**Important:** This document resumes the integrations performed to the available Temoa source code. Two kinds of integrations are required: creating new tables within the .sql database; adding pieces of code to the Temoa source code. The document indicates the exact .py file and the exact row where the pieces of code must be copied. Downloading an updated version of the Temoa source code from Github, it is possible that the line number changes slightly from what was indicated in the last update of the document. Be careful to copy the code correctly.

**EMISSION FACTORS**

Create a new table in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "CommodityEmissionFactor" (  "input\_comm" text,  "emis\_comm" text,  "ef" real,  "emis\_unit" text,  "ef\_notes" text,  PRIMARY KEY("input\_comm","ef","emis\_comm"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("emis\_comm") REFERENCES "commodities"("comm\_name")  ); |

**SERVICE DEMANDS PROJECTION**

Create new tables in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "Driver" (  "regions" text,  "periods" integer,  "driver\_name" text,  "driver" real,  "driver\_notes" text,  PRIMARY KEY("regions", "periods", "driver\_name"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods")  );  CREATE TABLE "Allocation" (  "regions" text,  "demand\_comm" text,  "driver\_name" text,  "allocation\_notes" text,  PRIMARY KEY("regions", "demand\_comm", "driver\_name"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("demand\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("driver\_name") REFERENCES "Driver"("driver\_name"),  );  CREATE TABLE "Elasticity" (  "regions" text,  "periods" integer,  "demand\_comm" text,  "elasticity" real,  "elaticity\_notes" text,  PRIMARY KEY("regions", "periods", "demand\_comm"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  FOREIGN KEY("demand\_comm") REFERENCES "commodities"("comm\_name")  ); |

**CAPACITY FACTOR**

Create a new table in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "CapacityFactor" (  "regions" text,  "tech" text,  "vintage" integer,  "cf" real,  "cf\_notes" text,  PRIMARY KEY("regions","tech","vintage"),  FOREIGN KEY("tech") REFERENCES "technologies"("tech"),  FOREIGN KEY("vintage") REFERENCES "time\_periods"("t\_periods")  ); |

Add to **temoa\_config.py** at row 157 the following rows:

|  |
| --- |
| ['param','CapacityFactor', '', '', 3], |

Add to **temoa\_initialize.py** at row 798 the following rows:

|  |
| --- |
| def CapacityFactorIndices(M):  indices = set(  (r, t, v)  for r in M.regions  for t in M.tech\_all  for v in M.vintage\_all  )  return indices |

Add to **temoa\_model.py** at row 141 the following rows:

|  |
| --- |
| M.CapacityFactor\_rtv = Set(dimen=3, initialize= CapacityFactorIndices)  M.CapacityFactor = Param(M.CapacityFactor\_rtv, default=1) |

Add to **temoa\_rules.py** at row 82 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

Add to **temoa\_rules.py** at row 91 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

Add to **temoa\_rules.py** at row 133 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

**MAX ACTIVITY GROUP**

Create new tables in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "MaxGenGroupWeight" (  "regions" text,  "tech" text,  "max\_group\_name" text,  "act\_fraction" REAL,  "notes" text,  PRIMARY KEY("tech","max\_group\_name","regions")  );  CREATE TABLE "MaxGenGroupLimit" (  "periods" integer,  "max\_group\_name" text,  "max\_act\_g" real,  "notes" text,  PRIMARY KEY("periods","max\_group\_name")  ); |

Add to **temoa\_config.py** at row 133 the following rows:

|  |
| --- |
| ['param','MaxGenGroupLimit', '', '', 2],  ['param','MaxGenGroupWeight', '', '', 3], |

Add to **temoa\_model.py** at row 225 the following rows:

|  |
| --- |
| M.MaxGenGroupWeight = Param(M.RegionalIndices, M.tech\_groups, M.groups, default=0)  M.MaxGenGroupLimit = Param(M.time\_optimize, M.groups) |

Add to **temoa\_model.py** at row 454 the following rows:

|  |
| --- |
| M.MaxActivityGroup\_pg = Set(  dimen=2, initialize=lambda M: M.MaxGenGroupLimit.sparse\_iterkeys()  )  M.MaxActivityGroup = Constraint(  M.MaxActivityGroup\_pg, rule=MaxActivityGroup\_Constraint  ) |

Add to **temoa\_rules.py** at row 1744 the following rows:

|  |
| --- |
| def MaxActivityGroup\_Constraint(M, p, g):  r"""  The MaxActivityGroup constraint sets a maximum activity limit for a user-defined  technology group. Each technology within each group is multiplied by a  weighting function, which determines what technology activity share can count  towards the constraint.  .. math::  :label: MaxActivityGroup  \sum\_{S,D,I,T,V,O} \textbf{FO}\_{p, s, d, i, t, v, o} \cdot WEIGHT\_{t|t \not \in T^{a}}  + \sum\_{I,T,V,O} \textbf{FOA}\_{p, i, t, v, o} \cdot WEIGHT\_{t \in T^{a}}  \le MGGL\_{p, g}  \forall \{p, g\} \in \Theta\_{\text{MaxActivityGroup}}  where :math:`g` represents the assigned technology group and :math:`MGGL`  refers to the :code:`MaxGenGroupLimit` parameter.  """  activity\_p = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.MaxGenGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t not in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  activity\_p\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.MaxGenGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_act = value(M.MaxGenGroupLimit[p, g])  expr = activity\_p + activity\_p\_annual <= max\_act  return expr |

**MIN/MAX INPUT/OUTPUT GROUP**

Create new tables in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "MinInputGroupWeight" (  "regions" text,  "tech" text,  "group\_name" text,  "gi\_min\_fraction" real,  "notes" text,  PRIMARY KEY("tech","group\_name","regions")  );  CREATE TABLE "MinInputGroup" (  "regions" text,  "periods" integer,  "input\_comm" text,  "group\_name" text,  "gi\_min" real,  "gi\_min\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","input\_comm","group\_name")  );  CREATE TABLE "MaxInputGroupWeight" (  "regions" text,  "tech" text,  "group\_name" text,  "gi\_max\_fraction" real,  "notes" text,  PRIMARY KEY("tech","group\_name","regions")  );  CREATE TABLE "MaxInputGroup" (  "regions" text,  "periods" integer,  "input\_comm" text,  "group\_name" text,  "gi\_max" real,  "gi\_max\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","input\_comm","group\_name")  );  CREATE TABLE "MaxOutputGroupWeight" (  "regions" text,  "tech" text,  "group\_name" text,  "go\_max\_fraction" real,  "notes" text,  PRIMARY KEY("tech","group\_name","regions")  );  CREATE TABLE "MaxOutputGroup" (  "regions" text,  "periods" integer,  "output\_comm" text,  "group\_name" text,  "go\_max" real,  "go\_max\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("output\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","output\_comm","group\_name")  ); |

Add to **temoa\_config.py** at row 149 the following rows:

|  |
| --- |
| ['param','MinInputGroupWeight', '', '', 3],  ['param','MinInputGroup', '', '', 4],  ['param','MaxInputGroupWeight', '', '', 3],  ['param','MaxInputGroup', '', '', 4],  ['param','MaxOutputGroupWeight', '', '', 3],  ['param','MaxOutputGroup', '', '', 4], |

Add to **temoa\_model.py** at row 166 the following rows:

|  |
| --- |
| M.MinInputGroupWeight = Param(M.regions, M.tech\_groups, M.groups, default=0)  M.MinInputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups)  M.MaxInputGroupWeight = Param(M.regions, M.tech\_groups, M.groups, default=0)  M.MaxInputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups)  M.MaxOutputGroupWeight = Param(M.regions, M.tech\_groups, M.groups, default=0)  M.MaxOutputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups) |

Add to **temoa\_model.py** at row 545 the following rows:

|  |
| --- |
| M.MinInputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MinInputGroup.sparse\_iterkeys()  )  M.MinInputGroupConstraint = Constraint(  M.MinInputGroup\_Constraint\_rpig, rule=MinInputGroup\_Constraint  )  M.MaxInputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MaxInputGroup.sparse\_iterkeys()  )  M.MaxInputGroupConstraint = Constraint(  M.MaxInputGroup\_Constraint\_rpig, rule=MaxInputGroup\_Constraint  )  M.MaxOutputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MaxOutputGroup.sparse\_iterkeys()  )  M.MaxOutputGroupConstraint = Constraint(  M.MaxOutputGroup\_Constraint\_rpig, rule=MaxOutputGroup\_Constraint  ) |

Add to **temoa\_rules.py** at row 2097 the following rows:

|  |
| --- |
| def MinInputGroup\_Constraint(M, r, p, i, g):  r"""  Allows users to specify minimum shares of commodity inputs to a group of technologies.  These shares can vary by model time period.  """  inp = sum(  M.V\_FlowOut[r, p, s, d, i, S\_t, S\_v, S\_o] \* M.MinInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, i, S\_t, S\_v, S\_o] \* M.MinInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  total\_inp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.MinInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.MinInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  min\_inp = value(M.MinInputGroup[r, p, i, g])  expr = (inp + inp\_annual) >= min\_inp \* (total\_inp + total\_inp\_annual)  return expr  def MaxInputGroup\_Constraint(M, r, p, i, g):  r"""  Allows users to specify maximum shares of commodity inputs to a group of technologies.  These shares can vary by model time period.  """  inp = sum(  M.V\_FlowOut[r, p, s, d, i, S\_t, S\_v, S\_o] \* M.MaxInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, i, S\_t, S\_v, S\_o] \* M.MaxInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  total\_inp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.MaxInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.MaxInputGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_inp = value(M.MaxInputGroup[r, p, i, g])  expr = (inp + inp\_annual) <= max\_inp \* (total\_inp + total\_inp\_annual)  return expr  def MaxOutputGroup\_Constraint(M, r, p, o, g):  r"""  Allows users to specify maximum shares of commodity outputs to a group of technologies.  These shares can vary by model time period.  """  outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, o] \* M.MaxOutputGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  for s in M.time\_season  for d in M.time\_of\_day  )  outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, o] \* M.MaxOutputGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  )  total\_outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.MaxOutputGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.MaxOutputGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_outp = value(M.MaxOutputGroup[r, p, o, g])  expr = (outp + outp\_annual) <= max\_outp \* (total\_outp + total\_outp\_annual)  return expr |

**MAX RESOURCE**

This constraint has been modified with respect to the previous version. In particular, the MaxResource\_Constraint definition has been modified in **temoa\_rules.py**, adding the following expression at the end of rows 1874 and 1885.

This is to consider in the computation the different lengths of the time periods.

|  |
| --- |
| \* M.PeriodLength[p] |