```
BeginPackage["BeamSolver`"];
```

```
Unprotect @@ Names["BeamSolver`*"];
ClearAll @@ Names["BeamSolver`*"];
```

```
beamQ;
beamSolve;
beamPutValues;
beamReactions;
beamPlots;
beamMaxValue;
beamInitialParameters;
removePrivateContext;
```

```
Begin["`Private`"];
```

```
Needs["StaticEquilibrium`"]
```

```
Needs["MyPlots`"]
```

```
noNullInsideQ[arg_] := Not@MemberQ[arg, Null]
```

```
loadingQ[arg_Association] :=
 MatchQ[arg["Type"], "Force" | "Moment" | "Distributed"] &&
  noNullInsideQ[arg]
loadingQ[___] := False
```

```
forceQ[arg_?loadingQ] :=
 MatchQ[arg["Type"], "Force"] &&
  ContainsAll[Keys@arg, {"Type", "x", "Value"}]
forceQ[___] := False
```

```
distributedQ[arg_?loadingQ] :=
 ContainsAll[Keys@arg,
  {"Type", "x begin", "x end", "Value"}]
distributedQ[___] := False
```

```
momentQ[arg_?loadingQ] :=
 MatchQ[arg["Type"], "Moment"] &&
  ContainsAll[Keys@arg, {"Type", "x", "Value"}]
momentQ[___] := False
```

```
supportQ[arg_Association] :=
 ContainsAll[Keys@arg, {"Type", "x"}]
supportQ[___] := False
```

```
beamQ[arg_Association] :=
 SubsetQ[Keys@arg, {"Supports", "Loadings", "Length",
    "E", "J"}] && MatchQ[arg["Loadings"],
   {__?loadingQ}] &&
  MatchQ[arg["Supports"], {__?supportQ}]
beamQ[___] := False
```

```
beamPutValues[beam_?beamQ] :=
 With[
  {hθs = {HeavisideTheta[0] → 1,
     HeavisideTheta[0.] → 1}},
  Map[Evaluate, With[{values = beam["Numbers"]},
     Map[# /. values /. hθs &, Most@beam]], Infinity] /.
   hθs]
```

```
pinQ[beam_?beamQ] :=
 #["Type"] & /@ beam["Supports"] === {"Pin", "Pin"}
pinQ[___] := False
```

```
fixedEndQ[beam_?beamQ] :=
 #["Type"] & /@ beam["Supports"] === {"FixedEnd"}
fixedEndQ[___] := False
```

```
resultantOfRaspred[arg_?distributedQ] := <|
  "Type" → "Force",
  "x" → (#["x end"] + #["x begin"]) / 2 &@arg,
  "Value" → #["Value"] (#["x end"] - #["x begin"]) &@
   arg|>
resultantOfRaspred[arg___] := arg
```

```
convertForcesForStaticEqulibrium[
  nagrList : {__?loadingQ}] :=
 Function[arg, {{0, arg[[2]], 0}, {arg[[1]], 0, 0}}] /@
  (Rest /@ Values@Map[resultantOfRaspred,
      Cases[nagrList,
       _?(forceQ[#] || distributedQ[#] &)]])
```

```
convertMomentsForStaticEqulibrium[
  nagrList : {__?loadingQ}] :=
 #[[2]] & /@
  (Rest /@ Values@Cases[nagrList, _?(momentQ[#] &)])
```

```
momentPoints[beam_?beamQ] :=
 {{0, 0, 0}, {beam["Length"], 0, 0}}
```

```
reactionsEquations[beam_?pinQ] :=
 StaticEquilibrium`zmomentEquation[
momentPoints[beam],
(convertForcesForStaticEqulibrium@beam["Loadings"])~
   Join~
   {{{0, reac`yA, 0}, {beam["Supports"][[1]]["x"], 0, 0}},
    {{0, reac`yB, 0}, {beam["Supports"][[2]]["x"], 0, 0}}},
  convertMomentsForStaticEqulibrium@beam["Loadings"]]

reactionsEquations[beam_?fixedEndQ] :=
 With[
  {forces =
    (convertForcesForStaticEqulibrium@
       beam["Loadings"])~Join~
     {{{0, reac`Y, 0}, {beam["Supports"][[1]]["x"],
        0, 0}}}},
StaticEquilibrium`zmomentEquation[
{{beam["Supports"][[1]]["x"], 0, 0}},
forces,
    (convertMomentsForStaticEqulibrium@
       beam["Loadings"])~Join~{reac`M}]~Join~
   StaticEquilibrium`projectionEquation[{"y"},
    forces[[All, 1]]]
]
```

```
reactionsSolver[reactions:{_Symbol, _Symbol}] :=
 Function[eqs, Solve[eqs, reactions][[1]]]
```

```
findReactions[beam_?pinQ] :=
 reactionsSolver[{reac`yA, reac`yB}]@
  reactionsEquations[beam]
findReactions[beam_?fixedEndQ] :=
 reactionsSolver[{reac`Y, reac`M}]@
  reactionsEquations[beam]
```

```
convertReactionsForBeam[beam_?pinQ] :=
 (Thread[f @@ {#["x"] & /@ beam["Supports"],
      Values@findReactions[beam]}]) /.
  f →
   (<|"Type" → "Force", "x" → #1, "Value" → #2,
      "Kind" → "Reaction"|> &)

convertReactionsForBeam[beam_?fixedEndQ] :=
 Module[{Y, M},
{{Y, M} = Values@findReactions[beam]};
{
<|"Type" → "Force", "x" → beam["Supports"][[1]]["x"],
    "Value" → Y, "Kind" → "Reaction"|>,
<|"Type" → "Moment", "x" → beam["Supports"][[1]]["x"],
    "Value" → M, "Kind" → "Reaction"|>
}
]
```

```
findAndInsertReactions[beam_?beamQ] :=
 Insert[beam,
  "Loadings" → Join[beam["Loadings"],
    convertReactionsForBeam[beam]], "Loadings"]
```

```
beamWithReactionsQ[beam_?beamQ] :=
 MemberQ[Flatten[Keys@beam["Loadings"]], "Kind"]
beamWithReactionsQ[___] := False
```

```
beamReactions[beam_?beamWithReactionsQ] :=
 Select[beam["Loadings"], MemberQ[Keys@#, "Kind"] &]
```

```
supportBCs[beam_?pinQ] :=
 y[#] == 0 & /@ (#["x"] & /@ beam["Supports"])
supportBCs[beam_?fixedEndQ] :=
 {y[#] == 0, y'[#] == 0} &@ (beam["Supports"][[1]]["x"])
```

```
endMoment[beamWithReactions_?beamWithReactionsQ,
  "Left"] :=
 With[
  {selected = Select[beamWithReactions["Loadings"],
     (#["Type"] === "Moment" &&
        (#["x"] === 0 || #["x"] === 0.)) &]},
  If[Length@selected > 0, -selected[[1]]["Value"], 0]]

endMoment[beamWithReactions_?beamWithReactionsQ,
  "Right"] := (*0*)
 With[
  {selected = Select[beamWithReactions["Loadings"],
     (#["Type"] === "Moment" &&
        (#["x"] === beamWithReactions["Length"])) &]},
  If[Length@selected > 0, selected[[1]]["Value"], 0]]
```

```
endBCs[beamWithReactions_?beamWithReactionsQ] :=
 {y''[0] == endMoment[beamWithReactions, "Left"],
  y''[beamWithReactions["Length"]] ==
   endMoment[beamWithReactions, "Right"]}
```

```
fullBoundaryConditions[
  beamWithReactions_?beamWithReactionsQ] :=
 Join[endBCs[beamWithReactions],
  supportBCs[beamWithReactions]]
```

```
insertBeforeNumbers[target_?beamQ, key_String, obj_] :=
 Insert[target, key → obj, -2]
```

```
findAndInsertBoundaryConditions[
  beamWithReactions_?beamWithReactionsQ] :=
 insertBeforeNumbers[beamWithReactions,
  "Boundary conditions",
  fullBoundaryConditions@beamWithReactions]
```

```
beamWithBoundaryConditionsQ[beam_?beamQ] :=
 MemberQ[Keys@beam, "Boundary conditions"]
beamWithBoundaryConditionsQ[___] := False
```

```
compensDistributed[arg_?distributedQ] :=
 -arg["Value"] HeavisideTheta[x - arg["x end"]]
```

```
equationTerm[beamWithReactions_?beamWithReactionsQ,
  arg_?forceQ] := arg["Value"] DiracDelta[x - arg["x"]]
equationTerm[beamWithReactions_?beamWithReactionsQ,
  arg_?momentQ] :=
 -arg["Value"] DiracDelta'[x - arg["x"]]
  If[arg["x"] === beamWithReactions["Length"], 0, 1]
equationTerm[beamWithReactions_?beamWithReactionsQ,
  arg_?distributedQ] :=
 arg["Value"] HeavisideTheta[x - arg["x begin"]] +
  compensDistributed[arg]
```

```
createEquation[beamWithReactions_?beamWithReactionsQ] :=
 y''''[x] ==
  Total[equationTerm[beamWithReactions, #] & /@
    beamWithReactions["Loadings"]]
```

```
createAndInsertEquation[
  beamWithReactions_?beamWithReactionsQ] :=
 insertBeforeNumbers[beamWithReactions, "Equation",
  createEquation@beamWithReactions]
```

```
desiredFunctions = {y'''[x], y''[x], y'[x], y[x]};
```

```
beamSolveDE[eq_, boundaryConditions_] :=
 Quiet[DSolveValue[Join[{eq}, boundaryConditions],
    desiredFunctions, x] /.
   {HeavisideTheta[0] → 1, HeavisideTheta[0.] → 1,
    DiracDelta[0.] → 0}, {Reduce::ratnz}]
beamSolveDE[beamWithReactionsEquationsAndBCs_ ?
   (beamWithBoundaryConditionsQ[#] &&
      beamWithReactionsQ[#] &)] :=
 beamSolveDE[beamWithReactionsEquationsAndBCs[
   "Equation"], beamWithReactionsEquationsAndBCs[
   "Boundary conditions"]]
```

```
createBeamFunction[expression_] :=
 Function[{x1}, Evaluate[(expression /. x → x1)]]
```

```
solveDEAndInsertSolution[
  beamWithReactionsEquationsAndBCs_ ?
   (beamWithBoundaryConditionsQ[#] &&
      beamWithReactionsQ[#] &)] :=
 insertBeforeNumbers[beamWithReactionsEquationsAndBCs,
  "Solutions", Map[createBeamFunction,
   Association @@
    Thread[Head /@ desiredFunctions →
      (beamSolveDE@beamWithReactionsEquationsAndBCs /.
        {Q_, M_, ϴ_, y_} :>
         ({Q, M, ϴ / (#["E"] × #["J"]),
            y / (#["E"] × #["J"])} &@
          beamWithReactionsEquationsAndBCs))]]]
```

```
symbolsInExpr[arg_] :=
 Cases[Variables@Level[arg, {-1}], _Symbol]
```

```
valuesQ[arg_List] :=
 MatchQ[arg, {Rule[_Symbol, _?NumberQ] ..}] &&
  noNullInsideQ[arg]
valuesQ[___] := False
```

```
beamSolve[beam_?beamQ] /;
  (ContainsAll[beam["Numbers"][[All, 1]],
     symbolsInExpr[beam]] &&
    valuesQ[beam["Numbers"]]) :=
 (solveDEAndInsertSolution@
    createAndInsertEquation@
     findAndInsertBoundaryConditions@
      findAndInsertReactions@beam) /.
  DiracDelta[0] → 0.(*fixes DiracDelta[0] bug
    when a force is applied in a support*)
```

```
solvedBeamQ[arg_?beamQ] :=
 MemberQ[Keys@arg, "Solutions"]
solvedBeamQ[___] := False
```

```
solvedBeamWithValuesQ[arg_?solvedBeamQ] :=
 Not@MemberQ[Keys@arg, "Numbers"]
solvedBeamWithValuesQ[___] := False
```

```
beamPlots[solvedBeam_?solvedBeamWithValuesQ,
  opts : OptionsPattern[]] :=
 myPlot[solvedBeam["Solutions"][#][x],
    {x, -0.001, 1.001 solvedBeam["Length"]},
    Exclusions → None,
    FrameLabel →
     {"x", # /. {y''' → "y'''", y'' → "y''", y' → "y'",
        y → "y"}}, AxesOrigin → {0, 0}, Filling → Axis,
    opts] & /@ (Head /@ desiredFunctions)
```

```
removePrivateContext[expression_] :=
 ToExpression@StringReplace[ToString[expression],
   "BeamSolver`Private`" → ""]
```

```
beamMaxValue[solvedBeamWithValues_?
    solvedBeamWithValuesQ,
   parameter : "y'''" | "y''" | "y'" | "y",
   interval : {xmin_, xmax_}] /;
  xmin ≥ 0 && xmax ≤ solvedBeamWithValues["Length"] :=
 removePrivateContext@
  MaximalBy[
   Quiet@Through[{NMinimize, NMaximize}[##]] &[
    {solvedBeamWithValues["Solutions"][
       ToExpression["BeamSolver`Private`" <>
         parameter]][x], xmin ≤ x ≤ xmax}, x],
   Abs[#[[1]]] &]

beamMaxValue[solvedBeamWithValues_?
   solvedBeamWithValuesQ,
  parameter : "y'''" | "y''" | "y'" | "y"] :=
 beamMaxValue[solvedBeamWithValues, parameter,
  {0, solvedBeamWithValues["Length"]}]
```

```
beamInitialParameters[
  solvedBeamWithValues_?solvedBeamWithValuesQ] :=
 Thread[{"\!\(\*SubscriptBox[\(ϴ\), \(0\)]\)",
    "\!\(\*SubscriptBox[\(y\), \(0\)]\)"} →
   Through[{solvedBeamWithValues["Solutions"][y'],
     solvedBeamWithValues["Solutions"][y]}[0.]]] /.
  DiracDelta[0.] → 0.
```

```
End[];
Protect @@ Names["BeamSolver`*"];
EndPackage[]
```