CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 20 Saving Data in Files
Contents

Contents 1
References 1
Saving Data in Files 2
  Simple Text Files 6
  StoreString – Saving Objects in ASCII 9
  BOSS – Binary Object Storage System 12
  XML Configuration Files 14
  OmniBase – Object Database 17

References

VisualWorks Developer's Guide, Chapter 21 Binary Object Files

OmniBase documentation,
http://www.gorisek.com/homepage/index.html

Copyright ©, All rights reserved. 2003 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

Saving Data in Files

Some Issues

- ASCII verses Binary Files
- Extracting data from objects verse saving objects
- Multiple program access of data
- Concurrent access to data file
- Nested Objects
- Circular references
- Handling changes in class of a saved object
- Searching for data in the saved file

Handling changes in class of a saved object

Save an object or data in an object to a file

Modify the class by:

- Adding methods
- Removing methods
- Adding instance variables
- Removing instance variables
- Renaming instance variables
- Reordering instance variables

Read the data in the file

Recreate the object

What happens?

In Smalltalk adding/removing methods does not affect recreating the object

Customer Class Used in Examples

```
Smalltalk defineClass: #Customer
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: 'name phone id '
   classInstanceVariableNames: 'nextId '
   imports: ''
   category: 'Course-GUI-Examples'

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
```

Customer class methods

```
name: nameString phone: phoneString
   ^self new
      name: nameString;
      phone: phoneString;
      id: self nextId

nextId
   nextId ifNil: [nextId := 0].
   ^nextId := nextId + 1.
```

Instance Methods

```
= aCustomer
   ^(name = aCustomer name) &
   (phone = aCustomer phone) &
   (id = aCustomer id)

id
   ^id

id: anInteger
   id := anInteger

name
   ^name

name: aString
   name := aString

objectTerminator
   ^Character cr

phone
   ^phone

phone: aString
   phone := aString
```

Simple Text Files

Let an Object know how to save/restore itself

Convert instance variables to strings

Use characters to separate instance variables

Saving

```
Customer>>saveOn: aStream
   aStream
      nextPutAll: name;
      nextPut: self fieldSeparator;
      nextPutAll: phone;
      nextPut: self fieldSeparator;
      nextPutAll: id printString;
      nextPut: self objectTerminator

Customer>>objectTerminator
   ^Character cr

Customer>>fieldSeparator
   ^$,
```

Restoring

```
Customer class>>readFrom: aStream
   ^super new readFrom: aStream

Customer>>readFrom: aStream
   name := aStream upTo: self fieldSeparator.
   phone := aStream upTo: self fieldSeparator.
   id := (aStream upTo: self objectTerminator) asNumber
```

Sample Use
Saving

```
| out customers dataFile |
dataFile := 'dataFile.txt' asFilename.
out := dataFile writeStream.
customers := (OrderedCollection new)
   add: (Customer name: 'roger' phone: '1234567');
   add: (Customer name: 'pete' phone: '1111111');
   add: (Customer name: 'jose' phone: '2222222');
   yourself.
customers do: [:each | each saveOn: out].
out close.
```

Restoring

```
| customers in dataFile |
dataFile := 'dataFile.txt' asFilename.
in := dataFile readStream.
customers := OrderedCollection new.
[in atEnd] whileFalse: [customers add: (Customer readFrom: in)].
```

dataFile.txt Contents

```
roger,1234567,34
pete,1111111,35
jose,2222222,36
```

Issues

What characters to use to separate instance variables?
What character to use to end an object data?
How to handle nested objects?
What happens when you add/remove instance fields to the class?

Properly selected separators allow other programs to manipulate data files

StoreString – Saving Objects in ASCII

```
storeString
   "Answer a String representation of the receiver from which
   the receiver can be reconstructed."
```

Automatically serializes objects into strings

This is appropriate only for smaller simpler objects.

Can handle nested objects

Cannot handle circular references of objects.

Uses order of instance variables, can handle
- Adding new instance variables at end
- Renaming instance variables

Cannot handle
- Reordering of instance variables
- Removing of instance variables
- Adding instance variable in the middle of the variable list

Calls storeOn: to perform work

Saving

```
| out customer dataFile |
dataFile := 'dataFile.txt' asFilename.
out := dataFile writeStream.
customer := Customer name: 'roger' phone: '1234567'.
out
   nextPutAll: customer storeString;
   close
```

Restoring

```
|  contents customer dataFile |
dataFile := 'dataFile.txt' asFilename.
contents := dataFile contentsOfEntireFile.
customer :=Compiler evaluate: contents.
customer inspect
```

dataFile.txt Contents

```
(Customer basicNew instVarAt: 1 put: 'roger'; instVarAt: 2 put:
'1234567'; instVarAt: 3 put: 37; yourself)
```

storeString Nested Object Example

Saving

```
| out customers dataFile |
dataFile := 'dataFile.txt' asFilename.
out := dataFile writeStream.
customers := (OrderedCollection new)
   add: (Customer name: 'roger' phone: '1234567');
   add: (Customer name: 'pete' phone: '1111111');
   add: (Customer name: 'jose' phone: '2222222');
   yourself.
out
   nextPutAll: customers storeString;
   close
```

Restoring

```
|  contents customers dataFile |
dataFile := 'dataFile.txt' asFilename.
contents := dataFile contentsOfEntireFile.
customers :=Compiler evaluate: contents.
```

dataFile.txt Contents

((Core.OrderedCollection new) add: (Customer basicNew instVarAt: 1 put: 'roger'; instVarAt: 2 put: '1234567'; instVarAt: 3 put: 39; yourself); add: (Customer basicNew instVarAt: 1 put: 'pete'; instVarAt: 2 put: '1111111'; instVarAt: 3 put: 40; yourself); add: (Customer basicNew instVarAt: 1 put: 'jose'; instVarAt: 2 put: '2222222'; instVarAt: 3 put: 41; yourself); yourself)

BOSS – Binary Object Storage System

Stores objects in binary

Handles nested objects

Handles circular references of objects

Very sensitive to changes in Class

You must provide code to convert between old and new format

Need to load the BOSS parcel

In the Parcel Manager it is located in Application development

See Chapter 21 Binary Object Files in VisualWorks Developer's Guide

Storing

```
| bos customers dataFile |
dataFile := 'dataFile.txt' asFilename.
bos  := BinaryObjectStorage onNew: dataFile writeStream.
customers := (OrderedCollection new)
   add: (Customer name: 'roger' phone: '1234567');
   add: (Customer name: 'pete' phone: '1111111');
   add: (Customer name: 'jose' phone: '2222222');
   yourself.
[bos nextPutAll: customers] ensure: [bos close]
```

Restoring

```
|  customers dataFile |
dataFile := 'dataFile.txt' asFilename.
bos  := BinaryObjectStorage onOldNoScan: dataFile readStream.
[customers := bos contents] ensure: [bos close].
```

Searching the file

```
|  nextCustomer dataFile |
dataFile := 'dataFile.txt' asFilename.
bos  := BinaryObjectStorage onOldNoScan: dataFile readStream.
[[bos atEnd]
   whileFalse:
      [nextCustomer := bos next.
      nextCustomer name = 'pete' ifTrue:[^nextCustomer]]]
ensure: [bos close]
```

XML Configuration Files

Download from main course page

Stores objects using XML

Handles nested objects

Handles circular references of objects

Very tolerant of changes in class of stored objects

Uses name of instance variable to restore object, so can

- Add instance variable
- Remove instance variables
- Reorder instance variables

Note the full name of the class is
Tools.XMLConfigFileSupport.XMLConfigFile

Saving Multiple Objects

```
| xmlFile |
xmlFile :=XMLConfigFile filename: 'dataFile.txt'.
xmlFile
   saveObject: (Customer name: 'roger' phone: '1234567');
   saveObject: (Customer name: 'pete' phone: '1111111');
   saveObject: (Customer name: 'jose' phone: '2222222');
   saveConfiguration.
```

Restoring

```
| xmlFile objects customer1 customer2  customer3 |
xmlFile :=XMLConfigFile filename: 'dataFile.txt'.
objects :=xmlFile loadConfiguration.
customer1 := objects first.
customer2 := objects at: 2.
customer3 := objects at: 3
```

Saving a Collection

```
| xmlFile |
xmlFile :=XMLConfigFile filename: 'dataFile.txt'.
customers := (OrderedCollection new)
   add: (Customer name: 'roger' phone: '1234567');
   add: (Customer name: 'pete' phone: '1111111');
   add: (Customer name: 'jose' phone: '2222222');
   yourself.
xmlFile
   saveObject: customers;
   saveConfiguration.
```

Restoring

```
| xmlFile |
xmlFile :=XMLConfigFile filename: 'dataFile.txt'.
objects :=xmlFile loadConfiguration.
customers := objects first.
```

OmniBase – Object Database

Handles nested objects

Handles circular references of objects

Very tolerant of changes in Classes of stored objects.

Can search for an object

Edit and save one object with having to resave all objects

Handles concurrent access

Supports transactions with rollback

Do not need separate database process

Download educational version at:
http://www.gorisek.com/homepage/index.html

Creating

```
| database |
database :=OmniBase createOn: 'data'.
database close.
```

You can create and use the database before closing
Creating is done once, so I do it here to get to over with

Simple adding to Database

```
| database |
database :=OmniBase openOn: 'data'.
[ OmniBase root
   at: 'test'
   put: (OrderedCollection newPersistent
      add: 'string object';
      add: 1;
      add: Date today;
      yourself) ] evaluateAndCommitIn: db newTransaction.
```

Accessing

```
| database date |
database :=OmniBase openOn: 'data'.
[ date:= (OmniBase root at: 'test') last ] evaluateIn: database newTransaction.
^date
```

Modifying

Modifying the collect at test

```
| database test |
database :=OmniBase openOn: 'data'.
[ test:=  (OmniBase root at: 'test').
test add: 'Hi Mom'.
test markDirty] evaluateAndCommitIn: database newTransaction.
database close.
```

Don’t have to resave all objects in the file

Object needs to be persistent and marked dirty

Customer Example
Adding the Customers

```
| database customers |
database :=OmniBase openOn: 'data'.
customers := (OrderedCollection new)
   add: (Customer name: 'roger' phone: '1234567');
   add: (Customer name: 'pete' phone: '1111111');
   add: (Customer name: 'jose' phone: '2222222');
   yourself.
[customers makePersistent.
customers do: [:each | each makePersistent].
OmniBase root
   at: 'customers'
   put: customers ] evaluateAndCommitIn: database newTransaction.
database close.
```

Accessing and Modifying a Customer

```
| database customers toChange |
database :=OmniBase openOn: 'data'.
[customers :=  OmniBase root at: 'customers'.
toChange := customers detect: [:each | each name = 'pete'].
toChange phone: '9999999'.
toChange markDirty]
    evaluateAndCommitIn: database newTransaction.
database close.
```