ManagingKnotData`

Introduction

This package provides a uniform interface to the Knot Atlas, Livingston's KnotInfo, and the package KnotTheory`. In the future, it may encompass other sources as well.

Implementation

```
(*<pre>*)
```

```
BeginPackage["KnotTheory`KnotAtlas`ManagingKnotData`", {"KnotTheory`", "WikiLink`"}];
```

```
FromWikiString;
```

```
FromKnotInfoString;
```

Usage messages

```
LoadInvariantRules::usage =
  "LoadInvariantRules[pagename] loads definitions for invariants
    from the page pagename (using the current WikiLink` connection).";
```

```
InvariantDefinitionTable::usage =
  "InvariantDefinitionTable[rules] generates an html table
    representing rules, suitable for input via LoadInvariantRules.";
```

```
InvariantNames::usage = "InvariantNames[rules] returns
    a list of the names of the invariants described by rules.";
```

```
RetrieveInvariant::usage =
  "RetrieveInvariant[invariant, knot, source] returns the value
    of the named invariant for the given knot, from the specified
    source. At present, the only sources understood are \"KnotAtlas\",
    \"KnotTheory`\" and \"KnotInfo\". More may come soon!";
```

```
RetrieveInvariants::usage =
  "RetrieveInvariants[invariantList, knotList, source] returns a list
    of triples, each of the form {\"InvariantName\", K, value}, from
    the specified source. At present, the only sources understood are
    \"KnotAtlas\", \"KnotTheory`\" and \"KnotInfo\". More may come soon!";
```

```
StoreInvariants::usage =
  "StoreInvariants[data, target] stores the data, given in the form
    produced by RetrieveInvariants, in the specified target. At
    present, the only target understood is \"KnotAtlas\". Perhaps soon
    they'll be a way to specify a Mathematica .m file as the target.";
```

```
KnotInvariantURL::usage =
  "The function must be overriden in order to use the generic \"url\"
    source. Given two arguments, the name of the invariant and a knot,
    it should return the URL at which the invariant can be found.
    (Post-processing may be done by overriding ParseKnotInvariantFromURL.)";
```

```
ParseKnotInvariantFromURL::usage =
 "This function may be overriden when using the generic \"url\" source.
   Given three arguments, the name of the invariant, a knot, and the text
   of the page returned from the URL specificied by KnotInvariantURL,
   this function should return the invariant as a Mathematica expression."
```

```
TransferUnknownInvariants::usage = "";
```

```
FindDataDiscrepancies::usage =
  "FindDataDiscrepancies[data1, data2] returns a list of conflicts between the two
     lists of data. The conflicts are given in the form {\"InvariantName\",
     K, value1, value2}, where value1 is the value given in data1, and
     value2 is the value given in data2. See also FindMissingData.\n" <>
   "FindDataDiscrepancies[invariantList, knotList, source1, source2] first
     makes two calls to RetrieveInvariants to generate data1 and data2.";
```

```
FindMissingData::usage =
  "FindMissingData[data1, data2] returns a sublist of data1 consisting
     of items for which there is no corresponding value
     in data2. See also FindDataDiscrepancies.\n" <>
   "FindMissingData[invariantList, knotList, source1, source2] first makes
     two calls to RetrieveInvariants to generate data1 and data2.";
```

```
ProcessKnotAtlasUploadQueue::usage =
  "ProcessKnotAtlasUploadQueue[pagename] starts processing the
    queue at pagename on the KnotAtlas. See the Knot Atlas page \"Upload
    Queues\" for further information. Options Repeat->numberOfRepeats
    and Timeout->numberOfSeconds can be used to control how many items
    will be processed, and the maximum amount of time spent on each.";
```

```
CreateDataPackage;
```

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

Reading invariant definitions from a table

```
namePattern = "<!-- Invariant name -->" ~~
    WhitespaceCharacter ... ~~ "<td>" ~~ n : ShortestMatch[__] ~~ "</td>" ⧴ n;
```

```
linePattern = "<!-- " ~~ t : (WordCharacter ..) ~~
     " =" ~~ ShortestMatch[__] ~~ "<td>" ~~ v : ShortestMatch[___] ~~ "</td>" ⧴ (t → v);
```

```
expressionTags = {"ReadWiki", "ReadLivingston", "KnotTheory", "KnotTheorySetter"};
```

```
ConstructInvariantRule[S_String] :=
 Module[{names = StringCases[S, namePattern], saveContext, rule},
  If[Length[names] ≠ 1, Return[$Failed]];
  saveContext = $Context;
  $Context = "Global`";
  rule = (names⟦1⟧ → DeleteCases[StringCases[S, linePattern], _ → ""] /.
      (t_String ? (MemberQ[expressionTags, #] &) → s_String) ⧴ (t → ToExpression[s]));
  $Context = saveContext;
  rule
 ]
```

```
QuantumInvariantRules = {
   (S_String /; StringMatchQ[S, "QuantumInvariant" ~~ __]) :→ Module[{Γ0, λ0, cases},
     cases = StringCases[S, ("QuantumInvariant/" ~~
          Γ : (LetterCharacter) ~~ n : (DigitCharacter ..) ~~ "/" ~~ μ__) :→ {Γ, n, μ}];
     If[Length[cases] == 0, {}, With[{Γ = Subscript[globalToExpression[
             "QuantumGroups`" <> cases[[1, 1]]], ToExpression[cases[[1, 2]]]],
          λ = ToExpression["{" <> cases[[1, 3]] <> "}"]},
        {"WikiPage" → S, "KnotTheorySetter" → Function[{K, p},
           KnotTheory`QuantumKnotInvariants`QuantumKnotInvariant[Γ,
                QuantumGroups`Irrep[Γ][λ]][K] = Function[{q}, p]; &], "KnotTheory" →
          Function[{K}, KnotTheory`QuantumKnotInvariants`QuantumKnotInvariant[
              Γ, QuantumGroups`Irrep[Γ][λ]][K][Global`q]]
        }
       ]
      ]
    ]
  }
```

```
LoadInvariantRules[pagename_String] := AllInvariants =
   (ConstructInvariantRule /@ Drop[StringSplit[WikiGetPageText[pagename], "<tr>"], 2]) ~
    Join ~ QuantumInvariantRules
```

```
LoadInvariantRules["Invariant_Definition_Table"];
```

Saving invariant definitions to a table

```
InvariantTags[rules_] := Union @@ (rules /. (_ → L_List) :→ First /@ L)
```

```
TableHeader[rules_] := "<tr>\n<th>Invariant name</th>\n" <>
   StringJoin @@ ("    <th>" <> # <> "</th>\n" & /@ InvariantTags[rules]) <> "</tr>\n"
```

```
whitespaces[n_] := StringJoin @@ Table[" ", {n}]
```

```
TableRow[rules_, i_] :=
  "<tr>\n<!-- Invariant name --> <td>" <> rules[[i, 1]] <> "</td>\n" <> StringJoin @@
    ("<!-- " <> # <> " =" <> whitespaces[20 - StringLength[#]] <> "--> <td>" <> ToString[
          # /. rules[[i, 2]] /. {# → ""}] <> "</td>\n" & /@ InvariantTags[rules]) <> "</tr>\n"
```

```
InvariantDefinitionTable[rules_] := "{{Invariant Definition Table Warning}}\n" <>
   "<table width=\"100%\">\n" <> TableHeader[rules] <>
   StringJoin @@ Table[TableRow[rules, i], {i, 1, Length[rules]}] <> "</table>"
```

Code for uploading, downloading, and comparing data

```
FromWikiString[S_String] /; StringMatchQ[S, "<math>" ~~ __ ~~ "</math>"] :=
 FixTeXFormExpression[
  ToExpression[StringReplace[S, "<math>" ~~ X__ ~~ "</math>" :→ X], TeXForm]]
```

```
Clear[FixTeXFormExpression]
FixTeXFormExpression[Times[a_, b__][c__]] := Times[a, b, c]
FixTeXFormExpression[x_] := x
```

```
FromWikiString[S_String] /; StringMatchQ[S, "<nowiki>" ~~ __ ~~ "</nowiki>"] :=
 StringReplace[S, "<nowiki>" ~~ X__ ~~ "</nowiki>" :→ X]
```

```
FromWikiString[S_String] /; StringMatchQ[S, "http://" ~~ __] := S
```

```
FromWikiString[S_String] := ToExpression[S]
```

```
FromKnotInfoString["Not Hyperbolic"] := NotHyperbolic
```

```
FromKnotInfoString[S_String ? (StringMatchQ[#, NumberString] &)] := ToExpression[S]
```

```
FromKnotInfoString[S_String] := S
```

```
FromKnotInfoString["infty"] = ∞;
```

```
InvariantNames[L_List] := Cases[L, (S_String → _List) :→ S]
```

```
InvariantRule[I_String] := InvariantRule[I] = Module[{rule}, rule = I /. AllInvariants;
   If[rule === I, Print["I don't recognise the invariant " <> I <> "."];
    Return[$Failed], rule]]
```

```
RetrieveInvariant[I_String, K_, "KnotTheory"] :=
 Module[{rule = InvariantRule[I], KnotTheory}, If[rule == $Failed, Return[$Failed]];
  KnotTheory = "KnotTheory" /. (I /. AllInvariants);
  If[KnotTheory == "KnotTheory", Print[
    "Sorry, I don't know how to calculate the invariant " <> I <> " using KnotTheory`."];
   Return[$Failed]];
  KnotTheory[K]]
```

```
ReadWikiFunction[I_String] :=
 ("ReadWiki" /. (I /. AllInvariants)) /. "ReadWiki" → FromWikiString
```

```
RetrieveInvariant[I_String, K_, "KnotAtlas"] := Module[{WikiPage, WikiResult},
  WikiPage = WikiPageForInvariant[I];
  If[WikiPage == $Failed, Return[$Failed]];
  WikiResult = WikiGetPageText["Data:" <> NameString[K] <> "/" <> WikiPage];
  ReadWikiFunction[I][WikiResult]]
```

```
RetrieveInvariants[Is : {__Rule}, Ks_List, "KnotAtlas"] :=
 RetrieveInvariants[InvariantNames[Is], Ks, "KnotAtlas"]
```

```
RetrieveInvariants[Is : {__String}, Ks_List, "KnotAtlas"] :=
 Module[{wikipages, pagenames, wikiResult, delegateReadWikiFunction},
  wikiPages = WikiPageForInvariant /@ Is;
  If[MemberQ[wikiPages, $Failed], Return[$Failed]];
  pagenames =
   Flatten[Outer["Data:" <> NameString[#2] <> "/" <> #1 &, wikiPages, Ks], 1];
  wikiResult = WikiGetPageTexts[pagenames];
  getResult[I_, K_] := Module[{c, r},
    c = Cases[wikiResult,
      {"Data:" <> NameString[K] <> "/" <> WikiPageForInvariant[I], r_} :> r];
    If[Length[c] == 1, c[[1]], ""]
   ];
  delegateReadWikiFunction[I_, K_] := With[{result = getResult[I, K]},
    If[result == "", Null, ReadWikiFunction[I][result]]
   ];
  Flatten[Outer[{#1, #2, delegateReadWikiFunction[#1, #2]} &, Is, Ks], 1]
 ]
```

```
RetrieveInvariants[pairs_List, "KnotAtlas"] :=
 Module[{wikipages, pagenames, wikiResult, delegateReadWikiFunction},
  pagenames =
   "Data:" <> NameString[#[[2]]] <> "/" <> WikiPageForInvariant[#[[1]]] & /@ pairs;
  wikiResult = WikiGetPageTexts[pagenames];
  getResult[I_, K_] := Module[{c, r},
    c = Cases[wikiResult,
      {"Data:" <> NameString[K] <> "/" <> WikiPageForInvariant[I], r_} :> r];
    If[Length[c] == 1, c[[1]], ""]
   ];
  delegateReadWikiFunction[I_, K_] := With[{result = getResult[I, K]},
    If[result == "", Null, ReadWikiFunction[I][result]]
   ];
  {#[[1]], #[[2]], delegateReadWikiFunction[#[[1]], #[[2]]]} & /@ pairs
 ]
```

```
KnotInfoGroup[Knot[n_Integer, _Integer]] /; (3 ≤ n ≤ 6) := "knots=3-6&"
KnotInfoGroup[Knot[7, _Integer]] := "knots=7&"
KnotInfoGroup[Knot[8, _Integer]] := "knots=8&"
KnotInfoGroup[Knot[9, _Integer]] := "knots=9&"
KnotInfoGroup[Knot[10, _Integer]] := "knots=10&"
KnotInfoGroup[Knot[11, Alternating, _Integer]] := "knots=11a&"
KnotInfoGroup[Knot[11, NonAlternating, _Integer]] := "knots=11n&"
KnotInfoGroup[Knot[12, Alternating, k_Integer]] :=
 "knots=12a" <> ToString[Ceiling[k / 200]]
KnotInfoGroup[Knot[12, NonAlternating, k_Integer]] :=
 "knots=12n" <> ToString[Ceiling[k / 200]]
```

```
TrimWhitespace[S_String] :=
 StringReplace[S, {StartOfString ~~ Whitespace :→ "", Whitespace ~~ EndOfString :→ ""}]
```

```
RetrieveInvariants[{I_String}, Ks_List, "KnotInfo"] :=
 Module[{groupstring, knotinfopage, knotinfotag, datatable},
  groupstring = StringJoin[Union[KnotInfoGroup /@ Ks]];
  knotinfotag = "KnotInfoTag" /. (I /. AllInvariants);
  If[knotinfotage == "KnotInfoTag",
   Print["Sorry, I don't know how to retrieve the invariant " <> I <> " from KnotInfo."];
   Return[$Failed]];
  knotinfopage = Import["http://www.indiana.edu/~knotinfo/results.cgi?" <>
     groupstring <> "name=1&" <> knotinfotag <> "=1&option=ptxt", "Text"];
  datatable = StringCases[knotinfopage,
     "<table" ~~ Except[">"] .. ~~ ">" ~~ Whitespace ~~ "Name," ~~
           ShortestMatch[__] ~~ "<br>" ~~ dt : ShortestMatch[__] ~~ "</table" :→ dt][[1]];
  StringCases[datatable, "& " ~~ knotname : ShortestMatch[__] ~~
      " & " ~~ value : ShortestMatch[__] ~~ "<br>" :→
    {I, Knot[knotname], FromKnotInfoString[TrimWhitespace[value]]}]
 ]
```

```
RetrieveInvariants[Is : {__String}, Ks_List, "KnotInfo"] /; Length[Is] > 1 :=
 DeleteCases[Join @@ (RetrieveInvariants[{#}, Ks, "KnotInfo"] & /@ Is), $Failed]
```

```
RetrieveInvariants[Is : {__Rule}, Ks_List, source_String] :=
 RetrieveInvariants[InvariantNames[Is], Ks, source]
```

```
RetrieveInvariants[Is : {__String}, Ks_List, source_] :=
 RetrieveInvariants[Flatten[Outer[List, Is, Ks], 1], source]
```

```
RetrieveInvariants[pairs : {{_String, _} ...}, source_String] :=
 {#[[1]], #[[2]], RetrieveInvariant[#[[1]], #[[2]], source]} & /@ pairs
```

```
Clear[WikiPageForInvariant];
WikiPageForInvariant[I_String] :=
 WikiPageForInvariant[I] = Module[{rule = InvariantRule[I], wikiPage},
   If[rule == $Failed, Return[$Failed]];
   wikiPage = "WikiPage" /. rule;
   If[wikiPage === "WikiPage", Print[
     "Sorry, I don't know how to store the invariant " <> I <> " in the Knot Atlas."];
    Return[$Failed]];
   wikiPage
  ]
```

```
General::spell1 : Possible spelling error: new symbol name "rule" is similar to existing symbol "Rule".
   More…
```

```
Options[StoreInvariants] = {Write → True};
```

```
StoreInvariants[Dall : {{_String, _, _} ...}, "KnotAtlas", opts___] :=
 Module[{D, invariants, unknownInvariants, wikiPages, uploadPairs},
  D = DeleteCases[Dall, {_, _, $Failed}];
  invariants = Union[Part[D, All, 1]];
  wikiPages = WikiPageForInvariant /@ invariants;
  If[MemberQ[wikiPages, $Failed], Return[$Failed]];
  uploadPairs = {"Data:" <> NameString[#[[2]]] <> "/" <> WikiPageForInvariant[#[[1]]],
      ToString[#[[3]], WikiForm]} & /@ D;
  If[! FreeQ[uploadPairs, $Failed], Print["Warning: tried to upload bad data -- "];
   Print[uploadPairs];
   Return[$Failed]];
  If[Write /. {opts} /. Options[StoreInvariants],
   WikiSetPageTexts[uploadPairs], uploadPairs]]
```

```
StoreInvariants[Dall : {{_String, _, _} ...}, "CSVString"] :=
 StringJoin @@ ("\"" <> #[[1]] <> "\"" <> ",\t" <> "\"" <> NameString[#[[2]]] <>
      "\"" <> ",\t\"" <> ToString[#[[3]], InputForm] <> "\"\n" & /@ Dall)
```

```
KnotTheorySetterForInvariant[I_String] :=
 KnotTheorySetterForInvariant[I] = Module[{rule = InvariantRule[I], setter},
   If[rule == $Failed, Return[$Failed]];
   setter = "KnotTheorySetter" /. rule;
   If[setter === "KnotTheorySetter",
    Print["Sorry, I don't know how to store the invariant " <>
      I <> " in the current KnotTheory`."];
    Return[$Failed]];
   setter
  ]
```

```
StoreInvariants[Dall : {{_String, _, _} ...}, "KnotTheory"] :=
 Module[{D},
  D = DeleteCases[Dall, {_, _, $Failed | Null}];
  invariants = Union[Part[D, All, 1]];
  setterFunctions = KnotTheorySetterForInvariant /@ invariants;
  If[MemberQ[setterFunctions, $Failed], Return[$Failed]];
  KnotTheorySetterForInvariant[#[[1]]][#[[2]], #[[3]]] & /@ D;
 ]
```

```
StoreInvariants[Dall : {{_String, _, _} ...}, "KnotTheoryInputString"] :=
 Module[{D},
  D = DeleteCases[Dall, {_, _, $Failed | Null}];
  invariants = Union[Part[D, All, 1]];
  setterFunctions = KnotTheorySetterForInvariant /@ invariants;
  If[MemberQ[setterFunctions, $Failed], Return[$Failed]];
  "#[[1]][#[[2]],#[[3]]]&/@ {\n" <> StringJoin @@ ((ToString[#, InputForm] <> "\n") & /@
      ({KnotTheorySetterForInvariant[#[[1]]], #[[2]], #[[3]]} & /@ D)) <> "}"
 ]
```

```
ParseKnotInvariantFromURL[I_, K_, data_] := data
```

```
RetrieveInvariant[I_String, K_, "url"] := Module[{url = KnotInvariantURL[I, K], data},
  If[url == "",
   Print["Sorry, I don't know where to find the value of the invariant " <>
     I <> " online. Trying defining more values for KnotInvariantURL."];
   Return[$Failed]];
  Off[FetchURL::conopen];
  data = Import[url, "Text"];
  If[data == $Failed, Return[$Failed]];
  Return[ParseKnotInvariantFromURL[I, K, data]];
 ]
```

```
take[l_, n_] := If[Length[l] > n, Take[l, n], l]
shuffle[l_] := l[[Ordering[Table[Random[], {Length[l]}]]]]
randomisedpartition[l_, n_] := shuffle[Partition[l, n, n, {1, 1}, {}]]
TransferUnknownInvariants[invariants : {___String}, knots_List,
  source : "KnotTheory", target_String] := Module[{needed, workingset, chunksize = 1,
   counter = 0, timer = 0. Second, interval = 300. Second, failures = {}},
  If[Length[knots] > 5000, Print["Large knot set, dividing into ",
    Ceiling[Length[knots] / 5000], " groups"];
   Return[Union[TransferUnknownInvariants[invariants, #, source, target] & /@
      randomisedpartition[knots, 5000]]]];
  Print["Checking to see what ", target, " already contains..."];
  Print["(took ",
   AbsoluteTiming[needed = Cases[RetrieveInvariants[invariants, knots, target],
       {i_, k_, Null} :> {i, k}]][[1]], ")"];
  Print["Starting to calculate ", Length[needed], " invariants..."];
  While[Length[needed] > 0,
   While[Length[needed] > 0 ∧ (timer < interval /. Second → 1),
    workingset = take[needed, chunksize];
    counter += Length[workingset];
    timer += AbsoluteTiming[failures = failures ~ Join ~
          StoreInvariants[RetrieveInvariants[workingset, source], target];][[1]];
    needed = Complement[needed, workingset];
   ];
   Print["Uploaded ", counter, " invariants in ", timer];
   If[2 chunksize ≤ counter, ++chunksize];
   counter = 0;
   timer = 0 Second;
  ];
  failures
 ]
```

```
FindDataDiscrepancies[Is : {__Rule}, Ks_List, source1_String, source2_String] :=
 FindDataDiscrepancies[InvariantNames[Is], Ks, source1, source2]
```

```
FindDataDiscrepancies[Is : {__String}, Ks_List, source1_String, source2_String] :=
 FindDataDiscrepancies[RetrieveInvariants[Is, Ks, source1],
  RetrieveInvariants[Is, Ks, source2]]
```

```
FindDataDiscrepancies[D1 : {{_String, _, _} ...}, D2 : {{_String, _, _} ...}] :=
 Module[{D1t, D2t, D, P, C},
  (*Mark the data,according to where it came from.*)
D1t = {#⟦1⟧, #⟦2⟧, 1, #⟦3⟧} & /@ D1;
  D2t = {#⟦1⟧, #⟦2⟧, 2, #⟦3⟧} & /@ D2;
  (*Combine the data,and split it into doublets
   (or singlets) corresponding to the same invariant and knot.*)
  D = Split[Sort[D1t ~ Join ~ D2t], SameQ[Take[#1, 2], Take[#2, 2]] &];
  (*Take only the pairs.*)P = Select[D, Length[#] == 2 &];
  (*Combine the pairs*)C = P /. {{I_, K_, 1, V1_}, {I_, K_, 2, V2_}} ⧴ {I, K, V1, V2};
  Select[C, #⟦3⟧ =!= #⟦4⟧ &]]
```

```
FindMissingData[D1 : {{_String, _, _} ...}, D2 : {{_String, _, _} ...}] :=
 Complement[D1, D2, SameTest → SameQ[Take[#1, 2], Take[#2, 2]] &]
```

```
Options[ProcessKnotAtlasUploadQueue] = {Timeout → 42 300, Repeats → ∞};
```

```
ProcessKnotAtlasUploadQueue[pagename_String, opts___Rule] :=
 Module[{n = 0, repeats = Repeats /. {opts} /. Options[ProcessKnotAtlasUploadQueue],
   timeout = Timeout /. {opts} /. Options[ProcessKnotAtlasUploadQueue]},
  While[(++n < repeats) ∧ (TimeConstrained[ProcessKnotAtlasUploadQueue[
        pagename, WikiGetPageText[pagename]], timeout] =!= Null)]
 ]
```

```
randomEntry[list_] := list⟦Random[Integer, {1, Length[list]}]⟧
```

```
randomEntry[list_ /; Length[list] == 0] := Null
```

```
ProcessKnotAtlasUploadQueue[pagename_String, contents_String] := Module[{item, result},
  result = ProcessKnotAtlasUploadQueueEntry[pagename,
    item = randomEntry[StringSplit[contents, StringExpression[EndOfLine]]]];
  If[result == $Failed,
   WikiStringReplace[pagename, item ~~ EndOfLine → ""];
   WikiSetPageText["Upload Queues Rejected Items",
    WikiGetPageText["Upload Queues Rejected Items"] <> "\n" <> item]
  ];
  result
 ]
```

```
ProcessKnotAtlasUploadQueueEntry[_, Null] := Null
```

```
globalToExpression[S_String] := Module[{saveContext, result},
  saveContext = $Context;
  $Context = "Global`";
  result = ToExpression[S];
  $Context = saveContext;
  result
 ]
```

```
ProcessKnotAtlasUploadQueueEntry[pagename_String, item_String] :=
 Module[{cases},
  cases = StringCases[item, "*\"" ~~ invariant : ShortestMatch[__] ~~
       "\", \"" ~~ knotset : ShortestMatch[__] ~~ "\"" :> {invariant, knotset}];
  If[Length[cases] == 0, Return[$Failed]];
  ProcessKnotAtlasUploadQueueEntry[pagename, item, #[[1]], #[[2]]] & /@ cases
 ]
```

```
commaSpaces = "," ~~ " " ...;
```

```
validKnotSetStringPatterns = Alternatives @@ {
    "All" ~~ ("Knots" | "Links") ~~ "[" ~~ DigitCharacter .. ~~ "]",
    "All" ~~ ("Knots" | "Links") ~~
      "[" ~~ DigitCharacter .. ~~ commaSpaces ~~ "Alternating" | "NonAlternating" ~~ "]",
    "All" ~~ ("Knots" | "Links") ~~ "[{" ~~ DigitCharacter .. ~~
        commaSpaces ~~ DigitCharacter .. ~~ "}]",
    "All" ~~ ("Knots" | "Links") ~~ "[{" ~~ DigitCharacter .. ~~
        commaSpaces ~~ DigitCharacter .. ~~
          "}" ~~ commaSpaces ~~ "Alternating" | "NonAlternating" ~~ "]",
    "TorusKnots[" ~~ DigitCharacter .. ~~ "]",
    "Select[" ~~ (s1__ /; knotsetStringSanityCheck[s1]) ~~
      commaSpaces ~~ "First[BR[#]]" ~~ ("<" | "=") ~~ "=" ~~ DigitCharacter .. ~~ "&]",
    "Take[" ~~ (s2__ /; knotsetStringSanityCheck[s2]) ~~
      commaSpaces ~~ DigitCharacter .. ~~ "]",
    "Take[" ~~ (s3__ /; knotsetStringSanityCheck[s3]) ~~
      commaSpaces ~~ "{" ~~ ("-" | "") ~~
         DigitCharacter .. ~~ commaSpaces ~~ ("-" | "") ~~ DigitCharacter .. ~~ "}" ~~ "]"
   };
```

```
knotsetStringSanityCheck[knotset_String] :=
 StringMatchQ[knotset, validKnotSetStringPatterns]
```

```
ProcessKnotAtlasUploadQueueEntry[pagename_String,
  item_String, invariant_String, knotset_String] := Module[{result},
  If[! knotsetStringSanityCheck[knotset], Print["The knot set string ",
    knotset, " doesn't pass the sanity test, so I won't try to interpret it."];
   Return[$Failed]];
  Print["Calculating ", invariant, " for everything in ", knotset];
  result = TransferUnknownInvariants[{invariant},
    globalToExpression[knotset], "KnotTheory", "KnotAtlas"];
  If[result == {}, WikiStringReplace[pagename, item ~~ EndOfLine → ""];
   WikiSetPageText["Upload Queues Completed Work",
    WikiGetPageText["Upload Queues Completed Work"] <> "\n" <> item]];
  item
 ]
```

```
CreateDataPackage[datasetname_String, invariant_String, knotset_List] :=
 CreateDataPackage[datasetname, {invariant}, knotset]
```

Note to self; this really needs an extra argument, for other needed packages.

```
CreateDataPackage[datasetname_String, invariants : {__String}, knotset_List] :=
 Module[{filename},
  filename = KnotTheoryDirectory[] <> "/" <> datasetname <> ".m";
  If[FileNames[datasetname <> ".m", {KnotTheoryDirectory[]}] =!= {},
   Print[
    "Warning! There's already a file called " <> filename <> "\nPlease double check the
       name, and delete the pre-existing file if appropriate."];
   Return[$Failed]];
  WriteString[filename,
   "BeginPackage[\"KnotTheory`" <> datasetname <> "`\",{\"KnotTheory`\"}]\n" <>
    "Message[KnotTheory::loading, \"" <> datasetname <> "`\"]\n" <> StoreInvariants[
     RetrieveInvariants[invariants, knotset, "KnotAtlas"], "KnotTheoryInputString"] <>
    "\nEndPackage[]"
  ];
  Close[filename]
 ]
```

```
End[];
```

```
EndPackage[];
```

```
(*</pre>[[Category:Source Code]]*)
```

```
$Aborted
```