CS 535 Object-Oriented Programming & Design
Fall Semester, 2001
Doc 16 Scripts and Data Files

References

VisualWorks Application Developer Guide, doc/vwadg.pdf in the VisualWorks installation. Chapter 20 pages 446-448

VisualWorks source code

Smalltalk as Script

Compiler evaluate: aString

Compiles and executes the Smalltalk code in aString

Examples

Evaluate these in a workspace

Compiler evaluate: ' 1 + 2'.

Compiler evaluate: 'Transcript show: (1 + 2) printString'

User strings via Dialog

```
| userScript |
userScript := Dialog
        request: 'Write a Smalltalk expression'
        initialAnswer: '1 + 2'.
Compiler evaluate: userScript.
```

Evaluating Blocks

Blocks can be created and executed

```
| script |
script := Compiler evaluate:  '[1 + 2]'.
script value
```

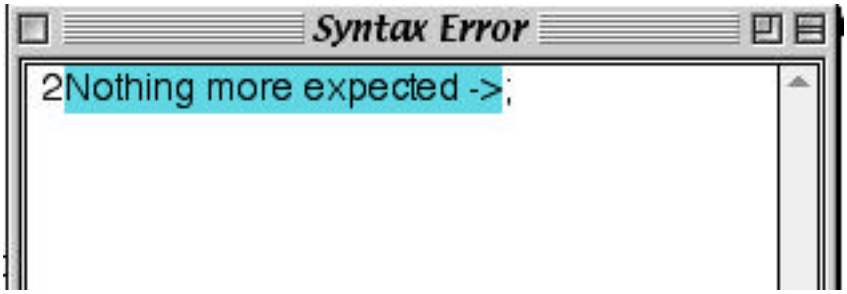
Embedding code in a Block

```
| userScript compiledCode |
userScript := Dialog
            request: 'Write a Smalltalk expression'
            initialAnswer: '1 + 2'.
compiledCode := Compiler evaluate: '[' , userScript , ']'.
compiledCode value
```

Errors in the Evaluated String
Syntax error

Syntax errors open a text window on the error

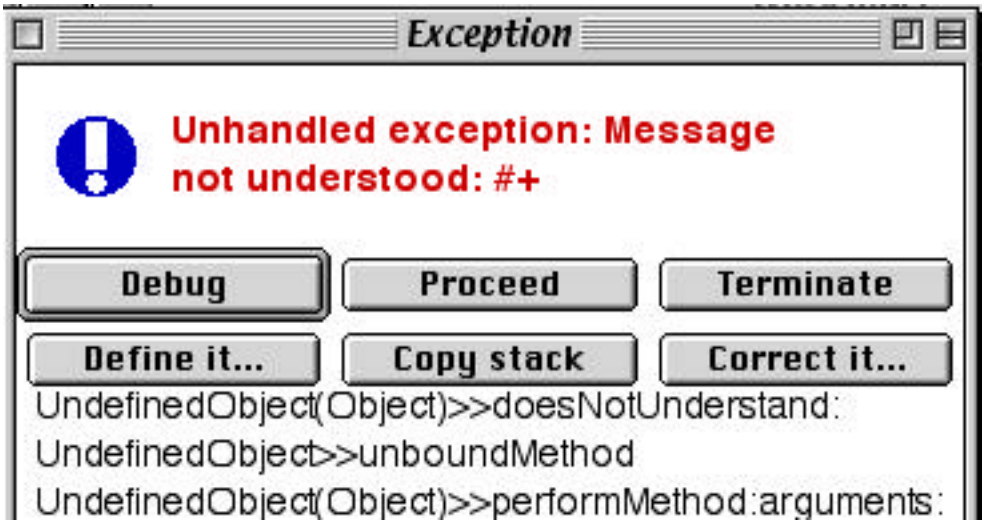
Compiler evaluate: '2;'



Runtime Errors

Runtime Errors open a debugger

Compiler evaluate: 'bar + 3'



Handling Errors

If the default action is not correct for your situation then

on:do: can be used to catch the errors

```
[Compiler evaluate: '2;']
   on: Notification
   do: [:error | error handling code]

[Compiler evaluate: 'foo + 2']
   on: Notification
   do: [:error | error handling code]
```

External Variables in the Script

There are several easy ways to provide scripts access to existing variables in a program

- Use block variables
- Use evaluate:for:logged:

Using Blocks

```
| scriptString scriptBlockString scriptBlock |
scriptString := 'price > 10
            ifTrue:[ "expensive"]
            ifFalse:[ "cheap"]'.
scriptBlockString := '[:price | ' , scriptString , ' ]'.
scriptBlock := Compiler evaluate: scriptBlockString.
scriptBlock value: 12
```

In the string literal assigned to scriptString, contains code that is to have a string literal ('expensive'), the inner string literals need to be quoted with two single quotes. If the script is not created from a string literal the double single quotes are not needed.

evaluate:for:logged:

Evaluates code as if it were part of an object

Used primarily for tools like debugger

Violates information hiding should be avoided

Can be used to add methods to objects

Example

```
Smalltalk.CS535 defineClass: #SampleClass
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: 'name '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Course-Examples'
```

Instance method

```
name: aString
   name := aString
```

Script

```
| dataObject |
dataObject := SampleClass new.
dataObject name: 'sam'.
script := ' "the name is: " , name '.
Compiler
   evaluate: script
   for: dataObject
   logged: false
```

Since the script is run as part of the object dataObject it can access instance variable 'name'

If the logged: parameter is true the execution of the code is recorded in the change file

Undefined Variables

Evaluate the following twice

Compiler evaluate: 'foobar'

The first time you will see in the transcript:

UndefinedObject #DoIt - foobar is undeclared

The second time this message will not appear.

What is going on?

When running code runs across an undefined variable it is stored in Undeclared. So the second time foobar already exists in Undeclared.

Just say no to Undeclared

To see if you have any undeclared variables run:

```
Undeclared inspect
```

The inspector allows you to check for existing references to undeclared variables

The inspector also allows you to remove variables from Undeclared that do not have any references.

Or you can run:

```
Undeclared purgeUnusedBindings
```

To see how many undeclared variables are in your image run:

```
Undeclared size
```

Some Simple Data Storage
Delimited Files

How to read data delineated by a special character

Example

Let the file 'names' contain the following text:

roger;tom;sally;rhea;ritu;smita

The following will read each name and put it into an ordered collection

```
| dataFile stream names readingBlock |
dataFile := 'names' asFilename.
stream := dataFile readStream.
separator := $;.
names := OrderedCollection new.
readingBlock := [
   [stream atEnd] whileFalse: [
      names add: (stream upTo: separator)]].
readingBlock valueNowOrOnUnwindDo: [stream close].
^names
```

ASCII Serialized Objects

Smalltalk will serialize objects

The serialize object
- Is stored in a string
- Can be used to recreate the object
- Can not handle object with circular references
- Limited ability to handle changes in a class

Serializing an Object

Send storeOn: to the object

storeOn: requires a writeStream as an argument

anObject storeOn: aWriteStream

Simple Example

```
| dataStream serializedFive  |
dataStream := WriteStream on: String new.
5 storeOn: dataStream.
serializedFive := dataStream contents
```

Deserializing an Object

Use the one of the following class methods of Object

readFrom: aStream
readFromString: aString

Simple Example

```
| dataStream serializedFive five |
dataStream := WriteStream on: String new.
5 storeOn: dataStream.
serializedFive := dataStream contents.
five := Object readFromString: serializedFive
```

More Complex Example
Class Used in Example

```
Smalltalk.CS535 defineClass: #Student
    superclass: #{Core.Object}
    indexedType: #none
    private: false
    instanceVariableNames: 'name phone graduationDate '
    classInstanceVariableNames: ''
    imports: ''
    category: 'Course-Examples'
```

CS535.Student class methods

```
name: aString phone: aPhoneNumberString graduation: aDate
    ^super new
        setName: aString
        setPhone: aPhoneNumberString
        setGraduation: aDate
```

CS535.Student instance methods

```
printOn: aStream
    aStream
        nextPutAll: 'Student: ';
        nextPutAll: name;
        nextPutAll: ', ' ;
        nextPutAll: phone;
        nextPutAll: ', ';
        nextPutAll: graduationDate printString

setName: aNameString setPhone: aPhoneString setGraduation: aDate
    name := aNameString.
    phone := aPhoneString.
    graduationDate := aDate
```

Example Continued

In this example we serialize a collection of different objects

```
| jeff data dataStream serializedObject deserializedObject |

jeff := Student
        name: 'jeff'
        phone: '543-3445'
        graduation: Date dateAndTimeNow.

data := OrderedCollection new
        add: 1;
        add: 'cat';
        add: jeff;
        yourself.
dataStream := WriteStream on: String new.
data storeOn: dataStream.
serializedObject := dataStream contents.
deserializedObject := Object readFromString: serializedObject.
deserializedObject at: 3
```

Result

deserializedObject at: 3 is a Student object.

serializedObject is '((Core.OrderedCollection new) add: 1; add: "cat"; add: (CS535.Student basicNew instVarAt: 1 put: "jeff"; instVarAt: 2 put: "543-3445"; instVarAt: 3 put: ((Core.Array new: 2) at: 1 put: (Core.Date readFromString: "11/7/2001"); at: 2 put: (Core.Time readFromString: "9:59:00 pm"); yourself); yourself); yourself)'

Changing the Shape of the Class

Changes you can make to the class and still read old objects

- Add/Delete methods
- Add instance variables at the end of the instance var list

Changes which make old objects unreadable

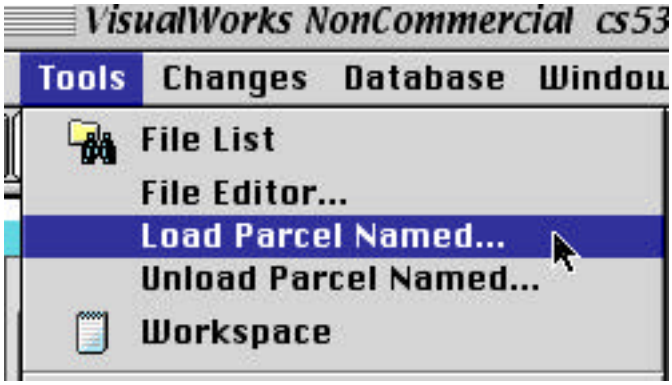
- Changing the name of a class
- Deleting instance variables
- Changing the order of instance variables
- Adding instance variables other than at the end of the list

Binary Object Storage System - BOSS

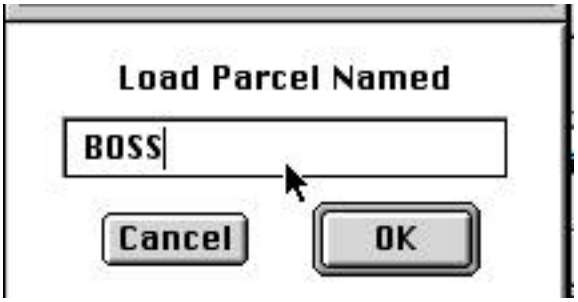
BOSS
- Stores objects in binary
- Can handle circular references
- Must load BOSS parcel before using

Loading BOSS Parcel

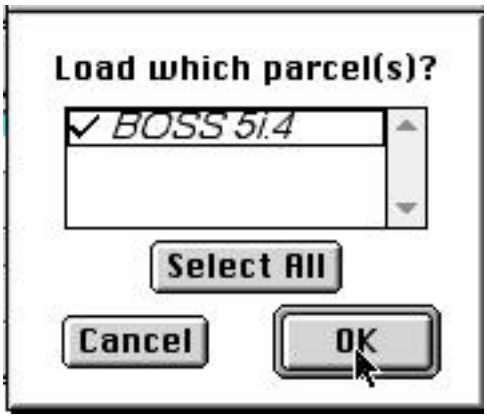
In the Tools menu in the launcher select the "Load Parcel Named…" menu item.



You will be asked which parcel to load, enter BOSS.



After a search of all parcels you will be asked if BOSS 5i4 is the one you wish to load.



Select it and click on yes.

Simple Examples of Using BOSS

```
| binaryFile binaryWriter binaryReader five |
binaryFile := 'binaryFile' asFilename.
binaryWriter :=
    BinaryObjectStorage onNew: (binaryFile writeStream).
binaryWriter nextPut: 5.
binaryWriter close.

binaryReader :=
    BinaryObjectStorage onOld: (binaryFile readStream).
five := binaryReader next.
binaryReader close.
^five
```

BinaryObjectStorage Methods

Instance Creation (Class Methods)

onNew: aWriteStream
Uses aWriteStream to write binary objects
Overwrites current contents of aWriteStream

onOld: aReadStream
Reads binary objects from aReadStream

Writing (Instance methods)

nextPut: anObject
Writes anObject in binary format onto the stream

nextPutAll: aCollection
Writes all elements of aCollection to the stream in binary format

Reading

next
Return the next object from the stream
Object is deserialized

skipNext
Skip the next object in the stream

contents
Return all objects in the stream from current location

atEnd
Return true if at the end of the stream

Student Example

```
jeff := Student
        name: 'jeff'
        phone: '543-3445'
        graduation: Date dateAndTimeNow.

binaryFile := 'binaryFile' asFilename.
binaryWriter :=
   BinaryObjectStorage onNew: (binaryFile writeStream).
binaryWriter nextPut: jeff.
binaryWriter close.

binaryWriter :=
   BinaryObjectStorage onOld: (binaryFile readStream).
recovered := binaryWriter next.
binaryWriter close.
recovered
```

Changing the Shape of a Class

Changes you can make to the class and still read old objects with no extra work

- Add/Delete methods

Changes you can make to the class and still read old objects with extra work

- Modify the list of instance variables

Changes which make old objects unreadable

- Changing the name of a class

How to handle changing the Instance Variable list Example with Student

First start with the instance variable list as:

```
instanceVariableNames: 'name phone graduationDate '
```

Now save an object:

```
jeff := Student
        name: 'jeff'
        phone: '543-3445'
        graduation: Date dateAndTimeNow.
binaryFile := 'binaryFile' asFilename.
binaryWriter :=
   BinaryObjectStorage onNew: (binaryFile writeStream).
binaryWriter nextPut: jeff.
binaryWriter close.
```

Now change the instance variable list to:

```
instanceVariableNames: 'name phone graduationDate foo '
```

If you try to read the object you will get an error

```
binaryWriter :=
   BinaryObjectStorage onOld: (binaryFile readStream).
recovered := binaryWriter next.
binaryWriter close.
recovered
```

Add the Class Method to Student

The following class method in Student will convert the old instance variable list to the new one.

After adding this to the class you will be able to read objects in the old format and the new format

```
binaryReaderBlockForVersion: oldVersion format: oldFormat
  oldVersion isNil
    ifTrue:
      [| inst |
      inst := self basicNew.
      ^[:oldInstanceVarList | | newList |
      newList := Array new: oldInstanceVarList size+1.
      newList at: 1 put: (oldInstanceVarList at: 1).
      newList at: 2 put: (oldInstanceVarList at: 2).
      newList at: 3 put: (oldInstanceVarList at: 3).
      oldInstanceVarList become: newList.
      oldInstanceVarList changeClassToThatOf: inst]].
  ^super
    binaryReaderBlockForVersion: oldVersion
    format: oldFormat
```

How does this Work

Each class has a format number.

Student format

Returns the format of the current version of the Student class

When you serialize an object with BOSS the classes format number is stored

When reading a serialize object BOSS checks the current and old format numbers

If the formats match the default process is used to deserialize the object

If the formats do not match then

```
binaryReaderBlockForVersion:format:
```

is called on the objects class

This method returns a block to perform the conversion

The Conversion Block

OldInstanceVarList is the list of the instance variable objects from the old object

```
^[:oldInstanceVarList | | newList |
```

Now put the list in the correct order for the new object.
This is the section of the code to change for your classes.
The rest of the code remains the same in the block

```
newList := Array new: oldInstanceVarList size+1.
newList at: 1 put: (oldInstanceVarList at: 1).
newList at: 2 put: (oldInstanceVarList at: 2).
newList at: 3 put: (oldInstanceVarList at: 3).
```

Now some magic. Transform the list into a student object

```
oldInstanceVarList become: newList.
oldInstanceVarList changeClassToThatOf: inst]].
```

Multiple Old Versions

If you have multiple old versions of objects that are stored you need to have a different convert block for each one.

The format: parameter is the format of the object you are converting

Use that to select the correct convert block

binaryReaderBlockForVersion: oldVersion format: oldFormat