```
BeginPackage["QuantumGroups`Utilities`DataPackage`", {"QuantumGroups`",
   "QuantumGroups`Utilities`Debugging`", "QuantumGroups`MatrixPresentations`",
   "QuantumGroups`RepresentationTensors`", "QuantumGroups`Representations`",
   "QuantumGroups`Algebra`", "QuantumGroups`QuantumRoots`"}];
```

```
ValuesAsString; PackageData; MatchingValues; DefiniteValues;
```

```
{PackageMatrixPresentations, PackageDecompositionMaps,
  PackageQuantumRoots, PackageRMatrix, PackagePartialRMatrixPresentation,
  PackageDirectSumDecompositions, PackageHighWeightVectors, PackageSubIrrepWeightBases,
  PackageBraidingMatrices, PackageBRPresentations, PackageBraidingMaps};
```

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

```
DefiniteValues[s_Symbol] := Cases[DownValues[s] ~Join~ SubValues[s],
  rule : (pattern_ /; FreeQ[pattern, Blank] :> _) :> rule]
```

```
MatchingValues[s_Symbol, p_] := Cases[DownValues[s] ~Join~ SubValues[s],
  rule : (pattern_ /; MatchQ[pattern /. HoldPattern -> Hold, Hold[p]] :> _) :> rule]
```

```
ConvertRuleToAssignmentString[a_HoldPattern :> b_] :=
 StringTake[ToString[a, InputForm, CharacterEncoding -> "PrintableASCII"], {13, -2}] <>
  ":=" <> ToString[b, InputForm, CharacterEncoding -> "PrintableASCII"] <> "\n"
```

```
ValuesAsString[s_Symbol, p_] :=
 StringJoin @@ (ConvertRuleToAssignmentString /@ MatchingValues[s, p])
```

```
WriteRule[filename_, a_HoldPattern :> b_] :=
 (WriteString[filename, StringTake[
     ToString[a, InputForm, CharacterEncoding -> "PrintableASCII"], {13, -2}] <> ":="];
  CautiousWriteString[filename, b, InputForm, CharacterEncoding -> "PrintableASCII"];
  WriteString[filename, "\n"];)
```

```
CautiousWriteString[filename_, s_Symbol, options___] :=
 WriteString[filename, ToString[s, options]]
CautiousWriteString[filename_, s_String, options___] := WriteString[filename, s]
CautiousWriteString[filename_, s_Integer, options___] :=
 WriteString[filename, ToString[s, options]]
CautiousWriteString[filename_, s_Real, options___] :=
 WriteString[filename, ToString[s, options]]
CautiousWriteString[filename_, x_Plus, options___] :=
 WriteString[filename, ToString[x, options]]
CautiousWriteString[filename_, x_Times, options___] :=
 WriteString[filename, ToString[x, options]]
CautiousWriteString[filename_, x_TensorProduct, options___] :=
 WriteString[filename, ToString[x, options]]
CautiousWriteString[filename_, x_CircleTimes, options___] :=
 WriteString[filename, ToString[x, options]]
CautiousWriteString[filename_, {}, options___] := WriteString[filename, "{}"]
CautiousWriteString[filename_, {x__}, options___] :=
 (WriteString[filename, "{"];
  (CautiousWriteString[filename, #, options];
     WriteString[filename, ", "]) & /@ Most[{x}];
  CautiousWriteString[filename, Last[{x}], options];
  WriteString[filename, "}"];)
CautiousWriteString[filename_, f_[x___], options___] :=
 (WriteString[filename, ToString[f, options]];
  WriteString[filename, "["];
  (CautiousWriteString[filename, #, options];
     WriteString[filename, ", "]) & /@ Most[{x}];
  CautiousWriteString[filename, Last[{x}], options];
  WriteString[filename, "]"];)
```

```
Options[PackageData] = {"Needs" → {}, "ExtraPackageCode" → "",
    "ExtraPrivateCode" → "", "LoadPreexistingPackage" → True,
    "Message" → "QuantumGroups::loading", "UseGzip" → False, "ByteCountLimit" → ∞};
PackageData[s_Symbol, p_, packagePath : {__String}, opts___] :=
 PackageData[{{s, p}}, QuantumGroupsDataDirectory[], packagePath, opts]
PackageData[s_Symbol, p_, baseDirectory_String, packagePath : {__String}, opts___] :=
 PackageData[{{s, p}}, baseDirectory, packagePath, opts]
PackageData[patterns : {{_Symbol, _} ..}, packagePath : {__String}, opts___] :=
 PackageData[patterns, QuantumGroupsDataDirectory[], packagePath, opts]
PackageData[patterns : {{_Symbol, _} ..}, baseDirectory_String,
  packagePath : {__String}, opts___] := Module[{fullPackagePath, package,
   directory, filename, contentsTop, contentsBottom, needs, extraPackageCode,
   extraPrivateCode, loadPreexistingPackage, message, useGzip, byteCountLimit},
  needs = "Needs" /. {opts} /. Options[PackageData];
  extraPackageCode = "ExtraPackageCode" /. {opts} /. Options[PackageData];
  extraPrivateCode = "ExtraPrivateCode" /. {opts} /. Options[PackageData];
  loadPreexistingPackage = "LoadPreexistingPackage" /. {opts} /. Options[PackageData];
  message = "Message" /. {opts} /. Options[PackageData];
  useGzip = "UseGzip" /. {opts} /. Options[PackageData];
```

```
  byteCountLimit = "ByteCountLimit" /. {opts} /. Options[PackageData];
  SetDirectory[baseDirectory];
  fullPackagePath = {"QuantumGroups", "Data"} ~Join~ packagePath;
  directory = ToFileName[Most[fullPackagePath]];
  If[Length[FileNames[fullPackagePath[[-2]], ToFileName[Drop[fullPackagePath, -2]]]] ==
    0, CreateDirectory[directory]];
  package = StringJoin @@ (Flatten[Transpose[
       {fullPackagePath, Table["`", {Length[fullPackagePath]}]}], 1]);
  filename = ToFileName[Most[fullPackagePath], Last[fullPackagePath] <> ".m"];
  If[loadPreexistingPackage ∧ ! MemberQ[$ContextPath, package],
   If[
    useGzip ∧ Length[FileNames[filename]] == 0 ∧ Length[FileNames[filename <> ".gz"]] ≠ 0,
    SetDirectory[directory];
    Run["gzip -d " <> Last[fullPackagePath] <> ".m.gz"];
    ResetDirectory[]
   ];
   If[Length[FileNames[filename]] ≠ 0, Get[package]]
  ];
  contentsTop = "BeginPackage[\"" <> package <> "\"" <>
    If[MatchQ[needs, {__String}], ", " <> ToString[needs, InputForm], ""] <> "]" <> "\n"
    <> "Message[" <> message <> ",\"" <> package <> "\"]" <> "\n"
    <> extraPackageCode <> "\n"
    <> "Begin[\"`Private`\"]" <> "\n"
    <> extraPrivateCode <> "\n";
  contentsBottom = "End[]\n" <> "EndPackage[]";
  If[Length[FileNames[filename]] ≠ 0, DeleteFile[filename]];
  If[useGzip ∧ Length[FileNames[filename <> ".gz"]] ≠ 0, DeleteFile[filename <> ".gz"]];
  WriteString[filename, contentsTop];
  (Function[{rule}, WriteRule[filename, rule]] /@ Cases[MatchingValues @@ #,
       ((p_ :→ v_) /; ByteCount[v] ≤ byteCountLimit)]) & /@ patterns;
  WriteString[filename, contentsBottom];
  Close[filename];
  If[useGzip, SetDirectory[directory];
   Run["gzip --rsync " <> Last[fullPackagePath] <> ".m"]; ResetDirectory[]];
  ResetDirectory[];]
```

```
PackageqDimensions[ℐ_n_] := PackageData[
  {{QuantumGroups`Representations`Private`fastQDimension, HoldPattern[
     QuantumGroups`Representations`Private`fastQDimension[ℐn][Irrep[ℐn][_]]]},
   {QuantumGroups`Representations`Private`recursiveQDimension, HoldPattern[
     QuantumGroups`Representations`Private`recursiveQDimension[ℐn][Irrep[ℐn][_]]]},
  {ToString[ℐ] <> ToString[n], "qDimensions"},
  "Needs" → {"QuantumGroups`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False
 ]
```

```
PackageMatrixPresentations[Γ_n_] := PackageData[
  MatrixPresentation, MatrixPresentation[Γn][_][Irrep[Γn][_], FundamentalBasis, _],
  {ToString[Γ] <> ToString[n], "MatrixPresentations"},
  "Needs" → {"QuantumGroups`", "QuantumGroups`MatrixPresentations`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Algebra`"},
  "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False
 ]
```

```
PackageDecompositionMaps[Γ_n_] := (PackageData[
   {{DecompositionMap, DecompositionMap[Γn, _, _]},
    {InverseDecompositionMap, HoldPattern[InverseDecompositionMap[Γn, _, _]]}},
   {ToString[Γ] <> ToString[n], "DecompositionMaps"},
   "Needs" → {"QuantumGroups`",
     "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`",
     "QuantumGroups`RepresentationTensors`", "QuantumGroups`MatrixPresentations`"},
   "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False, "ByteCountLimit" → 6 000 000
  ];
  PackageLargeDecompositionMaps[Γn, 6 000 000];
 )
```

```
weightToString[λ : {__Integer}] :=
 StringDrop[StringJoin @@ ((ToString[#] <> "$") & /@ λ), -1]
```

```
PackageLargeDecompositionMaps[ℐ_n_, limit_Integer] :=
 Module[{tensorPowerQ, tensorPowerPattern, largeValues,
   largeTensorPowers, largeTensorPowerData, otherLargeValues},
  (*tensorPowerQ checks if W looks like a tensor power of V*)
  tensorPowerQ[V_][W_] := MatchQ[W, V] || MatchQ[W, U_ ⊗ V /; tensorPowerQ[V][U]];
  tensorPowerPattern = U_ ⊗ (V : (Irrep[ℐ_n][λ_])) /; tensorPowerQ[V][U];
  largeValues = Cases[MatchingValues[DecompositionMap, DecompositionMap[ℐ_n, _, _]],
    ((p_ :> v_) /; (ByteCount[v] > limit)) :> p];
  largeTensorPowers = Cases[largeValues /. HoldPattern → Hold,
     Hold[DecompositionMap[ℐ_n, V_, _]] /;
      MatchQ[V, tensorPowerPattern]] /. Hold → HoldPattern;
  (* oops, watch out in the next line, V means two different things! *)
  largeTensorPowerData =
   largeTensorPowers /. HoldPattern → Hold /. Hold[DecompositionMap[_, V_, _]] :> V /.
    U : (_ ⊗ V : Irrep[_][λ_]) :> {λ, Count[U, Irrep[_][λ], ∞], U};
  otherLargeValues = Complement[largeValues, largeTensorPowers];
  If[Length[otherLargeValues] > 0,
   PackageData[{DecompositionMap, #} & /@ otherLargeValues,
    {ToString[ℐ] <> ToString[n], "DecompositionMaps", "Large"},
    "Needs" → {"QuantumGroups`", "QuantumGroups`Utilities`MatrixWrapper`",
      "QuantumGroups`Representations`", "QuantumGroups`RepresentationTensors`",
      "QuantumGroups`MatrixPresentations`"},
    "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False]];
  If[Length[largeTensorPowers] > 0,
   Function[{d}, PackageData[DecompositionMap, HoldPattern[DecompositionMap[##]] & @@
       {ℐ_n, d[[3]], _}, {ToString[ℐ] <> ToString[n], "DecompositionMaps",
       "w" <> weightToString[d[[1]]], "k" <> ToString[d[[2]]]},
      "Needs" → {"QuantumGroups`", "QuantumGroups`Utilities`MatrixWrapper`",
        "QuantumGroups`Representations`", "QuantumGroups`RepresentationTensors`",
        "QuantumGroups`MatrixPresentations`"},
      "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False]] /@ largeTensorPowerData];
 ]
```

```
PackageQuantumRoots[ℐ_n_] := PackageData[
  ExpandQuantumRoot, ExpandQuantumRoot[ℐ_n][_],
  {ToString[ℐ] <> ToString[n], "QuantumRoots"},
  "Needs" →
   {"QuantumGroups`", "QuantumGroups`QuantumRoots`", "QuantumGroups`Algebra`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageRMatrix[ℐ_n_] := PackageData[
  RMatrix, RMatrix[ℐ_n, _, _, _, _],
  {ToString[ℐ] <> ToString[n], "RMatrix"},
  "Needs" → {"QuantumGroups`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;", "UseGzip" → False
 ]
```

```
PackagePartialRMatrixPresentation[Γ_n_] := PackageData[
  QuantumGroups`RMatrix`Private`PartialRMatrixPresentation,
  QuantumGroups`RMatrix`Private`PartialRMatrixPresentation[Γn, __],
  {"tmp", ToString[Γ] <> ToString[n], "PartialRMatrixPresentation"},
  "Needs" →
   {"QuantumGroups`", "QuantumGroups`MatrixPresentations`", "QuantumGroups`RMatrix`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageDirectSumDecompositions[Γ_n_] := PackageData[
  QuantumGroups`MatrixPresentations`Private`DirectSumDecomposition,
  QuantumGroups`MatrixPresentations`Private`DirectSumDecomposition[Γn][__],
  {"tmp", ToString[Γ] <> ToString[n], "DirectSumDecompositions"},
  "Needs" → {"QuantumGroups`", "QuantumGroups`MatrixPresentations`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageSubIrrepWeightBases[Γ_n_] := PackageData[
  QuantumGroups`MatrixPresentations`Private`SubIrrepWeightBasis,
  QuantumGroups`MatrixPresentations`Private`SubIrrepWeightBasis[Γn][__],
  {"tmp", ToString[Γ] <> ToString[n], "SubIrrepWeightBases"},
  "Needs" → {"QuantumGroups`", "QuantumGroups`MatrixPresentations`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageBraidingMaps[Γ_n_] := PackageData[
  {{BraidingMap, BraidingMap[Γn, _, _]},
   {InverseBraidingMap, InverseBraidingMap[Γn, _, _]}},
  {ToString[Γ] <> ToString[n], "BraidingMaps"},
  "Needs" → {"QuantumGroups`",
    "QuantumGroups`MatrixPresentations`", "QuantumGroups`RepresentationTensors`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageBRPresentations[Γ_n_] := PackageData[
  KnotTheory`BR, KnotTheory`BR[_, _][Γn, __],
  {"tmp", ToString[Γ] <> ToString[n], "BRPresentations"},
  "Needs" → {"QuantumGroups`",
    "QuantumGroups`MatrixPresentations`", "QuantumGroups`RepresentationTensors`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageHighWeightVectors[ℐ_n_] := PackageData[
  HighWeightVectors, HighWeightVectors[ℐn][__],
  {"tmp", ToString[ℐ] <> ToString[n], "HighWeightVectors"},
  "Needs" → {"QuantumGroups`", "QuantumGroups`MatrixPresentations`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
PackageBraidingMatrices[ℐ_n_] := PackageData[
  QuantumGroups`Braiding`Private`BraidingMatrices,
  QuantumGroups`Braiding`Private`BraidingMatrices[ℐn][__],
  {"tmp", ToString[ℐ] <> ToString[n], "BraidingMatrices"},
  "Needs" →
   {"QuantumGroups`", "QuantumGroups`Braiding`", "QuantumGroups`MatrixPresentations`",
    "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Representations`"},
  "ExtraPrivateCode" → "q=Global`q;"
 ]
```

```
Unprotect[Get];
Get[package_String /; StringMatchQ[package, "QuantumGroups`Data`" ~~ __]] :=
 Get[StringDrop[package, StringLength["QuantumGroups`Data`"]]]
Protect[Get];
```

Deprecated:

```
Unprotect[Get];
useGetGzipHack = True;
Get[package_String /;
    (useGetGzipHack ∧ StringMatchQ[package, "QuantumGroups`Data`" ~~ __])] :=
 Module[{packageFragments, directory, filename, gzipExists = False},
  packageFragments = StringSplit[package, "`"];
  SetDirectory[QuantumGroupsDirectory[]];
  directory = ToFileName[Most[packageFragments]];
  filename = Last[packageFragments] <> ".m";
  If[Length[FileNames[filename <> ".gz", {directory}]] ≠ 0,
   gzipExists = True;
   SetDirectory[directory];
   Run["gzip -d " <> filename <> ".gz"];
   ResetDirectory[]
  ];
  useGetGzipHack = False;
  Get[package];
  useGetGzipHack = True;
  If[gzipExists,
   SetDirectory[directory];
   Run["gzip " <> filename];
   ResetDirectory[];
  ]
 ]
Protect[Get];
```

```
End[];
```

```
EndPackage[];
```