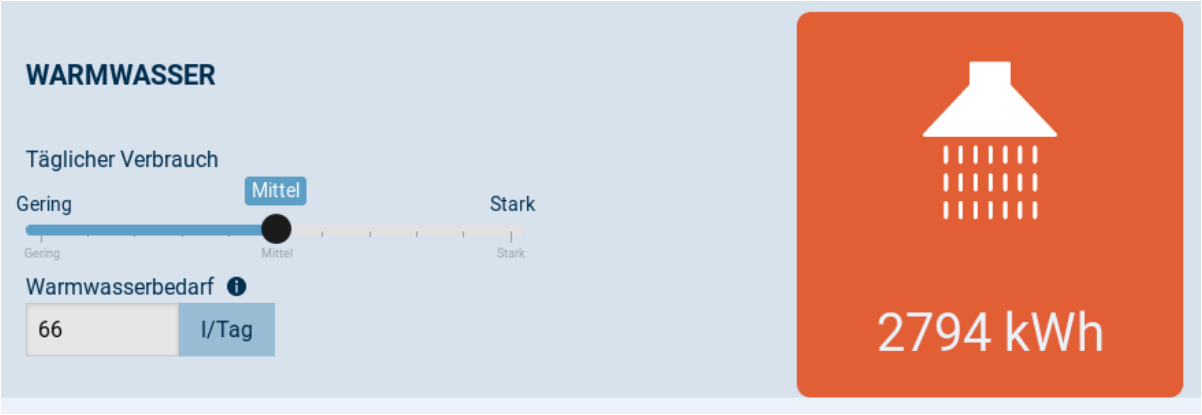
Weiterhin werden folgende Angaben ermittelt:

**Heizung:** Der Energiebedarf durch die benötigte Heizwärme kann eingestellt werden. Außerdem kann der eingebaute Heizungstyp (Heizkörper oder Fussbodenheizung) angegeben werden. Der Energiebedarf pro Jahr wird zusätzlich rechts noch einmal eingeblendet.




**Bemerkung:** Der Heizungstyp wird in der aktuellen Version des Tools nur dazu verwendet, um eine Warnung einzublenden falls die Heizungsoption „Wärmepumpe“ in Verbindung mit dem Heizungstyp „Heizkörper“ ausgewählt wurde (Diese Kombination ist technisch schwieriger umzusetzen).

**Warmwasser:** Der Warmwasserverbrauch pro Tag kann mittels dreier Stufen (gering, mittel, stark) eingestellt werden. Der resultierende Energiebedarf pro Jahr ist rechts eingeblendet.



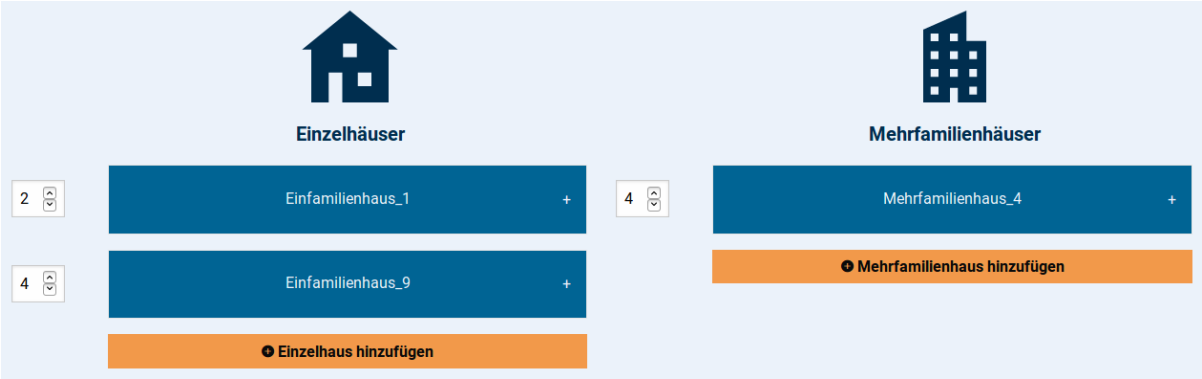

**Dachfläche:** Die verfügbare Dachfläche kann angegeben werden. Diese wird benötigt, um eine Obergrenze für die mögliche Größe einer zu installierenden Photovoltaikanlage berechnen zu können.

HH_dachfläche.png

## 3.3 Erstellung eines Viertels

Die Erstellung eines Viertels erfolgt, indem mehrere Haushalte zu einem Viertel zusammengefasst werden. Dazu können beliebig viele Einfamilienhäuser und Mehrfamilienhäuser dem zunächst leeren Viertel hinzugefügt werden. Alternativ kann ein bestehendes Viertel geladen werden und anschließend angepasst werden:

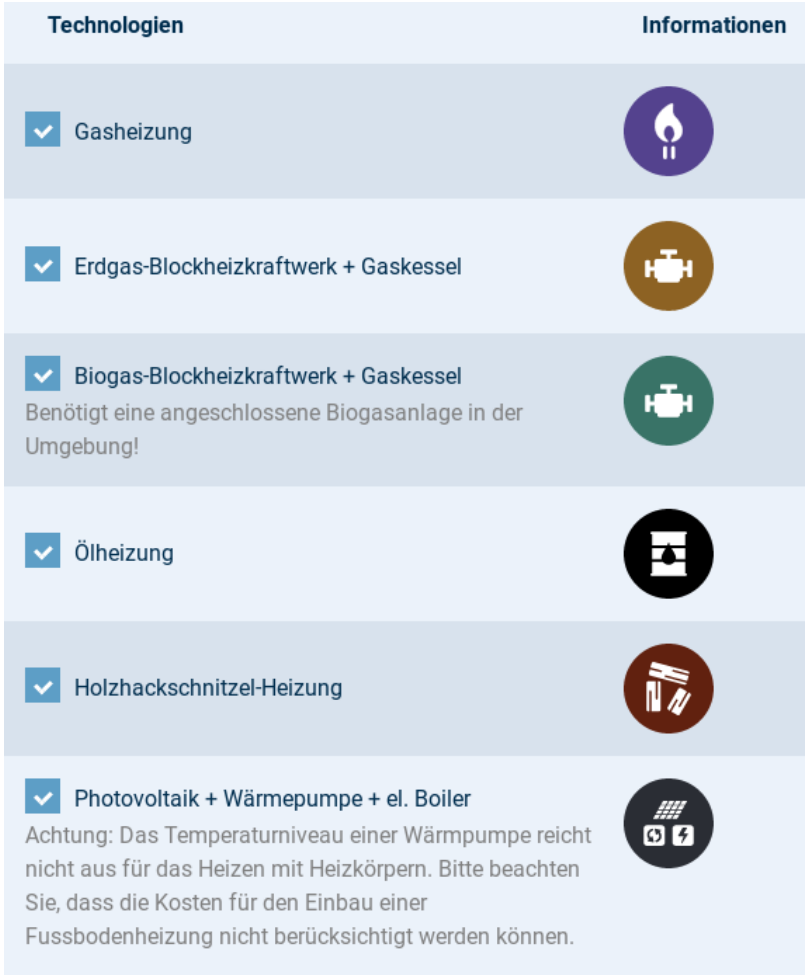

Bei Klick auf „Einfamilienhaushalt hinzufügen“ bzw. „Mehrfamilienhaushalt hinzufügen“ öffnet sich die Seite zur Erstellung eines Haushalts. Dort kann, wie unter *Erstellung eines Haushalts* beschrieben, ein Haushalt erstellt oder aus einer Liste geladen werden. Anschließend wird der gewählte Haushalt dem Viertel hinzugefügt. Die Anzahl eines Haushalts kann zusätzlich angepasst werden; der Haushalt ist entsprechend oft im Viertel vorhanden und der Energiebedarf ist entsprechend erhöht:




**Bemerkung:** Die Möglichkeit der Einstellung der Haushaltsanzahl soll eine archetypische Generierung verschiedener Haushaltstypen ermöglichen, ohne jeden Haushalt einzeln erstellen zu müssen.

## 3.4 Auswahl der Heizungstechnologien

In diesem Schritt werden die zu vergleichenden Technologien ausgewählt. Eventuelle Einschränkungen und Hinweise sind den Technologien in grau hinzugefügt:




Zur Verfügung stehen in der aktuellen Version folgende Technologien:

- Gasheizung
- Erdgas-Blockheizkraftwerk + Gaskessel
- Biogas-Blockheizkraftwerk + Gaskessel
- Ölheizung
- Holzhackschnitzel-Heizung
- Photovoltaik + Wärmepumpe + elektrischer Boiler

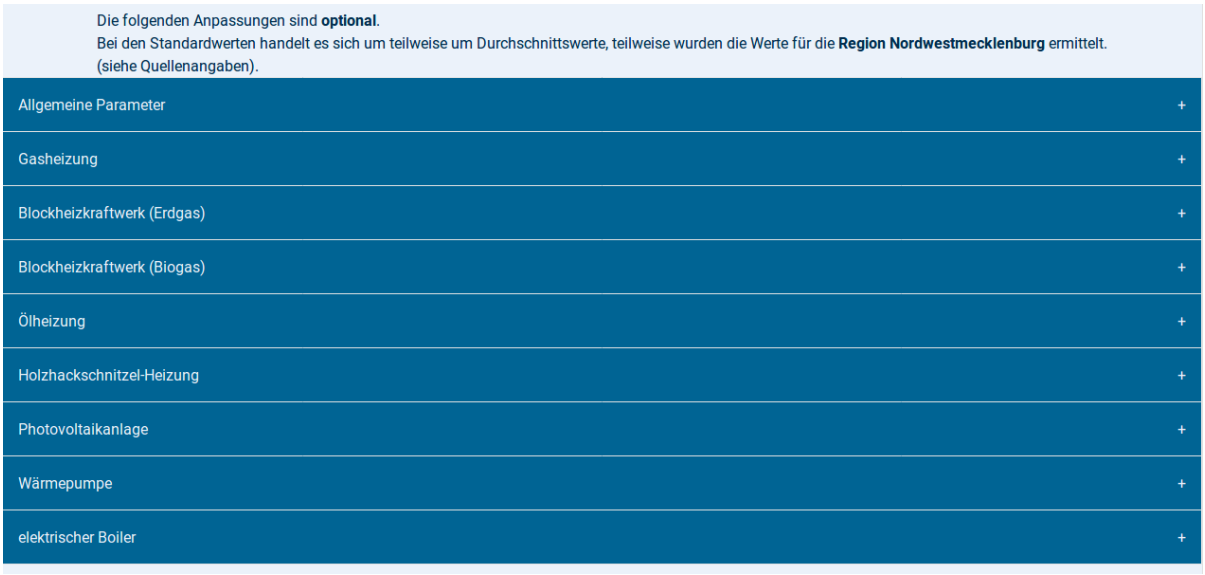
Erläuterungen zur Funktionsweise der einzelnen Technologien können per Klick auf das jeweilige Technologie-Symbol eingesehen werden. Die Technologien und die daraus resultierenden Szenarien werden unter *Energiesystem* genauer betrachtet. Für alle ausgewählten Technologien können anschließend die Parameter angepasst werden (siehe *Anpassung der Parameter*). Auf der Ergebnisseite (siehe *Ergebnisse*) werden schließlich die ausgewählten Technologien untereinander verglichen.

## 3.5 Anpassung der Parameter

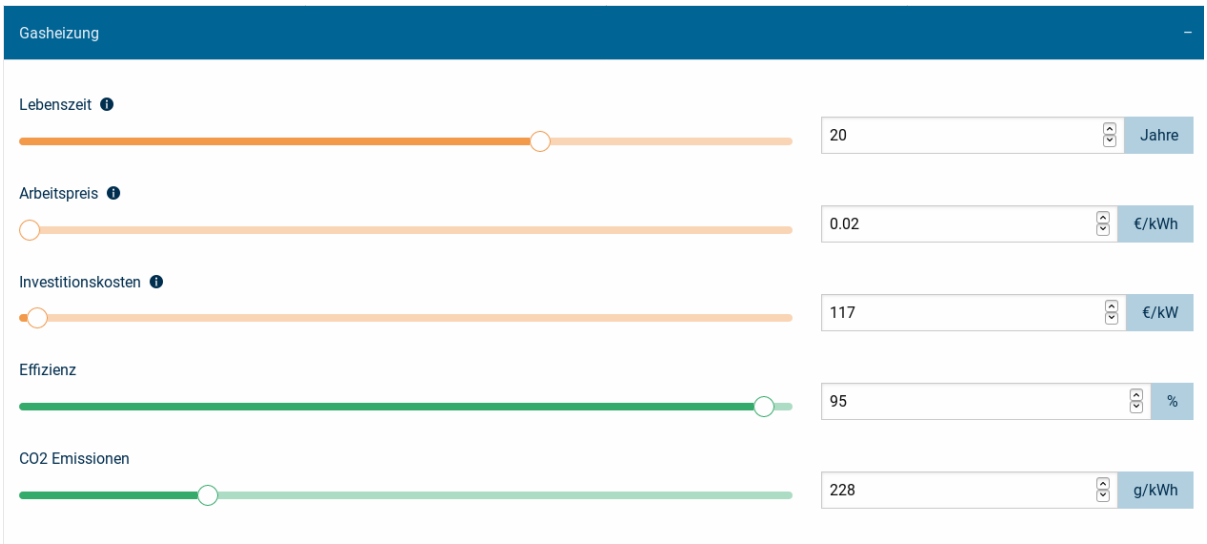
**Bemerkung:** [Expertenmodus] Alle Parameter sind auf Standardwerte eingestellt, die nach bestem Wissen und Gewissen recherchiert wurden (siehe auch Quellenangaben dazu). Änderungen an den Parametern sollten nur von „Exper-

ten“ vorgenommen werden. Für die Verlässlichkeit der daraus resultierenden Ergebnisse können wir nicht garantieren.

Alle einstellbaren Parameter sind nach Technologien gruppiert:




Per Klick auf eine Technologie, öffnen sich die zugehörigen Parameter und können angepasst werden:

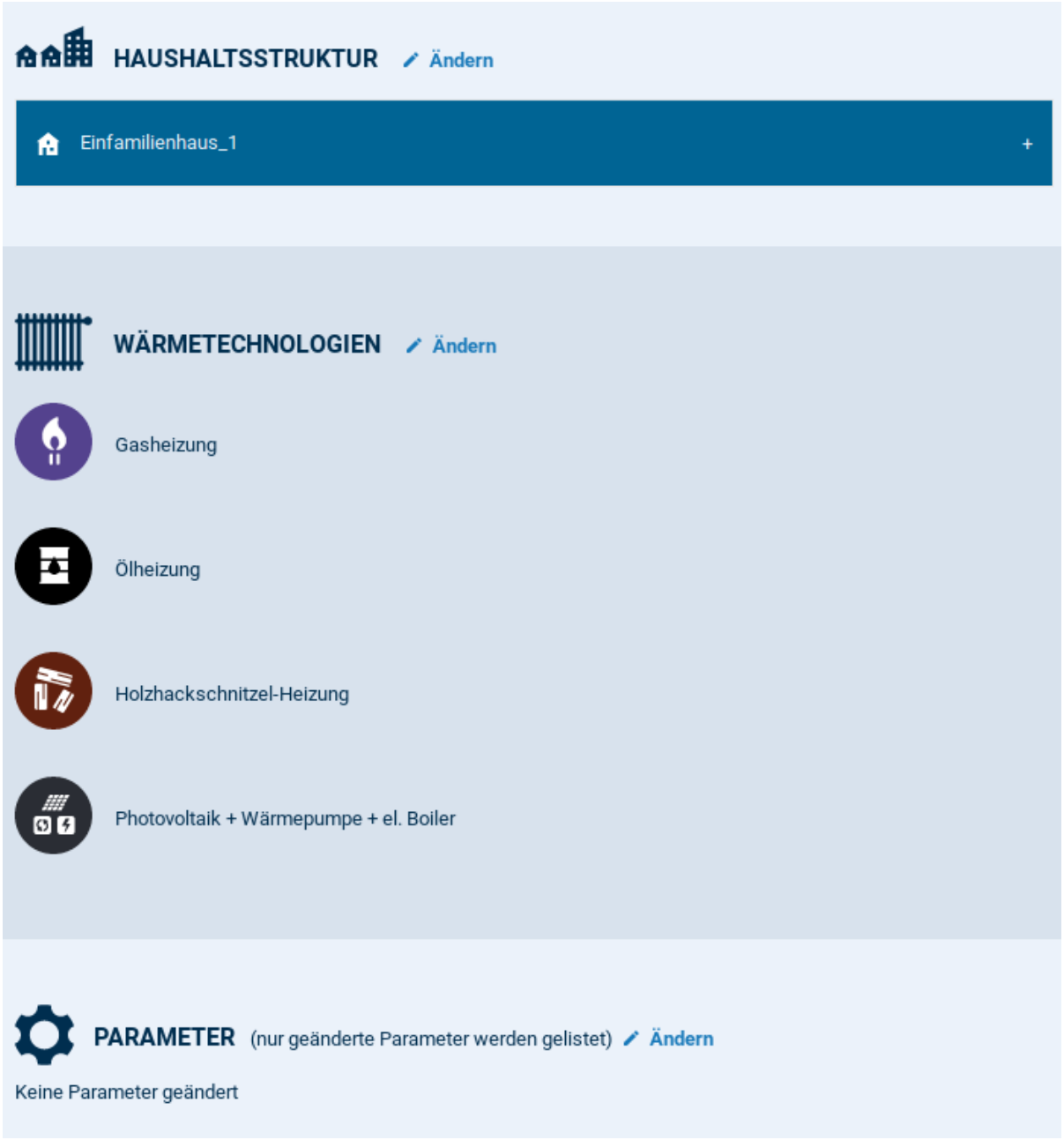



Die Parameter werden farblich unterschieden zwischen orange - entspricht ökonomischen Einstellungen - und grün - entspricht technologischen Einstellungen. Der Wert der jeweiligen Einstellung kann mittels Schieberegler oder per manueller Eingabe daneben angepasst werden. Bei der manuellen Eingabe sind minimal und maximal Werte zu beachten - die Einstellungen werden nach der Eingabe automatisch angepasst. Die zughörige Einheit des Parameters ist in der blauen Box rechts angegeben. Außerdem sind für einige Einstellungen kuzre Erläuterungen zum besseren Verständnis hinterlegt (?).

## 3.6 Zusammenfassung

In der Zusammenfassung werden alle Einstellungen des Szenarios (Haushalt/Viertel, Technologien und Parameter) aufgelistet. Ein Klick auf „Fertig“ startet anschließend die Berechnung der zugehörigen Ergebnisse. Sollte es noch

Änderungsbedarf an den Einstellungen geben, kann über das Feld „Ändern“ zu den jeweiligen Einstellungen zurückgesprungen werden.




**Bemerkung:** Werden Änderungen an vorherigen Einstellungen vorgenommen, werden in der aktuellen Version leider keine späteren Änderungen gespeichert. Dass heißt, werden Änderungen am Haushalt vorgenommen, müssen anschließend gegebenenfalls erneut Technologien und zugehörige Parameter angepasst werden. In einer zukünftigen Version soll dafür Abhilfe geschaffen werden.

## 3.7 Ergebnisse

Auf der Ergebnisseite werden die ausgewählten Technologien miteinander verglichen. Zur Zeit werden folgende Ergebnisse für jedes Szenario tabellarisch ausgegeben:

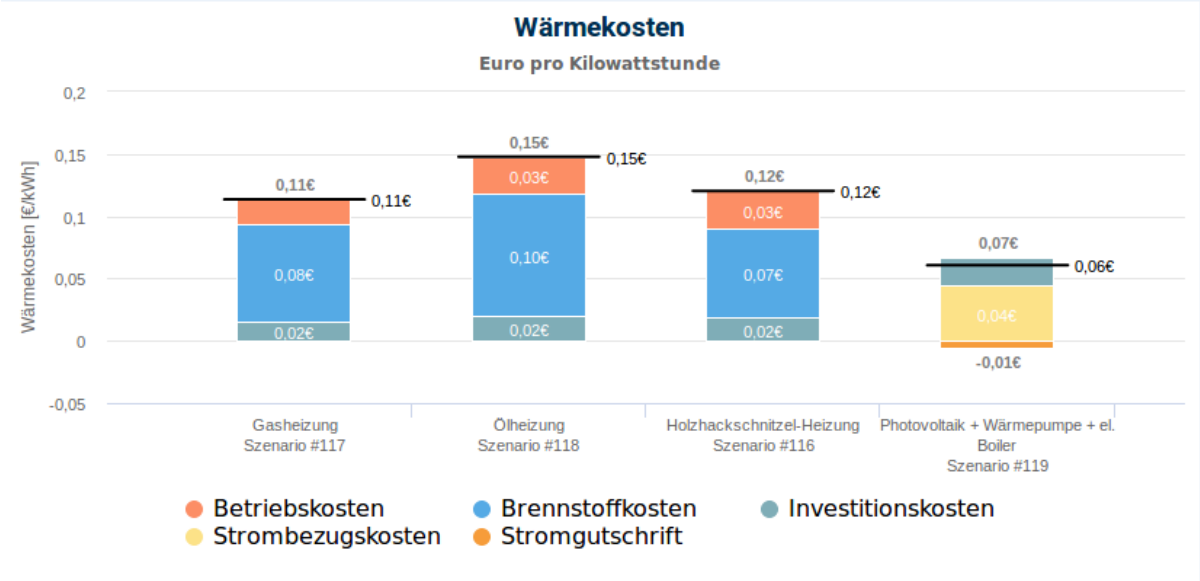
- Wärmekosten
- Investitionskosten
- Brennstoffkosten
- CO2 Emissionen
- Primärenergiefaktor
- Primärenergie

Zusätzlich werden allgemeine „weiche“ Faktoren als Vor- und Nachteile stichpunktartig ergänzt. Erläuterungen zu den einzelnen Vergleichen können über das ?-Symbol eingeholt werden. Die Technologien werden qualitativ verglichen; dazu werden die Technologien farblich als qualitativ „gut“ (grün), „neutral“ (beige) und „schlecht“ (rot) markiert:

| | **Gasheizung**<br>Szenario #117 | **Ölheizung**<br>Szenario #118 | **Holzhackschnitzel-Heizung**<br>Szenario #116 | **Photovoltaik + Wärmepumpe + el. Boiler**<br>Szenario #119 |
|---|---|---|---|---|
| **Wärmekosten** | 0,11 €/kWh | 0,15 €/kWh | 0,12 €/kWh | 0,06 €/kWh |
| **Investitionskosten** | 1.197 € | 1.539 € | 1.522 € | 1.540 € |
| **Brennstoffkosten** | 415 €/Jahr | 521 €/Jahr | 377 €/Jahr | 0 €/Jahr |
| **CO2 Emissionen** | 240 g/kWh | 351 g/kWh | 27 g/kWh | 116 g/kWh |
| **Primärenergiefaktor** | 1,3 (1,4)* | 1,3 (1,5)* | 0,5 | 0,5 |
| **Primärenergie** | 7.413 kWh | 7.826 kWh | 2.451 kWh | 2.114 kWh |
| **Vorteile** | *Hohe Effizienz und niedrige Heizkosten durch Brennwert-Nutzung*<br>*Kompakte Bauart*<br>*Geringe Investitionskosten durch ausgereifte Technik* | *Hoher Wirkungsgrad*<br>*Sicherer Betrieb durch ausgereifte Technik* | *Betrieb mit nachwachsendem Holz*<br>*Automatischer Betrieb durch genormte Pellets möglich* | *Nutzung erneuerbarer Energien*<br>*Unabhängigkeit von fossilen Energieträgern* |
| **Nachteile** | *Verbraucht fossile Energieträger*<br>*Ein Gas-Anschluss muss vorhanden oder möglich sein*<br>*Entwicklung der Gas-Preise nicht absehbar* | *Arbeitet überwiegend mit fossilen Energieträgern*<br>*Tankanlage braucht viel Platz*<br>*Ölpreisentwicklung nicht absehbar* | *Lagerplatz für Pellets*<br>*Regelmäßiges Asche-Austragen notwendig* | *Höhere Anschaffungskosten*<br>*Aufwendige Planung* |

**Warnung:** Der qualitative Vergleich findet nur unter den ausgewählten Szenarien statt! Generelle qualitative Aussagen über einzelne Szenarien können dadurch nicht getroffen werden. Werden zum Beispiel nur Technologien mit sehr hohen Investitionskosten verglichen, werden die Investionskosten einer Technologie als „gut“ eingestuft, obwohl diese im Vergleich mit einer anderen (nicht ausgwählten) Technologie eventuell sehr viel schlechter wären.

Die Wärmekosten werden zusätzlich als Grafik angezeigt, wobei die Kosten zusätzlich nach Betriebskosten, Brennstoffkosten, Investionskosten und Strombezugskosten aufgeschlüsselt werden, sowie mögliche Stromgutschriften von den Gesamtkosten abgezogen werden:




**Bemerkung:** Die tatsächlichen Wärmekosten jeder Technologie sind als schwarze Linie eingezeichnet. Diese sind notwendig um „negative Kosten“, in diesem Falle also eine Stromgutschrift, berücksichtigen zu können.

**Bemerkung:** Jedes Szenario erhält intern eine eindeutige ID, über diese ID können Szenarien miteinander verglichen werden. Zur Zeit kann dies leider nur manuell über den Browser geschehen, indem die zu vergleichenden Szenarien-IDs als komma-getrennte Liste zu der Ergebnis-URL hinzugeügt werden (z.B. https://wam.rl-institut.de/stemp/result/117,118,116,119 )

KAPITEL 4

# Installation

Das Tool kann bei Bedarf auch lokal eingerichtet werden. Zur Vereinfachung kann das Tool mittels Docker eingerichtet werden.

## 4.1 Voraussetzungen

- git ist installiert.
- Docker und Docker-Compose sind installiert.

## 4.2 Installation

Zunächst muss ein WAM-Server mittels Docker aufgesetzt werden. Anschließend kann das StEmp-MV Tool integriert werden.

### 4.2.1 Installation der WAM

Für die Installation wird folgende Ordnerstruktur eingerichtet:

```
wam_docker (Name beliebig)
+-- docker (Enthält Docker Konfiguration und WAM-Codebasis)
|   +-- docker-compose.yml
|   +-- WAM (WAM-Codebasis; hier können später weitere Apps integriert werden)
+-- config (Konfiguration für WAM und integrierte Apps)
|   +-- config.cfg
```

**Bemerkung:** Änderungen an dieser Struktur müssen in der Konfigurationsdatei bzw. in der Docker-Konfiguration (docker-compose.yml) berücksichtigt werden.

Die Einrichtung der Ordnerstruktur geschieht folgendermaßen:

```
mkdir wam_docker
cd wam_docker
mkdir docker
mkdir config
cd docker
git clone https://github.com/rl-institut/WAM.git
cp WAM/docker-compose.yml .
cp WAM/.config/config.cfg ../config/
```

Anschließend müssen die beiden Konfigurationsdateien (*docker-compose.yml* und *config.cfg*) angepasst werden. Abschließend werden mit folgendem Befehl (aus dem *docker*-Ordner heraus) die Images erstellt und die Container gestartet:

```
sudo docker-compose up -d --build
```

Der WAM-Server sollte nun unter 127.0.0.1:5000 erreichbar sein.

### 4.2.2 Einbindung der StEmp-MV App

Die StEmp-MV App kann nun folgendermaßen in die WAM-Struktur eingebunden werden (ausgehend vom Ordner *wam_docker*):

```
cd docker/WAM
git clone https://github.com/rl-institut/WAM_APP_stemp_mv.git stemp
```

---

**Bemerkung:** Die StEmp-MV App muss im Ordner *stemp* installiert sein, da sonst die internen Verweise nicht funktionieren. Sollte das Repository ohne Namenszusatz geklont worden sein, kann das Repository einfach umbenannt werden.

---

Die neue App *stemp* kann nun für die WAM aktiviert werden, in dem sie in der *docker-compose.yml* Datei unter *celery->build->args->WAM_APPS* hinzugefügt wird:

```
celery:
  restart: unless-stopped
  build:
    context: ./WAM
    args:
      - WAM_APPS=stemp
  ...
```

Folgender Konfigurationsblock muss der Konfigurationsdatei (*config/config.cfg*) zusätzlich hinzugefügt werden:

```
[STEMP]
  ACTIVATED_SCENARIOS=gas,bhkw,bio_bhkw,oil,woodchip,pv_heatpump
  DB_RESULTS=LOCAL
  DB_SCENARIOS=LOCAL
```

Dort können u.a. folgende Konfigurationen vorgenommen werden:

**ACTIVATED_SCENARIOS** die zur Verfügung stehenden Technologien

**DB_RESULTS** Name der Datenbank (muss unter *WAM->Databases* konfiguriert sein), in der die Ergebnisse gespeichert werden sollen

**DB_SCENARIOS** Name der Datenbank, in der die verwendeten Parameter liegen

Das WAM-Image muss nun neu kompiliert werden. Dabei wird die StEmp-MV App in das Image kopiert und alle zusätzlichen Abhängigkeiten der neuen App installiert:

```
cd docker
sudo docker-compose up -d --build
```

Der Server (mit integrierter StEmp-MV App) sollte jetzt wieder unter 127.0.0.1:5000 erreichbar sein.

Abschließend müssen alle benötigten Parameter, Zeitreihen und andere Daten *einmalig* in die Datenbank migriert werden. Dies geschieht über ein Skript das innerhalb des Docker Containers (der Container muss dafür in Betrieb sein) gestartet werden muss:

```
# Startet eine bash innerhalb des Containers:
sudo docker exec -it wam bash
# Fügt die WAM dem PYTHONPATH hinzu; notwendig damit das queries.py Skript die WAM␣
↪module nutzen kann
export PYTHONPATH=$PYTHONPATH:/code
python stemp/db_population/queries.py all
```

KAPITEL 5

# Energiesystem

Die Berechnungen dieses StEmp-Tools umfassen u.a. die Erzeugung, den Bedarf und die Speicherung von elektrischer und thermischer Energie über ein Kalenderjahr. Was auf den ersten Blick wie eine simple Bilanzierung von Energiemengen anmuten könnte, ist in Wirklichkeit ein Energiesystemmodell, das mit Hilfe einer linearen Optimierung gelöst wird.

Die Eingangsparameter für das Modell sind einerseits vorberechnete Daten wie beispielsweise Zeitreihen zur Einspeisung oder Heizwärmebedarf und andererseits die durch den/die BenutzerIn einstellbaren Größen wie z.B. Erdgaspreis oder Installationskosten der Technologien. Die Ergebnisse werden im Anschluss an die Optimierung aufbereitet und im Tool dargestellt.

Für die Optimierung wird das vom RLI mitentwickelte Open Energy System Modelling Framework (oemof) eingesetzt. oemof ist ein freies, offenes und gut dokumentiertes Framework für die Modellierung und Optimierung von Energieversorgungssystemen.

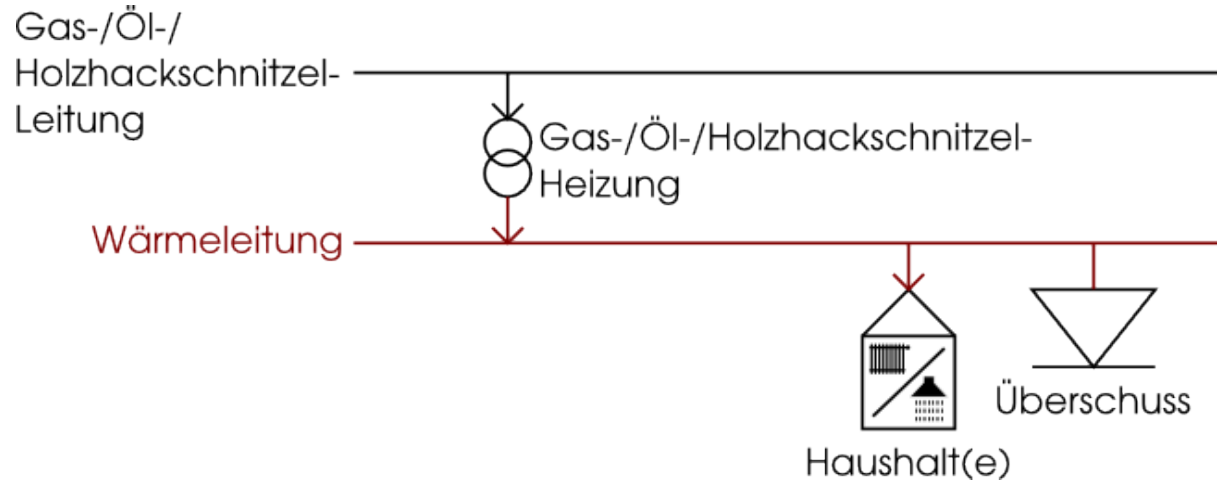
Im folgenden werden die zugrundeliegenden Energiesystememodelle für jede Technologie dargestellt und erläutert. Grundlage für alle Szenarien (mit leichter Abweichung im Falle des PV/Wärmepumpen-Szenarios) ist das folgende Verbrauchermodell:


Ein oder mehrere Haushalte sind an die Wärmeleitung angeschlossen und können über diese mit Wärme versorgt werden. Warmwasser- und Heizungsbedarf sind hierbei vereinfachend zusammengefasst als ein Wärmebedarf. Zusätzlich ist ein Wärmeüberschuss-Verbraucher angeschlossen, der Wärmeüberschüsse ableiten kann (notwendig aus Modellierersicht). Auf diesem Grundmodell aufbauend werden, je nach Technologie, verschiedene Heizsysteme und zusätzliche Leitungen angeschlossen.

**Bemerkung:** Im Falle eines Viertels werden alle verwendeten Haushalte zu einem einzelnen Haushalt zusammengefasst. Die verschiedenen Wärmebedarfe werden dabei übereinandergelegt und aufsummiert.

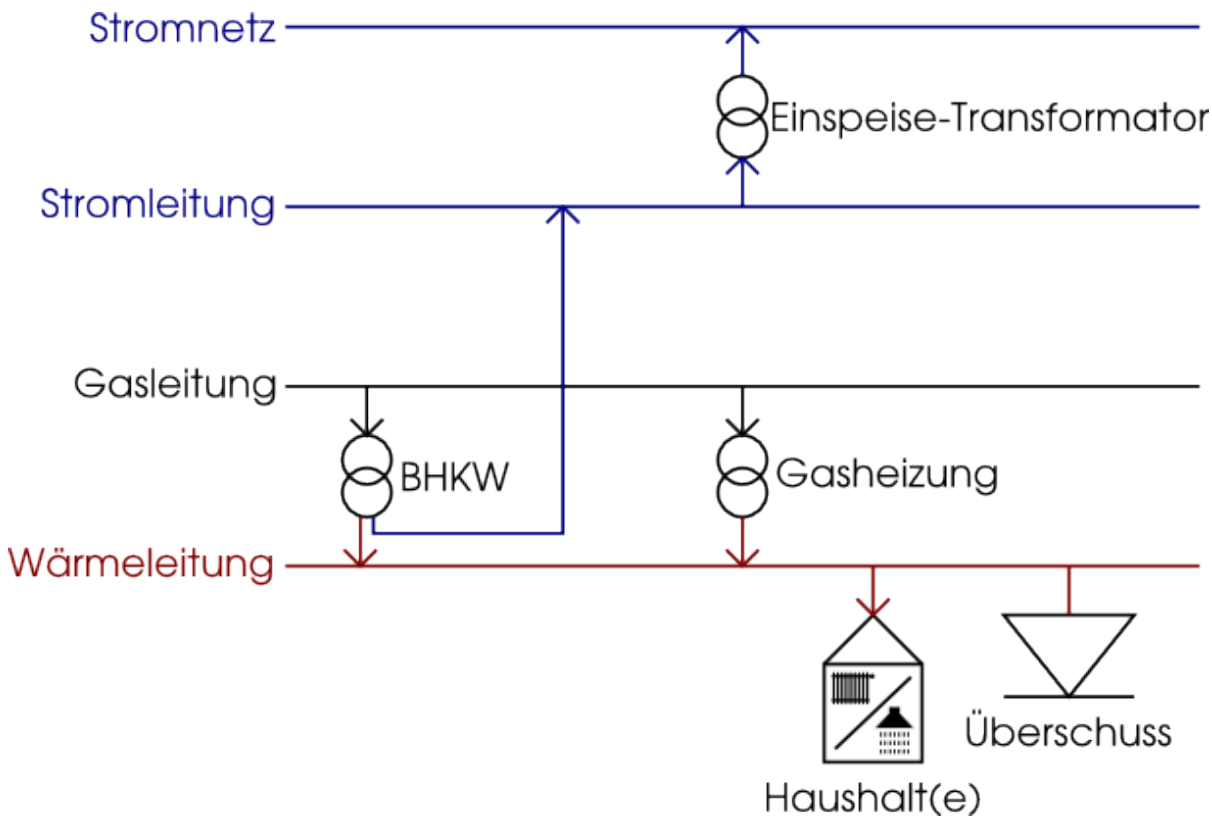
## 5.1 Gas-/Öl-/Holzhackschnitzelheizung

Über eine einfache Leitung und eine angeschlossene Heizung (einfacher Transformator) wird der Wärmebedarf versorgt. Das zugrundeliegende Modell für Gas-, Öl- und Holzhackschnitzel (Modell ist für alle drei Technologien identisch) ist wie folgt:




## 5.2 Blockheizkraftwerk

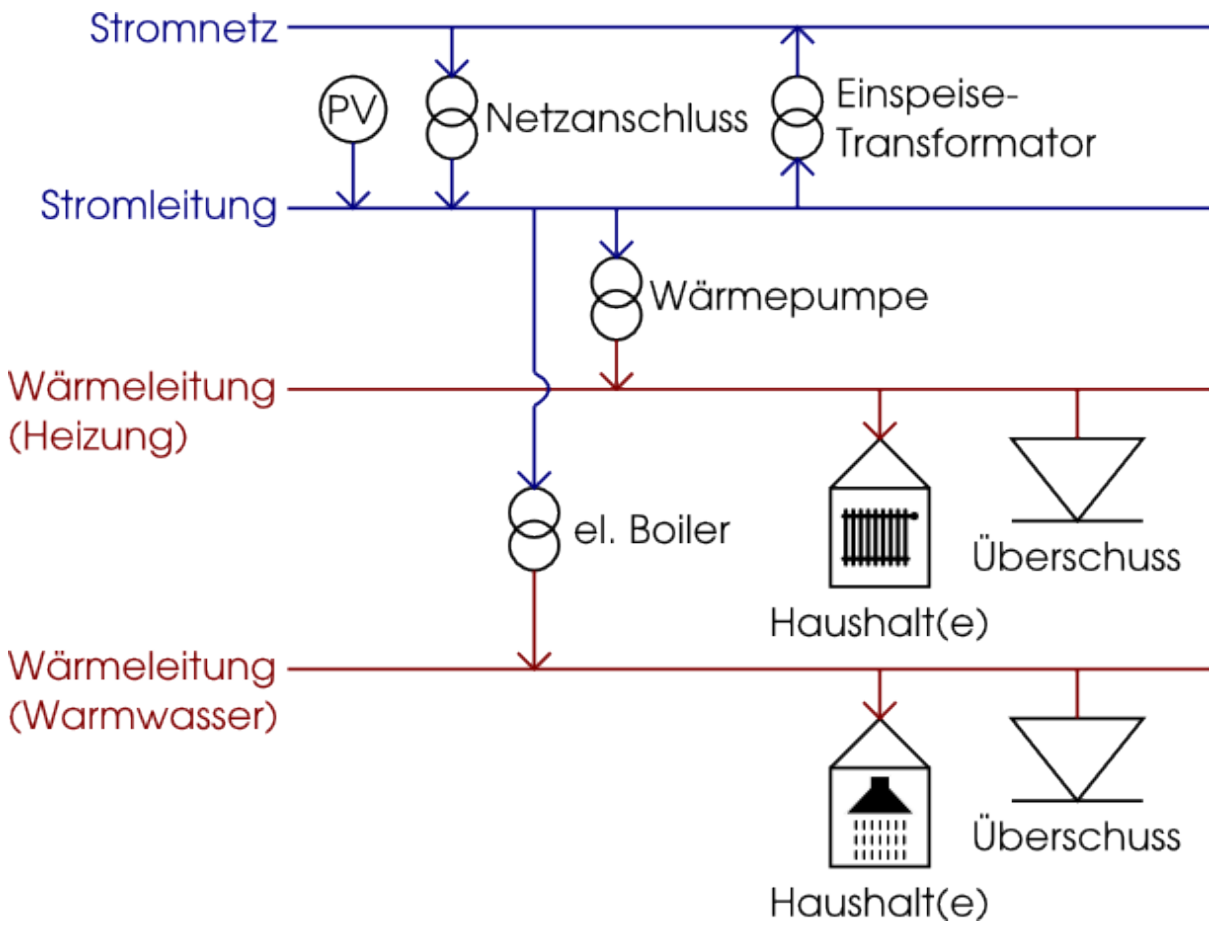
Das Modell für ein Blockheizkraftwerk (Erdgas und Biogas) beinhaltet zwei zusätzliche Stromleitungen. Die (lokale) Stromerzeugung des Kraftwerks kann mithilfe eines Transformators (gewinnbringend) in das Stromnetz eingespeist werden. Zusätzlich zum Blockheizkraftwerk ist eine einfache Gasheizung installiert, die im Falle von extremen Wärmebedarf eingeschaltet werden kann.




**Bemerkung:** Die zusätzliche Gasheizung dient dazu Spitzenbedarf abzufangen. Anderernfalls müsste das Blockheizkraftwerk so überdimensioniert werden, dass es auch extremen Wärmebedarf decken kann. Das Blockheizkraftwerk könnte dann nicht mehr wirtschaftlich betrieben werden.

## 5.3 Photovoltaik-Wärmepumpe

Für die Photovoltaik-Wärmepumpen-Technologie muss das Verbrauchermodell angepasst werden. Der Wärmedarf muss nach Warmwasser und Heizung getrennt werden, da die Wärmepumpe allein nicht die nötige Heizwärme für Warmwasser bereitstellen kann. Für diesen Fall ist zusätzlich ein elektrischer Boiler angeschlossen. Wie im Falle des Blockheizkraftwerkmodells sind zwei zusätzliche Stromleitungen (lokal und Netzanschluss) vorhanden. Der elektrische Boiler und die Wärmepumpe können Strom über die lokale Stromleitung beziehen, die ihrerseits Strom (im günstigen Fall) über die angeschlossene Photovoltaikanlage oder (im ungünstigen Fall) über den Netzanschluss bezieht. Über einen weiteren Transformator können Stromüberschüsse aus der Photovoltaikanlage (gewinnbringend) in das Netz eingespeist werden.


Stromnetz
PV
Netzanschluss
Einspeise-
Transformator
Stromleitung
Wärmepumpe
Wärmeleitung
(Heizung)
el. Boiler
Überschuss
Haushalt(e)
Wärmeleitung
(Warmwasser)
Überschuss
Haushalt(e)

KAPITEL 6

# Datengrundlage

In diesem Tool werden ausschließlich offene Daten verwendet. Eine Liste aller verwendeten Daten findet sich unter Annahmen auf der Projektseite. Jeder Datensatz enthält für eine größtmögliche Transparenz entsprechende Metadaten nach dem OpenEnergyPlatform Metadaten-Standard Version 1.4.

KAPITEL 7

# Übertragung des Tools auf andere Regionen

Das Tool kann durch seinen offenen Charakter und seine geringen Abhängigkeiten grundsätzlich mit geringem Aufwand auf andere Regionen übertragen werden. Vorraussetzung für eine Adaption auf eine andere Region sind standortabhängige Einspeisezeitreihen für Photovoltaik und Temperaturdaten.

Notwendige Schritte (grob):

- Neue Zeitreihen für lokale Temperaturdaten und Stromerzeugung durch Photovoltaik einarbeiten
- Eventuell Standard-Parameter an lokale Gegebenheiten anpassen
- Beschriftungen im Tool auf neue Region anpassen

KAPITEL 8

# Für EntwicklerInnen

Der Quellcode ist auf GitHub verfügbar. Details zur Installation finden sich unter *Installation*.

## 8.1 Technologien

## 8.2 Django-Kosmos

Diese StEmp-Web-Applikation ist Teil des WAM-Applikationen-Kosmos des Reiner Lemoine Instituts (RLI). Die Bezeichnung WAM steht hierbei für *Web Applications and Maps* und stellt eine Kategorie von Applikationen dar, welche am RLI erstellt werden. Dies sind hauptsächlich Web-Applikationen, die mit Hilfe von Karten arbeiten und oder bei denen im Hintergrund Berechnungen stattfinden, dessen Ergebnis an die NutzerInnen visuell zurückgegeben wird. Einige der WAM-Applikationen sind mit dem RLI-eigenen Web-Applikationen-Framework mit dem Namen WAM erstellt worden, andere WAM-Applikationen basieren auf anderen Frameworks oder sind freie Implementationen ohne Framework-Basis.

Diese StEmp-Web-Applikation des RLI nutzt das Git-Projekt WAM als Basis-Projekt. Das verwendete WAM-Basis-Projekt baut auf dem Django-Web-Framework auf, welches Python als Programmiersprache verwendet. Django hat eine gewisse Lernkurve, deswegen kann es Sinn machen, sich bei Bedarf zuerst mit dem zugrunde liegenden Web-Framework Django zu beschäftigen, bevor mit der Arbeit an einer auf Django basierenden WAM-Applikation begonnen wird. Für das Erlernen des Umgangs mit Django gibt es sehr viel Lernmaterial Dritter, weswegen im Folgenden in dieser Dokumentation auf einige dieser Anleitungen und Dokumentationen zum Erlernen von Django verwiesen werden soll. Anschließend wird im nächsten Abschnitt auf der auf Django basierenden WAM-Projektbasis eingegangen.

Tutorials und Informationen zu Django:

- Das Django getting started Tutorial
- Tutorial einer einfachen Geo-Applikation in Django
- Die offizielle Django-Dokumentation
- Die Django-Design-Philosophie

## 8.3 WAM-Kosmos

Wie im vorherigen Abschnitt *Django-Kosmos* bereits kurz angesprochen, verwendet dieses Projekt das RLI eigene Web-Applikationen-Framework WAM. Das WAM-Framework nutzt hierfür Django also Unterbau. Der Django-Unterbau ist dahingehend umgearbeitet, dass er besonders gut auf die Bedürfnisse für die Entwicklung von Applikationen am RLI zugeschnitten ist und er sich von Projekt zu Projekt als Projektbasis wiederverwenden lässt. Im Folgenden soll deshalb kurz auf das WAM-Framework eingegangen werden.

Das WAM-Framework setzt mehrere zugrundlegende Prinzipien konsequent um:

- die initiale Konfigurationsarbeit für auf der WAM basierende Web-Applikationen soll minimiert und wenn möglich automatisiert werden.
- das Zusammenfassen von häufig benötigten Funktionalitäten und die Integration dieser Funktionalitäten in die WAM-Projektbasis, für die einfache Verwendung von auf der WAM basierenden Web-Applikationen.
- eine gemeinsame Projektbasis in der multiple Applikationen angedockt sind minimieren den Wartungsaufwand des Gesamtsystems zur Laufzeit.

Der Grund für die Umsetzung dieser Prinzipien ist die Minimierung von Aufwänden bei der Erstellung und dem Betrieb von Web-Applikationen, welche am RLI und Allgemein im Bereich der Erneuerbaren-Energien-Forschung und -Entwicklung benötigt werden. Durch diese Herangehensweise profitieren bereits die beiden (stemp_abw, stemp_mv) im Rahmen des ENavi-Projektes vom RLI entwickelten StEmp-Tools, welche beide das WAM-Framework als Unterbau nutzen.

Für die Sichtbarmachung von Aufbau und Nutzung des WAM-Frameworks gibt es eine eigenständige WAM-Dokumentation, welche weiterführende Informationen enthält.

KAPITEL 9

# stemp package

## 9.1 Subpackages

### 9.1.1 stemp.db_population package

This package is related to database migration. All data used in simulation (assumptions, timeseries, etc.) are stored locally in the repository and can be migrated easily via the *queries.py* script.

### 9.1.2 stemp.results package

#### Submodules

#### stemp.results.aggregations module

#### stemp.results.analyzer module

#### stemp.results.results module

### 9.1.3 stemp.scenarios package

#### Submodules

#### stemp.scenarios.basic_setup module

#### stemp.scenarios.bhkw module

#### stemp.scenarios.bio_bhkw module

#### stemp.scenarios.gas module

#### stemp.scenarios.heat module

Module to calculate heatpump efficiency

stemp.scenarios.heat.**calc_hp_heating_supply_temp**(*temp*, *heating_system*, ***kwargs*)
Generates an hourly supply temperature profile depending on the ambient temperature. For ambient temperatures below t_heat_period the supply temperature increases linearly to t_sup_max with decreasing ambient temperature.

**Parameter temp** – pandas.Series Timeseries of ambient temperature

**heating_system: str** specifies the heating system (floor heating or radiator)

stemp.scenarios.heat.**cop_heating_floor**(*temp*, *type_hp*, ***kwargs*)
Returns the COP of a heat pump for heating. Parameters temp – pandas Series with ambient or brine temperature type_hp – string specifying the heat pump type (air or brine)

#### stemp.scenarios.oil module

#### stemp.scenarios.pv_heatpump module

#### stemp.scenarios.simulation module

#### stemp.scenarios.woodchip module

### 9.1.4 stemp.visualizations package

#### Submodules

#### stemp.visualizations.dataframe module

#### stemp.visualizations.highcharts module

Module to present data from dataframe in Highcharts chart

**class** stemp.visualizations.highcharts.**LCOEHighchart**(*data*)
Bases: utils.highcharts.Highchart

Layout for LCOE visualization using highcharts

```
colors = {'Betriebskosten': '#fc8e65', 'Brennstoffkosten': '#55aae5', 'Investitionskos
setup = {'chart': {'type': 'column'}, 'plotOptions': {'column': {'dataLabels': {'color
sum_line_options = {'color': 'black', 'dataLabels': {'align': 'left', 'enabled': True,
```

## 9.2 stemp.app_settings module

App-specific settings

These settings are initialized when *wam.settings* are called, before app itself gets started.

**class** stemp.app_settings.**ScenarioModules**
Bases: object

Class to import scenario modules once needed

Scenarios cannot be imported within app_settings as not all resources are yet ready. Therefore, scenario modules are imported once when accessed later.

stemp.app_settings.**add_engine**(*db_connection*)
Creates and adds DB connection from configuration file by given name

First, it is checked which db configuration should be used (default, or specified in config file). Second, engine is created by DB parameters for given DB connection and added to slqahelper with connection name.

**Parameter** **db_connection** (*str*) – Database connection to set up

stemp.app_settings.**import_scenario**(*scenario*)
Scenario with given name is imported from scenario module

**Parameter** **scenario** (*str*) – Name of scenario module

**Rückgabe** Imported scenario module

**Rückgabetyp** module

## 9.3 stemp.apps module

**class** stemp.apps.**StempConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

**name = 'stemp'**

## 9.4 stemp.constants module

Collection of stemp-related constants and enumerations

**class** stemp.constants.**DemandType**
Bases: enum.IntEnum

Enumeration to differentiate between single household and district

**District = 1**

**Single = 0**

**label**()
Label for given demand type

**Rückgabe** Label of current demand type

**Rückgabetyp** str

**suffix**()
Suffix for current demand type

This is needed in order to load scenario parameters which are dependent on demand type.

**Rückgabe** Demand type suffix

**Rückgabetyp** str

**class** stemp.constants.**DistrictStatus**
Bases: enum.Enum

District status enumeration

**Changed = 'changed'**

**New = 'new'**

**Unchanged = 'unchanged'**

**class** stemp.constants.**HeatType**
Bases: enum.Enum

Heat type enumeration

**floor = 'Fussbodenheizung'**

**radiator = 'Heizkörper'**

**class** stemp.constants.**HouseType**
Bases: enum.Enum

House type enumeration

**EFH = 'Einfamilienhaus'**

**MFH = 'Mehrfamilienhaus'**

**class** stemp.constants.**ResultColor**(*quality*, *percentage*, *style*)
Bases: tuple

**percentage**
Alias for field number 1

**quality**
Alias for field number 0

**style**
Alias for field number 2

**class** stemp.constants.**WarmwaterConsumption**
Bases: enum.Enum

Enumeration for Warmwater consumption levels

**High = 2**

**Low = 0**

**Medium = 1**

**from_liters = <function WarmwaterConsumption.from_liters>**

**in_liters**()
Warmwater level in liters

**Rückgabe** Warmwater consumption in liters

**Rückgabetyp** int

stemp.constants.**get_roof_square_meters**(*household_square_meters*, *house_type*)
Calculates potential roof square meters available for PV

**Parameter**

- **household_square_meters** (*int*) – Square meters of current household
- **house_type** (HouseType) – Single household or district

**Rückgabe** Roof square meters available for PV

**Rückgabetyp** float

## 9.5 stemp.fields module

Module to hold customized django fields

**class** stemp.fields.**HouseholdField**(*hh*, *count=1*, *in_district=False*)
Bases: django.forms.fields.Field

Field for households

Is used in district list and summary. Shows current household and selected amounts.

**widget**
Alias von *stemp.widgets.HouseholdWidget*

**widget_attrs**(*widget*)
Given a Widget instance (*not* a Widget class), returns a dictionary of any HTML attributes that should be added to the Widget, based on this Field.

**class** stemp.fields.**SubmitField**(*widget=<class 'stemp.widgets.SubmitWidget'>*, ***kwargs*)
Bases: django.forms.fields.Field

Simple submit field to add submit input

## 9.6 stemp.forms module

Module to hold customized django forms

**class** stemp.forms.**ChoiceForm**(*name*, *label=None*, *choices=None*, *submit_on_change=True*, *initial=None*, *field=<class 'django.forms.fields.ChoiceField'>*, *widget=<class 'django.forms.widgets.Select'>*, **args*, ***kwargs*)
Bases: django.forms.forms.Form

Customized choice form

Adds foundation attributes and „onchange“ submission.

**base_fields = {}**

**declared_fields = {}**

**media**

**class** stemp.forms.**DistrictListForm**(*hh_dict*)
Bases: django.forms.forms.Form

Form to add multiple households of different type to current district

**base_fields = {}**

**declared_fields = {}**

**efh**()
Returns all EFH of current district

Used to differentiate between EFH/MFH in template.

**media**

**mfh**()
Returns all MFH of current district

Used to differentiate between EFH/MFH in template.

**class** stemp.forms.**DistrictSelectForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

Form to select a district

**base_fields = {'district': <django.forms.models.ModelChoiceField object>}**

**declared_fields = {'district': <django.forms.models.ModelChoiceField object>}**

**media**

**class** stemp.forms.**HouseholdForm**(*only_house_type=None*, **args*, ***kwargs*)

Bases: django.forms.models.ModelForm

Form to add/update a household

**class Media**

Bases: object

**js = ('stemp/js/household.js',)**

**class Meta**

Bases: object

**error_messages = {'heat_demand': {'required': 'Bitte geben Sie einen gültigen Wert**

**fields = '__all__'**

**model**

Alias von *stemp.models.Household*

**widgets = {'heat_demand': <django.forms.widgets.HiddenInput object>, 'roof_area': <**

**base_fields = {'heat_demand': <django.forms.fields.FloatField object>, 'heat_demand_ha**

**static create_hotwater_chart**()

**declared_fields = {'heat_demand_hand': <django.forms.fields.IntegerField object>, 'numb**

**media**

**class** stemp.forms.**HouseholdSelectForm**(*only_house_type=None*, **args*, ***kwargs*)

Bases: django.forms.forms.Form

Form to select a household from list

A summary is shown for current selected household (logic in .js file).

**class Media**

Bases: object

**js = ('stemp/js/household_select.js',)**

**base_fields = {'profile': <django.forms.models.ModelChoiceField object>}**

**declared_fields = {'profile': <django.forms.models.ModelChoiceField object>}**

**media**

**class** stemp.forms.**ParameterForm**(*parameters*, *data=None*, **args*, ***kwargs*)

Bases: django.forms.forms.Form

Form to show a grouped edit form for all possible parameters

Depending on parameter best fitting field is automatically detected and added to the form. The field type depends on parameter type (int, char, etc.). In case of int ant float types, slider field is added if min and max values are given.

**base_fields = {}**

**declared_fields = {}**

**delimiter = '-'**

**error_groups**()

**static get_changed_parameters**(*parameters*, *data*)
Compare original parameters with user entries and return diffs

**media**

**prepared_data**(*scenario=None*)
Reads out all (edited) parameters and returns them as dict.

Dict has the form dict[component: dict[parameter: value]].

**Parameter** **scenario** (*str*) – If given, only parameters used in scenario are returned

**Rückgabe** Nested dictionary of all related components and their parameter values

**Rückgabetyp** dict

**class** stemp.forms.**TechnologyForm**(*name*, *label=None*, *choices=None*, *initial=None*, *information=None*, **args*, ***kwargs*)
Bases: django.forms.forms.Form

Form for technology checkboxes

Additional information is loaded within TechnologyWidget.

**base_fields = {}**

**declared_fields = {}**

**media**

**class** stemp.forms.**ValueUnit**(*value*, *unit*)
Bases: tuple

**unit**
Alias for field number 1

**value**
Alias for field number 0

## 9.7 stemp.models module

Database models to set up households and districts and to store simulations including related scenario, parameters and results

**class** stemp.models.**District**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Multiple households combined to a district

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

**add_households**(*households*)
Adds multiple households to the district

**Parameter (dict[int, int])** (*households*) – Dictionary containing household-id as key and amount of this household type as value

**annual_heat_demand**()
Returns combined annual heat demand for all households in district

**annual_hot_water_demand**()
Returns combined annual hot water demand for all households in district

**annual_total_demand**()
Returns combined annual totol demand (heat and warmwater) for all related households in district

**contains_radiator**()
Returns *True* if any household in district contains a radiator

**districthouseholds_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**households**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**max_pv_size**
Returns summarized PV potential in district

**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**class** stemp.models.**DistrictHouseholds**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Relation between districts and related households (many-to-many)

Amount is given to provide possibility to add multiple households of same type as one entry.

**exception DoesNotExist**
Bases: `django.core.exceptions.ObjectDoesNotExist`

**exception MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

**amount**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**district**
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**district_id**

**household**
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**household_id**

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**class** stemp.models.**HeatProfile**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Model to hold heat profiles for different households and number of persons

**exception DoesNotExist**
Bases: `django.core.exceptions.ObjectDoesNotExist`

**exception MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**layout = {'title': 'Wärmeverbrauch', 'x_title': 'Zeit [h]', 'y_title': 'Wärmeverbrauch**

**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**profile**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**class** stemp.models.**Household**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Household model contains all data to a single household

**name** (*str*)

**Type** Name of the household

**house_type** (*str*)

**Type** EFH (Einfamilienhaus) or MFH (Mehrfamilienhaus)

**heat_demand** (*float*)

**Type** Total annual heat demand

**number_of_persons** (*int*)

**Type** Persons belonging to the household

**square_meters** (*int*)

**Type** Square meters of household

**heat_type** (*str*)

**Type** radiator or floor

**warm_water_per_day** (*int*)

**Type** Daily warm water consumption

**roof_area** (*float*)

**Type** Potential photovoltaik area on roof of household

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

**annual_heat_demand**()

**annual_hot_water_demand**()
Get annual hot water demand profile

Depending on warmwater consumption per day and number of persons in household.

**Rückgabe** Annual hot water profile

**Rückgabetyp** pandas.Series

**annual_total_demand**()

**contains_radiator**()

**district_set**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**districthouseholds_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**get_heat_demand_profile**()
Get heat demand profile depending on given house type (EFH/MFH)

**Rückgabe** Heat demand profile for given house type

**Rückgabetyp** pandas.Series

**get_heat_type_display**(*, *field=<django.db.models.fields.CharField: heat_type>*)

**get_house_type_display**(*, *field=<django.db.models.fields.CharField: house_type>*)

**get_oep_timeseries**(*name*)
Function to get household timeseries

This function allows one time DB access, in order to avoid multiple database look-ups.

**Rückgabe** Heat demand profile for given house type

**Rückgabetyp** pandas.Series

**heat_demand**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**heat_type**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**house_type**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**max_pv_size**
Photovoltaik potential for given roof area and PV-per-square-meters-ratio

**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**number_of_persons**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**roof_area**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**square_meters**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**timeseries = None**

**warm_water_demand**()

**warm_water_per_day**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**class** stemp.models.**Parameter**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Holds all parameters for a simulation run

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

**data**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**simulation_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

**class** stemp.models.**Scenario**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Holds name and date of simulation run

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

**get_next_by_last_change**(*, *field=<django.db.models.fields.DateTimeField: last_change>*, *is_next=True*, ***kwargs*)

**get_previous_by_last_change**(*, *field=<django.db.models.fields.DateTimeField: last_change>*, *is_next=False*, ***kwargs*)

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**last_change**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**simulation_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**class** stemp.models.**Simulation**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Connects szenario with parameters to oemof results.

As oemof results are stored via sqlalchemy, relation between simulation and result is built via „result_id“ (*not* Foreign-key, thus loose coupled).

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

**date**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**classmethod delete_containing_household**(*hh_id*)
Deletes all simulations which contain given household ID

**get_next_by_date**(*, *field=<django.db.models.fields.DateTimeField: date>*, *is_next=True*, ***kwargs*)

**get_previous_by_date**(*, *field=<django.db.models.fields.DateTimeField: date>*, *is_next=False*, ***kwargs*)

**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects = <django.db.models.manager.Manager object>**

**parameter**
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**parameter_id**

**result_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**scenario**
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**scenario_id**

## 9.8 stemp.oep_models module

Database models to store scenario parameters and timeseries (including hot water).

Initially these models should be seperated from WAM-server and migrated to OEP instead. But due to OEP-latency, models are migrated to WAM-serverr right now.

**class** stemp.oep_models.**OEPHotWater**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Model to hold hot water timeseries related to given liter

**data**

**id**

**liter**

**class** stemp.oep_models.**OEPScenario**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Contains all parameters related to scenarios

**Parameter**

- **scenario** – Related scenario name (gas/oil/...)

- **component** – Component which parameter belongs to (gas/bus/demand/...)
- **unit** – unit of given param
- **parameter_type** – costs/technologies - is used to distinguish parameters (different colors are used in parameters page)
- **parameter** – Name of the parameter
- **value_type** – int/str/float
- **value** – Value of given parameter

**component**

**classmethod get_scenario_parameters**(*scenario_name*, *demand_type*)

All scenario parameters for given scenario name and demand type are fetched

Scenario parameters are chained with default attributes; thus, default parameter values are used as fallback values, if scenario does not define special/individual values.

**id**

**parameter**

**parameter_type**

**scenario**

**unit**

**value**

**value_type**

**class** stemp.oep_models.**OEPTimeseries**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

Model to hold timeseries with related metadata (json)

**data**

**id**

**meta_data**

**name**

## 9.9 stemp.settings module

## 9.10 stemp.tasks module

## 9.11 stemp.urls module

## 9.12 stemp.user_data module

## 9.13 stemp.views module

## 9.14 stemp.views_admin module

## 9.15 stemp.views_dynamic module

Functions to handle dynamic AJAX-requests

stemp.views_dynamic.**get_heat_demand**(*request*)
Calculates heat demand related to square meters and household type

stemp.views_dynamic.**get_household_summary**(*request*)
Provides household summary widget for given household ID

stemp.views_dynamic.**get_next_household_name**(*request*)
Dynamic household naming

If household name is already present in DB, a counting number prefix is added

stemp.views_dynamic.**get_roof_area**(*request*)
Calculates roof area related to square meters and household type

stemp.views_dynamic.**get_square_meters**(*request*)
Calculates square meters related to number of persons

stemp.views_dynamic.**get_warm_water_energy**(*request*)
Calculates warm water consumption related to number of persons

## 9.16 stemp.widgets module

Additional widgets used by forms or individually

**class** stemp.widgets.**DistrictSubmitWidget**(*attrs=None*)
Bases: django.forms.widgets.Widget

**media**

**template_name = 'widgets/district_submit.html'**

**class** stemp.widgets.**HouseholdSummary**(*household*, *use_header=True*, *count=None*)
Bases: utils.widgets.CustomWidget

Widget to show summary of a household

If „use_header“ is True, an accordion is used to hide summary at first.

**get_context**()

**class** stemp.widgets.**HouseholdWidget**(*attrs=None*)
Bases: django.forms.widgets.Widget

**media**

**template_name = 'widgets/district_household.html'**

**class** stemp.widgets.**ParameterSummary**(*changed_parameters*)
Bases: utils.widgets.CustomWidget

Widget to show non-default parameters, changed by user

For each changed parameter, related icon and label is shown

**get_context**()

**template_name = 'widgets/summary_parameter.html'**

**class** stemp.widgets.**SelectWithDisabled**(*attrs=None*, *choices=()*)
Bases: django.forms.widgets.Select

Subclass of Django's select widget that allows disabling options. To disable an option, pass a dict instead of a string for its label, of the form: {'label': 'option label', 'disabled': True} From: https://djangosnippets.org/snippets/2453/

**create_option**(*name*, *value*, *label*, *selected*, *index*, *subindex=None*, *attrs=None*)

**media**

**class** stemp.widgets.**SliderInput**(*step_size=1*, *attrs=None*)
Bases: django.forms.widgets.NumberInput

Widget to add slider to number input field

Step size can be chosen, defaults to 1. Precision of number is adapted automatically from step size.

**get_context**(*name*, *value*, *attrs*)

**input_type = 'number'**

**media**

**template_name = 'widgets/slider.html'**

**class** stemp.widgets.**SubmitWidget**(*attrs=None*)
Bases: django.forms.widgets.Widget

Widget for submit button

From https://djangosnippets.org/snippets/2312/

**media**

**render**(*name*, *value*, *attrs=None*, *renderer=None*)
Render the widget as an HTML string.

**class** stemp.widgets.**TechnologySummary**(*scenario_config*)
Bases: utils.widgets.CustomWidget

Widget to show chosen technology as icon plus name

**get_context**()

**template_name = 'widgets/summary_technology.html'**

**class** stemp.widgets.**TechnologyWidget**(*attrs=None*, *choices=()*, *information=None*)

Bases: django.forms.widgets.CheckboxSelectMultiple

Widget to add key-value-pairs to select component From: https://www.abidibo.net/blog/2017/10/16/add-data-attributes-option-tags-django-admin-select-field/

**create_option**(*name*, *value*, *label*, *selected*, *index*, *subindex=None*, *attrs=None*)

**media**

**template_name = 'widgets/technology.html'**


GEFÖRDERT VOM

# Python-Modulindex

## S

# Stichwortverzeichnis

## E



## F



## G



## H

## I



## J



## L



## M



## N



## O

## W