StEmp-ABW Documentation

Release 1.0.4

Reiner Lemoine Institut

Oct 31, 2020

Inhalt

StEmp-ABW ist ein Stakeholder-Empowerment-Tool für die Region Anhalt-Bitterfeld-Wittenberg (ABW). Es wurde vom Reiner Lemoine Institut (RLI) im Rahmen des Kopernikus-Projekts „ENavi“ entwickelt. Sie finden das Tool auf den WAM-Seiten des RLI.

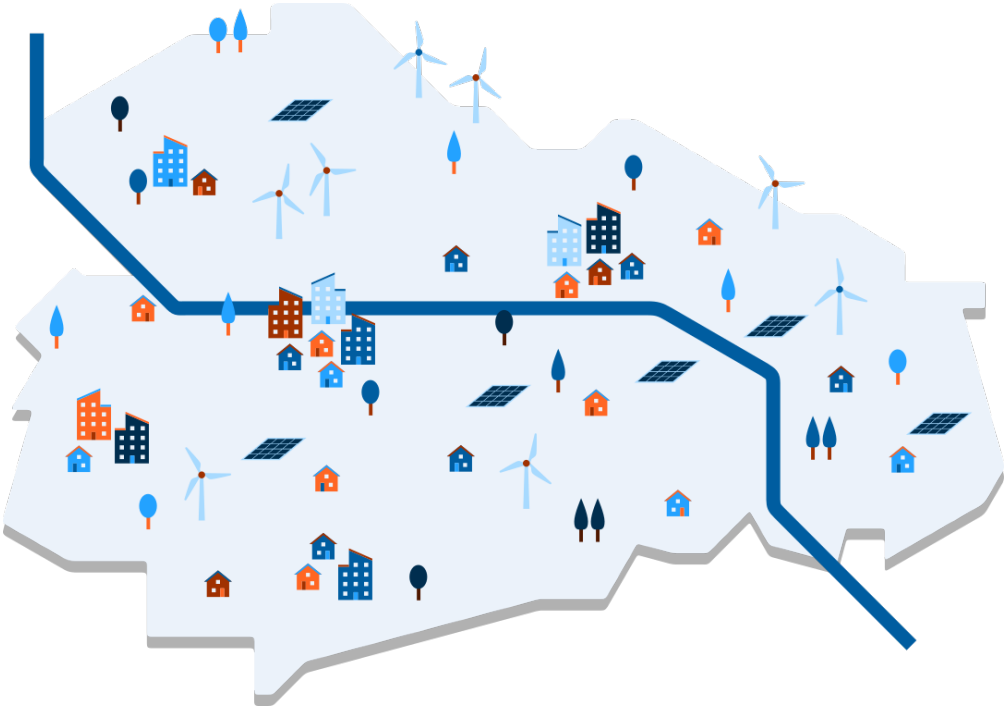
In dieser Dokumentation finden Sie methodische und technische Hintergrundinformationen und Anleitungen.

Wollen Sie uns Rückmeldung geben? Hierfür können Sie gern unser Feedback-Formular verwenden.

CHAPTER 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 mit Unterstützung der Energieavantgarde Anhalt e.V. (EAA) entwickelt, das es Akteurinnen und Akteuren der Energiewende ermöglicht, sich an Planungsprozessen zu beteiligen.



Über das Menü links gelangen Sie auf die entsprechenden Seiten.

1.1 Motivation

Die Energiedialoge der Energieavantgarde Anhalt haben insbesondere im direkten Gespräch mit den Bürgerinnen und Bürgern und in der Mitglieder- und Stakeholderberatung gezeigt, dass die Möglichkeiten zur Partizipation in der Entwicklung eines zukunftsweisenden regionalen Energiesystems zwar formal gegeben sind, in der Praxis aber aufwändig und häufig unbefriedigend sind. Der ganzheitliche Blick auf die Anforderungen und Chancen eines regionalen und regenerativen Energiesystems fehlt meist. Stattdessen werden landschaftliche Veränderungen (bspw. durch Windenergieanlagen oder Freiland-PV-Anlagen), lokale Auswirkungen auf Nachbarschaften, Naturschutz- und Denkmalschutzbelange sowie Auswirkungen der „Energiewende“ auf Energiepreise etc. meist unverbunden und teils konfrontativ debattiert. Die Rechtsunsicherheit im Umgang mit den Inhalten der Regionalplanung und Raumordnung ist nur ein Ausdruck dieser komplexen Problemlage.

Verbesserungspotenzial sehen die Befragten in diesem Themenfeld vor allem in der Darstellung der unmittelbaren Betroffenheit einzelner Regionen unter Berücksichtigung anderer Vorgaben in der Flächennutzung, wie beispielsweise der Mindestabstand von Windenergieanlagen zu Siedlungen. Außerdem ist eine Betrachtung variabler Szenarien des regionalen Energiebalancekreises bezogen auf Energiedargebot, Anlagen der Energieumwandlung, Energiemix durch gemeinsame Beeinflussung der Ziel- und Eingangsdaten, gemeinsames Nachvollziehen und Festlegen von Restriktionen und Potenzialen gewünscht.

1.2 Ziele

Mit dem Stemp - Tool für die Region Anhalt verfolgen das Reiner Lemoine Institut und die Energieavantgarde Anhalt folgende Ziele:

- Identifikation der Werte und Interessen der regionalen Anspruchsgruppen.
- Aktivierung der regionalen Anspruchsgruppen bzgl. der Mitgestaltung der Energiewende in der Region Anhalt.
- Verbesserung des Verständnisses von Zusammenhängen zwischen energiewirtschaftlichen, ökologischen und kulturlandschaftlichen Systemen.
- Seriöse und reale Partizipation der regionalen Anspruchsgruppen, insb. der Bürgerinnen und Bürger, sowohl am Planungsprozess sowie an der Nutzung erneuerbarer Energieanlagen.
- Befähigung, bessere Entscheidungen bzgl. der regionalen Energiewende verhandeln und treffen zu können.
- Ermöglichung einer holistischen Planung und Genehmigung von erneuerbaren Energieanlagen.
- Bereitstellung von Daten und Quellcode, um die Umsetzung ähnlicher Tools zu erleichtern.

1.3 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.“
```

1.4 Lizenz

CHAPTER 2

Was ist ein StEmp-Tool?

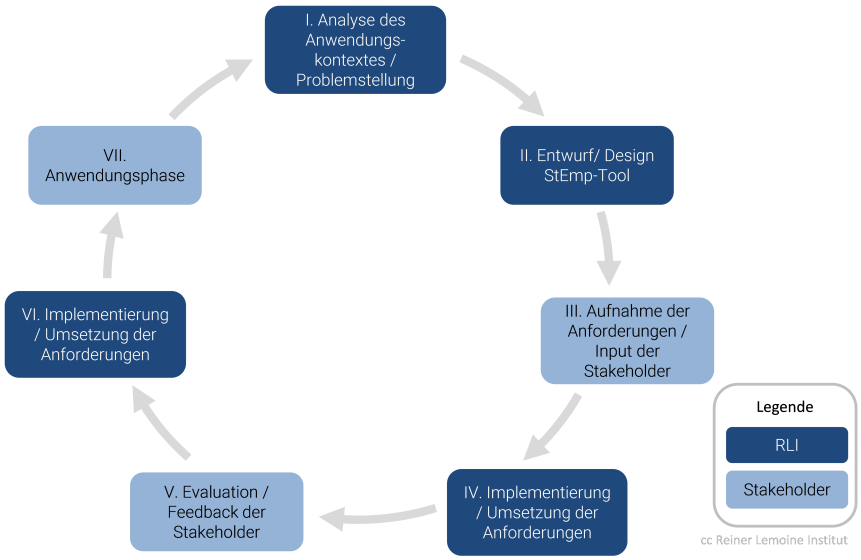
Ein Stakeholder-Empowerment-Tool - kurz StEmp-Tool - ist ein fachliches Berechnungsprogramm, das verschiedene Akteure auf verständlicher Basis unterstützen soll. Ziel ist es dabei, eine Grundlage für die Partizipation unterschiedlicher Interessensgruppen zu schaffen, indem relevante Daten so aufbereitet 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

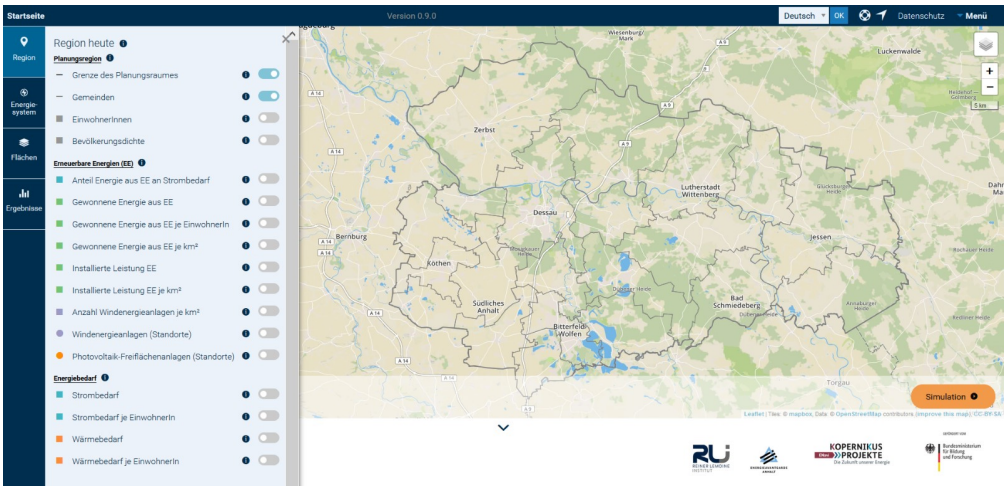
CHAPTER 3

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 hat unterschiedliche Funktionalitäten, die in der Legende links aufgeführt sind. Es ist möglich, sich die Region, deren Energiesystem und die Flächennutzung anzeigen zu lassen. Eine weitere Anwendung ist die Optimierung des Energiesystems unter selbst vorgegebenen Bedingungen. Von dieser kann man sich die Ergebnisse nach der Simulation anzeigen und abspeichern lassen. Die einzelnen Funktionen sind im Weiteren näher beschrieben. Durch Hovern über die dunkelblau hinterlegten Informationsbuttons ''i" können weitere Informationen abgefragt werden. Diese öffnen sich nach wenigen Sekunden, wenn man die Maus darüber bewegt. Über den Button ''Startseite" gelangt man wieder auf die Anfangsseite der App, wo man einige Informationen zur App nachlesen kann. Unter ''Datenschutz" lassen sich die aktuellen diesbezüglichen Bestimmungen anzeigen. Im ''Menu" finden sich Kontaktdaten sowie die Angabe von Quellen und Annahmen, die der Anwendung zu Grunde liegen. Außerdem kann man sich hier zurück ''Zum Tool" navigieren lassen.



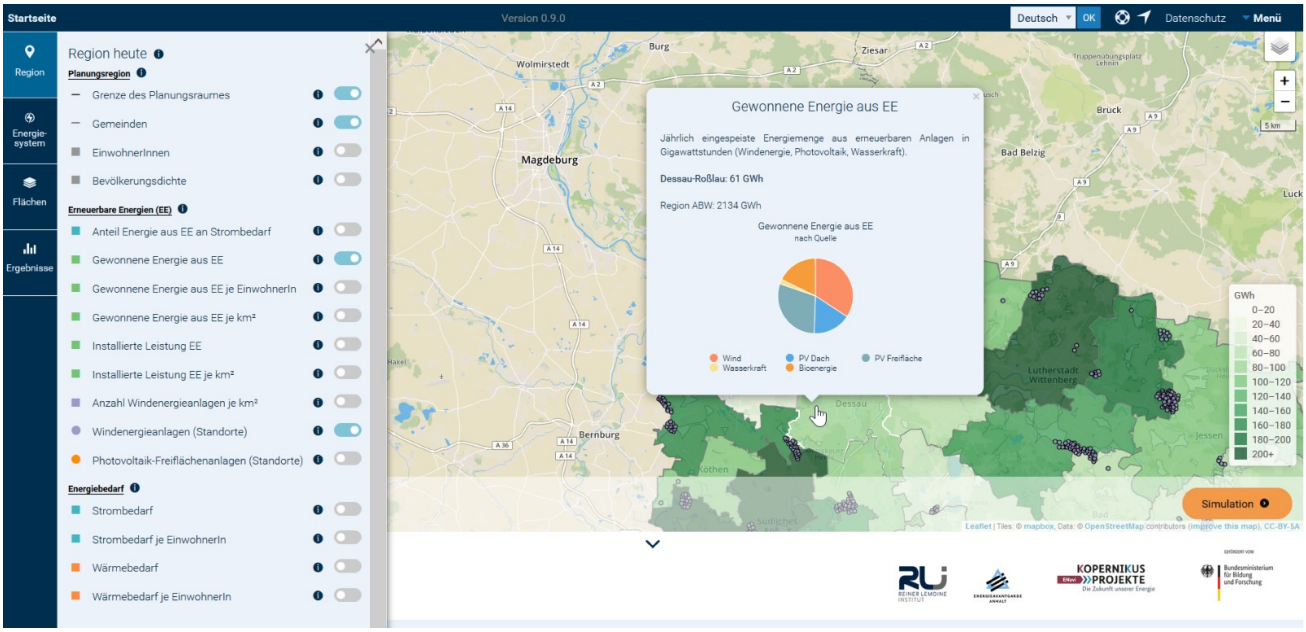
3.2 Panels

Die App umfasst die Panels ‘’Region’‘, ‘’Energiesystem’‘, ‘’Flächen” und ‘’Ergebnisse’‘, deren Aufbau und Handhabung nachfolgend beschrieben ist.

3.2.1 Region

Diese Funktionalität stellt grafisch aufbereitete Informationen zur Region und deren Energiesystem dar. Dabei werden Eckdaten der Region, Informationen zur Einspeisung aus Erneuerbaren Energien und zum Energiebedarf der einzelnen Gemeinden bereitgestellt.

Die Abbildung zeigt eine beispielhafte Nutzung. Links im Panel kann der Nutzer auswählen, welche der Daten dargestellt werden sollen. Hier ist ‘Gewonnene Energie aus EE’ angewählt, welche auf der Karte gemeindescharf dargestellt ist. Außerdem werden die Standorte der Windenergieanlagen angezeigt. Durch Anklicken einer Gemeinde öffnet sich eine Informationstafel über die angewählte Funktionalität, hier Erzeugung aus EE. Dort finden sich nähere Informationen zu der ausgewählten Gemeinde, in diesem Fall die jährlich eingespeiste Energiemenge und die Anteile einzelner Energieträger. Im Beispiel wird der größte Anteil der 190 GWh durch Windenergie bereit gestellt.



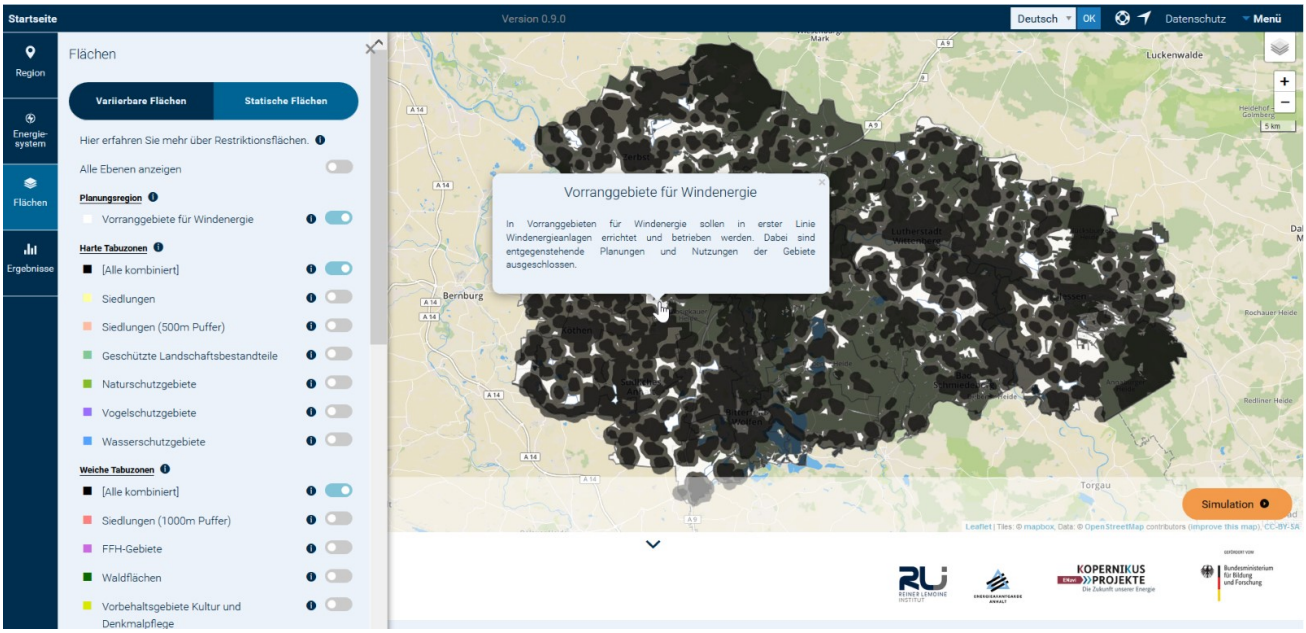
3.2.2 Energiesystem

Das Panel ‘’Energiesystem” ermöglicht die Variation und Untersuchung desselben in der Region. Als erster Schritt kann ein Vergleichs-Szenario ausgewählt werden, in Relation zu welchem die Ergebnisse angezeigt werden. Danach ist es möglich, über die Reiter ‘’Stromerzeugung’‘, ‘’Strombedarf’‘, ‘’Wärmeerzeugung” und ‘’Wärmebedarf” die entsprechenden Größen individuell anzupassen. In den Bedarfsfällen können jeweils die prozentualen Einsparungen zum Vergleichsszenario in Haushalten sowie in Gewerbe, Handel und Dienstleistungen (GHD) definiert werden. Bei der Stromerzeugung dagegen gibt es mehr Einstellparameter. Für die einzelnen Erzeuger sind die Leistungen des Vergleichsszenarios voreingestellt. Es wird außerdem automatisch das Potenzial der erneuerbaren Erzeuger, gemessen an ausgewähltem Zubau-Szenario und Flächennutzung, errechnet. Die Regler können entsprechend verschoben werden, um die gewünschte installierte Kapazität der Erzeuger einzustellen. Das Zubau-Szenario bezieht sich auf Windenergieanlagen und beeinflusst auch die Flächennutzung, was im folgenden Unterkapitel näher erläutert wird. Bei der Wärmeerzeugung kann der prozentuale Anteil von Power-to-Heat in Haushalten und GHD verändert werden.

3.2.3 Flächen

Dieses Panel beinhaltet zwei Ebenen, die statische und die variierbare. ‘’Statische Flächen” zeigt optisch aufbereitete Informationen zur Flächennutzung in der Region. Dabei sind vor allem für Planungen von Windergieanlagen relevante Informationen zusammengefasst. Die Informationen, die dargestellt werden können, sind in verschiedene Kategorien eingeteilt. Unter ‘’Planungsregion” können die Vorranggebiete für Energie angezeigt werden. Dies sind die Gebiete in der Region, die sich am besten zur Errichtung von Windenergieanlagen eignen und in denen keine Konflikte mit anderen Flächennutzungsoptionen auftreten. Unter der Überschrift ‘’Harte Tabuzonen” sind Gebiete zusammengefasst, in denen nach aktueller Rechtslage kein Bau von Windenergieanlagen möglich ist. Unter ‘’Weiche Tabuzonen” aufgeführte Flächen unterliegen zwar aktuell genauso dem Verbot der Bebauung durch Windenergieanlagen. Hier besteht allerdings mehr Gestaltungsspielraum und durch Anpassung der rechtlichen Rahmenbedingungen könnten diese nutzbar gemacht werden. Unter ‘’Einzelfallprüfung” sind solche Gebiete zusammengefasst, bei denen es einer individuellen Abwägung der Interessen bedarf, um zu entscheiden, ob hier der Bau von Windenergieanlagen erlaubt werden sollte.

Die Abbildung zeigt die Darstellung aller harten und weichen Tabuzonen in dunkelgrau und grau, sowie der Voranggebiete für Windenergie in weiß. Dieses Panel kann veranschaulichen, wie begrenzt die tatsächlich für Windenergieanlagen zur Verfügung stehenden Flächen sind und wie hoch die Flächenkonkurrenz im untersuchten Gebiet. Diese Visualisierung kann außerdem als Grundlage für Diskussionen über Regularien und Rahmenbedingungen genutzt werden.



Den Reiter ‘’Variierbare Flächen” kann man wie das Energiesystem-Panel dazu nutzen, ein eigenes Szenario näher zu definieren, um es dann mit einem Referenz-Szenario zu vergleichen. Diese Option kann allerdings nur dann genutzt werden, wenn als Zubau-Szenario für Windenergie die Option ‘’Freier Zubau” ausgewählt wurde. Dann kann einerseits der gesetzlich vorgeschriebene Abstand von Windenergieanlagen zu Gemeinden variiert und andererseits ausgewählt werden, ob Waldflächen für die Bebauung freigegeben werden sollen. Verändert man diese Optionen, so gleicht sich auch das Potenzial der installierbaren Leistung von Windenergieanlagen unter ‘’Energiesystem” an.

3.2.4 Ergebnisse

Wurden unter ‘’Energiesystem” und/oder ‘’Flächen” Änderungen zum Vergleichsszenario vorgenommen, so kann eine Simulation durch Anklicken des entsprechenden Buttons gestartet werden. Diese berechnet dann Erzeugungs- und Verbrauchstdaten mit dem neuen Erzeugungspool. Die Ergebnisse der Simulation können im gleichnamigen Panel eingesehen werden.

Die unter ''Ergebnisse für die Planungsregion" zusammengefassten Reiter können dazu genutzt werden, sich diese auf der Karte anzeigen zu lassen. Durch Anwählen von ''Mehr Ergebnisse +" können Details eingesehen werden. In einer ''Übersicht" ist die anteilige Erzeugung der unterschiedlichen Erzeugerarten des neu definierten Szenarios dem Vergleichsszenario gegenüber gestellt. Unter den Reitern ''Energieerzeugung'', ''Wärmebedarf" und ''Strombedarf" ist eine genauere monatliche Darstellung der entsprechenden Größen zu finden.

Diese Funktionalität kann dazu genutzt werden, Zukunftsszenarien zu simulieren und unterschiedliche Arten zu beleuchten, vorgegebene Zielsetzungen zu erreichen. So können sowohl der Zubau an erneuerbaren Erzeugern als auch die Reduktion des Verbrauchs zu einer Verminderung der Treibhausgase führen. Den Verantwortlichen der Region steht somit ein Instrument zur Verfügung, unterschiedliche Zukunftsszenarien zu simulieren und hinsichtlich vorher festzulegender Kriterien zu vergleichen.

3.2.5 Sprachen

Im Tool bringt Sprachpakete für Deutsch und Englisch mit und kann bei Bedarf erweitert werden. Für Details siehe *Sprachpakete*.

CHAPTER 4

Installation

4.1 Voraussetzungen

- Python Version >=3.6
- WAM (Installation in der folgenden Anleitung enthalten)
- Git (Linux/Windows) und Git Bash (Windows), falls direkt vom Quellcode installiert wird

4.2 Installation (Quellcode-Weg)

1. Ordner anlegen und in diesen wechseln:

- Linux und Windows (Git Bash):

```
mkdir ~/StEmp-ABW/
cd ~/StEmp-ABW/
```

2. GitHub-Repository der WAM klonen und in den Master-Branch wechseln:

- Linux und Windows (Git Bash):

```
git clone https://github.com/rl-institut/WAM.git WAM
cd ./WAM/
git checkout master
```

3. Einrichtung der WAM.

Entsprechend der WAM-Dokumentation (PostgreSQL, PostGIS, Celery, Umgebungsvariablen, benötigte Pakete etc.). Danach an dieser Stelle weiter.

Anmerkung: In der WAM-Dokumentation wird u. a. beschrieben, wie Sie via Conda eine virtuelle Umgebung erstellen und diese aktivieren. In den nächsten Schritten gehen wir davon aus, dass diese aktiviert ist.

4. GitHub-Repository des StEmp-Tools in das WAM-Verzeichnis klonen und in den Master-Branch wechseln:

- Linux und Windows (Git Bash):

```
git clone https://github.com/rl-institut/WAM_APP_stemp_abw.git stemp_abw
cd ./stemp_abw/
git checkout master
```

Anmerkung: falls noch nicht getan, im der Umgebungsvariable `WAM_APPS` den Applikationennamen `stemp_abw` eintragen - vgl. mit WAM-Dokumentation.

5. Datenbankeinrichtung

- Linux und Windows (Git Bash):

Anmerkung: Falls nicht bereits geschehen ins WAM-Stammverzeichnis wechseln, in dem sich die Datei *manage.py* befindet.

```
python manage.py makemigrations
python manage.py migrate
```

6. Datenmigration

Die benötigten Daten liegen auf Zenodo und können mit dem bereitgestellten Skript `get_fixtures_stemp_abw.py` über Django fixtures installiert werden.

- Linux und Windows (Git Bash):

```
python manage.py get_fixtures_stemp_abw
python manage.py loaddata stemp-abw_data_<VERSION>.json
```

Anmerkung: Falls der Downloadvorgang (`get_fixtures_stemp_abw`) fehlschlägt, dann einfach den Datensatz von Zenodo herunterladen, entpacken und die resultierende JSON-Datei in das Fixtureverzeichnis (`stemp_abw/fixtures`) kopieren.

Anmerkung: Der Ladevorgang der JSON-Daten (`loaddata`) kann bis zu 15 Minuten dauern.

7. Django-Server starten

- Linux und Windows (Git Bash):

```
./manage.py runserver 8888
```

Per Browser kann nun auf das Tool zugegriffen werden: http://127.0.0.1:8888/

Note: Kompatibilität: Die Versionsnummern des verwendeten Tools und der Daten müssen übereinstimmen.

CHAPTER 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.

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.

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

5.1 Struktur und Vereinfachungen

Das Energiesystemmodell wird über Komponenten definiert (bspw. Erzeuger und Verbraucher), die an Busse angeschlossen werden an welchen ein Austausch stattfindet. Aus diesem Modell wird ein lineares Optimierungsproblem erstellt, das durch einen Solver gelöst wird.

Aus verschiedenen Gründen sind Vereinfachungen notwendig, um ein sinnvolles Gleichgewicht aus Genauigkeit und Rechenzeit herzustellen.

Zeitraum Die Optimierung erstreckt sich über den Zeitraum von 1 Kalenderjahr in 1h-Schritten.

Perfect Foresight Aus den gegebenen Randbedingungen wird ein Gesamtproblem erstellt, der Zustand aller Komponenten des Energiesystems wie z.B. Erzeuger ist zu jedem Zeitpunkt bekannt (im Gegensatz zu bspw. Rolling-Horizon-Verfahren)

Modelltopologie Die antreibenden Modelldaten selbst liegen gemeindescharf vor, werden zugunsten der Rechenzeit jedoch zusammengefasst und an einen zentralen elektrischen Bus angeschlossen. So werden z.B. die Einspeisezeitreihen der Windenergieanlagen aller 20 Gemeinden in einer einzigen Komponente aggregiert und mit diesem Bus verbunden. Im Ergebnis besteht das Modell aus einem Einspeiser pro Technologie (Wind, FF-PV, Dach-PV, ...) und einem Verbraucher pro Sektor

(Haushalte, GHD u. Landwirtschaft, Industrie). Für die Ergebnisdarstellung in der Karte findet eine Desaggregation statt.

Mehr über die Integration des Energiesystems in das Tool erfahren sie unter *Für EntwicklerInnen*.

CHAPTER 6

Szenarien und Methoden

6.1 Szenarien

Das Grundszenario ist der Status quo (Stand der Daten: Ende 2017), der als Ausgangspunkt für Variationen dient. Mögliche Variationen umfassen die Anpassung der Energieerzeugung (und diese beeinflussende EE-Potenzialflächen) sowie des Energieverbrauchs.

Es können weitere Szenarien definiert werden, die anschließend im Tool ausgewählt werden können. Mehr über die Integration der Szenarien erfahren Sie unter *Szenarien* (für EntwicklerInnen).

6.2 Methoden

Im Folgenden werden die wichtigsten Methoden skizziert, die für die Aufbereitung der Rohdaten zu verwendbaren Daten innerhalb des Tools und Parametrierung des Energiesystems verwendet wurden.

6.2.1 Erzeugung der Tooldaten

Wenn Sie beabsichtigen neue Eingangsdaten zu erzeugen, können Sie die folgenden Schritte selbst ausführen oder alternative Daten und Tools verwenden, siehe Kapitel *Übertragung des Tools auf andere Regionen*.

Kraftwerksdaten

1. Herunterladen des aktuellen Kraftwerksdatensatzes von Open Power Systems Data (OPSD) (Stand EE: Ende 2017, Stand konventionelle Kraftwerke: Ende 2018).
2. Zuordnung der Kraftwerke zu Gemeinden anhand der Koordinaten
3. Aggregation von Leistung und Anzahl nach Technologie und Gemeinde

Datentabelle Ergebnisse: *stemp_abw.models.MunData*

Anmerkung: Für die räumliche Zuordnung auf Gemeindeebene ist die Genuigkeit des OPSD-Datensatzes ausreichend, nicht jedoch für eine exakte Geopositionierung. Hierfür sollte auf OSM-Daten oder zukünftig auf das Marktstammdatenregister zurückgegriffen werden, s. *Übertragung des Tools auf andere Regionen*.

Verbrauchsdaten Strom

1. Herunterladen des Datensatzes zu Lastgebieten (Load Areas), von der OpenEnergy Platform, welcher den jährlichen Verbrauch je OSM-Lastgebiet und Sektor enthält (Paper zum Hintergrund).
2. Herunterladen des Datensatzes zu industriellen Großverbrauchern (Large scale consumer), von der OpenEnergy Platform, welcher den jährlichen Verbrauch für große Industriegebiete enthält.
3. Zuordnung der Load Areas und Large scale consumer zu Gemeinden anhand der Koordinaten
4. Aggregation des Verbrauchs nach Sektor und Gemeinde

Die Datensätze in 1 und 2 wurden im Projekt open_eGo erstellt (Abschlussbericht hier abrufbar).

Datentabelle Ergebnisse: *stemp_abw.models.MunData*

Verbrauchsdaten Wärme

Die Wärmebedarfe von Haushalten wurden mit Hilfe folgender Daten und Studien gemeindescharf bestimmt:

- Wohngebäudebestand (Alter, Größe, Wohneinheiten, Leerstand etc.) nach Zensus 2011
- Einwohner nach Regionalstatistik
- Wohngebäudetypologie nach IWU
- Temperaturen nach DWD1

Die gemeindescharfen Wärmebedarfe für Gewerbe, Handel, Dienstleistungen und Landwirtschaft basieren auf folgenden Studien und Daten:

- Bevölkerung und Erwerbstätigkeit nach STALA1
- Energieverbrauch des Sektors GHD nach BMWi
- Arbeitsmarktstatistiken der Bundesagentur für Arbeit
- Temperaturen nach DWD1

Datentabelle Ergebnisse: *stemp_abw.models.MunData*

Einspeisezeitreihen

Verwendetes Wetterjahr: 2013 (in Region ABW ca. 100 % Globalstrahlung nach DWD2 und 100 % Windenergieertrag nach IEE1 im Vergleich zum langjährigen Mittel)

Windenergie

1. Wetterdaten herunterladen und aufbereiten (hier coastdat2).
2. Zuordnung der Gemeinden zu Zellen des Wettermodells
3. Kennlinien von betrachteten Anlagentypen auswählen: Hier entsprechend der Marktanteile in Deutschland nach IEE2 (2017):

- 40 % Enercon (E-82 mit 85 m und 98 m Nabenhöhe und E-115 mit 122 m Nabenhöhe) und 24 % Vestas (V90 mit 80 m und 100 m Nabenhöhe und V112 mit 119 m Nabenhöhe), hochskaliert auf 100 %: Enercon 63 %, Vestas 37 %.
- Die einzelnen Anlagentypen wurden anhand des Anlagenbestandes (Kraftwerksdaten wie oben beschrieben) vereinfacht in 2 Klassen <2,5 MW (87 %) und >2,5 MW (13 %) sortiert und die o.g. 6 Typen entsprechend gewichtet.
- Für Zukunftsszenarien: Enercon E-141 mit 159 m Nabenhöhe verwendet

4. Erzeugung von normierten Zeitreihen (stündlich) pro Technologie und Gemeinde für a) Status quo und b) Zukunftsszenerien.
5. Erhöhung der Repräsentativität durch Skalierung der Status-quo-Zeitreihen auf langjähriges Mittel der Jahresvolllaststunden von Sachsen-Anhalt anhand von Erhebungen der AEE1 (2011-2015: 1630 h).

Dieser Prozess wurde mit Hilfe von reegis automatisiert durchgeführt.

Photovoltaik

1. Normierte Einspeisezeitreihen herunterladen von renewables.ninja (Wetterdatensatz: CM-SAF SARAH)
2. Anlagen-Setting:
   - 20 % Systemverluste nach ISE
   - Tilt: 45° (Dach), 35° (Freifläche/Flachdach, optimale Ausrichtung für DE)
   - Azimut: 180°, Berücksichtigung verschiedener Ausrichtungen auf Dächern durch nachträgliche Ertragskorrektur mit Minderungsfaktor von 0,85 nach FfE.
3. Korrektur der Zeitreihen anhand der mittleren Jahresvolllaststunden nach AEE2, für Sachsen-Anhalt (2011-2015: 998 h), da mit Wetterdatensatz CM-SAF SARAH produzierte Einspeisung tendenziell zu niedrig ist (Pfenninger et al.):
   - Aus Kraftwersdaten folgt: Anteil Dachanlagen an Gesamtleistung: 20%, Anteil Freiflächenanlagen an Gesamtleistung: 80%
   - Anhand dieser Gewichtung werden die Zeitreihen skaliert, sodass beim aktuellen Bestand für alle 20 Gemeinden die mittlere Vollaststundenzahl (s.o.) von 998 h erreicht werden.
   - Vernachlässigt: Modul- und Wechselrichterkonfiguration, Flach- und Fassadenbauweise, Degradation, Tracking, variabler Airmass-Faktor
4. Erzeugung von normierten Zeitreihen (stündlich) pro Typ (Dach, Freifläche) und Gemeinde, die sowohl für den Status quo als auch Zukunftsszenerien verwendet werden.

Laufwasserkraft

1. Es wird eine konstante Einspeiseleistung über das gesamte Jahr an allen Anlagen angenommen.
2. Erzeugung konstante, normierte Zeitreihe mit mittlerer Jahresvolllaststundenzahl in Sachsen-Anhalt nach STALA2 (2012-2017: 3833 h).

Konventionelle Kraftwerke

Unterteilung in 2 Klassen nach Netto-Stromerzeugungsleistung:

- **>=10 Megawatt:** 2 Erdgaskraftwerke (106 MW, 40 MW), 1 Braunkohlekraftwerk (49 MW). Es wird eine stromgeführte Betriebsweise mit konstanter Einspeiseleistung angenommen mit typischen Werten für die Jahresvolllaststunden:
  - Erdgaskraftwerk (Turbine): 1250 h/a
  - Erdgaskraftwerk (GuD): 2900 h/a

  - Braunkohlekraftwerk: 7000 h/a

- **<10 Megawatt:** Hierzu zählen 21 Anlagen im Leistungsbereich von 750 kW bis 9,9 MW. Es werden eine wärmegeführte Betriebsweise und 5000 Jahresvolllaststunden angenommen.

Datentabelle Ergebnisse: *stemp_abw.models.FeedinTs*

Die normierten Zeitreihen werden beim Start des Tools geladen und anhand der eingestellten EE-Kapazitäten ad hoc auf absolute Zeitreihen skaliert (s. `stemp_abw.simulation.esys.prepare_feedin_timeseries()`).

Verbrauchszeitreihen Strom

1. Voraussetzung: Verbrauchsdaten (s.o.)
2. Erzeugung von absoluten Verbrauchszeitreihen (stündlich) mit Hilfe von BDEW-Standardlastprofilen, hierbei wurden die Typen H0 für Haushalte, G0 für GHD und L0 für Landwirtschaft verwendet. Für industrielle Verbraucher wurde ein Stufenlastprofil angenommen.

Datentabelle Ergebnisse: *stemp_abw.models.DemandTs*

Schritt 2 wurde mit Hilfe der demandlib durchgeführt.

Die Zeitreihen werden beim Start des Tools geladen und entsprechend den Tool-Einstellungen im Bereich Verbrauch skaliert (s. `stemp_abw.simulation.esys.prepare_demand_timeseries()`).

6.2.2 Flächen und Potenziale

Details zur Ermittlung der Potenzialflächen für erneuerbare Energieanlagen finden sie im Bereich *EE-Flächen und -Potenziale*.

CHAPTER 7

EE-Flächen und -Potenziale

7.1 Windenergieanlagen

Die Windenergie leistet in der Region Anhalt-Bitterfeld-Wittenberg (ABW) den größten Beitrag zur erneuerbaren Stromgewinnung - Ende 2017 deckten 438 Anlagen mit einer installierten Leistung von 717 Megawatt bilanziell bereits rund 45 % des regionalen Strombedarfs. Der weitere Ausbau sowie das Repowering bestehender Anlagen spielt eine entscheidende Rolle bei der Transformation des Energiesystems hin zu einer erneuerbaren Energieversorgung.

7.1.1 Potenzialflächen

Grundsätzlich gilt: WEA sind privilegierte Vorhaben nach §35 BauGB. Jedoch bedarf es einer Abwägung von wichtigen Belangen wie bspw. Natur- und Anwohnerschutz sowie der Steuerung der Windenergieplanung durch Ausweisung sog. Konzentrationsgebiete in Regional- und Flächennutzungsplänen mit Ausschlusswirkung im restlichen Planungsraum.

In ABW obliegt es der Regionalen Planungsgemeinschaft, derartige Vorrang-/Eignungsgebiete (VR/EG) auszuweisen und in Form eines Teilplans (Teilplan Wind ABW 2018) zu veröffentlichen. Der Planungsraum umfasst die Landkreise Wittenberg, Anhalt-Bitterfeld sowie die kreisfreie Stadt Dessau-Roßlau (Regionalplan ABW 2018).

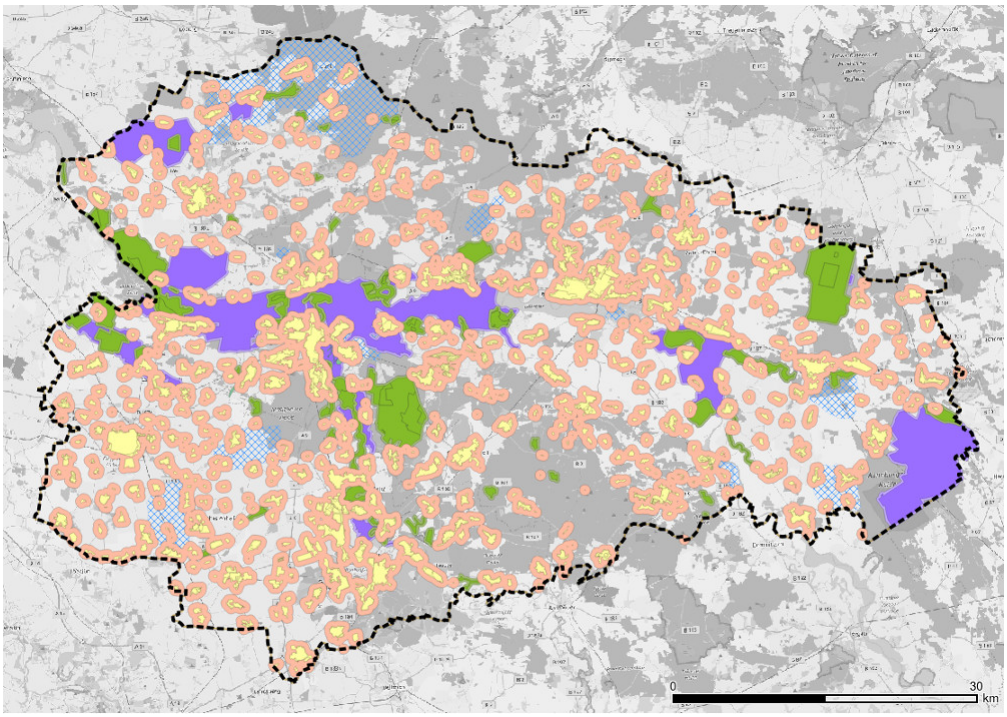
Vereinfacht dargestellt werden vom Planungsraum Restriktionsflächen (sog. “harte” und “weiche” Tabuzonen, s.u.) abgezogen, die aus verschiedenen Gründen (z.B. Naturschutz, vorhandener Infrastruktur, andersartigen Interessen etc.) für die Windenergienutzung nicht zur Verfügung stehen. Es resultieren potenziell geeignete Flächen - der sog. “Suchraum” - die rund 10 % des gesamten Planungsraums ausmachen. Diese Flächen werden einer Einzelfallprüfung (s.u.) unterzogen, die wiederum zu einem Ausschluss von etwa 90 % des Suchraums führen. Resultierend stehen somit für sie Ausweisung von VR/EG noch ca. 1 % des Planungsraums zur Verfügung, tatsächlich ausgewiesen als VR/EG wurden 3590 ha (0,98 %), vgl. Teilplan Wind ABW 2018.

7.1.2 Restriktionsflächen

Im Folgenden wird die Zusammensetzung der oben genannten Restriktionsflächen detaillierter beschrieben:

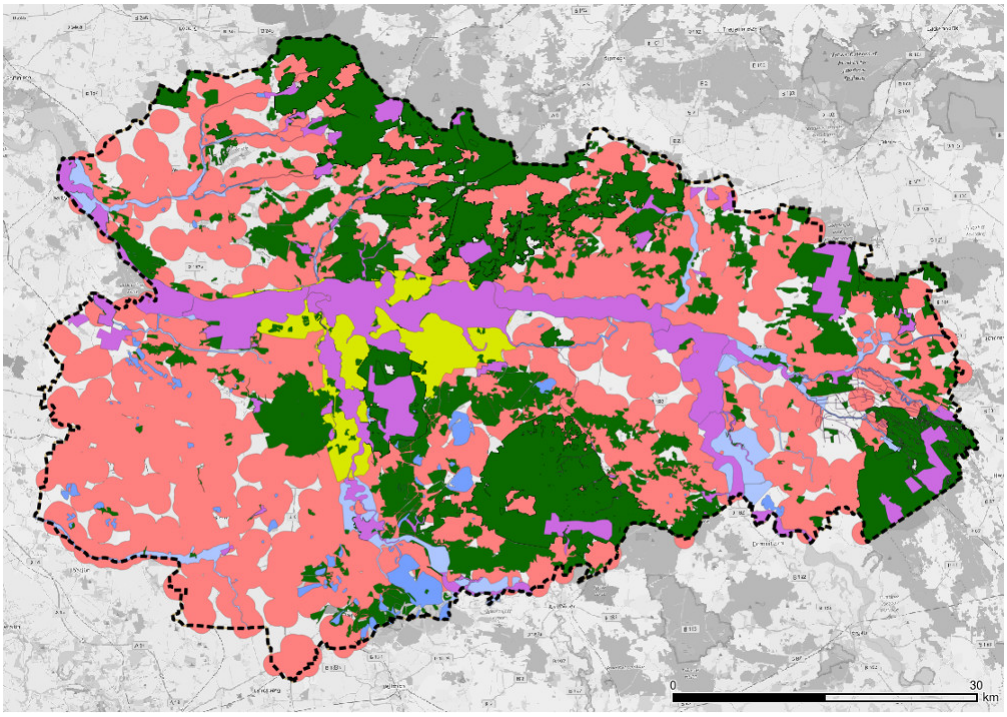
Harte Restriktionen (Tabuzonen)

- Siedlungsflächen mit überwiegender Wohn- und Erholungsnutzung, Kur- und Klinikgebiete (+500 m Puffer (Teilplan Wind ABW 2018))
- Verkehrs-, Sonder- und Hubschrauberlandeplätze
- Naturschutzgebiete (+200 m Puffer (UBA 2013))
- Geschützte Landschaftsbestandteile
- SPA-/Vogelschutzgebiete (+200 m Puffer (UBA 2013))
- Trinkwasserschutzgebiete Zone I + II



Weiche Restriktionen (Tabuzonen)

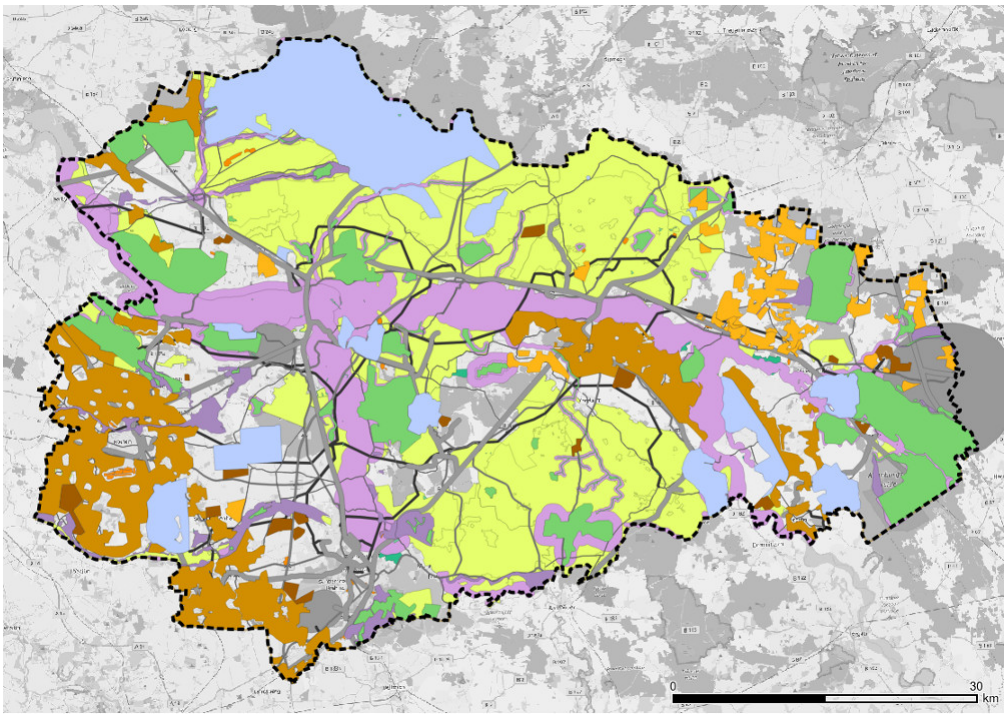
- 1000 m-Pufferzone um Siedlungsflächen mit überwiegender Wohn- und Erholungsnutzung, Kur- und Klinikgebiete (Teilplan Wind ABW 2018)
- Flora-Fauna-Habitat (FFH)-Gebiete
- Wald gem. § 2 WaldG LSA
- UNESCO-Welterbegebiete
- Überschwemmungsgebiete
- Gewässer, stehend >1 ha (+65 m Puffer (UBA 2013))
- Fließgewässer 1. Ordnung (+65 m Puffer (UBA 2013))



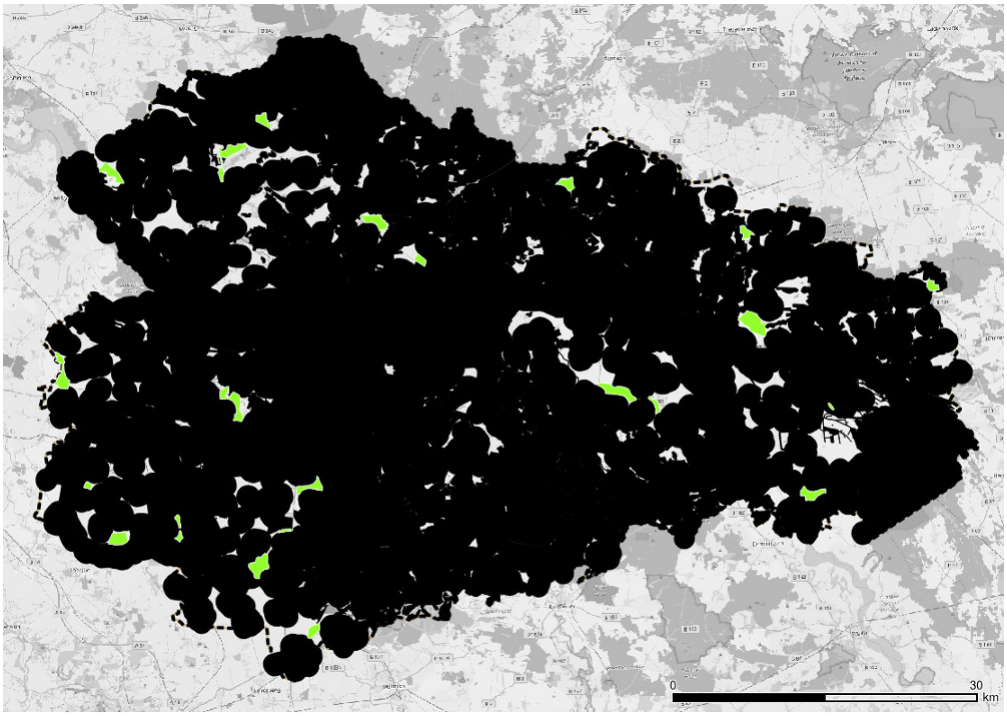
Abwägungsflächen/Einzelfallprüfung

- Schutzabstände zu FFH-Gebieten (je nach Vorkommen von Bechsteinfledermaus oder Großes Mausohr +200. . . 1000 m (Teilplan Wind ABW 2018))
- Landschaftsschutzgebiete
- Naturparke
- Biosphärenreservate
- Vorbehaltsgebiete für den Aufbau eines ökologischen Verbundsystems
- Vorranggebiete für Natur und Landschaft
- Naturdenkmale
- Flächennaturdenkmale
- Wasserschutzgebiet Zone III, Vorranggebiet für Wassergewinnung
- Vorrang- und Vorbehaltsgebiete für die Landwirtschaft
- Vorrang- und Vorbehaltsgebiete für die Rohstoffgewinnung
- Kommunale Planungen und Planabsichten, Landschaftsbild, private Belange, Erfordernisse der Raumordnung
- **Technische Infrastruktur und Vorbelastung**
  - BAB (+100 m Puffer (UBA 2013))
  - Bundesstraßen (+80 m Puffer (UBA 2013))
  - Land- und Kreisstraßen (+20 m Puffer (Teilplan Wind ABW 2018))
  - Schienenwege (+250 m Puffer (UBA 2013))
  - Bahnanlagen

- Flugverkehrsanlagen (je nach Nutzung +1760 m (zivil) (UBA 2013) … 6 km (militärisch) (Teilplan Wind ABW 2018) Puffer)
- Hochspannungsleitungen (+120 m Puffer (UBA 2013))
- Photovoltaik-Freiflächenanlagen



Folgende Grafik zeigt die nach Ausschluss von harten und weichen Tabuzonen (schwarz) verbleibenden Flächen sowie die letztendlich ausgewiesenen Vorrang-/Eignungsgebiete (VR/EG) (grün):



Die Restriktionsflächen finden Sie im Tool unter **Flächen -> Statische Flächen**.

7.1.3 Repowering

In Anlehnung an MLV 2018 ermöglicht das Tool die Einstellung verschiedener Repowering-Szenarien, die im Folgenden beschrieben werden. Zu beachten ist, dass diese Varianten teilweise unvereinbar mit der aktuellen Rechts- und Planungslage sind (vgl. LEntwG LSA, MULE 2019). Entsprechend dem visionären Charakter des Tools sollen sie vielmehr alternative Pfade beim Ausbau der Windenergie aufzeigen:

1. **Kein Repowering:** Es wird kein Repowering vorgenommen (nur heutige Anlagen vorhanden, mittlere Volllaststunden 2011-2015 für gesamte Region: 1630 (AEE 2018)).
2. **1:1-Repowering (standorttreu):** Standorttreues Repowering aller heute in Betrieb befindlichen Altanlagen durch eine neue Anlage, sowohl innerhalb als auch außerhalb von Vorranggebieten (VR/EG) für Windenergie.
3. **Volle Nutzung VR/EG:** In allen aktuellen Vorranggebieten (VR/EG) für Windenergie wird ein Maximum an Neuanlagen installiert. Alle Anlagen außerhalb dieser Gebiete werden abgebaut.
4. **Variabler Zubau:** Bei diesem Szenario können neben den Vorranggebieten (VR/EG) zusätzliche Flächen für die Windenergie freigegeben werden, die sich aus veränderten Siedlungsabständen und/oder der Nutzung von Waldflächen ergeben.

Die jeweiligen Potenziale können dem Abschnitt Ergebnisse entnommen werden.

7.1.4 Randbedingungen

Für den Windenergieausbau innerhalb des Tools gelten folgende Vereinfachungen:

- Die **Altersstruktur** der bestehenden Windenergieanlagen wird vernachlässigt, d.h. es wird kein sukzessiver Zubau berücksichtigt.
- Es wird keine Optimierung der Anlagenanordnung vorgenommen. Stattdessen wird **pauschal ein Flächenbedarf** von 20 Hektar pro Anlage zugrunde gelegt (basierend auf MLV 2018, UBA 2013 und BMWi 2017).

7.1.5 Musteranlage

Als Neuanlage wird in den **Repowering-Szenarien 2-4** vereinfacht eine Enercon E-141 (4,2 MW) mit einer Nabenhöhe von 159 m angenommen (im Mittel 2500 Volllaststunden in der gesamten Region).

Durch die höhere Effizienz neuer Anlagen kann bei gleicher installierter Leistung ein höherer Ertrag erreicht werden. Wenn Sie im Tool also das Repowering bzw. den freien Zubau aktiviert haben, führt dies auch ohne zusätzliche Kapazitäten stets zu einem höheren Ertrag.

7.1.6 Ergebnisse

1. **Kein Repowering:** Keine Änderungen
2. **1:1-Repowering (standorttreu):** 438 WEA * 4,2 MW/WEA = **1840 MW**
3. **Volle Nutzung VR/EG:** 3590 ha / 20 ha/WEA * 4,2 MW/WEA = **752 MW**
4. **Variabler Zubau:** Die Flächenpotenziale und maximal installierbare Leistung hängen von den vorgenommenen Flächeneinstellungen ab und sind in folgender Tabelle aufgeführt. Grundlage bilden die harten und weichen Tabuzonen. Während die Rechtslage und damit der Ausschluss der harten und weichen Tabuzonen klar geregelt ist, konnten jene Flächen, die einer Abwägung/Einzelfallprüfung unterliegen, im Projekt nicht eindeutig bewertet werden: Die Bewertung im Teilplan Wind ABW 2018 erfolgt anhand eines Punktesystems, das

als Grundlage nichtöffentliche Daten verwendet. Daher wird in Anlehnung an den Teilplan Wind ABW 2018 angenommen, dass lediglich 10 % der (nach Ausschluss harter und weicher Tabuzonen) resultierenden Flächen zur Verfügung stehen.

Abstand zu Siedlungen [m]	**Wald verwenden?**	**Ergebnisse**		
		Fläche [ha]	Installierbare Anzahl WEA (10% der Fläche)	Installierbare Leistung [MW]
500	nein	87847	439	1844
1000	nein	30713	154	752
1500	nein	7461	37	155
500	ja	147652	738	3100
1000	ja	69057	345	1449
1500	ja	25629	128	538

Der im Menü-Regler des Tools angezeigte Wert stellt die maximal installierbare unter den gewählten Randbedingungen dar.

7.2 Freiflächen-Photovoltaikanlagen

Freiflächen-Photovoltaikanlagen leisten mit einer installierten Kapazität von 445 Megawatt und einem Anteil von rund 18 % am Strombedarf der Region bereits Ende 2017 einen entscheidenden Beitrag. Der weitere Ausbau ist ein wichtiger Bestandteil auf dem Weg zu einer vollständig erneuerbaren Energieversorgung, welcher in Abwägung mit der Raum- und Umweltwirkung erfolgen muss.

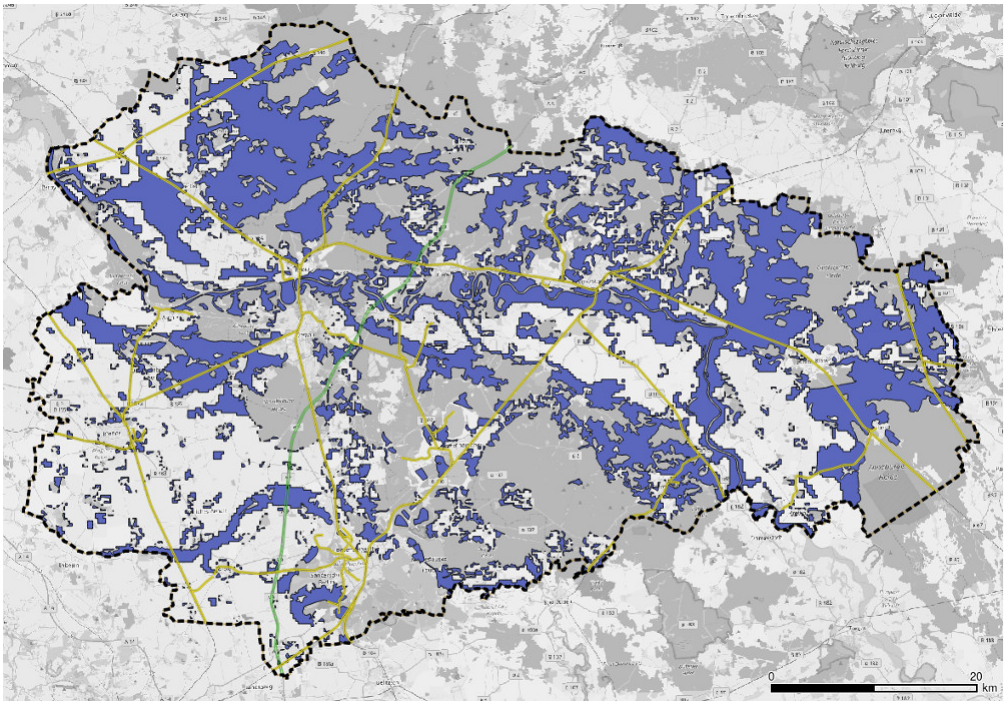
7.2.1 Potenzialflächen

Die Potenzialflächen werden in Anlehnung an die aktuelle Förderkulisse nach §37 EEG (2017) bestimmt. Es werden die folgenden Flächen berücksichtigt:

110 m-Streifen entlang von Bundesautobahnen und Schienenwegen Eine 110 m-Pufferzone um BAB und Schienenwege, abzüglich einer 40 m-Zone um BAB (§9 FStrG) und 10 m-Zone bei Schienenwegen (ZSW 2018) wird genutzt. Es werden weiterhin eine Breite von 24 m (BAB) respektive 12 m (Schienenwege) angenommen.

Böden mit geringem ackerbaulichen Ertragspotenzial (Bodengüte) Große Flächenpotenziale sind in den landwirtschaftlich genutzten Gebieten zu finden. In Anlehnung an ZSW 2019 werden Flächen mit geringem ackerbaulichen Ertragspotenzial (Bodengüte) herangezogen. Hierfür werden die Ackerflächen und Wiesen aus Corine Land Cover (CLC 2018) mit den Flächen sehr und äußerst geringer Bodengüte (<50) laut Soil Quality Rating der Bundesanstalt für Geowissenschaften und Rohstoffe (BGR 2014) verschnitten.

In der aktuellen Planung sollen laut Klima- und Energiekonzept des Landes Sachsen-Anhalt (MULE 2019) "die Errichtung von Photovoltaik-Freiflächenanlagen auf landwirtschaftlich genutzten Flächen weitestgehend vermieden werden". Um dieser Einschränkung Rechnung zu tragen, werden nach (ZSW 2019) lediglich **0,5..1,0 % als raumverträglich verfügbar** angenommen.



Weitere in der Förderkulisse des EEG enthaltenen Standorte Konversionsflächen und bundeseigene Immobilien finden hier aufgrund des vergleichsweise geringen Potenzials und mangelhafter Datenverfügbarkeit keine Berücksichtigung.

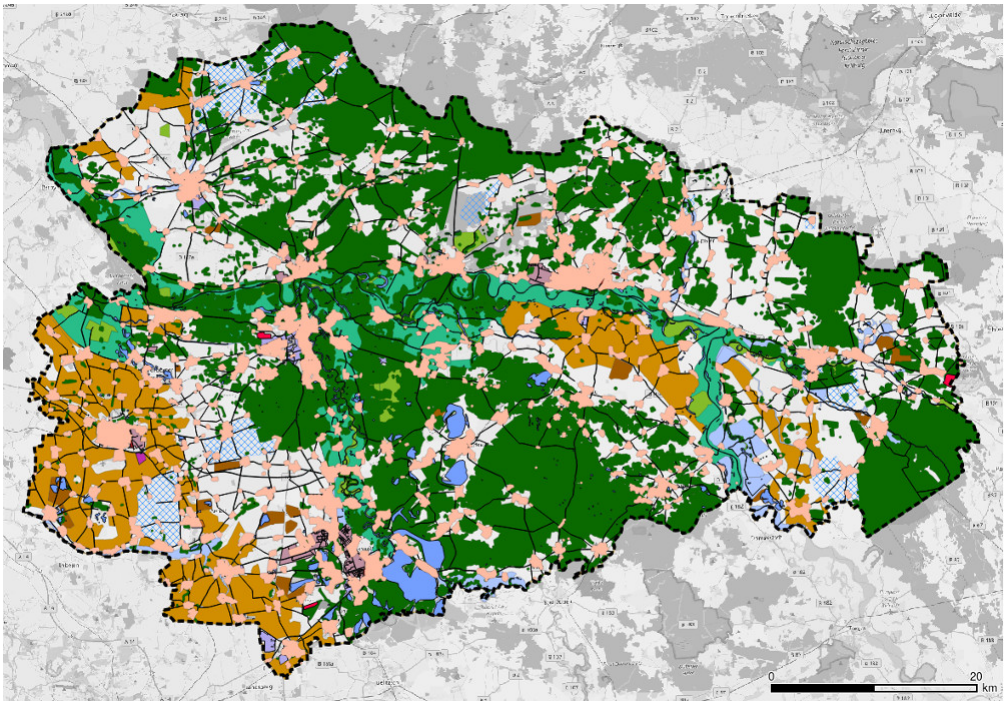
7.2.2 Restriktionsflächen

Den Potenzialflächen entgegen stehen Restriktionsflächen, in welchen die Errichtung von Photovoltaikanlagen ausgeschlossen (hart) oder unwahrscheinlich (weich) ist. Hierbei wurden sowohl die aktuelle Rechtslage als auch Planungskriterien einbezogen (vgl. BMVI 2015).

Harte Restriktionen

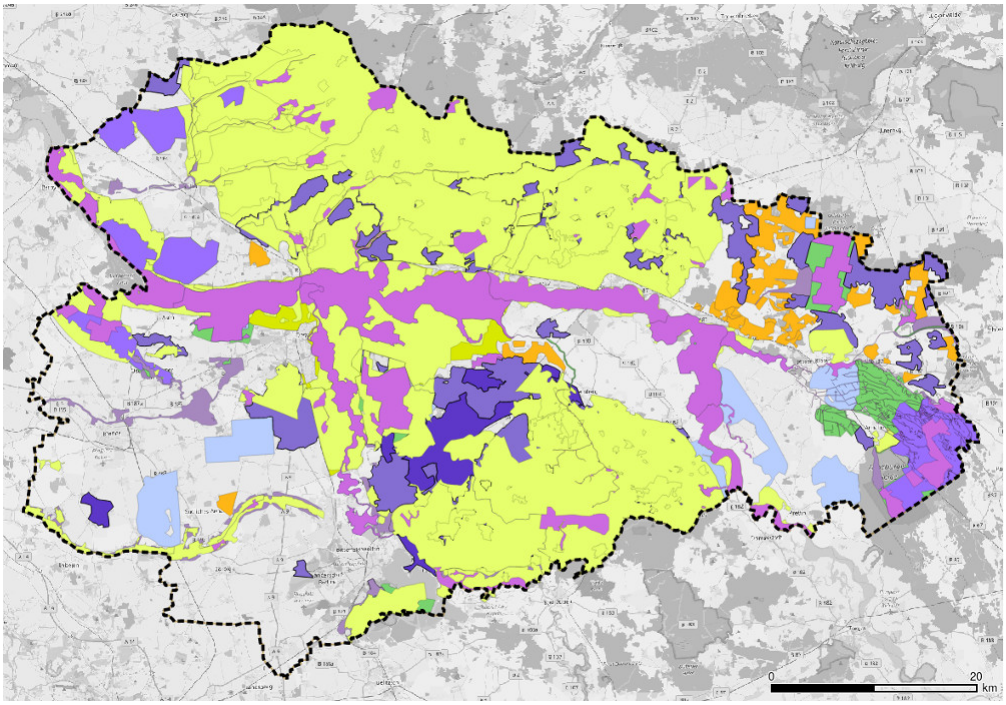
- Siedlungsflächen (+100 m Puffer)
- **Verkehrswege**
  - BAB: (24 m Breite, +40 m Puffer)
  - Bundes-, Land- und Kreisstraßen (+20 m Puffer)
  - Nebenstraßen (5 m Breite)
  - Schienenwege (12 m Breite, +10 m Puffer)
- Bahnanlagen
- Flugverkehrsanlagen
- Gewässer, stehend und fließend (+20 m Puffer)
- Wald (+100 m Puffer)
- Naturschutzgebiete
- Nationalpark (in ABW n.V.)
- Biosphärenreservate
- Überschwemmungsgebiete

- Flächenhafte Naturdenkmale
- Wasserschutzgebiete
- **Harte Restriktionsflächen nach Regionalplan ABW 2018**
  - Vorranggebiete für die Landwirtschaft
  - Vorranggebiete für die Rohstoffgewinnung
  - Landesbedeutsame Industrie- und Gewerbestandorte (Bestand + Planung)
  - Region bedeutsame Standort für Industrie und Gewerbe (Bestand + Planung)



Weiche Restriktionen

- FFH-Gebiete
- SPA-/Vogelschutzgebiete
- Landschaftsschutzgebiete
- Vorbehaltsgebiete für den Aufbau eine ökologischen Verbundsystems
- Vorranggebiete für Forstwirtschaft
- Naturparke
- Vorranggebiete für Natur und Landschaft
- Vorranggebiete für Wassergewinnung
- Vorbehaltsgebiete für Tourismus und Erholung
- UNESCO Weltkulturerbegebiet (Gartenreich Dessau-Wörlitz)
- Vorbehaltsgebiete für Landwirtschaft



7.2.3 Annahmen und Randbedingungen

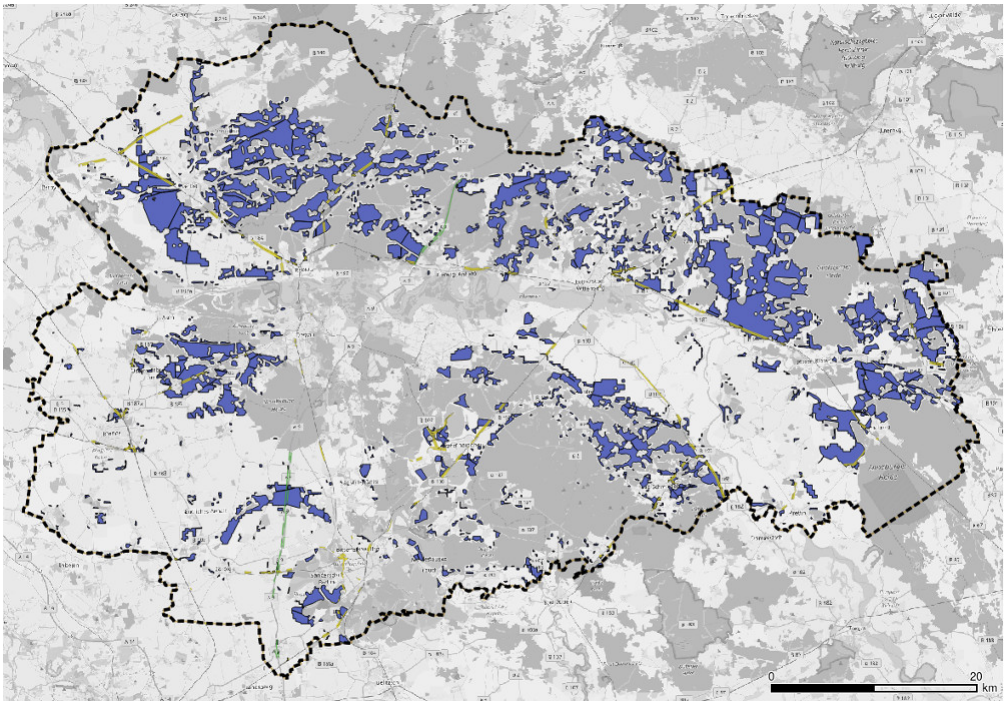
- Aus wirtschaftlichen Gründen werden nur Gebiete >1 ha berücksichtigt
- Bereis bestehende Anlagen und damit genutzte Flächen werden vernachlässigt
- Für den spezifischen Flächenbedarf werden 1,5 Hektar pro installiertem Megawatt (peak) angenommen. Gängige Werte reichen von 1,5 ha/MWp (ZSW 2019) bis zu 2..2,5 ha/MWp (BMWi 2017). Unter anderem aufgrund von Effizienzsteigerungen ist zukünftig von einem geringeren Flächenbedarf auszugehen (0,8 ha/MWp in 2030 nach ZSW 2019).
- Aus den weiter oben beschriebenen Gründen werden von Äckern und Wiesen nach (ZSW 2019) nur ein Teil als raumverträglich angenommen. Hierfür werden pauschal **1,0 %** angesetzt. Die gesamte Acker- und Wiesenfläche in ABW beträgt nach CLC 2018 208.578 ha, die maximal verfügbare Potenzialfläche ist demnach auf rund 2086 ha limitiert **(A)**.
- Gegenseitige Überschneidungen von Potenzialflächen sind minimal und werden daher vernachlässigt.

7.2.4 Ergebnisse

Um das tatsächlich verfügbaren Flächen zu bestimmen, werden die Restriktionsflächen von den Potenzialflächen abgezogen. Es ergeben sich die folgenden Potenziale für Flächen und maximal installierbare Leistung:

	Harte Restriktionen		Harte + Weiche Restriktionen	
	Fläche [ha]	Leistung [MWp]	Fläche [ha]	Leistung [MWp]
Bundesautobahn	226,0	150,7	138,0	92,0
Schienenwege	1959,0	1306,0	963,0	642,0
Äcker und Wiesen	45352,0	30234,7	13997,0	9331,3
Summe	**47537,0**		**15098,0**	
Bundesautobahn	226,0	150,7	138,0	92,0
Schienenwege	1959,0	1306,0	963,0	642,0
Äcker und Wiesen	*2086,0*	1390,7	*2086,0*	1390,7
Summe inkl. (A)	**4271,0**	**2847,4**	**3187,0**	**2124,7**

Der im Menü-Regler angezeigte Wert stellt die maximal installierbare Nennleistung unter Berücksichtigung der harten und weichen Restriktionsflächen dar (2125 MW).



CHAPTER 8

Datengrundlage

In diesem Tool werden ausschließlich offene Daten verwendet. Eine Liste aller verwendeten Daten kann hier im Tool eingesehen werden, jeder Datensatz enthält für eine größtmögliche Transparenz entsprechende Metadaten nach dem OpenEnergyPlatform Metadaten-Standard Version 1.4.

Statt jede Primärquelle einzeln aufzuführen, sind in den oben verlinkten Quellen die finalen, bereits aufbereiteten Datensätze gelistet. Jeder Datensatz hat eigene Metadaten und eine eigene Lizenz, die jeweils die zugrunde liegenden Quellen sind in den Metadaten vermerkt.

Alle Daten sind auf Zenodo verfügbar, siehe hierzu *Datenmigration*.

Das Format der Datenbanktabellen inkl. Spaltenbeschreibungen kann den Django-Modellen entnommen werden, s. `stemp_abw.models`.

Sofern nicht anders vermerkt, stehen alle Daten unter der Lizenz CC-BY-4.0.

CHAPTER 9

Übertragung des Tools auf andere Regionen

Das Tool kann durch seinen offenen Charakter grundsätzlich auf andere Regionen übertragen werden.

Der aufwändigste Teil ist hierbei die Recherche und Aufbereitung der Daten, siehe auch *Datengrundlage*.

Es werden **statische Geodaten** (im Tool Flächen -> Statische Flächen) verwendet, deren Verfügbarkeit je nach Region unterschiedlich ausfällt. Zudem müssen die Daten oft von von vielen verschiedenen Anbietern (Regionalplanung, Landesämter, Umweltbehörden, . . .) bezogen werden.

Die *EE-Flächen und -Potenziale* werden u.a. durch Verschneidung von geeigneten Flächen mit Restriktionsflächen erzeugt. Die Aufbereitung und Verarbeitung der Geodaten (z.B. mit QGIS) nimmt hierbei den meiste Zeit in Anspruch.

Weiterhin werden **Kraftwerksdaten** benötigt, die in vollständiger, georeferenzierter Form derzeit (Stand Sep. 2019) noch nicht verfügbar sind. Mit der Einführung des Marktstammdatenregisters (MaStR) ist hier ein wichtiger Schritt erfolgt, die Georeferenzierung dauert jedoch noch an. Das RLI hat ein offenes Tool entwickelt, mit dessen Hilfe der aktuelle Stand des heruntergeladen werden kann. Ein ständig aktualisierter Datensatz kann auch über die Website des RLI heruntergeladen werden. Alternativ (wie hier verwendet) kann auf den Datensatz von Open Power Systems Data (OPSD) zurückgegriffen werden.

Aus diesen Daten können unter Verwendung passender Wetterdaten vom Tool benötigte **Einspeisezeitreihen** für fluktuierende erneuerbare Energieanlagen generiert werden. Alternativ kann auf geeignete Tools zurückgegriffen werden, die fertige Zeitreihen generien, z.B. open_FRED (verwendet u.a. windpowerlib und pvlib) oder renewables.ninja.

Erforderliche **Verbrauchsdaten** können zum Beispiel in stark aggregierter Form von Statistischen Landesämtern oder detaillierter von der OpenEnergy Platform bezogen werden. Anhand von Standardlastprofilen können Zeitreihen generiert werden, etwa mit der demandlib.

Bei der Erstellung dieser Datensätze und darüber hinaus werden weitere Annahmen benötigt.

Bei allen verwendeten Daten ist stets die Lizenz zu beachten. Dürfen die Daten verändert oder veröffentlicht werden? Wenn ja, unter welchen Bedingungen? Wie müssen abgeleitete Daten lizenziert werden? Siehe hierzu auch 1 und 2.

Ein geeigneter Einstiegspunkt für die Energiesystemmodellierung ist die openmod initiative.

CHAPTER 10

Für EntwicklerInnen

Der Quellcode zu dieser StEmp-Web-Applikation ist auf GitHub verfügbar.

10.1 Technologien

10.2 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

10.3 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.

10.4 Tool-Struktur

In den vorherigen Abschnitten wurde die Pfadabhängigkeit dieses Projektes, mit dem Django-Framework und dem darauf aufbauenden WAM-Framework herausgearbeitet. An dieser Stelle soll nun auf die eigentliche Struktur dieses Projektes eingegangen werden. Dies setzt unter anderem Kenntnisse des Django-Frameworks und des WAM-Frameworks voraus. Es kann deshalb Sinn machen, erst nach einer Einarbeitung in Django und WAM sich diesem Abschnitt verstärkt zu widmen, falls die Voraussetzungen zum Zeitpunkt des ersten Lesens noch nicht in ausreichender Tiefe vorhanden sind.

Zuerst soll in diesem Abschnitt kurz daran erinnert werden, dass die WAM als Projektbasis dient und das auf der WAM aufbauende Applikationen im Ordner der WAM-Projekbasis zu finden sind. Das bedeutet, dass die WAM-Projektbasis der Gastgeber (Host) von multiplen WAM-Applikationen sein kann. Somit kann auf jeder WAM-Instanz ein bis viele WAM-Applikationen laufen. Diese Logik folgt der Logik von Django, also der Trennung von Projektbasis und Applikationen, welche auf dieser Projektbasis laufen. Eine WAM-Applikation, kann dabei auf zwei Arten in eine WAM-Projektbasis integriert werden:

- eine WAM-Applikation wird von Grund auf neu angelegt. Dies erfolgt mit den Bordmitteln von Django (Stichwort: `python manage.py startapp appname`).
- eine WAM-Applikation wird von einer bestehenden WAM-Applikation abgeleitet und in dem WAM-Projektbasis-Ordner manuell angelegt.

In beiden Fällen muss die neu zu erstellende Applikation konfiguriert und mit der WAM-Projektbasis verknüpft werden. Weitergehende Infos zur Installation und Inbetriebnahme einer neuen WAM-Applikation finden sich in der WAM-Dokumentation. In dieser Dokumentation soll deswegen vielmehr auf die konkrete Struktur dieses Projektes eingegangen werden, um EntwicklerInnen an die konkrete Codebasis heranzuführen. Die Strukturbetrachtung findet hierbei aus verschiedenen Blickwinkeln statt, um die Komplexität des Projektes besser durchdringen zu können.

In einer ersten groben Betrachtung widmet sich dieses Dokument im Folgenden der Ordnerstruktur des Projektes:

```
.
├── config
├── dataio
├── doc
├── fixtures
├── locale
├── management
├── migrations
├── results
├── simulation
├── static
├── templates
├── views
└── visualizations
```

Im Rootordner "." finden sich die für Django typischen Dateien, darunter sind aber auch einige Dateien, welche projektspezifischer Natur sind. Unter den projektspezifischen Dateien sind queries.py (welches Hilfsfunktionen für wiederkehrende Prozesse enthält) und sessions.py (in der User-Sessions gehandhabt werden) hervorzuheben.

Bei den Ordnern (Modulen) verhält es sich ähnlich. Einige sind typisch für Django (doc, fixtures, locale, management, migrations, static, templates, views), andere spezifisch für dieses Projekt (config, dataio, results, simulation, visualizations). Im Folgenden soll ausschließlich auf die projektspezifischen Module kurz eingegangen werden:

- *config:* Konfigurationsmodul, in dem Layer-, Label- und Kartenparameter definiert werden.
- *dataio:* Modul, in dem das Laden von statischen Daten gehandhabt wird.
- *results:* Modul, in dem die Resultate der Simulation behandelt werden.
- *simulation:* Modul, in dem die Simulation mit der Energiesystemmodellierungsframework oemof realisiert wird.
- *visualizations:* In diesem Modul befindet sich der Python-Wrapper für die JS-Chartsbibliothek.

Nach diesem kurzen strukturellen Überblick folgt nun ein funktionaler Überblick der wichtigsten Komponenten des Projektes. Eine komplette Beschreibung aller Schnittstellen findet sich im Kapitel API dieser Dokumentation.

10.5 User-Session

Beim ersten Besuch des Tools (oder einer anderen WAM-Applikation) wird eine User-Session mit einer eindeutigen ID erzeugt (fired by `stemp_abw.views.MapView.get()`). Die ID wird beim Client in einem Cookie für spätere Seitenbesuche abgelegt und nach Ablauf eines Gültigkeitszeitraums erneuert. Die Sessions werden nur für die Bereitstellung von Inhalten im Tool verwendet und in keiner Form anderweitig verwendet oder ausgewertet!

10.6 Geo-Ebenen (Layer)

Ebenen mit räumlichen Informationen werden an 4 Stellen im Tool verwendet:

1. Regions-Informationen (Panel "Region")
2. Statische Flächen (Panel "Flächen" -> "Statische Flächen")
3. Weißflächen (Panel "Flächen" -> "Variierbare Flächen")
4. Ergebnisse (Panel "Ergebnisse")

10.6.1 Hinzufügen eines neuen Layers

Wenn ein neuer Layer hinzugefügt werden soll, dann muss an sechs Stellen Code hinzugefügt und eine Migration (neues Modell) durchgeführt werden. Die sechs Stellen sind:

- models.py
- config/labels.cfg
- config/layers_<Panelname>.cfg
- templates/stemp_abw/popups/<Templatename-des-Popups>.html
- views/detail_views.py
- views/serial_views.py

Als Referenz für die Implementation von weiteren Layern, können folgende drei Commits exemplarisch herangezogen werden:

- Add layer for reg_mun_gen_count_wind_density_result #38
- Add layer for reg_mun_gen_cap_re_density_result #38
- Add layer for reg_mun_gen_cap_re_result #38

Wie sich aus den Commits entnehmen lässt folgt das Hinzufügen von weiteren Layern einem definierten Ablauf, welcher die Layer automatisch in das gewählte Panel hinzufügt, ohne das hierfür der HTML-Code des Panels angefasst werden muss. In den folgenden Abschnitten soll auf die einzelnen Schritte vertiefend eingegangen werden, indem exemplarisch auf die Erstellung eines Layers eingegangen wird.

10.6.2 Erstellung eines neuen Modells in *models.py*

Die Basis eines jeden neuen Layers ist ein Modell, aus dem der Layer seine Daten speist. Bei den Modellen handelt es sich um den bekannten Modellmechanismus aus Django. In diesem Projekt werden mit zwei Arten von Modellen gearbeitet:

- Modelle, welche mit einer Datenbanktabelle (via ORM-Mechanismus) korrespondieren
- Proxymodelle, welche von anderen Modellen erben und nicht direkt mit einer eigenen Datenbanktabelle korrespondieren, sondern mit den Datenbanktabellen der vererbten Modelle

In beiden Modellarten können über den *@property*-Dekorator weitere Eigenschaften definiert werden. In diesem Projekt ist dies z.B. in den Proxymodellen der Fall, hier werden neue Werte mit Hilfe der arithmetischen Grundrechenarten aus bestehenden Werten ermittelt und zurückgegeben.

Im Folgenden zwei Beispiele für das Modell RegMun und dem davon erbenden Proxymodell RegMunDemElEnergy:

- Klassendefinition des *RegMun*-Modells, mit Datenbanktabelle *stemp_abw_regmun*:

```
class RegMun(LayerModel):
    name = 'reg_mun'
    ags = models.IntegerField(primary_key=True)
    geom = geomodels.MultiPolygonField(srid=3035)
    geom_centroid = geomodels.PointField(srid=3035, null=True)
    gen = models.CharField(max_length=254)
```

Jedes Modell hat mindestens zwei definierte Eigenschaften *name* und *geom*. Mit der Eigenschaft *name* wird der Name definiert, welcher im Konfigurationsmodell (*config/*) Verwendung findet. Für die Benennung und Verwendung der Datenbanktabelle wiederum wird der Appname (*stemp_abw*) mit dem Klassennamen (*RegMun*) zu einem eindeutigen Tabellennamen von Django automatisiert verbunden (*stemp_abw_regmun*). Somit ist Obacht geboten, denn wir haben an zwei Stellen die Vergabe von Namensräumen für dasselbe Modell, einmal automatisiert für die Handhabung der

Daten und einmal manuell für die automatisierte Konfiguration und Verwendung des Modells in einem Layer. Mit der Eigenschaft *geom* wird die Geometrie des Layers mit dem Modell verknüpft. Alle weiteren Eigenschaften sind optional.

- Klassendefinition des *RegMunGenEnergyRe*-Proxymodells, ohne eigene Datenbanktabelle:

```
class RegMunDemElEnergy(RegMun):
    name = 'reg_mun_dem_el_energy'

    class Meta:
        proxy = True

    @property
    def dem_el_energy(self):
        return round((self.mundata.dem_el_energy_hh +
                      self.mundata.dem_el_energy_rca +
                      self.mundata.dem_el_energy_ind) / 1e3)

    @property
    def dem_el_energy_region(self):
        result = MunData.objects.aggregate(Sum('dem_el_energy_hh'))['dem_el_
↪energy_hh__sum'] + \
                 MunData.objects.aggregate(Sum('dem_el_energy_rca'))['dem_el_
↪energy_rca__sum'] + \
                 MunData.objects.aggregate(Sum('dem_el_energy_ind'))['dem_el_
↪energy_ind__sum']
        return round(result / 1e3)
```

In jedem Proxymodell wird ein eigener Name (*name*) als Eigenschaft vergeben, die Geometrie (*geom*) wird in der Regel geerbt. Das Proxymodell wird über *class Meta* als Proxyklasse gekennzeichnet. Weitere Schritte, für die Kennzeichnung eines Modells als Proxymodell, sind nicht nötig. An dem Beispiel von *RegMunGenEnergyRe* lässt sich die bereits erwähnte Verwendung des *@property*-Dekorators exemplarisch in den Methodendefinitionen von *dem_el_energy* und *dem_el_energy_region* alesen.

Nach der Erstellung eines oder mehrerer Modelle, sollte eine Datenbankmigration mit *python manage.py makemigrations* und *python manage.py migrate* durchgeführt werden, falls dies nötig ist. Der Befehl *python manage.py makemigrations* gibt Aufschluss darüber.

10.6.3 Die Registrierung und automatische Erstellung des Layers in einem Panel

Dieses Projekt verfügt über die Möglichkeit einen neuen Layer automatisiert einem bestimmten Panel hinzuzufügen. Dies wird durch die Definition des Layers in zwei Konfigurationsdateien ermöglicht:

- config/labels.cfg
- config/layers_<Panelname>.cfg

In *config/labels.cfg* wird hierbei das zu verwendende Panel, die Bezeichnung des Layers im Panel (*title*) und die (Tooltip-)Beschreibung des Layers im Panel (*text*) definiert. Eine vertiefende Beschreibung der Datenstruktur und ihrer Verwendung kann dem Dateikommentar in *config/labels.cfg* entnommen werden.

In *config/layers_<Panelname>.cfg* wird der Layer anhand des Modell konfiguriert und das Aussehen definiert. Im Folgenden eine generelle Übersicht:

```
Format:
[<GROUP_ID>]
    [[<LAYER_ID>]]
        model = <DATA MODEL NAME (property 'name' of model)>
```

(continues on next page)

(continued from previous page)

```
        geom_type = <TYPE OF GEOMETRY (line, point, poly)>
        show = <SHOW LAYER ON STARTUP (0/1)>
        sources = <COMMA-SEPARATED SOURCES ID(s) (PK from database)>, (0 = no source)
        [[[style]]]
            <CSS STYLE OPTIONS>
        [[[accuracy]]]
            <ACCURACY OF LAYER DISPLAY -> GEOJSON PARAMS>
        [[[choropleth]]]
            unit = <LEGEND TITLE>
            data_column = <MODEL PROPERTY USED AS DATA>
            color_schema = <COLORBREWER COLOR SCHEMA>
            min = <MIN VALUE FOR COLOR AND LEGEND (int or float)>
            max = <MAX VALUE FOR COLOR AND LEGEND (int or float)>
            step = <STEP SIZE FOR COLOR AND LEGEND (int or float)>
            reverse = <REVERSE COLOR SCHEMA (true/false)>
```

Anhand des konkreten Beispiels von RegMunDemElEnergy in config/layers_region.cfg soll an dieser Stelle exemplarisch auf die Konfiguration eines Layers eingegangen werden, welcher im Panel *Region* Verwendung findet:

```
[layer_grp_demand]
    [[reg_mun_dem_el_energy]]
        model = reg_mun_dem_el_energy
        geom_type = poly
        show = 0
        sources = 0
        [[[style]]]
            fillColor = '#41b6c4'
            weight = 1
            opacity = 1
            color = gray
            fillOpacity = 0.7
        [[[accuracy]]]
            precision = 5
            simplify = 0
        [[[choropleth]]]
            unit = 'GWh'
            data_column = dem_el_energy
            color_schema = YlGnBu
            min = 0
            max = 500
            step = 50
            reverse = false
```

[layer_grp_demand]: jedes Panel besteht aus Layergruppen. Die Bezeichnung und die Beschreibung einer Layergruppe wird, wie bei den Layern, in *config/labels.cfg* definiert. Der Layergruppenname wird je Layergruppe nur einmal angegeben.

[[reg_mun_dem_el_energy]]: der Name des Layers.

model = reg_mun_dem_el_energy: der Modellname (*name*) des Layers aus der Modelldefinition.

geom_type = poly: der Geometrietyp des Layers. Es stehen *line*, *point*, *poly* zur Verfügung.

show = 0: fragt ab, ob der Layer beim Start der Applikation sichtbar sein soll. In der Regel wird hier 0 angegeben. Mögliche Werte: 0 oder 1 (falseltrue).

sources = 0: jedem Layer kann auf bestimmte Quellen zu den Daten verweisen, welche im Gesamten über die URL *<Hostname>/stemp_abw/sources/* im Browser zugänglich sind. Die Quellen werden im Backend (*<Hostname>/admin/*) angelegt. Es können pro Layer mehrere Quellen verwendet werden (*1, 2, 3, ... n*). Die Angabe

erfolgt kommagetrennt und entspricht dem Primärschlüssel (PK) der jeweiligen Quelle in der Datenbank. In unserem Beispiel wird keine Quelle angegeben (deswegen der Wert 0).

[[[style]]]: in diesem Abschnitt wird das grundlegende Styling eines Layers definiert.

fillColor = '#41b6c4': der Parameter *fillColor* definiert die Grundfarbe des Layers und nimmt als Wert alle Werte entgegen, welche vom CSS *color*-Attribut entgegen genommen werden können (z.B. Hexadezimalwerte und sprechende Bezeichnungen).

weight = 1?: der Parameter *weight* definiert die Randstärke eines Layers. Ein Wert von 10 steht hierbei beispielsweise für eine Randstärke von 10 Pixeln. In der Regel steht der Wert bei 1.

opacity = 1: der Transparenzwert des Randes eines Layers. Bei dem Wert handelt es sich um einen Dezimalwert von 0 bis 1. Dieser Wert ist in der Regel 1.

color = gray: mit dem Parameter *color* wird die Farbe des Randes definiert. Dieser Wert ist in der Regel grau (*gray*).

fillOpacity = 0.7: der Transparenzwert eines Layers. Bei dem Wert handelt es sich um einen Dezimalwert von 0 bis 1. Dieser Wert liegt in der Regel bei 0.7, damit der Layer teildurchsichtig ist.

[[[accuracy]]]: in diesem Abschnitt wird die Genauigkeit definiert, mit der die Geometriedaten eines Layers angezeigt werden sollen.

precision = 5: der Parameter *precision* wird als Ganzzahl angegeben und definiert die Anzahl von Nachkommastellen, welche bei den Geometriewerten eines Layers berücksichtigt werden sollen. Dieser Wert ist in der Regel 5. Der Parameter *precision* spiegelt hierbei das Verhalten des Attributes precision aus der Django GEOS API, welcher in diesem Projekt als Unterbau Verwendung findet.

simplify = 0: der Parameter *simplify* definiert inwieweit die Geometrie eines Layers vereinfacht werden soll. Weil dieser Prozess rechenintensiv ist wird er in der Regel in diesem Projekt nicht verwendet und deswegen der Wert auf 0 gesetzt. Der Parameter *simplify* spiegelt hierbei das Verhalten des Attributes simplify aus der Django GEOS API, welcher in diesem Projekt als Unterbau Verwendung findet.

[[[choropleth]]]: in diesem Abschnitt wird, falls es sich bei dem Layer um eine Choroplethkarte handelt, diese definiert. Jede Choroplethkarte hat zusätzlich noch rechts unten eine Legende, welche eine Farbskala mit ihren Werten beschreibt.

unit = 'GWh': Einheit, welche in der Legende als Maßeinheit verwendet wird. Der Wert wird als String angegeben.

data_column = dem_el_energy: Der Parameter *data_column* enthält den *property*-Wert, welcher als Wert in der Choroplethkarte auf Gemeindeebene Verwendung finden soll. Der *property*-Wert wird zwar im Modell definiert, aber in *views/serial_views.py* für die Verwendung im Layer explizit ausgewiesen.

color_schema = YlGnBu: Der Parameter *color_schema* definiert das Farbschema, welches in der jeweiligen Choroplethkarte Verwendung findet. Mögliche Werte richten sich nach den von Cynthia Brewer entwickelten Farbschemata. Mit dem von Frau Brewer entwickelten Online-Tool colorbrewer2.org lassen sich die passenden Farbschemata und ihre Bezeichnungen ermitteln. Um diese Funktionalität zur Verfügung zu stellen, verwendet dieses Projekt die JavaScript-Farbbibliothek Chroma.js als Unterbau.

min = 0: der Parameter *min* definiert einen Minimalwert für die Choroplethkarte. Dieser Minimalwert sollte sich am Minimalwert aller Werte aus *data_column* orientieren.

max = 500: der Parameter *max* definiert einen Maximalwert für die Choroplethkarte. Dieser Maximalwert sollte sich am Maximalwert aller Werte aus *data_column* orientieren.

step = 50: der Parameter *step* definiert die Schrittgröße einer Farbabstufung einer Choropletkarte. Hierbei sollten sinnvolle Werte verwendet werden, welche mehrfach in das Intervall von Maximalwert minus Minimalwert passen. In unserem Beispiel hat das Intervall eine Länge von 500, eine Schrittgröße von 50 und somit zehn Farbabstufungen in der Choroplethkarte.

reverse = false: der Parameter *reverse* definiert, ob das verwendet Farbschema gedreht werden soll. Mögliche Werte sind hierbei *false* (nein) und *true* (ja). Ein Farbschema das z.B. bei dem Minimalwert blau und beim Maximalwert rot

ist, wird durch den Wert *true* vertauscht, so dass der Minimalwert rot und der Maximalwert blau ist.

10.6.4 Die Verwendung von angepassten Popup-Fenstern in Layern

In jedem Layer können Popup-Fenster verwendet werden, welche die einzelnen Elemente eines Layers genauer beschreiben. In diesen Popup-Fenstern können des Weiteren Charts verwendet werden, welche sich aus den Layerdaten speisen.

Standardmäßig ist ein Standard-Popup definiert, welcher Verwendung findet. Dieser kann angepasst werden, indem ein eigenes Popup-Template verwendet wird. Hierbei wird der von Django zur Verfügung gestellte Templatemechanimus verwendet, um das Standard-Popup zu erweitern.

Die Templates der Popups befinden sich im Ordner *templates/stemp_abw/popups/*. Falls für einen neuen Layer ein angepasstes Popup erstellt werden soll, bietet es sich an, eine bestehendes Popup-Template als Vorlage zu verwenden.

Im Folgenden soll exemplarisch auf das Popup-Template von RegMunGenEnergyRe eingegangen werden:

```
{% extends 'stemp_abw/popups/base_layer_popup.html' %}

{% block gen %}
  <div class="cell">
    <p>{{ layer.gen }}: {{ layer.gen_energy_re }} GWh</p>
  </div>
  <div>
    Region ABW: {{ layer.gen_energy_re_region }} GWh
  </div>
{% endblock %}

{% block vis %}
<div class="cell" style="height: 252px;">
  {{ chart }}
</div>
{% endblock%}
```

Im ersten Abschnitt "{% extends ..." wird vom Basis-Popup geerbt.

Im Block *gen* werden Angaben zur erzeugten Energie "layer.gen_energy_re" der Gemeinde "layer.gen" im Verhältnis zum Gesamtgebiet von ABW "layer.gen_energy_re_region" gemacht.

Im Block *vis* wird ein Chart (*chart*) eingebunden, welcher in der Detailview in *views/detail_views.py* definiert wird.

10.6.5 Die Erstellung der Detailansicht

Alle Detailansichten finden sich in *views/detail_views.py*. In der Detailansicht werden Modell und Template verbunden, damit das passende Popup bei einem Klick auf eine Element in einem bestimmten Layer angezeigt wird.

Einfache Detailansichten enthalten nur die Werte für das zu verwendende Modell (*model*) und das zugrunde liegende Template (*template_name*).

Komplexere Detailansichten enthalten darüber hinaus auch Methoden für die Übergabe des Django context (*get_context_data*) und die Erstellung eines Charts (*build_chart*), welcher mittels *{{ chart }}*-Tag im Template Verwendung findet.

10.6.6 Die Definition der zu serialisierenden Daten

Die Daten einer jeden Ansicht werden serialisiert und an einem bestimmten Endpunkt zur Verfügung gestellt, damit von der Applikation via AJAX-Abruf darauf zugegriffen werden kann.

Im Folgenden soll hierbei exemplarisch auf die Serialisierungsansicht von RegMunGenEnergyRe eingegangen werden:

```
class RegMunGenEnergyReData(GeoJSONLayerView):
    model = models.RegMunGenEnergyRe
    properties = [
        'name',
        'gen',
        'gen_energy_re',
        'gen_energy_re_region'
    ]
```

Als erstes wird das Modell (*model*) definiert, welches Verwendung finden soll.

In einem zweiten Schritt werden alle *properties* aus dem Modell definiert, welche serialisiert werden sollen, um an dem Endpunkt zur Verfügung zu stehen.

Bei den Layern der Gemeinden orientieren sich die Endpunkte an den Amtlichen Gemeindeschlüsseln (AGS). Die Endpunkte bei der Gemeinde Dessau mit dem AGS-Wert 15001000 sind somit:

```
stemp_abw/popup/reg_mun_gen_energy_re/15001000/
stemp_abw/popupjs/reg_mun_gen_energy_re/15001000/
```

Unter *stemp_abw/popup/* finden sich hierbei die menschenlesbaren Daten für das Popup und unter *stemp_abw/popupjs/* befinden sich Daten, wenn ein Chart in einem Popup Verwendung findet.

10.7 Energiesystem

- Wo werden die Komponenten definiert?

10.8 Szenarien

Die Szenarien werden im Modell/der Tabelle *stemp_abw.models.Scenario* definiert. Wie sich der API-Dokumentation entnehmen lässt, gehören zu jedem Szenario-Datensatz weitere Datensätze:

- Szenario-Daten: *stemp_abw.models.ScenarioData*
- Ergebnisse: *stemp_abw.models.SimulationResults*
- EE-Potenzialflächen: *stemp_abw.models.REPotentialAreas*
- Repowering-Szenario: *stemp_abw.models.RepoweringScenario*

Das Einfügen der Szenarien-betreffenden Datensätze erfolgt durch das Skript *queries.py*. Standardmäßig ist hier nur das Szenario *Status quo* vorhanden, kann jedoch beliebig erweitert werden.

Anmerkung: Die Szenario-Daten enthalten eine eindeutige UUID, die aus dem Hash des Daten-JSON erzeugt wird. Beim Start der Optimierung durch die Userin wird geprüft, ob für diese UUID bereits Ergebnisse vorliegen. Ist dies der Fall, werden diese geladen statt das Energiesystem erneut zu optimieren. Auf diese Weise kann die Darstellung der Ergebnisse erheblich verkürzt werden (s. `stemp_abw.sessions.Simulation.load_or_simulate()`).

10.9 Hilfetexte

Die StEmp-ABW-Applikation ist gespickt mit Hilfetexten, welche an folgenden Stellen Verwendung finden:

- Layer
- Layergruppen
- Komponenten
- Komponentengruppen
- Panels
- Tooltips
- Szenarien
- Charts

Die Hilfetexte werden hierbei in der Datei *labels.cfg* definiert, welche sich im jeweiligen Sprachunterordner im locale-Ordner (*stemp_abw/locale*) befindet. Je Sprache gibt es hierbei genau eine *labels.cfg*-Datei. Die zu verwendende Formatierung sieht hierbei wie folgt aus

```
[groups]
 [[<GROUP_ID>]]
     title = <TITLE OF GROUP>
     text = '''<DESCRIPTIVE TEXT OF GROUP WITH EACH LINE HAVING ABOUT 50 CHARACTERS>''
↪'
[entities]
 [[<ENTITY_ID>]]
     title = <TITLE OF ENTITY>
     text = '''<DESCRIPTIVE TEXT OF ENTITY WITH EACH LINE HAVING ABOUT 50 CHARACTERS>'
↪''
     text2 = '''<ANOTHER TEXT OF ENTITY> (Supported in components)'''
     reveal_id = <ID OF REVEAL WINDOW> (Supported in components.
                 If provided, the tooltip is replaced by a reveal
                 window with content from markdown file in
                 config/reveals)
     reveal_icon = <ION ICON FOR REVEAL BUTTON, MUST BE PROVIDED IF
                    reveal_id IS SET>
     icon = <ICON FILE NAME> (Supported in components, located in
                              static/stemp_abw/img/energy/icons/)
```

Die Hilfetexte werden über die StEmp-ABW-eigene i18n-Funktionalität realisiert, indem sie dynamisch als ConfigObj-Instanzen in stemp_abw/app_settings.py in der passenden Sprache eingebunden werden. Mehr zur Mehrsprachigkeitsfunktionalität in stemp_abw/app_settings.py auch im Abschnitt *Sprachpakete*.

10.10 Konfigurationsdateien

Neben den in den vorherigen Abschnitten erwähnten existieren weitere Konfigurationsdateien, die von der WAM eingelesen werden:

10.10.1 app.cfg

Die app.cfg dient als Setup-Datei für den WAM-Launcher, der WAM-Launcher ist die Startseite der WAM-Projektbasis in der Apps, welche in einer WAM installiert sind, aufgelistet werden.

Dabei sind folgende Variablen zu konfigurieren - Beispiel anhand von StEmp-ABW:

```
category = app
name = 'StEmp-Tool Anhalt-Bitterfeld-Wittenberg'
icon = 'stemp_abw/img/app_stemp_abw_icon.png'
email = 'jonathan.amme@rl-institut.de'
```

category: Definition der Kategorie. Standardname ist *app*.

name: Name des Projektes.

icon: Pfad und Dateiname zum Icon der App des Projektes, welches im WAM-Launcher angezeigt wird.

email: E-Mailadresse der/des Appveranwortlichen.

10.10.2 settings.py

Neben der Standard-Django settings.py in der WAM-Projektbasis (*wam/settings.py*) gibt es im stemp_abw-Projektordner ebenfalls eine *stemp_abw/settings.py*. Die darin enthaltenen Konstanten werden zu den Konstanten in *wam/settings.py* der WAM-Projektbasis hinzugeladen, so das appspezifische Konfiguration zu den globalen WAM-Konstanten hinzugefügt und über *wam.settings* importierbar sind.

Alle in stemp_abw/settings.py hinzugefügten Konstanten werden somit zur Laufzeit zu Konstanten in wam/settings.py.

10.10.3 app_settings.py

Die Konstanten und Funktionen in stemp_abw/app_settings.py wiederum sind appspezifisch für StEmp-ABW und bestehen hauptsächlich aus Konstanten und Funktionen (Callables), welche einen Teil der Mehrsprachigkeitsfunktionalität in StEmp-ABW realisieren sowie Teile der App-Computing-Funktionalität via cfg-Dateien mappen. Mehr zur Mehrsprachigkeitsfunktionalität in stemp_abw/app_settings.py auch im Abschnitt *Sprachpakete*.

10.11 Sprachpakete

Die StEmp-ABW-Applikation enthält Sprachpakete für Deutsch und Englisch und ist somit zweisprachig. Beide Sprachpakete befinden sich im Ordner *stemp_abw/locale*. StEmp-ABW verwendet hierbei sowohl den Django-i18n-Mechanismus als auch einen eigenen Implementationsteil, welcher auf ConfigObj-Dateien basiert.

Für die Funktionsweise des Django-i18n-Teils wird an dieser Stelle auf die offizielle Dokumentation verwiesen: Link.

Der StEmp-ABW Implementationsteil für Mehrsprachigkeit ist hierbei wie folgt. In der Datei *stemp_abw/app_settings.py* befinden sich folgende Konstanten:

1. DEFAULT_LANGUAGE
2. LANGUAGE_STORE

Die Konstante DEFAULT_LANGUAGE definiert die Standardsprache aus der settings.py WAM-Projektbasis. Zur Zeit ist diese Deutsch (de-DE).

Die Konstante LANGUAGE_STORE enthält alle zur Verfügung stehenden Sprachen. Zur Zeit sind dies en und de-DE.

Wenn jetzt in der Navigationsleiste der App eine Sprache ausgewählt und mit OK bestätigt wird, dann wird ein Post-Anfrage an eine Django-i18n-redirect-view gestellt, welche darauf u.a. die Sprache im Browser-Cookie auf die gewählte Sprache umstellt und die aktuelle Seite neu lädt. Dieser Mechnismus wird von den Callables in app_settings.py genutzt um dynamisch die passenden configObj in der passenden Sprache in locale zu verwenden. Die Callables in app_settings.py sind dabei alle Funktionen, welche ein ConfigObj oder eine Markdowndatei als Rückgabewert zurückgeben.

CHAPTER 11

What's New

Eine Liste neuer Funktionen, Verbesserungen und Bugfixes für jedes Release.

Releases

11.1 unreleased (??. ?? 2020)

11.1.1 Neue Features

- Ergebnis-Ebenen: Es werden nun Änderungen ggü. dem Status quo sowie Popups angezeigt #38

11.1.2 Änderungen

11.2 v1.0.4 (30. Oktober 2020)

11.2.1 Änderungen

- Mapbox Tiles entfernt, new default: OSM

11.3 v1.0.3 (06. April 2020)

11.3.1 Änderungen

- Tabelle Potenzialflächen für PV-Freiflächenanlagen korrigiert

11.4 v1.0.2 (06. April 2020)

11.4.1 Änderungen

- Potenzialflächen für PV-Freiflächenanlagen korrigiert #108:
  - mehr verfügbare Flächen auf Äcker und Wiesen
  - geringerer spezifischer Flächenbedarf (1,5 statt 2,5 ha/MWp) Details s. *EE-Flächen und -Potenziale*.

11.5 v1.0.1 (21. Februar 2020)

11.5.1 Änderungen

- Bugfix: Erträge aus Windenergie wurden falsch berechnet, wenn die installierbare Kapazität durch Änderung der variablen Flächen nach unten korrigiert wurden #106
- Tooltip Text Simulation-Button geändert

11.6 v1.0.0 (07. Oktober 2019)

Erste stabile Version

11.6.1 Neue Features

- Installation der vom Tool verwenden Daten ermöglicht via Django fixtures #71
- Dokumentation fertiggestellt #30 inkl. API docs #89
- Quellen den Layern zugeordnet und in den Popups verlinkt #104

11.6.2 Änderungen

- Wind-Potenziale korrigiert: Es werden nun lediglich 10 % der nach Abzug der harten und weichen Tabuzonen resultierenden Fläche als Potenzialfläche angenommen (vorher: 50 %).
- Quellenliste in Popup gefixt
- Diverse Bugfixes
- typo fixes in the english version

11.7 v0.9.0 (17. September 2019)

11.7.1 Neue Features

- Mehrsprachenunterstützung #72
- API docs hinzugefügt #89
- **ausführlichere Erklärtexte zu Potenzialen von**
  - Freiflächen-Photovoltaikanlagen #75
- El. Energiebedarf Industrie änderbar #86

11.7.2 Änderungen

- PV-Freiflächen-Potenziale korrigiert #75
- Einwohnerzahlen und Wärmebedarfe korrigiert #74
- Verbesserungen an der Benutzeroberfläche
- diverse Bugfixes

11.8 v0.2.0 (07. Juli 2019)

11.8.1 Neue Features

- Feedback-Formular
- der el. Energiebedarf kann nun auch erhöht werden
- ausführlichere Erklärtexte (z.B. bei Windenergieanlagen und konventionellen Kraftwerken)

11.8.2 Änderungen

- Verbesserungen an der Benutzeroberfläche
- diverse Bugfixes

11.9 v0.1.0 (25. Juni 2019)

Die erste öffentliche Version des Tools

11.9.1 Änderungen

keine

CHAPTER 12

stemp_abw

12.1 stemp_abw package

12.1.1 Subpackages

stemp_abw.config package

Submodules

stemp_abw.config.leaflet module

stemp_abw.config.prepare_context module

stemp_abw.config.prepare_context.**component_data**()

stemp_abw.config.prepare_context.**create_panel_reveal_info_button**(*reveal_id*, *reveal_icon*)
Creates reveal window with trigger button with content from markdown file (panel info button, e.g. in wind slider)

stemp_abw.config.prepare_context.**prepare_layer_data**()

stemp_abw.config.prepare_context.**prepare_scenario_data**()
Create scenarios for scenario dropdown menu (tool initialization only)

stemp_abw.config.prepare_texts module

stemp_abw.config.prepare_texts.**label_data**()

stemp_abw.config.prepare_texts.**text_data**()
Create reveal window with trigger button with content from markdown file (general app info buttons, e.g. in top navigation bar)

Module contents

stemp_abw.dataio package

Submodules

stemp_abw.dataio.load_static module

stemp_abw.dataio.load_static.**load_mun_data**()
Load municipality statistics

stemp_abw.dataio.load_static.**load_repowering_scenarios**()
Load repowering scenarios

stemp_abw.dataio.load_static.**load_timeseries**()
Load and format time series for all municipalities

Notes

- renewable timeseries from DB are normalized
- demand timeseries from DB are absolute in MW

Module contents

stemp_abw.management package

Subpackages

stemp_abw.management.commands package

Submodules

stemp_abw.management.commands.get_fixtures_stemp_abw module

class stemp_abw.management.commands.get_fixtures_stemp_abw.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: django.core.management.base.BaseCommand

add_argument(*parser*)

handle(**args*, ***options*)
The actual logic of the command. Subclasses must implement this method.

help = 'This command downloads all available fixtures for stemp_abw'

Module contents

Module contents

stemp_abw.results package

Submodules

stemp_abw.results.io module

stemp_abw.results.io.**oemof_json_to_results**(*results_json*)
Convert stored oemof results json back to result dicts

Parameters **results_json** (JSON formatted str) – Format see stemp_abw.results.tools.oemof_results_to_json()

Returns

- dict – Results from optimization as returned by oemof.outputlib.processing.results() (sequences only without timestamps, no scalars)
- dict – Parameters of optimization as returned by oemof.outputlib.processing.parameter_as_dict() (scalars only, no sequences)

stemp_abw.results.io.**oemof_results_to_json**(*results*, *param_results*)
Convert oemof raw results to json

Parameters

- **results** (dict) – Results from optimization as returned by oemof.outputlib.processing.results()
- **param_results** (dict) – Parameters of optimization as returned by oemof.outputlib.processing.parameter_as_dict()

Returns

Serialized results, format: {'param_results':

{'node_from_1':

{'node_to_1': {'scalars': {'param_1': val_1,

'param_2': val_2}

}

}

},

'results':

{'node_from_1':

{'node_to_1': {'sequences': [n values] }

}

}

}

Return type JSON formatted str

Notes

- Node keys in results and para_results must be provided as strings using `oemof.outputlib.processing.convert_keys_to_strings()`
- In the sequences, timestamps are not preserved (values only)

stemp_abw.results.result_charts module

stemp_abw.results.results module

stemp_abw.results.serializers module

class stemp_abw.results.serializers.**ResultLayerDataSerializer**(*options*)
Bases: `object`

Serializer for GeoJSON result layers

get_features(*queryset*)

serialize(*queryset*)
Serialize queryset

Module contents

stemp_abw.simulation package

Submodules

stemp_abw.simulation.bookkeeping module

stemp_abw.simulation.esys module

stemp_abw.simulation.simulation module

Module contents

stemp_abw.templatetags package

Submodules

stemp_abw.templatetags.language_tags module

stemp_abw.templatetags.language_tags.**language_store**()

Module contents

stemp_abw.views package

Submodules

stemp_abw.views.detail_views module

class stemp_abw.views.detail_views.**GenPVGroundDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.GenPVGround*

class stemp_abw.views.detail_views.**GenWECDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.GenWEC*

class stemp_abw.views.detail_views.**MasterDetailView**(***kwargs*)
Bases: django.views.generic.detail.DetailView

chart_session_store(*context*)

context_object_name = 'layer'

get_context_data(***kwargs*)
Insert the single object into the context dict.

get_source_data(*metadata*, *app_name*)
This method takes a metadata ConfigObj and returns a list with 0 OR n-amount of Source objects, if primary keys (PK)s of sources records in database are provided in ConfigObj object. Values in the metadata config file should correspond to PKs as list of values (1,2,3,...n). if the sole value 0 is provided in the metadata config file then the returned list is empty.

Parameters

- **metadata** (ConfigObj) –
- **app_name** (str) –

Returns List with 0 OR n-amount of Source objects.

Return type list of wam.meta.models.Source

mode = None

template_name = 'stemp_abw/popups/base_layer_popup.html'

class stemp_abw.views.detail_views.**RegBioReserveDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegBioReserve*

class stemp_abw.views.detail_views.**RegBirdProtAreaB200DetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegBirdProtAreaB200*

class stemp_abw.views.detail_views.**RegBirdProtAreaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegBirdProtArea*

class stemp_abw.views.detail_views.**RegDeadZoneHardDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegDeadZoneHard*

class stemp_abw.views.detail_views.**RegDeadZoneSoftDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegDeadZoneSoft*

class stemp_abw.views.detail_views.**RegFFHProtAreaBDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegFFHProtAreaB*

class stemp_abw.views.detail_views.**RegFFHProtAreaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegFFHProtArea*

class stemp_abw.views.detail_views.**RegForestDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegForest*

class stemp_abw.views.detail_views.**RegInfrasAviationDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegInfrasAviation*

class stemp_abw.views.detail_views.**RegInfrasHvgridDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegInfrasHvgrid*

class stemp_abw.views.detail_views.**RegInfrasRailwayDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegInfrasRailway*

class stemp_abw.views.detail_views.**RegInfrasRoadDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegInfrasRoad*

class stemp_abw.views.detail_views.**RegLandscProtAreaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegLandscProtArea*

class stemp_abw.views.detail_views.**RegLandscProtAreaPartsDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegLandscProtAreaParts*

class stemp_abw.views.detail_views.**RegMunDemElEnergyDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemElEnergy*

template_name = 'stemp_abw/popups/dem_el_energy.html'

class stemp_abw.views.detail_views.**RegMunDemElEnergyPerCapitaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemElEnergyPerCapita*

template_name = 'stemp_abw/popups/dem_el_energy_per_capita.html'

class stemp_abw.views.detail_views.**RegMunDemElEnergyPerCapitaResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemElEnergyPerCapitaResult*

template_name = 'stemp_abw/popups/result_dem_el_energy_per_capita.html'

class stemp_abw.views.detail_views.**RegMunDemElEnergyResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemElEnergyResult*

template_name = 'stemp_abw/popups/result_dem_el_energy.html'

class stemp_abw.views.detail_views.**RegMunDemThEnergyDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemThEnergy*

template_name = 'stemp_abw/popups/dem_th_energy.html'

class stemp_abw.views.detail_views.**RegMunDemThEnergyPerCapitaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunDemThEnergyPerCapita*

template_name = 'stemp_abw/popups/dem_th_energy_per_capita.html'

class stemp_abw.views.detail_views.**RegMunDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegMun*

class stemp_abw.views.detail_views.**RegMunEnergyReElDemShareDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunEnergyReElDemShare*

template_name = 'stemp_abw/popups/energy_re_el_dem_share.html'

class stemp_abw.views.detail_views.**RegMunEnergyReElDemShareResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunEnergyReElDemShareResult*

template_name = 'stemp_abw/popups/result_energy_re_el_dem_share.html'

class stemp_abw.views.detail_views.**RegMunGenCapReDensityDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenCapReDensity*

template_name = 'stemp_abw/popups/gen_cap_re_density.html'

class stemp_abw.views.detail_views.**RegMunGenCapReDensityResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenCapReDensityResult*

template_name = 'stemp_abw/popups/result_gen_cap_re_density.html'

class stemp_abw.views.detail_views.**RegMunGenCapReDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenCapRe*

template_name = 'stemp_abw/popups/gen_cap_re.html'

class stemp_abw.views.detail_views.**RegMunGenCapReResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenCapReResult*

template_name = 'stemp_abw/popups/result_gen_cap_re.html'

class stemp_abw.views.detail_views.**RegMunGenCountWindDensityDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegMunGenCountWindDensity*

template_name = 'stemp_abw/popups/gen_count_wind_density.html'

class stemp_abw.views.detail_views.**RegMunGenCountWindDensityResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegMunGenCountWindDensityResult*

template_name = 'stemp_abw/popups/result_gen_count_wind_density.html'

class stemp_abw.views.detail_views.**RegMunGenEnergyReDensityDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenEnergyReDensity*

template_name = 'stemp_abw/popups/gen_energy_re_density.html'

class stemp_abw.views.detail_views.**RegMunGenEnergyReDensityResultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(**kwargs)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenEnergyReDensityResult*

template_name = 'stemp_abw/popups/result_gen_energy_re_density.html'

class stemp_abw.views.detail_views.**RegMunGenEnergyReDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(**kwargs)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenEnergyRe*

template_name = 'stemp_abw/popups/gen_energy_re.html'

class stemp_abw.views.detail_views.**RegMunGenEnergyRePerCapitaDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(**kwargs)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenEnergyRePerCapita*

template_name = 'stemp_abw/popups/gen_energy_re_per_capita.html'

class stemp_abw.views.detail_views.**RegMunGenEnergyReResultDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(**kwargs)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunGenEnergyReResult*

template_name = 'stemp_abw/popups/result_gen_energy_re.html'

class stemp_abw.views.detail_views.**RegMunPopDensityDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegMunPopDensity*

template_name = 'stemp_abw/popups/pop_density.html'

class stemp_abw.views.detail_views.**RegMunPopDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

build_chart()

get_context_data(**kwargs)
Insert the single object into the context dict.

model
alias of *stemp_abw.models.RegMunPop*

template_name = 'stemp_abw/popups/pop.html'

class stemp_abw.views.detail_views.**RegNatureMonumDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegNatureMonum*

class stemp_abw.views.detail_views.**RegNatureParkDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegNaturePark*

class stemp_abw.views.detail_views.**RegNatureProtAreaDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegNatureProtArea*

class stemp_abw.views.detail_views.**RegPrioAreaAgriDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaAgri*

class stemp_abw.views.detail_views.**RegPrioAreaCultDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaCult*

class stemp_abw.views.detail_views.**RegPrioAreaFloodProtDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaFloodProt*

class stemp_abw.views.detail_views.**RegPrioAreaNatureDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaNature*

class stemp_abw.views.detail_views.**RegPrioAreaResDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaRes*

class stemp_abw.views.detail_views.**RegPrioAreaWECDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaWEC*

class stemp_abw.views.detail_views.**RegPrioAreaWaterDetailView**(***kwargs*)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegPrioAreaWater*

class stemp_abw.views.detail_views.**RegResidAreaB1000DetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegResidAreaB1000*

class stemp_abw.views.detail_views.**RegResidAreaB500DetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegResidAreaB500*

class stemp_abw.views.detail_views.**RegResidAreaDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegResidArea*

class stemp_abw.views.detail_views.**RegRetentAreaAgriDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegRetentAreaAgri*

class stemp_abw.views.detail_views.**RegRetentAreaEcosysDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegRetentAreaEcosys*

class stemp_abw.views.detail_views.**RegSurfaceWaterDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegSurfaceWater*

class stemp_abw.views.detail_views.**RegWaterProtAreaDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RegWaterProtArea*

class stemp_abw.views.detail_views.**RpAbwBoundDetailView**(**kwargs)
Bases: *stemp_abw.views.detail_views.MasterDetailView*

model
alias of *stemp_abw.models.RpAbwBound*

stemp_abw.views.serial_views module

class stemp_abw.views.serial_views.**GenPVGroundData**(**kwargs)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.GenPVGround*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**GenWECData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.GenWEC*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**GeoJSONResultLayerData**(**args*, ***kwargs*)
Bases: django.views.generic.list.ListView

Serial view with custom data in djgeojson's GeoJSON response

Modified version of GeoJSONResponseMixin - add custom data column before creating GeoJSON response. Municipalities (model RegMun) is used as base model.

Different from static serial layer views (examples see above) which use a specific model each, this view uses the dummy model *stemp_abw.models.ResultLayerModel* on initialization. The property "model" which is required by djgeojson is set dynamically using class method *stemp_abw.models.ResultLayerModel.name_init()*.

model_name
Name string of model (used as property "name" in dummy model stemp_abw.models.ResultLayerModel), see model for detailed description.

Type str

custom_property
Property (column) to be added to model, must be a column in layer results DataFrame results_df.

Type str

properties
Properties for each feature to be contained in the GeoJSON.

Type list of str

srid
SRID of CRS

Type int

geometry_field
Geometry field of model to be used for the GeoJSON

Type str

Notes

Attributes *model_name* and *custom_property* must be attributes of subclass.

dispatch(*request*, **args*, ***kwargs*)

geometry_field = 'geom'

model_name = None

properties = ['name', 'gen']

render_to_response(*context*, ***response_kwargs*)
Return a response, using the *response_class* for this view, with a template rendered with the given context.

Pass response_kwargs to the constructor of the response class.

result_property = None

class stemp_abw.views.serial_views.**GeoJSONSingleDatasetLayerView**(***kwargs*)
Bases: djgeojson.views.GeoJSONResponseMixin, django.views.generic.detail.DetailView

Serial view for single objects of djgeojson's GeoJSON response

Modified version of GeoJSONResponseMixin - filter queryset before creating GeoJSON response.

render_to_response(*context*, ***response_kwargs*)
Returns a JSON response, transforming 'context' to make the payload.

class stemp_abw.views.serial_views.**REPotentialAreasData**(***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONSingleDatasetLayerView*

geometry_field = 'geom'

model
alias of *stemp_abw.models.REPotentialAreas*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegBioReserveData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegBioReserve*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegBirdProtAreaB200Data**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegBirdProtAreaB200*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegBirdProtAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegBirdProtArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegDeadZoneHardData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegDeadZoneHard*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegDeadZoneSoftData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegDeadZoneSoft*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegFFHProtAreaBData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegFFHProtAreaB*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegFFHProtAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegFFHProtArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegForestData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegForest*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegInfrasAviationData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegInfrasAviation*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegInfrasHvgridData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegInfrasHvgrid*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegInfrasRailwayData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegInfrasRailway*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegInfrasRoadData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegInfrasRoad*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegLandscProtAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegLandscProtArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegLandscProtAreaPartsData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegLandscProtAreaParts*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegMunData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegMun*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegMunDemElEnergyData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunDemElEnergy*

properties = ['name', 'gen', 'dem_el_energy', 'dem_el_energy_region']

class stemp_abw.views.serial_views.**RegMunDemElEnergyPerCapitaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunDemElEnergyPerCapita*

properties = ['name', 'gen', 'dem_el_energy_per_capita', 'dem_el_energy_per_capita_reg

class stemp_abw.views.serial_views.**RegMunDemElEnergyPerCapitaResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_dem_el_energy_per_capita_result'

result_property = 'dem_el_energy_per_capita_result'

class stemp_abw.views.serial_views.**RegMunDemElEnergyPerCapitaResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunDemElEnergyPerCapitaDeltaResult*

properties = ['name', 'gen', 'dem_el_energy_per_capita_result_delta']

class stemp_abw.views.serial_views.**RegMunDemElEnergyResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_dem_el_energy_result'

result_property = 'dem_el_energy_result'

class stemp_abw.views.serial_views.**RegMunDemElEnergyResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunDemElEnergyDeltaResult*

properties = ['name', 'gen', 'dem_el_energy_result_delta']

class stemp_abw.views.serial_views.**RegMunDemThEnergyData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunDemThEnergy*

properties = ['name', 'gen', 'dem_th_energy', 'dem_th_energy_region']

class stemp_abw.views.serial_views.**RegMunDemThEnergyPerCapitaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunDemThEnergyPerCapita*

properties = ['name', 'gen', 'dem_th_energy_per_capita', 'dem_th_energy_per_capita_reg

class stemp_abw.views.serial_views.**RegMunEnergyReElDemShareData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunEnergyReElDemShare*

properties = ['name', 'gen', 'energy_re_el_dem_share', 'energy_re_el_dem_share_region'

class stemp_abw.views.serial_views.**RegMunEnergyReElDemShareResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_energy_re_el_dem_share_result'

result_property = 'energy_re_el_dem_share_result'

class stemp_abw.views.serial_views.**RegMunEnergyReElDemShareResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunEnergyReElDemShareDeltaResult*

properties = ['name', 'gen', 'energy_re_el_dem_share_result_delta']

class stemp_abw.views.serial_views.**RegMunGenCapReData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenCapRe*

properties = ['name', 'gen', 'gen_cap_re', 'gen_cap_re_region']

class stemp_abw.views.serial_views.**RegMunGenCapReDensityData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenCapReDensity*

properties = ['name', 'gen', 'gen_cap_re_density', 'gen_cap_re_density_region']

class stemp_abw.views.serial_views.**RegMunGenCapReDensityResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_gen_cap_re_density_result'

result_property = 'gen_cap_re_density_result'

class stemp_abw.views.serial_views.**RegMunGenCapReDensityResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunGenCapReDensityDeltaResult*

properties = ['name', 'gen', 'gen_cap_re_density_result_delta']

class stemp_abw.views.serial_views.**RegMunGenCapReResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_gen_cap_re_result'

result_property = 'gen_cap_re_result'

class stemp_abw.views.serial_views.**RegMunGenCapReResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunGenCapReDeltaResult*

properties = ['name', 'gen', 'gen_cap_re_result_delta']

class stemp_abw.views.serial_views.**RegMunGenCountWindDensityData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenCountWindDensity*

properties = ['name', 'gen', 'gen_count_wind_density', 'gen_count_wind_density_region'

class stemp_abw.views.serial_views.**RegMunGenCountWindDensityResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_gen_count_wind_density_result'

result_property = 'gen_count_wind_density_result'

class stemp_abw.views.serial_views.**RegMunGenCountWindDensityResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunGenCountWindDensityDeltaResult*

properties = ['name', 'gen', 'gen_count_wind_density_result_delta']

class stemp_abw.views.serial_views.**RegMunGenEnergyReData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenEnergyRe*

properties = ['name', 'gen', 'gen_energy_re', 'gen_energy_re_region']

class stemp_abw.views.serial_views.**RegMunGenEnergyReDensityData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenEnergyReDensity*

properties = ['name', 'gen', 'gen_energy_re_density', 'gen_energy_re_density_region']

class stemp_abw.views.serial_views.**RegMunGenEnergyReDensityResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_gen_energy_re_density_result'

result_property = 'gen_energy_re_density_result'

class stemp_abw.views.serial_views.**RegMunGenEnergyReDensityResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunGenEnergyReDensityDeltaResult*

properties = ['name', 'gen', 'gen_energy_re_density_result_delta']

class stemp_abw.views.serial_views.**RegMunGenEnergyRePerCapitaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunGenEnergyRePerCapita*

properties = ['name', 'gen', 'gen_energy_re_per_capita', 'gen_energy_re_per_capita_reg

class stemp_abw.views.serial_views.**RegMunGenEnergyReResultData**(**args*, ***kwargs*)
Bases: *stemp_abw.views.serial_views.GeoJSONResultLayerData*

model_name = 'reg_mun_gen_energy_re_result'

result_property = 'gen_energy_re_result'

class stemp_abw.views.serial_views.**RegMunGenEnergyReResultDeltaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom_centroid'

model
alias of *stemp_abw.models.RegMunGenEnergyReDeltaResult*

properties = ['name', 'gen', 'gen_energy_re_result_delta']

class stemp_abw.views.serial_views.**RegMunPopData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunPop*

properties = ['name', 'gen', 'pop', 'pop_region']

class stemp_abw.views.serial_views.**RegMunPopDensityData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

model
alias of *stemp_abw.models.RegMunPopDensity*

properties = ['name', 'gen', 'pop_density', 'pop_density_region']

class stemp_abw.views.serial_views.**RegNatureMonumData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegNatureMonum*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegNatureParkData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegNaturePark*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegNatureProtAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegNatureProtArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaAgriData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaAgri*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaCultData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaCult*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaFloodProtData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaFloodProt*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaNatureData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaNature*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaResData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaRes*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaWECData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaWEC*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegPrioAreaWaterData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegPrioAreaWater*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegResidAreaB1000Data**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegResidAreaB1000*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegResidAreaB500Data**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegResidAreaB500*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegResidAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegResidArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegRetentAreaAgriData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegRetentAreaAgri*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegRetentAreaEcosysData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegRetentAreaEcosys*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegSurfaceWaterData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegSurfaceWater*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**RegWaterProtAreaData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RegWaterProtArea*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**ResultChartsData**(***kwargs*)
Bases: django.views.generic.base.View

dispatch(*request*, **args*, ***kwargs*)

static get(*request*)

model = None

class stemp_abw.views.serial_views.**RpAbwBoundData**(***kwargs*)
Bases: djgeojson.views.GeoJSONLayerView

geometry_field = 'geom'

model
alias of *stemp_abw.models.RpAbwBound*

precision = 5

properties = ['name']

srid = 4326

class stemp_abw.views.serial_views.**SimulationStatus**(***kwargs*)
Bases: django.views.generic.base.View

Serial view to get simulation status

Returns GeoJSON response with simulation status, format: {'sim_status': <STATUS>} where <STATUS> is one of * 'init' (tool initialized, results contain those from SQ scenario) * 'up_to_date' (results are up-to-date) * 'outdated' (results are outdated)

Return type django.http.JsonResponse

See also:

Status

dispatch(*request*, **args*, ***kwargs*)

static get(*request*)

model = None

Module contents

stemp_abw.visualizations package

Submodules

stemp_abw.visualizations.highcharts module

class stemp_abw.visualizations.highcharts.**HCPiechart**(*theme='results'*, *data=None*, *tooltip_text=''*, *setup_labels=None*, ***kwargs*)
Bases: *stemp_abw.visualizations.highcharts.HCStemp*

setup = {'chart': {'backgroundColor': 'rgba(255, 255, 255, 0.0)', 'type': 'pie'}, '

class stemp_abw.visualizations.highcharts.**HCStackedColumn**(*theme='results'*, *data=None*, *tooltip_text=''*, *setup_labels=None*, ***kwargs*)
Bases: *stemp_abw.visualizations.highcharts.HCStemp*

setup = {'chart': {'backgroundColor': 'rgba(255, 255, 255, 0.0)', 'type': 'column'}

class stemp_abw.visualizations.highcharts.**HCStemp**(*theme='results'*, *data=None*, *tooltip_text=''*, *setup_labels=None*, ***kwargs*)
Bases: utils.highcharts.Highchart

setup = {}

tooltip

class stemp_abw.visualizations.highcharts.**HCTimeseries**(*theme='results'*, *data=None*, *tooltip_text=''*, *setup_labels=None*, ***kwargs*)

Bases: *stemp_abw.visualizations.highcharts.HCStemp*

setup = {'chart': {'backgroundColor': 'rgba(255, 255, 255, 0.0)', 'type': 'line'},

Module contents

12.1.2 Submodules

12.1.3 stemp_abw.admin module

12.1.4 stemp_abw.app_settings module

stemp_abw.app_settings.**get_language_or_fallback**()

stemp_abw.app_settings.**labels**()

stemp_abw.app_settings.**layer_areas_metadata**()

stemp_abw.app_settings.**layer_region_metadata**()

stemp_abw.app_settings.**layer_result_metadata**()

stemp_abw.app_settings.**month_labels**()

stemp_abw.app_settings.**node_labels**()

stemp_abw.app_settings.**text_files**()

stemp_abw.app_settings.**text_files_dir**()

12.1.5 stemp_abw.apps module

12.1.6 stemp_abw.forms module

class stemp_abw.forms.**AreaGroupForm**(*components=None*, **args*, ***kwargs*)

Bases: django.forms.forms.Form

Form for layer group (variable layers)

base_fields = {}

declared_fields = {}

media

class stemp_abw.forms.**ComponentGroupForm**(*components=None*, **args*, ***kwargs*)

Bases: django.forms.forms.Form

Form for esys components

base_fields = {}

declared_fields = {}

media

class stemp_abw.forms.**LayerGroupForm**(*cat_name=None*, *layers=None*, **args*, ***kwargs*)
Bases: django.forms.forms.Form

Form for layer group (regional info)

base_fields = {}

declared_fields = {}

media

class stemp_abw.forms.**ScenarioDropdownForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form

Form for scneario dropdown menu (predefined scenarios only)

base_fields = {}

declared_fields = {}

media

12.1.7 stemp_abw.helpers module

stemp_abw.helpers.**order_dict**(*dictionary*)
Order dictionary recursively

12.1.8 stemp_abw.models module

class stemp_abw.models.**DemandTs**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Demand timeseries (hourly, partly normalized - see columns)

id
DB id

timestamp
timestamp

ags
Municipality key (Amtlicher Gemeindeschlüssel), refers to *stemp_abw.models.RegMun*

el_hh
El. demand of households

el_rca
El. demand of retail, commercial and agricultural sector (GHD)

el_ind
El. demand of industry

th_hh_efh
Heat demand of households in single-family houses (Einfamilienhäuser), absolute, in MW

th_hh_mfh
Heat demand of households in multi-family houses (Mehrfamilienhäuser), absolute, in MW

th_hh_efh_spec
Heat demand of households in single-family houses (Einfamilienhäuser), area-specific in kWh/m^2

th_hh_mfh_spec
Heat demand of households in multi-family houses (Mehrfamilienhäuser), area-specific in kWh/m^2

th_rca
Heat demand of retail, commercial and agricultural sector (GHD) in MW

th_ind
Heat demand of industry in MW

Notes

Timeseries are stored per timestep and ags -> one dataset is uniquely identified by timestamp and municipality's ags.

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

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

ags
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.

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

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

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

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

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

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *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>

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

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

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

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

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

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

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

class stemp_abw.models.**FeedinTs**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Feedin timeseries (hourly, partly normalized - see columns)

id
DB id

timestamp
timestamp

ags
Municipality key (Amtlicher Gemeindeschlüssel), refers to *stemp_abw.models.RegMun*

pv_ground
Photovoltaics (ground-mounted systems) normalized (relative values)

pv_roof
Photovoltaics (roof-mounted systems) normalized (relative values)

hydro
Run-of-river plants normalized (relative values)

wind_sq
Wind turbines (status quo) normalized (relative values)

wind_fs
Wind turbines (future scenarios) normalized (relative values)

bio
Biogas/biomass plants (incl. landfill and sewage) normalized (relative values)

conventional
Conventional plants (>=10 MW: power-led, <10 MW: heat-led) NOT normalized (absolute values)

Notes

Timeseries are stored per timestep and ags -> one dataset is uniquely identified by timestamp and municipality's ags.

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

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

ags
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.

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

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

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

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

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

hydro
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>

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

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

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

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

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

class stemp_abw.models.**GenPVGround**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'gen_pv_ground'

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

class stemp_abw.models.**GenWEC**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'gen_wec'

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

class stemp_abw.models.**LayerModel**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

class Meta
Bases: object

abstract = False

name

class stemp_abw.models.**MunData**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Statistical data of municipalities (status quo)

ags
Municipality key (Amtlicher Gemeindeschlüssel), refers to *stemp_abw.models.RegMun*

area
Total area in km^2

pop_2011
Population (2011) according to Zensus

pop_2017
Population (2017) according to GV-ISys

pop_2030
Population (2030) forecast according to MLV Sachsen-Anhalt

pop_2050
Population (2050), linearly extrapolated using 2017 and 2030

total_living_space
Total living space (Wohnfläche) in m^2

gen_count_wind
Count of wind turbines

gen_count_pv_roof_small
Count of small (<=30 kVA) roof-mounted PV systems

gen_count_pv_roof_large
Count of large (>30 kVA, <=300 kVA) roof-mounted PV systems

gen_count_pv_ground
Count of ground-mounted PV systems (>300 kVA)

gen_count_hydro
Count of run-of-river systems

gen_count_bio
Count of biogas/biomass systems

gen_count_conventional_large
Count of large (>=10 MW) conventional plants in MW

gen_count_conventional_small
Count of small (<10 MW) conventional plants in MW. Simplified assumption: 1 plant per municipality

gen_count_sewage_landfill_gas
Count of sewage/landfill gas systems

gen_count_storage
Count of storages

gen_capacity_wind
Total nominal power of wind turbines in MVA

gen_capacity_pv_roof_small
Total nominal power of small roof-mounted PV systems in MW

gen_capacity_pv_roof_large
Total nominal power of large roof-mounted PV systems in MW

gen_capacity_pv_ground
Total nominal power of ground-mounted PV systems in MW

gen_capacity_hydro
Total nominal power of run-of-river systems in MW

gen_capacity_bio
Total nominal power of biogas/biomass systems in MW

gen_capacity_conventional_large
Total nominal power of large (>=10 MW) conventional plants in MW

gen_capacity_conventional_small
Total nominal power of small (<10 MW) conventional plants in MW

gen_capacity_sewage_landfill_gas
Total nominal power of sewage/landfill gas systems in MW

gen_capacity_storage
Total storage capacity of storages in MWh

gen_el_energy_wind
Annual el. energy fed in by wind turbines in MWh

gen_el_energy_pv_roof
Annual el. energy fed in by roof-mounted PV systems in MWh

gen_el_energy_pv_ground
Annual el. energy fed in by ground-mounted PV systems in MWh

gen_el_energy_hydro
Annual el. energy fed in by run-of-river systems in MWh

gen_el_energy_bio
Annual el. energy fed in by biomass/biogas systems incl. sewage and landfill gas in MWh

gen_el_energy_conventional
Annual el. energy fed in by conventional power plants in MWh (large >=10 MW and small <10 MW).

dem_el_peak_load_hh
El. peak demand of households in MW

dem_el_peak_load_rca
El. peak demand of retail, commercial and agricultural sector (GHD) in MW

dem_el_peak_load_ind
El. peak demand of industry in MW

dem_el_energy_hh
Annual el. energy consumed by households in MWh

dem_el_energy_rca
Annual el. energy consumed by retail, commercial and agricultural sector (GHD) in MWh

dem_el_energy_ind
Annual el. energy consumed by industry in MWh

dem_th_peak_load_hh
Heat peak demand of households in MW

dem_th_peak_load_rca
Heat peak demand of retail, commercial and agricultural sector (GHD) in MW

dem_th_peak_load_ind
Heat peak demand of industry in MW

dem_th_energy_hh
Annual heat consumed by households in MWh

dem_th_energy_hh_efh
Annual heat consumed by single-family households (Einfamilienhäuser) in MWh

dem_th_energy_hh_mfh
Annual heat consumed by multi-family households (Mehrfamilienhäuser) in MWh

dem_th_energy_hh_efh_spec
Annual heat consumed by single-family households (Einfamilienhäuser), area-specific in kWh/m^2

dem_th_energy_hh_mfh_spec
Annual heat consumed by multi-family households (Mehrfamilienhäuser), area-specific in kWh/m^2

dem_th_energy_rca
Annual heat consumed by retail, commercial and agricultural sector (GHD) in MWh

dem_th_energy_ind
Annual heat consumed by industry in MWh

dem_th_energy_hh_per_capita
Annual heat demand of households per capita in MWh

dem_th_energy_total_per_capita
Annual heat demand of households, retail, commercial and agricultural sector per capita in MWh

reg_prio_area_wec_area
Area sum of priority areas (parts) in ha

reg_prio_area_wec_count
Count of priority area (parts)

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

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

ags
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gen_el_energy_wind
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>

pop_2011
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pop_2017
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pop_2030
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pop_2050
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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

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

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

class stemp_abw.models.**Powerplant**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Power plants (status quo)

id
DB id

ags
Municipality key (Amtlicher Gemeindeschlüssel), refers to *stemp_abw.models.RegMun*

capacity
Nominal power in MW

chp
Indicates if plant is of type CHP (combined heat and power)

com_month
Month of commissioning

com_year
Year of commissioning

comment
Comment

decom_month
Month of decommissioning

decom_year
Year of decommissioning

efficiency
Efficiency

energy_source_level_1
Indicates if renewable or conventional

energy_source_level_2
Indicates energy source

energy_source_level_3
More specific energy source

geometry
SRID: EPSG:4326 (WGS84)

Type Geometry

technology
Technology

thermal_capacity
Nominal thermal nominal power, if applicable

coastdat2
No. of coastdat2 weather cell (reegis)

capacity_in
Capacity of inflow

federal_state
Abbreviation of federal state name (2 letters according to ISO 3166-2:DE)

Notes

Most of the attributes correspond to the OPSD dataset, some were added by reegis.

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

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

ags
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.

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

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

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

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

coastdat2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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

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

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

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

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

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

energy_source_level_1
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

energy_source_level_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

energy_source_level_3
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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

geometry

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>

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

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

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

class stemp_abw.models.**REPotentialAreas**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Potential areas for renewable plants

id
DB id

area_params
TODO: Define format App settings for usable areas (area panel)

mun_data
TODO: Define format Available potentials (per technology) TO BE SPECIFIED

geom
SRID: EPSG:3035 (ETRS89/LAEA)

Type Geometry

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

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

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

geom

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

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

name = 're_pot_areas'

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

scenario_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_abw.models.**RegBioReserve**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *schutzzone*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

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

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

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

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

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

geom

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

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

name = 'reg_bio_reserve'

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

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

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

class stemp_abw.models.**RegBirdProtArea**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_bird_prot_area'

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

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

class stemp_abw.models.**RegBirdProtAreaB200**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_bird_prot_area_b200'

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

class stemp_abw.models.**RegDeadZoneHard**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_dead_zone_hard'

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

class stemp_abw.models.**RegDeadZoneSoft**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_dead_zone_soft'

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

class stemp_abw.models.**RegFFHProtArea**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_ffh_prot_area'

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

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

class stemp_abw.models.**RegFFHProtAreaB**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_ffh_prot_area_b'

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

class stemp_abw.models.**RegForest**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_forest'

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

class stemp_abw.models.**RegInfrasAviation**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_infras_aviation'

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

class stemp_abw.models.**RegInfrasHvgrid**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_infras_hvgrid'

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

class stemp_abw.models.**RegInfrasRailway**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_infras_railway'

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

class stemp_abw.models.**RegInfrasRoad**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_infras_road'

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

class stemp_abw.models.**RegLandscProtArea**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_landsc_prot_area'

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

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

class stemp_abw.models.**RegLandscProtAreaParts**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

name = 'reg_landsc_prot_area_parts'

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

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

class stemp_abw.models.**RegMun**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

demandts_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.

feedints_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.

gen

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

geom

geom_centroid

mundata

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

name = 'reg_mun'

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

powerplant_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_abw.models.**RegMunDemElEnergy**(*ags*, *geom*, *geom_centroid*, *gen*)

Bases: *stemp_abw.models.RegMun*

exception DoesNotExist

Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned

Bases: stemp_abw.models.MultipleObjectsReturned

dem_el_energy

dem_el_energy_region

name = 'reg_mun_dem_el_energy'

class stemp_abw.models.**RegMunDemElEnergyDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)

Bases: *stemp_abw.models.RegMun*

exception DoesNotExist

Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

dem_el_energy_result_delta

name = 'reg_mun_dem_el_energy_result_delta'

class stemp_abw.models.**RegMunDemElEnergyPerCapita**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunDemElEnergy*, *stemp_abw.models.RegMunPop*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

dem_el_energy_per_capita

dem_el_energy_per_capita_region

name = 'reg_mun_dem_el_energy_per_capita'

class stemp_abw.models.**RegMunDemElEnergyPerCapitaDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

dem_el_energy_per_capita_result_delta

name = 'reg_mun_dem_el_energy_per_capita_result_delta'

class stemp_abw.models.**RegMunDemElEnergyPerCapitaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_dem_el_energy_per_capita_result'

class stemp_abw.models.**RegMunDemElEnergyResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_dem_el_energy_result'

class stemp_abw.models.**RegMunDemThEnergy**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

dem_th_energy

dem_th_energy_region

name = 'reg_mun_dem_th_energy'

class stemp_abw.models.**RegMunDemThEnergyPerCapita**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunDemThEnergy*, *stemp_abw.models.RegMunPopDensity*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

dem_th_energy_per_capita

dem_th_energy_per_capita_region

name = 'reg_mun_dem_th_energy_per_capita'

class stemp_abw.models.**RegMunEnergyReElDemShare**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunGenEnergyRe*, *stemp_abw.models.RegMunDemElEnergy*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

energy_re_el_dem_share

energy_re_el_dem_share_region

name = 'reg_mun_energy_re_el_dem_share'

class stemp_abw.models.**RegMunEnergyReElDemShareDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

energy_re_el_dem_share_result_delta

name = 'reg_mun_energy_re_el_dem_share_result_delta'

class stemp_abw.models.**RegMunEnergyReElDemShareResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_energy_re_el_dem_share_result'

class stemp_abw.models.**RegMunGenCapRe**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_cap_re

gen_cap_re_region

name = 'reg_mun_gen_cap_re'

class stemp_abw.models.**RegMunGenCapReDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_cap_re_result_delta

name = 'reg_mun_gen_cap_re_result_delta'

class stemp_abw.models.**RegMunGenCapReDensity**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunGenCapRe*, *stemp_abw.models.RegMunPopDensity*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

gen_cap_re_density

gen_cap_re_density_region

name = 'reg_mun_gen_cap_re_density'

class stemp_abw.models.**RegMunGenCapReDensityDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_cap_re_density_result_delta

name = 'reg_mun_gen_cap_re_density_result_delta'

class stemp_abw.models.**RegMunGenCapReDensityResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_gen_cap_re_density_result'

class stemp_abw.models.**RegMunGenCapReResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_gen_cap_re_result'

class stemp_abw.models.**RegMunGenCountWindDensity**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunPopDensity*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_count_wind_density

gen_count_wind_density_region

name = 'reg_mun_gen_count_wind_density'

class stemp_abw.models.**RegMunGenCountWindDensityDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_count_wind_density_result_delta

name = 'reg_mun_gen_count_wind_density_result_delta'

class stemp_abw.models.**RegMunGenCountWindDensityResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_gen_count_wind_density_result'

class stemp_abw.models.**RegMunGenEnergyRe**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_energy_re

gen_energy_re_region

name = 'reg_mun_gen_energy_re'

class stemp_abw.models.**RegMunGenEnergyReDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_energy_re_result_delta

name = 'reg_mun_gen_energy_re_result_delta'

class stemp_abw.models.**RegMunGenEnergyReDensity**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunGenEnergyRe*, *stemp_abw.models.RegMunPopDensity*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

gen_energy_re_density

gen_energy_re_density_region

name = 'reg_mun_gen_energy_re_density'

class stemp_abw.models.**RegMunGenEnergyReDensityDeltaResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

gen_energy_re_density_result_delta

name = 'reg_mun_gen_energy_re_density_result_delta'

class stemp_abw.models.**RegMunGenEnergyReDensityResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_gen_energy_re_density_result'

class stemp_abw.models.**RegMunGenEnergyRePerCapita**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunGenEnergyRe*, *stemp_abw.models.RegMunPop*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist, stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned, stemp_abw.models.MultipleObjectsReturned

gen_energy_re_per_capita

gen_energy_re_per_capita_region

name = 'reg_mun_gen_energy_re_per_capita'

class stemp_abw.models.**RegMunGenEnergyReResult**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMun*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_gen_energy_re_result'

class stemp_abw.models.**RegMunPop**(**args*, ***kwargs*)
Bases: *stemp_abw.models.RegMun*

This is a proxy model for RegMun which got same relations to the DB table but changes the model name. This is needed to load the appropriate DetailView when clicking on a map feature (serialized property in the data view). - See Also: https://github.com/rl-institut/WAM_APP_stemp_abw/issues/2 - All other model classes which heir from RegMun work like this.

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = 'reg_mun_pop'

pop

pop_region

class stemp_abw.models.**RegMunPopDensity**(*ags*, *geom*, *geom_centroid*, *gen*)
Bases: *stemp_abw.models.RegMunPop*

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

area_region

name = 'reg_mun_pop_density'

pop_density

pop_density_region

class stemp_abw.models.**RegNatureMonum**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_nature_monum'

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

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

class stemp_abw.models.**RegNaturePark**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_nature_park'

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

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

class stemp_abw.models.**RegNatureProtArea**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *schutzzone*, *erfassungs*, *info_konta*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

geom

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

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

name = 'reg_nature_prot_area'

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

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

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

class stemp_abw.models.**RegPrioAreaAgri**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_agri'

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

class stemp_abw.models.**RegPrioAreaCult**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_cult'

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

class stemp_abw.models.**RegPrioAreaFloodProt**(*id*, *geom*, *bemerkunge*, *bezeich_2*, *bezeich_3*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

bezeich_3
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_flood_prot'

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

class stemp_abw.models.**RegPrioAreaNature**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_nature'

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

class stemp_abw.models.**RegPrioAreaRes**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_res'

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

class stemp_abw.models.**RegPrioAreaWEC**(*id*, *geom*, *bezeich_2*, *bezeich_3*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

bezeich_3
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_wec'

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

class stemp_abw.models.**RegPrioAreaWater**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_prio_area_water'

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

class stemp_abw.models.**RegResidArea**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_resid_area'

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

class stemp_abw.models.**RegResidAreaB1000**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_resid_area_b1000'

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

class stemp_abw.models.**RegResidAreaB500**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_resid_area_b500'

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

class stemp_abw.models.**RegRetentAreaAgri**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_retent_area_agri'

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

class stemp_abw.models.**RegRetentAreaEcosys**(*id*, *geom*, *bezeich_2*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

bezeich_2
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

geom

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

name = 'reg_retent_area_ecosys'

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

class stemp_abw.models.**RegSurfaceWater**(**args*, ***kwargs*)
Bases: *stemp_abw.models.LayerModel*

Surface water

Oberflächengewässer (Fließgewässer 1. Ordnung, stehende Gewässer > 1 ha).

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'reg_surface_water'

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

class stemp_abw.models.**RegWaterProtArea**(*id*, *geom*, *gebietsnam*, *gebietsnum*, *rechtsgrun*, *schutzzone*, *erfassungs*, *amtsblatt*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

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

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

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

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

geom

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

name = 'reg_water_prot_area'

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

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

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

class stemp_abw.models.**RepoweringScenario**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Repowering scenario

TODO: Add doctring

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.

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

desc_en
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.

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

name_en
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>

scenario_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_abw.models.**ResultLayerModel**(**args*, ***kwargs*)
Bases: *stemp_abw.models.RegMun*

This model is a dummy proxy model for displaying layer results

Notes

It bases the municipalities' model *stemp_abw.models.RegMun* which is required (geom, names) for all result layers. The result data column cannot be defined using property decorator as the results are stored in stemp_abw.results.results.Results which is connected to a session and not accessible from models. Instead, the result column is dynamically added in the serial view *stemp_abw.views.serial_views.GeoJSONResultLayerData*.

exception DoesNotExist
Bases: stemp_abw.models.DoesNotExist

exception MultipleObjectsReturned
Bases: stemp_abw.models.MultipleObjectsReturned

name = None

classmethod name_init(*name*)
Class method to set model name property which is needed to match the layer configuration (config/layers_results.cfg) and control (associated layer switch in GUI).

Parameters **name** (str) – Model name as used in config/layers_results.cfg

class stemp_abw.models.**RpAbwBound**(*id*, *geom*)
Bases: *stemp_abw.models.LayerModel*

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

geom

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

name = 'rpabw'

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

class stemp_abw.models.**Scenario**(*energy system configuration*)
Bases: django.db.models.base.Model

id
DB id

created
Timestamp of creation

Type DateTime

name
Name of scenario

Type String

is_user_scenario
True, if scenario was created by a user (default)

Type Bool

data
Reference to ScenarioData

results
Reference to SimulationResults

re_potential_areas
Reference to REPotentialAreas

repowering_scenario
Reference to RepoweringScenario

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

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

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

data
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.

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

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

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

get_previous_by_created(*, *field=<django.db.models.fields.DateTimeField: created>*, *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.

is_user_scenario
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>

re_potential_areas
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.

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

repowering_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.

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

results
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.

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

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

Scenario data

id
DB id

data
TODO: Define format Scenario data, format as defined <HERE>

Type json

data_uuid
UUID for scenario data to quickly compare settings to avoid blowing up postgreSQL

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.

data_uuid
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>

scenario_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_abw.models.**SimulationResults**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Results of a scenario simulation

id
DB id

data
Result data, format as defined <HERE>

Type json

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>

scenario_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.

12.1.9 stemp_abw.queries module

12.1.10 stemp_abw.sessions module

12.1.11 stemp_abw.settings module

12.1.12 stemp_abw.tests module

12.1.13 stemp_abw.urls module

12.1.14 stemp_abw.widgets module

class stemp_abw.widgets.**EsysSwitchWidget**(*attrs=None*)
Bases: `django.forms.widgets.NumberInput`

media

template_name = 'widgets/esys_switch.html'

class stemp_abw.widgets.**LayerSelectWidget**(*attrs=None*, *check_test=None*)
Bases: `django.forms.widgets.CheckboxInput`

media

template_name = 'widgets/layer_switch.html'

class stemp_abw.widgets.**SliderWidget**(*attrs=None*)
Bases: `django.forms.widgets.NumberInput`

media

template_name = 'widgets/slider_abw.html'

12.1.15 Module contents

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Python Module Index

S

Index

C

D

E

F

G

H

N

O

P

R

S

T

W