GETTING STARTED WITH P-COLLECTIONS

A brief tutorial.

Abstract

This tutorial gives you an impression how to create, manage and use p-collections, which are portable artifacts representing (usually) large sets of XML documents. P-collections associate their member nodes with external properties which are stored outside the nodes and may be queried by XML and non-XML technologies alike. The tutorial shows you how to create, update and filter p-collections.

Hans-Jürgen Rennau, hrennau@yahoo.de

Getting started with p-collections

vsn 2015-02-08-a

This tutorial shows you how to take advantage of **p-collections** when developing **topic tools**. Topic tools are command-line tools based on the `ttools` framework. P-collections are artifacts representing a collection of XML documents associated with additional, "external" properties which can be used for filtering the collection. For a general introduction into the concept of p-collections see the presentation "Node search preceding node construction – XQuery inviting non-XML technologies", or the article with the same title, both found in the `doc` folder of `https://github.com/hrennau/TopicTools`.

General prerequisites

You must have installed the BaseX XQuery processor. You can download it from here:

http://basex.org/products/download/all-downloads/

In general, BaseX is only required by the administrative operations of the `ttools` framework. The topic tools themselves, in the development of which `ttools` assists you, are ordinary XQuery applications which can be run by any standard-conforming XQuery processor.

Additional prerequisites when using p-collections

The use and maintenance of p-collections, however, may require certain non-standard features of the XQuery processor. Such requirements depend on the actual operation (e.g. node insertion versus other operations) and the technology used to implement the collection. Currently, two technologies are supported:

- XML – p-collection is represented by an XML document
- SQL – p-collection is represented by a set of relational tables

The constraints are summarized as follows:

Processor	**NCAT model**	**Operations**
BaseX	XML, SQL	all
SaxonPE	XML	all
SaxonEE	XML	all
SaxonHE	XML	filtering; copying (XML to XML)
other	XML	filtering; copying (XML to XML)

In general, the operation of node insertion requires dynamic evaluation of XQuery expressions, and the use of SQL-based p-collections requires SQL access. For any XQuery processor not explicitly listed in the table above, which offers (a) a command-line interface, (b) extension functions for dynamic evaluation and/or SQL access, the support for handling p-collections will be extended when users signal interest. Support must indeed be added on a per-processor basis as there is neither an official

nor an unofficial standard (like EXPath) prescribing the signatures of functions offering dynamic evaluation or SQL access.

Note

When you are aware of an XQuery processor (apart from BaseX and Saxon) offering a command-line interface and extension functions for dynamic evaluation and/or SQL access, you are encouraged to contact me and request extended support for this processor, including extended support for dealing with p-collections.

Support for SQL-based p-collections is currently further restricted to the RDBMS MySQL, version 5.6 or higher. This means that in order to use SQL-based p-collections you must

- use the BaseX processor
- have installed the MySQL database
- have installed the official Java connector for MySQL

The MySQL connector is

mysql-connector-java-VERSION.bin.jar

version 5.134 or greater, which can be downloaded from here:

http://dev.mysql.com/downloads/connector/j/

The jar file must be placed in the `bin` directory of the BaseX installation.

When you want to follow this tutorial actively and repeat the example invocations, it is recommended to use the BaseX processor. Use of BaseX is also assumed by the text.

Note

This tutorial cannot replace the study of the “gettingStartingWithTopicTools” tutorial, which gives a general introduction to working with `ttools`. However, knowledge of the other tutorial is not assumed by the present one.

Scenario: topic tool `pcollect`

The `ttools` framework offers two APIs for dealing with p-collections: the node search API and the collection management API. These APIs are "inherited" by every topic tool based on `ttools`. Therefore `ttools` does not ship with a special application for dealing with p-collections. But this implies that working with p-collections requires *some* topic tool, which need not have any particular relationship to p-collections.

We create a new topic tool named `pcollect`, regarding it as a container which can gradually be filled with operations useful in the management of p-collections. As a start, we shall add a single operation for reporting the contents of NODLs found in the file system (NODLs are documents describing p-collections).

Step #1 – create the application

The first step of developing a topic tool is its initialization. You specify the tool directory, and (optionally) you also supply the name of a first module and the name(s) of first operations. Suppose the application directory is `/tta/pcollect`, the first module is `nodl.mod.xq` and the first operation will be `nodls`. You initialize your application by the following call:

```
basex -b "request=new?dir=/tta/pcollect, mod=nodl,ops=nodls"
    /tt/bin/ttools.xq
```

`ttools`:

```
================================================================

Topic tool created:  pcollect
Tool directory:      /tta/pcollect

The tool can already be called. Example:
   basex -b "request=?" /tta/pcollect/pcollect.xq

Use operation 'add' for adding module prototypes. Example:
   basex -b "request=add?dir=/tta/pcollect, mod=fooMod, ops=fooOp barOp foobarOp"
/tt/bin/ttools.xq
================================================================
```

Note

If you run the examples with the SaxonPE or SaxonEE processor, you need to supply the `flavor` parameter of the `new` operation. The value is the string "saxonpe" or "saxonee", followed by the version number "95" or "96". Example:

```
basex -b "request=new?dir=/tta/pcollect,mod=nodl,ops=nodls,
   flavor=saxonpe96" /tt/bin/ttools.xq
```

Step #2 – survey the available operations

What is our plan? We will first create, feed and filter an XML-based p-collection. Then we will create, feed and filter an SQL-based collection, which contains the same XML documents and associates them with external properties according to the same rules. This exercise will show us that the technology on which p-collections are based is completely hidden from the XQuery code – the XQuery code is exactly the same in both cases.

As a starting point we survey the available operations. Our new tool provides already various operations:

- built-in operations "inherited" from the `ttools` framework
- the operation `nodls` which is already in place, although bound to a dummy implementation generated by `ttools`

To see the available operations, we call the `_help` operation:

Call:

```
basex -b "request=?" /tta/pcollect/pcollect.xq
```

`pcollect`:

```
TOOL: pcollect

OPERATIONS    PARAMS
=========================================================================
_copyNcat     nodl?, query?, toNodl
_createNcat   nodl
_dcat         dfd+
_deleteNcat   nodl
_docs         dcat*, doc*, docs*, fdocs*
_doctypes     attNames, dcat*, doc*, docs*, elemNames, fdocs*, sortBy?
_feedNcat     dirs*, nodl
_help         default, mode, ops?, type
_nodlSample   model?
_search       nodl, query?
nodls         dcat*, doc*, docs*, fdocs*
=========================================================================
```

We will use the following built-in operations:

- `_nodlSample` – provides us with the sample of a NODL document which we will edit in order to define a p-collection
- `_createNcat` – instantiates a p-collection
- `_feedNcat` – feeds XML documents into the collection, associating them with external properties
- `_doctype` – reports the document types of a set of documents; can be applied to a filtered p-collection

Finally, we will add a first, primitive implementation of `nodls`, creating a mere start to be elaborated, though not within this short tutorial.

Step #3 – procure a sample NODL

We request a sample NODL for XML-based p-collections.

Call:

```
basex -b request=_nodlSample?model=xml /tta/pcollect/pcollect.xq >
   /nodls/xmls.nodl
```

`pcollect`:

```
<nodl xmlns="http://www.infospace.org/pcollection">
  <collection name="COLLECTION_NAME" uri="" formats="xml" doc="A collection of
FOOs."/>
  <pmodel>
    <property name="foo" type="xs:string*" maxLength="100" expr="//foo"/>
    <property name="bar" type="xs:string" maxLength="100" expr="//bar"/>
    <property name="foobar" type="xs:string*" maxLength="100" expr="//foobar"/>
    <anyProperty/>
  </pmodel>
  <nodeDescriptor kind="SELECT_ONE: uri|text"/>
  <ncatModel>
    <xmlNcat documentURI="DOC_URI" asElement="*foo* *bar*"/>
  </ncatModel>
</nodl>
```

Step #4 – edit the NODL

We want to define a p-collection which associates its members with simple, generic information about their content. The following table defines the external properties involved.

Property name	Semantics	Type	XQuery expression
name	local name of the root element	xs:string	local-name(/*)
namespace	namespace of the root element	xs:string	namespace-uri(/*)
namespaces	all namespaces used in the document	xs:string*	distinct-values(//*/namespace-uri(.))
enames	all local element names	xs:string*	distinct-values(//*/local-name(.))
anames	all local attribute names	xs:string*	distinct-values(//@*/local-name(.))
ecount	number of element nodes	xs:integer	count(//*)
acount	number of attribute nodes	xs:integer	count(//@*)

The collection name will be `xmls`, and the NCAT – the XML document implementing the p-collection - will have the URI:

```
/ncats/ncat-xmls.xml
```

Now we edit the NODL accordingly:

```
<nodl xmlns="http://www.infospace.org/pcollection">
  <collection name="xmls" uri="" formats="xml"
              doc="A collection of arbitrary XML documents."/>
  <pmodel>
    <property name="name" type="xs:string" maxLength="100"
                          expr="local-name(/*)"/>
    <property name="namespace" type="xs:string" maxLength="100"
                          expr="namespace-uri(/*)"/>
    <property name="namespaces" type="xs:string*" maxLength="100"
                          expr="distinct-values(//*/namespace-uri(.))"/>
    <property name="enames" type="xs:string+" maxLength="100"
                          expr="distinct-values(//*/local-name(.))"/>
    <property name="anames" type="xs:string*" maxLength="100"
                          expr="distinct-values(//@*/local-name(.))"/>
    <property name="ecount" type="xs:integer" maxLength="100"
                          expr="count(//*)"/>
    <property name="acount" type="xs:integer" maxLength="100"
                          expr="count(//@*)"/>
  </pmodel>
  <nodeDescriptor kind="uri"/>
  <ncatModel>
    <xmlNcat documentURI="/ncats/ncat-xmls.xml" asElement="*"/>
  </ncatModel>
</nodl>
```

Step #5 – create the p-collection

We create the p-collection `xmls`.

Call:

```
basex -b request=_createNcat?nodl=/nodls/xmls.nodl
   /tta/pcollect/pcollect.xq
```

`pcollect`:

```
<z:createNcat xmlns:z="http://www.ttools.org/structure"
              msg="XML NCAT written"
              documentURI="file:/ncats/ncat-xmls.xml"/>
```

Step #6 – populate the p-collection

We feed the p-collection all documents with a file name matching "*.xml" and found in or under a particular directory, e.g. `/projects`.

Call:

```
basex -b "request=_feedNcat?nodl=/nodls/xmls.nodl,docs=|/projects/*.xml"
   /tta/pcollect/pcollect.xq
```

`pcollect`:

```
<feedNcat name="xmls" countNewPnodes="1486" oldNcatSize="0"
newNcatSize="1486"/>
```

The response informs that the p-collection now contains 1486 p-nodes. The following listing shows an example p-node.

```
<pnode
     node_uri="file:/projects/nicole-dev/exc/merge/data/
     pixell_ugc_Prod_request_job_100001__2014-12-30__2015-01-13T19-15-05.135__7.xml">
  <name>BookingExportConfiguration</name>
  <namespace>http://infospace.de/middleware/BookingExportConfiguration</namespace>
  <namespaces>
    <item>http://infospace.de/middleware/BookingExportConfiguration</item>
    <item/>
  </namespaces>
  <enames>
    <item>BookingExportConfiguration</item>
    <item>JobList</item>
    <item>Job</item>
    <item>ArrivalDateSpan</item>
    <item>OrganisationUnitList</item>
    <item>OrganisationUnit</item>
    <item>ExpectedFilename</item>
  </enames>
  <anames>
    <item>MinInclusiveDate</item>
    <item>MaxInclusiveDate</item>
    <item>AgencyId</item>
  </anames>
  <ecount>7</ecount>
  <acount>3</acount>
</pnode>
```

Step #7 – query the p-collection

The built-in operation `_doctype` reports the "document type" of the supplied documents, defined as the concatenated name and namespace URI of the root element. The input can be provided by various parameters, each with a different type. In order to learn the details, we resort again to the `_help` operation, using parameters `type` and `default` which request information about the data type and a default value, and parameter `ops` which restricts the output to operations with matching names.

Call:

```
basex -b "request=?type,default,ops=_doct*" /tta/pcollect/pcollect.xq
```

pcollect:

```
TOOL: pcollect

OPERATIONS   PARAMS
================================================================================
_doctypes    attNames=false..... : xs:boolean
             dcat............... : docCAT* (sep=WS)
             doc................ : docURI* (sep=WS)
             docs............... : docDFD* (sep=SC)
             elemNames=false..... : xs:boolean
             fdocs.............. : docSEARCH* (sep=SC)
             sortBy=name........ : xs:string?; facets: values=name,namespace
               At least 1 of these parameters must be set: dcat, doc, docs, fdocs
================================================================================
```

Input parameter `fdocs` is typed `docSearch`, which means a filtered collection. We call `_doctype`, supplying it with our p-collection filtered by a query. The syntax of a filtered p-collection is the collection URI, followed by a ? character, followed by the filter text.

Call:

```
basex -b "request=_doctype?fdocs=/nodls/xmls.nodl?
enames~*SearchEngine* && not(name=msgType || namespace~*items*)"
   /tta/pcollect/pcollect.xq
```

pcollect:

```
<z:doctypes xmlns:z="http://www.ttools.org/structure" countDocs="3" countDoctypes="3">
  <z:doctype name="msgs@http://www.ttools.org/logspy/ns/xquery-functions" count="1">
    <doc href="file:///C:/projects/data/msgs-serl/msgs-serl-t01.xml"/>
  </z:doctype>
  <z:doctype name="SearchEngineOfferListRS@" count="1">
    <doc href="file:///C:/projects/xq/workshop-2013-dev/data/regionList.xml"/>
  </z:doctype>
  <z:doctype
name="SearchEngineOfferListRS@http://traveltainment.de/middleware/xml/SearchEngineOfferListRS"
count="1">
    <doc href="file:///C:/projects/xq/workshop-2013-dev/examples/someTips/regionList.xml"/>
  </z:doctype>
</z:doctypes>
```

Note that the pattern matching operator is the ~ character, not the = character.

Step #8 – write NODL for SQL-based collection

We want to create an equivalent p-collection based on relational tables. As a starting-point, we request a NODL sample.

Call:

```
basex -b "request=_nodlSample?model=sql" /tta/pcollect/pcollect.xq
```

pcollect:

```
<nodl xmlns="http://www.infospace.org/pcollection">
  <collection name="COLLECTION_NAME" uri="" formats="xml" doc="A collection of
FOOs."/>
  <pmodel>
    <property name="foo" type="xs:string*" maxLength="100" expr="//foo"/>
    <property name="bar" type="xs:string" maxLength="100" expr="//bar"/>
    <property name="foobar" type="xs:string*" maxLength="100" expr="//foobar"/>
    <anyProperty/>
  </pmodel>
  <nodeDescriptor kind="SELECT_ONE: uri|text"/>
  <ncatModel>
    <sqlNcat rdbms="MySQL" host="localhost" db="DB" user="USER"
password="PASSWORD"/>
  </ncatModel>
</nodl>
```

We edit it by

- copying the <collection>, <pmodel> and <nodeDescriptor> elements from the XML-based NODL
- inserting into the <sqlNcat> element connection data and a database name:
  - host: localhost
  - database: pcollect
  - user: root
  - password: admin

The resulting NODL looks like this:

```
<nodl xmlns="http://www.infospace.org/pcollection">
  <collection name="xmls" uri="" formats="xml"
              doc="A collection of arbitrary XML documents."/>
  <pmodel>
    <property name="name" type="xs:string" maxLength="100"
                          expr="local-name(/*)"/>
    <property name="namespace" type="xs:string" maxLength="100"
                          expr="namespace-uri(/*)"/>
    <property name="namespaces" type="xs:string*" maxLength="100"
                          expr="distinct-values(//*/namespace-uri(.))"/>
    <property name="enames" type="xs:string+" maxLength="100"
                          expr="distinct-values(//*/local-name(.))"/>
    <property name="anames" type="xs:string*" maxLength="100"
                          expr="distinct-values(//@*/local-name(.))"/>
    <property name="ecount" type="xs:integer" maxLength="100"
                          expr="count(//*)"/>
    <property name="acount" type="xs:integer" maxLength="100"
                          expr="count(//@*)"/>
  </pmodel>
  <nodeDescriptor kind="uri"/>
  <ncatModel>
    <sqlNcat rdbms="MySQL" host="localhost" db="pcollect" user="root"
```

```
            password="admin"/>
  </ncatModel>
</nodl>
```

Step #9 – create SQL-based p-collection

We create the SQL-based collection as we created the XML-based collection. The only difference is the value of the `nodl` parameter.

Call:

```
basex -b request=_createNcat?nodl=/nodls/xmls-sql.nodl
   /tta/pcollect/pcollect.xq
```

`pcollect`:

```
<z:createSqlNcat xmlns:z="http://www.ttools.org/structure"
                 countTables="4"/>
```

We start a `mysql` session in order to observe the results of our actions.

```
mysql> use pcollect
Database changed
mysql> show tables;
+----------------------+
| Tables_in_pcollect   |
+----------------------+
| xmls_ncat            |
| xmls_ncat_anames     |
| xmls_ncat_enames     |
| xmls_ncat_namespaces |
+----------------------+
4 rows in set (0.00 sec)
```

Step #10 – populate SQL-based p-collection

We populate the SQL-based p-collection with the same documents as the XML-based p-collection. Again, we invoke the same operation (`_feedNcat`), with the only difference being the value of the `nodl` parameter.

Call:

```
basex -b "request=_feedNcat?nodl=/nodls/xmls-sql.nodl,
                            docs=|/projects/*.xml"
   /tta/pcollect/pcollect.xq
```

pcollect:

```
<feedNcat name="xmls" countProcessed="1486"/>
```

A quick check of what happened in the database:

```
mysql> select count(*);
+----------+
| count(*) |
+----------+
|     1486 |
+----------+
1 row in set (0.03 sec)

mysql> select count(*);
+----------+
| count(*) |
+----------+
|    24576 |
+----------+
1 row in set (0.02 sec)

mysql> select count(*);
+----------+
| count(*) |
+----------+
|    13194 |
+----------+
1 row in set (0.01 sec)

mysql>
```

Step #11 – query SQL-based p-collection

Again, we call operation _doctype and supply it with input documents from a filtered collection. We apply the same filter to the SQL-based collection, as we applied to the XML-based collection.

Call:

```
basex -b "request=_doctype?fdocs=/nodls/xmls-sql.nodl?
enames~*SearchEngine* && not(name=msgType || namespace~*items*)"
   /tta/pcollect/pcollect.xq
```

pcollect:

```
<z:doctypes xmlns:z="http://www.ttools.org/structure" countDocs="3" countDoctypes="3">
  <z:doctype name="msgs@http://www.ttools.org/logspy/ns/xquery-functions" count="1">
    <doc href="file:///C:/projects/data/msgs-serl/msgs-serl-t01.xml"/>
  </z:doctype>
  <z:doctype name="SearchEngineOfferListRS@" count="1">
    <doc href="file:///C:/projects/xq/workshop-2013-dev/data/regionList.xml"/>
  </z:doctype>
```

```
  <z:doctype
name="SearchEngineOfferListRS@http://traveltainment.de/middleware/xml/SearchEngineOfferListRS"
count="1">
    <doc href="file:///C:/projects/xq/workshop-2013-dev/examples/someTips/regionList.xml"/>
  </z:doctype>
</z:doctypes>
```

The result is identical to the result obtained when filtering the XML-based collection.

Step #12 – implement first operation `pcollect`

By now we have used our new topic tool `pcollect` in various ways, although we have not yet written a single line of code. The explanation is that we have used built-in operations which every topic tool inherits from the `ttools` framework.

To conclude our tutorial, we implement our first operation `nodls`, which so far only exists as a dummy version, generated by `ttools` during the instantiation of the topic tool. We consult the `_help` operation in order to learn the generated interface.

Call:

```
basex -b "request=?type,default,ops=nodls" /tta/pcollect/pcollect.xq
```

`pcollect`:

```
TOOL: pcollect

OPERATIONS   PARAMS
================================================================================
nodls        dcat............... : docCAT* (sep=WS)
             doc................ : docURI* (sep=WS)
             docs............... : docDFD* (sep=SC)
             fdocs.............. : docSEARCH* (sep=SC)
               At least 1 of these parameters must be set: dcat, doc, docs, fdocs
================================================================================
```

These operation parameters should be regarded as a “suggestion” which ttools makes, showing you how you can enable your tool users to specify input documents in a very flexible way:

- `dcat` – as a “document catalog” containing document URIs (such a catalog can be produced using built-in operation `_dcat`)
- `doc` – as one or more document URIs, whitespace-separated
- `docs` – as a directory filter
- fdocs – as a filtered p-collection

We test the generated dummy version of the operation.

Call:

```
basex -b "request=nodls?docs=|/projects/*.xml *.nodl"
   /tta/pcollect/pcollect.xq
```

pcollect:

```
<z:nodls xmlns:z="http://www.ttools.org/pcollect/ns/structure"
         countDocs="19"/>
```

The dummy version simply returns the number of input documents supplied (see attribute @countDocs). Here is the generated code:

```
declare function f:nodls($request as element())
        as element() {
    let $docs := tt:getParams($request, 'doc docs dcat fdocs')
    return
        <z:nodls countDocs="{count($docs)}">{
           ()
        }</z:nodls>
};
```

We now edit the code so that it extracts from the input documents all NODL documents and creates a small report about them, stating collection name, NODL URI and property names.

```
declare function f:nodls($request as element())
        as element() {
    let $docs := tt:getParams($request, 'doc docs dcat fdocs')
    let $nodls := $docs/pc:nodl
    let $nodlsInfo :=
        for $nodl in $nodls
        let $colName := $nodl/pc:collection/@name
        let $pnames :=
            for $p in $nodl/pc:pmodel/pc:property/@name
            order by $p return $p
        order by $colName
        return
            <z:nodl col="{$colName}"
                    nodl="{$nodl/root()/document-uri(.)}"
                    ncatModel="{$nodl/pc:ncatModel/*/local-name(.)}"
                    pnames="{$pnames}"/>
    return
        <z:nodls countDocs="{count($docs)}"
                 countNodls="{count($nodls)}">{
           $nodlsInfo
        }</z:nodls>
};
```

We repeat the previous call of our tool and observe the output created by the edited code.

Call:

```
basex -b "request=nodls?docs=|/projects/*.xml *.nodl"
   /tta/pcollect/pcollect.xq
```

pcollect:

```
<z:nodls xmlns:z="http://www.ttools.org/pcollect/ns/structure"
         countDocs="1488" countNodls="2">
  <z:nodl col="xsds" nodl="file:///C:/projects/infospace/sql/xsds-sql.nodl"
          ncatModel="sqlNcat"
          pnames="agroup att ctype elem enum group stype tns"/>
  <z:nodl col="xsds"
          nodl="file:///C:/projects/infospace/sql/xsds-xml.nodl"
          ncatModel="xmlNcat"
          pnames="agroup att ctype elem enum group stype tns"/>
</z:nodls>
```