TinyDb_export_import

Oct 2018

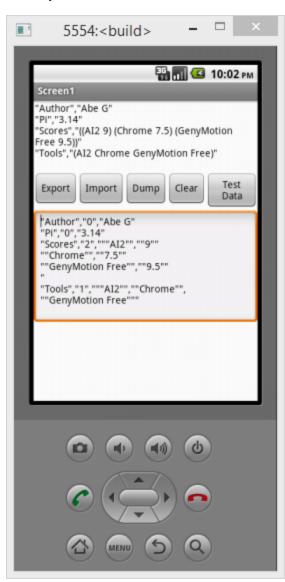
```
Background
Sample Run
   My sample data
Designer
   Components
       IblDump
       IvwDump
       btnExport
       btnImport
       btnDump
       btnClear
       <u>btnTestData</u>
       txbExpImp
Blocks
   When btnExport.Click
       <u>exportTinyDB</u>
           <u>dumpTinyDB</u>
           Dimensions
              maxSubDimension
           Exportable
   When btnImport.Click
       <u>importTinyDB</u>
           Importable
   All blocks (in case I missed something)
Gallery Link
Other Projects
```

Background

This app was developed to show how to export and import the complete contents of a TinyDB into and out of a block of text, suitable for writing and reading to and from an external text file for backup and recovery purposes.

This is not meant as an introduction to TinyDB or to list processing. It is meant to be useful in its own right as a tool. It's not necessary to fully understand the internals of a tool to use it, though it helps to understand the limitations of a tool.

Sample Run



This is the screen display of a sample run, using my sample data.

The top section contains a dump of my TinyDB sample, with four tag/value pairs, shown in the lazy and non-reversible Al2 list to text transformation, using parentheses and spaces to wrap list items.

The middle section has some buttons to exercise the various facilities, explained later.

The bottom section has the exported text of the TinyDB contents, complete with tags, values, and an extra dimension column needed to help reconstitute the values when re-importing the text into a new TinyDB.

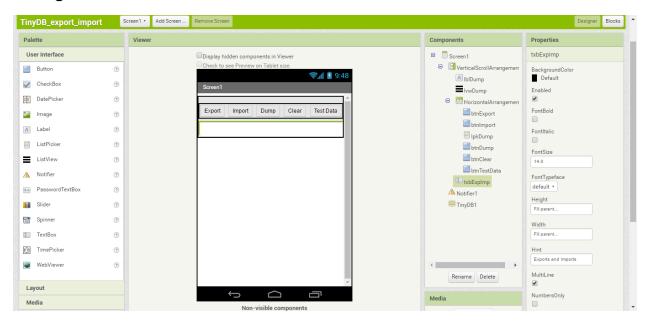
My sample data

I included four pieces of data, to show off how I handle simple values, lists, and tables.

- Simple values (0 dimensions)
 - Tag = "Author", value = "Abe G"
 - Tag = "Pi", value = "3.14"
- Lists (1 dimension)
 - Tag = "Tools",
 - Value =
 - "AI2"
 - "Chrome"
 - "Genymotion Free"
- Tables (2 dimensions)
 - Tag = "Scores",
 - Value =
 - "Ai2", 9
 - "Chrome", 7.5
 - "Genymotion Free", 9.5

I have listed my sample data here in ascending order by dimension, from simplest structure (plain values) to complex (two dimensional tables.) My sample data displayed by the app comes out in tag order, which TinyDB returns alphabetically by tag.

Designer



Components

IbIDump

A label for output of a dump of TinyDb. A dump is a haphazard lazy export, too jumbled for re-import.

IvwDump

A ListView for output of a dump of TinyDb. A dump is a haphazard lazy export, too jumbled for re-import.

btnExport

The Export button. It formats TinyDB into one piece of text, then sends it to the output text box.

btnImport

The Import button. It takes the text from the output text box, reforms it into its original tags and values, and loads them into TinyDB.

btnDump

The Dump button. It sends the contents of TinyDB to IbIDump.

btnClear

The Clear button. It clears TinyDB.

btnTestData

The Test Data button. It loads test data into TinyDB.

txbExpImp

The Export/Import Text box. Target for export, source for Import. A Text Box is better for this than a Label, because you can cut and paste it in an emulator.

Blocks

When btnExport.Click

```
when btnExport v .Click
do set [blDump v . Text v to call exportTinyDB v
```

The <u>Export button</u> clears label <u>IbIDump</u> to make room on the screen for <u>txbExpImp</u>, which is loaded from procedure <u>exportTinyDB</u>.

```
to exportTinyDB
         o initialize local (exportables) to
result
                                             create empty list
                   initialize local dump to
                                             call dumpTinyDB *
                     for each (item) in list
                                            get dump *
                          initialize local (tag) to
                                                      select list item list
                                                                                          index
                            initialize local value to
                                                      select list item list
                                                                           get item
                                                                                         index
                             initialize local Dim to
                              set Dim v to
                                                 call Dimensions x x
                                                                          get value
                               o add items to list list
                                                          get exportables
                                                          call exportable *
                                                  item 🌘
                                                                                get (tag 🔻
                                                                         tag
                                                                         dim
                                                                                get Dim
                                                                       value
                                                                                get value
                         list to csv table list
                                               get exportables
```

This is a value procedure, returning a Comma Separated Values (CSV) text representation of a table of the exportable representation of each (tag,dimension,value) row.

It works in three phases:

- Extract all (tag/value) pairs from TinyDB into local variable dump using procedure dumpTinyDB
- 2. For each dumped pair (tag/value), get the dimension (0/1/2) of that value
- 3. Build an <u>exportable</u> row suitable for adding to a 2 dimensional table representing the tag, dimension, value

Finally, it returns the CSV Table text.

dumpTinyDB

```
result initialize local dump to create empty list initialize local tags to call TinyDB1 v .GetTags in do for each tag in list get tags v do add items to list list get dump v item make a list get tag v valueIfTagNotThere get tag v valueIfTagNotThere
```

This value procedure returns a two column table with each row containing a TinyDB tag in column 1, and its corresponding TinyDB value in column 2. Notice that this works only in Tinydb1. If you want to use multiple TinyDB instances with different name spaces, you will have to add that yourself.

Dimensions

```
Return 0,1, or 2, depending on its input being a scalar, list, or two dimensional table.

To Dimensions X

result if ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is a list? thing ( get X Y )

then ( is
```

To determine the number of dimensions of a structure, we divide up the question recursively using a helper procedure <u>maxSubDimension</u>. If the structure is not a list, it has dimension 0. Otherwise, it's at least a list and its dimension is 1 more than the highest dimension of the items in that list.

maxSubDimension

```
Return the highest dimension of any of the items in a list.

Return the highest dimension of any of the items in a list.

Return the highest dimension of any of the items in a list.

Go initialize local Dim to (O) initialize local subDim to (O) ini
```

This can get a bit mind boggling. We are about to use recursion, where a procedure calls another procedure that may have called it. The recursion must eventually end, as long as it is working with a value from TinyDB, which filters out abnormal lists like the notorious <u>Ouroborous</u>.

We return a local variable Dim, which is initially 0, but is compared to the <u>dimension</u> of each item in the input list, and is set to the maximum of itself or that new dimension value.

After exiting the for each loop, variable Dim will have the maximum value of the dimensions of all the items in the given list.

Exportable

```
to exportable
                     tag
                            dim
                                   value
result
             make a list
                            get tag
                            get dim
                                        get dim
                                     get value
                            then
                             else
                                                 get dim
                                              list to csv row list
                                                                    get value 🔻
                                     then
                                              list to csv table list
                                                                     get value
```

To make an exportable representation of a TinyDB tag and its value, we have to flatten the value so it will fit into a piece of text in the 3rd position of a list (tag, dimension, flattened value). Fortunately, we have available the dimension (0/1/2) of the value in our second input parameter.

A 0 dimensional value is okay in itself, and can be used directly. A 1 dimensional value can be run through the **list to csv row** block to turn into a CSV row text. A 2 dimensional value can be run through the **list to csv table** block to turn into a CSV table text. If the dimension is higher than 2, all bets are off, and the app developer should read up on the relational model then redesign his data structures.

The output of this procedure will be a 3 item list (tag, dimension, flat text value).

When btnImport.Click

```
when btnImport v .Click
do call importTinyDB v text (txbExpImp v . Text v)
```

The Import button takes the contents of text box <u>txbExpImp</u> and loads it into TinyDB1 through procedure <u>importTinyDB</u>.

importTinyDB

```
to (importTinyDB)
    initialize local importable to list from csv table text get text
          for each row in list
                                 get importable
               o initialize local tag to
                                            select list item list
                                                                 get row
                                                                               index
                   initialize local dim to
                                            select list item list
                                                                               index
                                                                 get row
                 initialize local value to
                                            select list item list
                                                                 get row
                                                                               index
                    call TinyDB1 .StoreValue
                                                     get [tag 🔻
                                                     call [importable *
                                    valueToStore
                                                                    dim
                                                                            get dim 🔻
                                                                    CSV
                                                                            get value
```

This procedure does the opposite of the <u>exportTinyDB</u> procedure. It accepts a CSV Table text block and runs it through a list from csv table block to get a three column (tag,dimension, exported value) table.

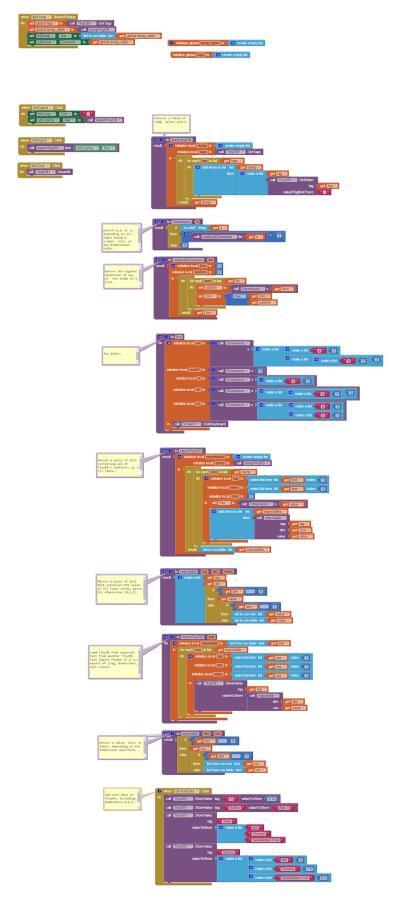
For each row in that table, that row is transformed back into its original tag/value pair using the <u>importable</u> procedure and the given dimension (0/1/2) number of that value, then stored into TinyDB1.

Importable

```
then get csv
else if get dim v = v 0
then get csv
else if get dim v = v 1
then list from csv row text get csv
else list from csv table text get csv
else list from csv table text get csv
else list from csv table text get csv
```

This value procedure does the opposite of the <u>exportable</u> procedure, reconstituting a simple value, list, or table from a text value and its dimension number, using the **list from csv row** or **list from csv table** blocks, as appropriate.

All blocks (in case I missed something)



Gallery Link

ai2.appinventor.mit.edu/?galleryId=6019316599488512

Other Projects

 $\underline{https://docs.google.com/document/d/1acg2M5KdunKjJgM3Rxpy_Rf6vT6OozxdIWglgbmzroA/ed} \\ \underline{it?usp=sharing}$