Introduction to swapped datatype

by Viktor Horvath 7/19/2017

Swapped datatype is designed to create a simple cell array interface for a local datastorage. It can be initialized similarly to a cell array object, but it provides additional functionalities that help access data and keep track of occupancy, etc. In the following some examples of usage are provided.

Usage of swapped cell array

Initialization using standard object call

```
X = swapped(10);
disp(X)
```

```
  swapped with properties:

    folder: '.'
      UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
    bitmap: [10×10 double]
      path: '.\eca7132e-8770-4e19-a75d-a706dd7688dc'
      size: [10 10]
     numel: 100
```

This call generates an empty cell array of 10x10 size = 100 elements.

Elements can be accessed through the standard cell array interface.

```
X{1}
```

```
ans =
     []
```

The data structure does not hold content, therefore it has not been written out to the harddisk. Writing to the harddisk occurs only when there is change, e.g. assignment to the array elements.

```
X{1} = zeros(1000);
X
```

```
X =
  swapped with properties:

      folder: '.'
        UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
      bitmap: [10×10 double]
        path: '.\eca7132e-8770-4e19-a75d-a706dd7688dc'
        size: [10 10]
       numel: 100
   occupancy: 66611
```

```
X.path
```

This is an assigment that would require about 7.6 MBytes of operative memory to store if it were in RAM. Note that the occupancy after this assignment is only about 65 kBytes, because MATLAB was able to compress the data. The output was written under the folder given in the `X.path` property, into the file `1.mat`. Note that we take advantage of storing data in the .mat format. This allows you to store any object types in the swapped cell array.

Let's assign to the index `1, 10` the text `'Hello world!'`

```
X{1, 10} = 'Hello world!';
X
```

```
X =
  swapped with properties:

        folder: '.'
          UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
        bitmap: [10×10 double]
          path: '.\eca7132e-8770-4e19-a75d-a706dd7688dc'
          size: [10 10]
         numel: 100
     occupancy: 68547
```

Note that the size increased, again slightly. The data was written into the file `91.mat` that is the linear index, equivalent of the subs index `1, 10`.

Next, let's clear this cell array and try to load it back into the variable `Y`. The main object file is saved under the `<UUID>.mat` file in the folder specified in the *read-only* property `X.folder`. To load the cell array object we first need to save the `UUID` of the cell array into another variable, `source_UUID`.

```
source_UUID = X.UUID
```

```
source_UUID = 'eca7132e-8770-4e19-a75d-a706dd7688dc'
```

```
clear X;
Y = swapped([source_UUID, '.mat']);
Y
```

```
Y =
  swapped with properties:

        folder: ''
          UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
        bitmap: [10×10 double]
          path: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
          size: [10 10]
         numel: 100
     occupancy: 69275
```

Let's check if the data we assigned as variable `X` is still available.

```
size(Y{1})
```

```
ans =
```

```
         1000        1000
```

```
Y{1, 10}
```

```
ans = 'Hello world!'
```

We can also try to index these elements within the swapped cell array.

```
Y{1}(10, 10)
```

```
ans = 0
```

```
Y{1, 10}(2)
```

```
ans = 'e'
```

The indexing is propagated to the stored element in accordance with cell array indexing standards.

The name of the temporary file `<UUID>.mat` can be changed when the variable is offline. The name change does not affect the main variable, however the directory name that holds the data files must be kept the same name. To illustrate this, let's clear the variable Y and rename the object file `<UUID>.mat`. Then load it, and assign some the value '`Name has been changed.`' at index 2, 5.

```
clear Y;
movefile([source_UUID, '.mat'], 'test.mat');
Z = swapped('test.mat');
Z{2, 5} = 'Name has been changed.'
Z
```

```
Z =
  swapped with properties:

        folder: ''
          UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
        bitmap: [10×10 double]
          path: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
          size: [10 10]
         numel: 100
     occupancy: 71235
```

Data can also be removed from the cell array and the harddisk by assigning empty array to an element. Let's remove the first element, `Z{1}`.

```
Z{1} = [];
Z
```

```
Z =
  swapped with properties:

        folder: ''
          UUID: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
        bitmap: [10×10 double]
```

```
          path: 'eca7132e-8770-4e19-a75d-a706dd7688dc'
          size: [10 10]
         numel: 100
     occupancy: 20984
```

Note that the size has been reduced and the `1.mat` file was removed from the harddisk from the storage folder.

To accelerate checking wheter an element is occupied in the swapped cell array, the curly braces, `()`,can be used:

```
Z(1)
```

```
ans = 0
```

```
Z(2, 5)
```

```
ans = 1
```

```
Z(1, 10)
```

```
ans = 1
```

This should help when the array is iterated through.

Data storage under a specific folder

If the default folder (current MATLAB path) is not desired, a specific folder can also be used.

```
F = swapped('./tmp/dataset 1', 6);
F{1} = rand(100);
F
```

```
F =
  swapped with properties:

        folder: '.\tmp\dataset 1'
          UUID: '0691b7a0-0356-48a8-a3c8-356d120f52fe'
        bitmap: [6×6 double]
          path: '.\tmp\dataset 1\0691b7a0-0356-48a8-a3c8-356d120f52fe'
          size: [6 6]
         numel: 36
     occupancy: 96668
```

MATLAB will create all necessary folders on the path as soon as the assignment takes place.

Multiple handles to the same data storage

The swapped cell array is a datatype that inherits the handle datatype. This allows us to use multiple handles pointing to the same data storage. To illustrate this I will create a new swapped cell array object and assign it to another variable.

```
A = swapped(2);
A
```

```
A =
  swapped with properties:
```

```
      folder: '.'
        UUID: '0691b7a0-0356-48a8-a3c8-356d120f52fe'
      bitmap: [2×2 double]
        path: '.\0691b7a0-0356-48a8-a3c8-356d120f52fe'
        size: [2 2]
       numel: 4
```

```
B = A;
B
```

```
B =
  swapped with properties:

      folder: '.'
        UUID: '0691b7a0-0356-48a8-a3c8-356d120f52fe'
      bitmap: [2×2 double]
        path: '.\0691b7a0-0356-48a8-a3c8-356d120f52fe'
        size: [2 2]
       numel: 4
```

```
B{1} = 10;
A{1}
```

```
ans = 10
```

Note that both `A` and `B` are pointing to the same underlying datastore which can be managed through either variables.

If one of them is cleared, the data is still available through the other one.

```
clear B;
A{1}
```

```
ans = 10
```

Recovery when MATLAB (or the computer) shutdown unexpectedly

When there is an unexpected shutdown the main variable does not get to be written out in its final state. However, since most of the object properties are generated dynamically this is not a problem as long as the initial write out is complete. During the initialization the data storage folder is scanned and the swapped cell array occupancy matrix (`.bitmap` property) is populated as elements are detected. The following warning is emitted to show that there was an issue upon initialization and the datastructure was reconstructed.

```
Warning: Inconsistency between object and data, reconstructing bitmap

> In swapped/cast_swapped (line 25)

  In swapped (line 80)
```