Subject:

serialization of closure-functions in BaseX

Motivation:

W3C's Xpath- and Xquery-specifications are excellent for handling XML-data and BaseX is an excellent implementation. Unfortunately, Xquery's data typing-mechanism is too primitive for serious functional programming. The lack of a generic data typing framework and the lack of nice specifications for several data container types makes serious programming in Xquery not as convenient and transparent as it should and could be.

Missing container data types are, to my opinion: a simple data-container[i], a data-set[ii] and most important a data-record[iii]. In the end, the use of closure-functions appears to be, in my opinion, the most elegant and maintenance friendly implementation within Xquery itself for these missing data-structures. The construction of closure-functions itself is pretty obscure but is nothing more than a trick that can be mastered, I managed. And one only need to do this trick once for each data-item.

Since version 8.4 of BaseX, with the introduction of the extended 'fn:serialize'-function, most of the generic data typing issues, that I use anyway, are tackled for the standard data types. It would however be nice if closure-functions could also fit in with the generic data functions, such as fn:deep-equal and fn:serialize. For now I could settle for an serialization-function that also could handle closure-functions but …

Closure-functions

Closure-functions are no more than self-contained argument lists or body-less functions which can easily be linked to any function with the same list structure as its parameter list. Closure-functions actually fulfill the role of user-defined data-structures or one could even say the role of data-objects.

Example of a closure-function in Xquery:

```
function( $f) { $f( $data-item-1, $data-item-2, ……)}
```

Unfortunately, neither Xquery nor BaseX can distinguish closure-functions from regular functions: they are all functions on the internal typing-level. Internally a closure-function is not recognized as a datatype.

Annotation

One way to 'solve' this on the Xquery-level is by the use of annotations. By annotating our closure-functions we can:

- Determine that the function actually is a closure-function.
- Trace back to its source for extra information about the closure-function if necessary.
- Pass a self-defined name of this user-defined datatype.

Our closure-function would look like this:

%UDT[1]:datatype('myDataTypeName') function($f) { $f($data-item-1, $data-item-2, ……)}

Serialization

For serialization of a closure-function one could use for example the following layout:

myDataTypeName::[serialized-result-of-$data-item-1, serialized-result-of-$data-item-2, ….]

Addendum

[i] A simple data container is a data-item that can hold a various number of data-items. Since a data container is a data-item, it can hold other data containers. In that respect, Xquery's sequence is actually not a data-type.

[ii] A dataset is a data-item that can hold a various number of unique data-items.

[iii] A datarecord is a data-item that can hold a specified number of data-items. Since XML is primary intended for external, un- or semi-typed data-structures, using this for internal, strictly typed data-structures seems to me just not a good idea.

[1] UDT: user defined data type. For example.