SEP # : 40 – Cell Arrays

Title: Cell Arrays
Version: 1.8
Author: Bruno JOFRET
Review:
Commented:
State:
Scilab-Version: 6.0
Vote:
Created: March 24, 2010

Table of Contents

Abstract

This details the evolution of Scilab cell arrays data type.
The main evolution consists in introducing a new syntax for cell arrays using braces.

Rationale

Scilab is a matrix oriented programming language so you can easily declare matrices containing numerical elements or string of characters, but the elements must be of the same type. There is also a need to have « container » data types that will allow mixing of different types.

Those datatypes are listed below:

- list
- tlist
- mlist
- struct
- cell

Considering Scilab < 6.0 implementations, cells and structs are only wrappings around mlist. So there is no « native » datatype for this, and they can only be managed through functions and they are not efficient.

In the version 6.0 of Scilab we will introduce the capability to declare/address cells as objects matrices.

Functionalities

Empty cell management

This describes how to create an empty cell.

- Before Scilab 6.0:
  ```
  emptyCell = cell()
  ```
- After Scilab 6.0:
  ```
  emptyCell = cell()
  ```
  or
  ```
  emptyCell = {}
  ```

Cell Creation

Example 1-D cell

This describes how to create a one-dimensional cell containing a double value: 42.

42

- Before Scilab 6.0:
  ```
  c = makecell([1,1], 42)
  ```
  or
  ```
  c = cell(1,1)
  c(1).entries = 42
  ```
- After Scilab 6.0:
  ```
  c = {42}
  ```

Notice that an easy-way to create cell is to use insertion capability (through .entries field) in Scilab version before Scilab 6.0. We will detail the new insertion process later on, and especially backward compatibility.

Example 2-D cell

This describes how to create a two-dimensional cell. The cell will have 3 rows and 2 columns. It will represent the table below :

[1 , 2 , 3]	'Scilab'
1 + 2s + 3s^2	%t
int32(42)	list('foo', [])

- Before Scilab 6.0:
  ```
  c = makecell([3,2], 1:3, 'Scilab', poly(1:3, 's', 'coeff'),
  %t, int32(42), list('foo', []))
  ```
  or
  ```
  c = cell(3,2);
  c(1,1).entries = 1:3;
  c(1,2).entries = 'Scilab';
  c(2,1).entries = poly(1:3, 's', 'coeff');
  c(2,2).entries = %t;
  c(3,1).entries = int32(42);
  ```
- After Scilab 6.0:
  ```
  c = {1:3 , 'Scilab' ; poly(1:3, 's', 'coeff') , %t ;
  int32(42), list('foo', [])}
  ```

Example n-D Cell

This describes how to create a multi-dimensional cell. For our purpose we will create a 4 x 3 x 2 cell. It will represent the two tables below. One in front (last dimension is 1) and the other in back (last dimension is 2).

- Front:

1	5	9
2	6	10
3	7	11
4	8	12

- Back:

13	17	21
14	18	22
15	19	23
16	20	24

- Before Scilab 6.0:
  ```
  c = cell([4,3,2]);
  c(1,1,1).entries = 1
  c(2,1,1).entries = 2
  c(3,1,1).entries = 3
  c(4,1,1).entries = 4
  c(1,2,1).entries = 5
  c(2,2,1).entries = 6
  ```

```
c(3,2,1).entries = 7
c(4,2,1).entries = 8
c(1,3,1).entries = 9
c(2,3,1).entries = 10
c(3,3,1).entries = 11
c(4,3,1).entries = 12
c(1,1,2).entries = 13
c(2,1,2).entries = 14
c(3,1,2).entries = 15
c(4,1,2).entries = 16
c(1,2,2).entries = 17
c(2,2,2).entries = 18
c(3,2,2).entries = 19
c(4,2,2).entries = 20
c(1,3,2).entries = 21
c(2,3,2).entries = 22
c(3,3,2).entries = 23
c(4,3,2).entries = 24
```
or
```
c = cell([4,3,2]);
for i = 1:24
c(i).entries = i
end
```
or
```
c= makecell([4,3,2],
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,2
4)
```
- After Scilab 6.0 :
```
c = cell([4,3,2]);
for i = 1:24
c{i} = i
end
```
or
```
c = cell([4,3,2]);
c(:,:,1) = {1,5,9;2,6,10;3,7,11;4,8,12};
c(:,:,2) = {13,17,21;14,18,22;15,19,23;16,20,24}
```

Notice that you can only initialize a n-dimensional cell by filling values inside. Or in best case, decrease dimension down to 2 in order to write values using braces.

Sub-cell Extraction

This describes how to extract sub-cell from an existing cell.
We assume we have the following cell defined and called c:

[1 , 2 , 3]	'Scilab'
1 + 2s + 3s^2	%t
int32(42)	list('foo', [])

- Before Scilab 6.0:

```
e1 = c(1, 1)
e2 = c(2, 1)
e3 = c(3, 1)
e4 = c(1, 2)
e5 = c(2, 2)
e6 = c(3, 2)
```

- After Scilab 6.0:

```
e1 = c(1, 1)
e2 = c(2, 1)
e3 = c(3, 1)
e4 = c(1, 2)
e5 = c(2, 2)
e6 = c(3, 2)
```

All ei are cells.
e1 is a cell containing a matrix value: [1, 2, 3]
e2 is a cell containing one polynomial value: 1 + 2s + 3s^2
…

Values Extraction

This describes how to extract values from an existing cell.

We assume we have the following cell defined and called c:

[1 , 2 , 3]	'Scilab'
1 + 2s + 3s^2	%t
int32(42)	list('foo', [])

- Before Scilab 6.0:

```
e1 = c(1, 1).entries
e2 = c(2, 1).entries
e3 = c(3, 1).entries
e4 = c(1, 2).entries
e5 = c(2, 2).entries
e6 = c(3, 2).entries
```

- After Scilab 6.0:

```
e1 = c{1, 1}
e2 = c{2, 1}
e3 = c{3, 1}
e4 = c{1, 2}
e5 = c{2, 2}
e6 = c{3, 2}
```

Notice, that we are extracting values and no more sub-cells. The use of braces replaces the « old » .entries field.

Sub-Cell Insertion

This describes how to insert a sub-cell in an existing cell.

- Before Scilab 6.0:
  ```
  c = cell([2,2])
  c(2,2) = makecell([1,1], 123)
  ```
- After Scilab 6.0:
  ```
  c = cell([2,2])
  c(2,2) = {123}
  or
  clear c
  c(2,2) = {123}
  ```

We will add the cell automated creation in Scilab 6.0, as shown in the second instruction without creating the cell before filling it, like matrices actually do.

Values Insertion

This describes how to insert a sub-cell in an existing cell.

- Before Scilab 6.0:
  ```
  c = cell([2,2])
  c(2,2).entries = 123
  ```
- After Scilab 6.0:
  ```
  c = cell([2,2])
  c{2,2} = 123
  or
  clear c
  c{2,2} = 123
  ```

We will add the cell automated creation in Scilab 6.0, as shown in the second instruction without creating the cell before filling it, like Matrices actually do.

Breaking backward compatibility

Brace and bracket equivalence

Considering we are introducing the braces in order to have an easy way to write cells, braces characters will no more be equivalent to bracket ones.

- Before Scilab 6.0:
  ```
  -->[1,2;3,4] == {1,2;3,4}
   ans  =

    T T
    T T
  ```
- After Scilab 6.0:
  ```
  -->[1,2;3,4] == {1,2;3,4}

  Undefined operation for the given operands.

  check or define function %s_o_ce for overloading.
  ```

Values access through .entries field

Before Scilab 6.0 cells are based on mlist datatype. Knowing this, all data are stored in a field called “entries”, so the user has to access this field to extract its data.

By introducing braces to direct access the data, we will remove the access through the “entries” field.

- Before Scilab 6.0:
```
-->c = makecell([2,2],1,2,3,4);

-->c(2,2).entries == 4
 ans  =

 T
```
- After Scilab 6.0:
```
-->c = {1,2;3,4}

-->c{2,2} == 4
 ans  =

 T
```

Function makecell to create cell

Before Scilab 6.0 the only way to create cells in a single call was through the makecell function. Considering we are adding an easiest way to do this, we will remove the makecell function.

Cell comparison

Before Scilab 6.0, due to the implementation based on mlist, comparing 2 cells will return a matrix of 3 booleans representing the type (“ce”), the matrix of dimensions, and the list of elements. According to the new implementation, this “feature” will be removed after Scilab 6.0.

Changelog

```
1.0      - Initial Version
1.1      - First set of functionalities
1.2      - Empty cells
1.3      - Cell creation
1.4      - Extraction
1.5      - Refactoring document change Table to document hierarchy
1.6      - Insertion
1.7      - Comments by Vincent Couvert
1.8      - Adding know breaking compatibilities
```

Copyright

Bruno JOFRET