Table of Contents

XML for Data storage

by LaurenceBoyd Mar 20, 2014 6:21 am 497095900

XML

XML is a flexible, convenient, way of storing data, once you understand how.
Programs typically evolve thru several versions. This often results in different data items being stored by different program versions. This has meant careful tracking of data version vs program versions, as programs could not correctly read data from the wrong data version.
XML is flexible as names are stored with the data. Names which are understood can be read, while others are ignored. The downside is that missing variables may require defaults.
XML is a very convenient to use. The rules are quite simple.
XML consists of elements:

text

Every opening is paired with a closing

```
<XML>
<!-- Mar 12, 2014 10:56:25 C:\Data\Sudoku\3.40.88.Sudoku -->
  <Sudoku>
    <Size> 9 </Size>
    <Diagonal> Off </Diagonal>
    <Possible> Off </Possible>
    <Reduce> Off </Reduce>
    <Cell> <Row> 1 </Row><Col> 1 </Col><Value> 7 </Value>
```

```
                    </Cell>
    <Cell> <Row> 1 </Row><Col> 7 </Col><Value> 9 </Value>
                    </Cell>
    <Cell> <Row> 1 </Row><Col> 8 </Col><Value> 3 </Value>
                    </Cell>
    <Cell> <Row> 2 </Row><Col> 4 </Col><Value> 6 </Value>
                    </Cell>
    <Cell> <Row> 2 </Row><Col> 5 </Col><Value> 4 </Value>
                    </Cell>
    <Cell> <Row> 2 </Row><Col> 6 </Col><Value> 7 </Value><Stage> 2
</Stage> </Cell>
    <Cell> <Row> 3 </Row><Col> 1 </Col><Value> 1 </Value>
                    </Cell>
    <Cell> <Row> 9 </Row><Col> 2 </Col><Value> 4 </Value>
                    </Cell>
    <Cell> <Row> 9 </Row><Col> 3 </Col><Value> 1 </Value>
                    </Cell>
  </Sudoku>
</XML>
```

Table of Contents

Reading Data

Reading XML data is a little more work than writing it, but still not difficult. Since you wrote the file, you only need to be able to read what you wrote.
For this code segment, variable names with initial letter capitalized are assumed to have been declared global.

```
sub FileLoad
```

```
 filedialog "Load file...", FilePath$+"*.Sudoku", FullName$
 call GetFileName$
 if FileName$="" then notice "No file chosen!": end
 open FullName$ for input as #File
 FileOpen = True
 call ProcessFile
end sub

sub GetFileName$
 for pos = len(FullName$) to 1 step -1
   if mid$(FullName$, pos, 1) = "\" then exit for
 next pos
 FileName$ = mid$(FullName$,pos+1)
 FilePath$ = left$(FullName$,pos)
end sub

sub ProcessFile
 do
   call FindTag
   select case Tag$
     case "<XML>"
     case "<Sudoku>":call InitCellArrays :call LoadPuzzle
     case "</XML>"  :close #File: FileOpen = False
   end select
 loop until FileOpen = False
end sub

sub LoadPuzzle
 do
   call FindTag
   select case Tag$
     case "<Cell>"     :call LoadCell
     case "<Diagonal>" :DiagActive = FindContentLogic()
     case "</Diagonal>"
     case "<Possible>" :PossibleShow = FindContentLogic()
     case "</Possible>"
     case "<Reduce>"   :ReduceActive = FindContentLogic()
     case "</Reduce>"
     case "<Size>"     :call CheckSize
     case "</Size>"
     case "</Sudoku>"  :call ShowRefresh: exit sub
   end select
 loop until FileOpen = False
end sub

sub LoadCell
```

```
 stage = 1
 do
   call FindTag
   select case Tag$
     case "<Row>"   :row = FindContentValue()
     case "</Row>"
     case "<Col>"   :col = FindContentValue()
     case "</Col>"
     case "<Value>" :val = FindContentValue()
     case "</Value>"
     case "<Stage>" :stage = FindContentValue()
     case "</Stage>"
     case "</Cell>" :CellValue(row,col) = val: CellStage(row,col) =
 stage
exit sub
   end select
 loop until FileOpen = False
end sub

sub LoadInputBuffer
 if eof(#File) <> 0 then close #File: FileOpen = False: exit sub
 do until (len(InputBuffer$) > 0 ) or (eof(#File) <> 0 )
'skip blank lines
   line input #File, InputBuffer$
 loop
end sub

sub FindTag
 if len(InputBuffer$) = 0 then call LoadInputBuffer
 for i = 1 to len(InputBuffer$)
   if mid$(InputBuffer$, i, 1) = "<" then exit for
 next i
 for j=i+1 to len(InputBuffer$)
   if mid$(InputBuffer$, j, 1) = ">" then exit for
 next j
 Tag$ = mid$(InputBuffer$, i, j-i+1)
 InputBuffer$ = mid$(InputBuffer$, j+1) 'remove tag
end sub

function FindContentString$()
 for i = 1 to len(InputBuffer$)
   if mid$(InputBuffer$, i, 1) = "<" then exit for
 next i
 FindContentString$ = trim$(mid$(InputBuffer$, 1, i-1))
 InputBuffer$ = mid$(InputBuffer$, i) 'remove content
end function
```

```
function FindContentValue()
 for i = 1 to len(InputBuffer$)
   if mid$(InputBuffer$, i, 1) = "<" then exit for
 next i
 FindContentValue = val(mid$(InputBuffer$, 1, i-1))
 InputBuffer$ = mid$(InputBuffer$, i) 'remove content
end function

function FindContentLogic()
 FindContentLogic = False
 if FindContentString$() = "on" then FindContentLogic = True
end function
```

20 Mar 2014 9:18