ROX - REXX Object eXtensions

Patrick Mueller
IBM Software Solutions Division
Cary, North Carolina
pmuellr@vnet.ibm.com

April 27, 1994

Contents

1 Introduction 3

1.1 What is **ROX**? 3

1.2 What **ROX** isn't 3

1.3 Object creation 3

1.4 Method invocation 4

1.5 Variables 4

1.6 Class Inheritance 5

1.7 self and super 6

2 Installation and Removal 6

3 Function Reference 6

3.1 Function Package Functions 7

3.2 Class Definition Functions 8

3.3 Object Lifecycle Functions 9

4 Format of .rox Files 10

5 C Programming Interface 11

6 Utilities Provided 13

7 Classes and Testers Provided 14

8 History 16

A Sample .rox file 18

B Sample ROX class usage 21

C Output of previous samples 23

1 Introduction

1.1 What is ROX?

ROX is a function package for REXX that allows for object oriented (OO) programming in REXX. You should have some basic familiarity with OO programming before diving into **ROX**.

ROX allows classes to be defined. The classes have a number of features.

- they may inherit from other classes
- they specify variables that will be maintained for each object created of the given class
- they specify methods written as REXX code

Classes are defined in files with an extension of .rox. See *Format of .rox Files* on page 10 for the format of the .rox files.

1.2 What ROX isn't

ROX is not a new language - it is simply a function package that can be used from the OS/2 [1] REXX language providing some OO capabilities.

ROX provides NO facilities for interacting with other object oriented systems such as SOM or Smalltalk.

ROX has no distributed (cross-process, or cross-platform) capabilities.

1.3 Object creation

Objects are created and destroyed with the RoxCreate() and RoxDestroy() functions, described in *Object Lifecycle Functions* on page 9. The RoxCreate() function takes the name of the class to create the object from, and any number of additional parameters to initialize the object. The RoxCreate() function returns an object reference. This object reference is a regular REXX string, with a particular value which the **ROX** functions can use to dereference the object. This object reference is used as the first parameter for method invocation.

When an object is created, the *init* method for the class is invoked. Likewise, when an object is destroyed, the *deinit* method for the class is invoked. If the

[1] OS/2 is a trademark of International Business Machines Corporation.

init or *deinit* methods are not defined in the class, they will be searched for in inherited classes.

1.4 Method invocation

Once an object is created, you can send messages to it. This is also commonly referred to as invoking a method. The message is the name of the method, along with parameters that the method should be passed. To invoke a method, use REXX function call invocation. The name of the function is the name of the method, prefixed by ".". The first parameter to the function is an object reference, and any other method specific parameters can be passed as well.

It's time for a short example. In this example, we create an object of class *dog*, passing an additional parameter on the RoxCreate() function which is the name of the dog. The *init* method of the dog class will be invoked, passing the name as the first parameter. Next, the *bark* method of the dog class is invoked, in both function invocation formats available in REXX. Both invocations do the same thing.

```
jackson = RoxCreate("dog","Jackson")
call .bark jackson
g = .bark(jackson)
```

As noted before, during object creation, the *init* message is sent to the object. In order to allow an object's inherited classes to initialize themselves, the init and deinit methods may be invoked as functions whose names are a class name and the method name, concatenated together, with a "." in between them. For example, assuming the dog class inherits from the animal class, the dog init method can call the animal init method by invoking the function *animal.init*.

1.5 Variables

Classes specify both the methods that can be used on an object and the state variables associated with the object. The variables are plain old REXX variables, whose values are available to methods of the classes. The variables are non-stem variables, such as *name*, *size*, etc,. Stem variables are handled via per-instance variables (see below). Any number of variables may be associated with a class (and thus an object).

Per-instance variables are variables that can be added to an object in an ad hoc manner. For instance, one object of class X might have object variables *x.0*, *x.1*, *x.2*, where another object of class might have object variables *x.0*, *x.1*.

Per-instance variables are added to an object with the function RoxAddVar(). Per-instance variables are the only way to store stem variables with an object - stem variables can NOT be defined with a class.

When a method is invoked, the variables of the object will be available to the REXX code of the method. If the value of a variable changes in the method, the changed value will be saved with the object.

It's time for another example. In this example, we'll describe a simple class in the format acceptable for .rox files. The class is dog, and it has two variables - name and breed. They will be used to hold the name of the dog, and the dog's breed. We also define three methods - name, breed and describe. The name and breed functions either set or return the current value of the variable, depending on whether any parameters are passed to them. The describe method prints a line describing the dog.

```
:class dog
:vars name breed
:method name
   if (arg() = 1) then
      name = arg(1)
   return name
:method breed
   if (arg() = 1) then
      breed = arg(1)
   return breed
:method describe
   say "The dog's name is" name".  It is a" breed"."
   return ""
```

Below is some REXX code that uses the class dog. The result of the method describe invocation is that the line "The dog's name is Jackson. It is a Chocolate Labrador Retriever." will be printed on the screen.

```
Jackson = RoxCreate("dog")
x = .name(Jackson,"Jackson")
x = .breed(Jackson,"Chocolate Labrador Retriever")
x = .describe(Jackson)
```

1.6 Class Inheritance

Classes can inherit other classes in their definitions. This technique expands the variables and methods available to the class to the set of variables and

methods defined in any inherited classes. A class can inherit from more than one class. **ROX** has no scoping facility, so if classes are inherited that have the same method, the method will be available in the derived class (the one that inherits the other classes), but the actual method invoked is undefined. One of the methods will be invoked, but it's not possible to determine which one.

1.7 self and super

Two special variables are available to all methods. They are *self* and *super*. self refers to the receiver of the method (the object which the methods was invoked on). super also refers to the receiver of the method, however, if super is used as the receiver of a method, the method to be invoked will be searched for starting at the inherited classes of the class of the method currently running. self and super are similiar to the self and super variables in Smalltalk.

2 Installation and Removal

The **ROX** REXX function package is contained in the file rox.dll. This file needs to be placed in a directory along your LIBPATH. To get access to the functions in the **ROX** function package, execute the following REXX code:

```
rc = RxFuncAdd("RoxLoadFuncs","rox","RoxLoadFuncs")
rc = RoxLoadFuncs()
```

To unload the DLL, you should first call the RoxDropFuncs() function, then exit all CMD.EXE shells. After exiting all the command shells, the DLL will be dropped by OS/2 and can be deleted or replaced.

3 Function Reference

The functions provided by the **ROX** function package fall into the following categories:

- function package functions
- class definition functions
- object lifecycle functions

3.1 Function Package Functions

The following functions load, drop and query the version number of the **ROX** function package.

RoxLoadFuncs() - load the ROX function package

```
rc = RoxLoadFuncs()
```

Loads all the functions in the **ROX** package.

If ANY parameters are passed to this function, it will bypass the program, author, and copyright information normally displayed. All parameters are ignored (except to determine whether or not to bypass displaying the information).

RoxDropFuncs() - drop the ROX function package

```
rc = RoxDropFuncs()
```

Drops all the functions in the **ROX** package.

RoxVersion() - returns version number of the ROX function package

```
vers = RoxVersion()
```

Returns the current version number of the **ROX** package.

RoxStats() - generates execution profile info

```
rc = RoxStats(<parm>)
```

This function can be used to generate profile information on stderr. A parameter should be passed to start profile information, no parameter should be passed to stop profile information. For example:

```
rc = RoxStats("") /* start profiling */
rc = RoxStats()   /* end profiling */
```

The profile information can be analyzed with the RoxProf.cmd utility.

Returns "".

3.2 Class Definition Functions

The following functions are used to add class definitions to the system. Generally you will only need to use RoxLoad() and RoxQueryClassLoaded(). The other functions are used by RoxLoad() to to load .rox files.

RoxLoad() - load class definitions in a .rox file

```
rc = RoxLoad(roxFileName)
```

This function loads the named file as a class definition. See the section of .rox file definitions for the layout of the file.

This function is implemented as a REXX .cmd file.

RoxQueryClassLoaded() - query whether class is loaded

```
bool = RoxQueryClassLoaded(className)
```

Returns 1 if the class named className is available in the system. Returns 0 otherwise.

RoxAddClass() - add a class

```
rc = RoxAddClass(className)
```

This function adds the named class to the system.

RoxClassAddInherit() - add an inherited class to a class definition

```
rc = RoxClassAddInherit(className,inheritedClassName)
```

This function specifies that the class named className should inherit from the class named inheritedClassName.

RoxClassAddMethod() - add a method to a class definition

```
rc = RoxClassAddMethod(className,methodName,methodCode)
```

This function adds the named method, with the REXX code for the method to the named class.

RoxClassAddMethodDll() - add a method (in a DLL) to a class definition

```
rc = RoxClassAddMethod(className,methodName,dllName,entryPoint)
```

This function loads the dll, gets the address of the function given with the name entryPoint, and adds this to the named class.

RoxClassAddVar() - add an instance variable to a class definition

```
rc = RoxClassAddVar(className,varName)
```

This function adds the named instance variable to the named class.

3.3 Object Lifecycle Functions

RoxCreate() - create an object

```
object = RoxCreate(className<,p1<,p2< . . . >>>)
```

This function creates an object of the class named className. Any number of parameters, specific to the class, can be passed.

RoxDestroy() - destroy an object

```
rc = RoxDestroy(object)
```

This function destroys an object.

RoxSend() - send a message to an object

```
result = RoxSend(messageName,object,<,p1<,p2<. . . >>>)
```

This function sends the named message to the object specified. Any number of parameters, specific to the message and class, can be passed.

RoxSendThread() - send a message to an object

```
result = RoxSendThread(messageName,object,<,p1<,p2<. . . >>>)
```

Same as RoxSend(), but starts a new thread to process the message. No useful return value is returned.

RoxClass() - return class of given object

```
class = RoxClass(object)
```

This function returns the name of the class of the object.

RoxAddVar() - add a variable to an object

```
result = RoxAddVar(object,varName)
```

This function will the named variable to the set of instance variables associated with the object. Be careful not to add extra blanks to varName when passing it in. The characters in the variable name, up to the first ".", will be uppercased, to conform with REXX variable conventions. The remainder of the variable name is left as is.

4 Format of .rox Files

Classes are defined in files with an extension of .rox. A .rox file may contain one or more class definitions.

Classes defined in .rox files may be loaded by using the RoxLoad function (see *Utilities Provided* on 13).

The format of .rox files is a tagged file. The character ':' in column one indicates a tag. The rest of the line after the ':' indicates the type of tag.

The characters ':*', when located in column one, indicate a comment.

The following tags may be used in a .rox file:

```
:include <file>
```

This tag indicates that the file specified in the tag should be loaded as a .rox file. Useful for including inherited class definitions from separate files.

:class

This tag indicates the start of a new class definitions. Any :inherits, :vars, and :method tags following this tag, up to the end of the current .rox file, are associated with this class.

:inherits <class> <class> ...

This tag indicates the classes that should be inherited from. More than one class may be specified. This tag may be used more than once within a class definition.

:vars <var> <var> ...

This tag indicates the variables associated with the class. More than one variable may be specified. This tag may be used more than once within a class definition. Note stem variables may NOT be used. Use RoxAddVar() to add stem variables to an object.

:method <methodName>

This tag indicates that the code for the method named <methodName> follows. The code for the method ends at the next tag (including :* comment), or end of file.

5 C Programming Interface

ROX methods can be implemented in compiled languages, such as C, via a DLL. The function RoxClassAddMethodDll() adds a method to a class that points to a function in a DLL. The function in the DLL must have the following signature:

```
/*-------------------------------------------------
 * typedef for function that handles method invocation
 *-------------------------------------------------*/
typedef ULONG APIENTRY RoxMethodHandlerType(
   void      *object,
   PUCHAR     name,
   ULONG      argc,
   PRXSTRING  argv,
   PRXSTRING  retString
   );
```

The parameters passed to the method are:

object a pointer to the **ROX** object receiver

name the name of the method

argc the number of arguments passed to the method

argv array of RXSTRINGs that make up the parameters

retString pointer to the return value

Most of these parameters will be familiar to those of you who have written external functions for REXX in C. The only new one is the object parameter. It can be used in the following functions:

```
ULONG RoxVariableGet(
   void      *object,
   PRXSTRING  name,
   PRXSTRING  value
   );

ULONG RoxVariableSet(
   void      *object,
   PRXSTRING  name,
   PRXSTRING  value
   );
```

The functions above are used to query and set variables for an object. The functions return 0 when successful, !0 when not successful. The data pointed to by the value parameter returned from RoxVariableGet() must not be modified.

A sample of a compiled class is provided in roxsem.c.

A DLL can provide a self-loading function named RoxDllEntryPoint, with the following function signature.

```
ULONG APIENTRY RoxDllEntryPoint(
   ULONG init
   )
```

Currently the init parameter is ignored.

This function gets called when the REXX function RoxLoadDLL() is invoked. This function takes the name of the DLL (usually sans ".DLL", although you may specify an absolute path, including the ".DLL" suffix) and calls the RoxDllEntryPoint function.

This function in the DLL can call any of the functions defined in the **ROX** function package through their C bindings. The call is made as if the call was being made to a REXX external function. For example, to call RoxAddClass(), you invoke it in C as:

```
RXSTRING parm, result;

parm.strptr = "myClassName";
parm.strlength = strlen(parm.strptr);

RoxAddClass(NULL,1,&parm,NULL,&result);
```

Note that the function name and queue name (first and fourth parameters) may be passed as NULL.

Be careful how the return value is freed. See the sample roxsem.c code for examples.

Two *platform* independent functions are provided to allocate and free memory. The functions are:

```
void APIENTRY *osMalloc(
   int size
   );

void APIENTRY osFree(
   void *ptr
   );
```

The include file "roxapi.h" prototypes these functions, and the library "rox.lib" contains them.

6 Utilities Provided

The following utilities are provided with **ROX**:

RoxLoad.cmd

This program can only be used as a REXX function. It can not be called from the OS/2 command line. One parameter must be passed to the function - the name of a .rox file to load. The file will be searched for in the current directory, and then the directories specified in the ROXPATH environment variable.

RoxInfo.cmd

Prints a short reference of the class definitions in .rox files. Multiple .rox files may be passed as parameters, and wildcards may be specified. For every class in the .rox file, the following information will be provided:

- Classes inherited by the class. These classes will be listed in an indentation style which indicates the *tree* of class inheritance.
- Variables defined and inherited by the class. Inherited variables are marked with a prefix of "*".
- Methods defined and inherited by the class. Inherited methods are marked with a prefix of "*".

RoxProf.cmd

Analyzes the profile information generated by RoxStats(). Use "RoxProf ?" for help.

7 Classes and Testers Provided

list.rox

Implements a simple list class. The program testcoll.cmd tests this class, by passing it a parameter of "list". The list class inherits the collection class in collect.rox.

wordlist.rox

Implements a simple list class, similiar to the list class. The difference is that the list class can contain arbitrary strings, whereas the wordlist class can only contain strings with no blanks in them. The program testcoll.cmd tests this class, by passing it a parameter of "wordlist". The wordlist class inherits the collection class in collect.rox.

set.rox

Implements a simple set class. The program testcoll.cmd tests this class, by passing it a parameter of "set". The set class inherits the collection class in collect.rox.

collect.rox

Implements a simple collection class, that can be inherited by other, more specific collection classes, and will provide additional capabilities.

sessions.rox

This file implements some of the classes from Roger Sessions' book on OO with C and C++ (reference included in the .rox file). The program sessions.cmd tests the classes.

spinner.rox

This class implements a character spinner, which can be used as a progress indicator. Also uses roxsem.dll. This class is tested with testspin.cmd. The demo shows code testing a collection along with a spinner running independently in another thread.

testthrd.cmd

This program tests the thread capabilities of **ROX**.

cmdline.cmd

This program uses cmdline.rox as a command line reader with history. Use the up and down arrows to cycle through previous lines entered.

roxsocks.cmd & roxsockc.cmd

These programs demonstrate tcp/ip server and client programs X socket class (in socket.rox).

8 History

04/14/94 - version 1.8

- fixed problem with super calls
- removed RoxVarSynch()
- added RoxAddVar() and per-instance variables
- cut execution time in half with new memory management scheme
- added RoxStats() and RoxProf.cmd

01/06/94 - version 1.7

- minor documentation cleanup
- cleanup of internal structure of **ROX** - no external changes - most notably, no performance changes

10/22/93 - version 1.6

- fixed infinite loop when no variables set in an init method - ObjectSaveState/RoxStemSynch ping-ponged. Reported by Zvi Weiss as a problem when a syntax error occurred in an init method.
- changed compiled classes/methods stuff to have just one type of class, and either compiled or REXX macros. Compiled macros added with RoxClassAddMethodCompiled().

09/14/93 - version 1.5

- more thread reentrancy fixes
- added compiled class capability

08/31/93 - version 1.4

- added RoxSendThread() function
- first attempt at making everything thread reentrant (still some more to go).

08/27/93 - version 1.3

- print error when invalid object reference is passed to a method
- added exception handling, to try to catch method invocation on objects which are no longer alive

08/24/93 - version 1.2

- fixed problems with re-adding and re-registering classes and methods

08/22/93 - version 1.1

- fixed super behaviour
- added multiple inheritance capability
- added class-specific init and deinit methods
- added RoxStemSynch() - requires user notify the system when stem variables are added or dropped as instance variables
- added RoxInfo.cmd utility
- documentation turned into .inf file and enhanced

08/18/93 - version 1.0

- initial release

A Sample .rox file

Below is a the 'sessions.rox' file, which contains class defintions inspired by Roger Sessions' book on class development.

```
:*-----------------------------------------------------------------
:* REXX Object eXtensions :
:* classes described in Roger Sessions' book "Class Construction in
:* C and C++", Prentice-Hall, ISBN 0-13-630104-5.
:*-----------------------------------------------------------------

:*=================================================================

:class performer

:vars minSalary

:method setMinimumSalary
   minSalary = arg(1)

   if (0 = datatype(minSalary,"W")) then
      minSalary = 1000

   return self

:method bargain
   say "   I get" minSalary * 2 "dollars a performance."

   return self

:*=================================================================

:class animal

:vars name sound soundTimes

:method init
   name       = arg(1)
   soundTimes = arg(2)
   sound      = arg(3)

   if (name = "") then
      name = "unnamed"

   if (0 = datatype(soundTimes,"W")) then
      soundTimes = 1
```

```
   if (sound = "") then
      sound = "..."

   return

:method says
   say name "says:"

   do i = 1 to soundTimes
      say "   "sound
   end

   return self


:*================================================================

:class dog

:inherits animal performer

:method init
   rc = animal.init(self,arg(1),arg(2),arg(3))
   return

:method scratch
   say "   Ooooh... what an itch."
   return self

:*================================================================

:class littleDog

:inherits dog

:method init
   rc = dog.init(self,arg(1),arg(2),arg(3))
   return

:method trick
   say "   Watch my trick: I can roll over."
   return self

:*================================================================

:class bigDog
```

```
:inherits dog

:method init
   rc = dog.init(self,arg(1),arg(2),arg(3))
   return

:method trick
   say "   Watch my trick: I can fetch the letter carrier."
   return self

:*=============================================================

:class usedCarDealer

:inherits animal

:method init
   rc = animal.init(self,arg(1),arg(2),arg(3))
   return

:method makeSale
   say "   ... and only $500 more if you want the wheels."
   return self
```

B Sample ROX class usage

Below is a the 'sessions.cmd' file, which uses the classes defined in the 'sessions.rox' file.

```
/*----------------------------------------------------------------
 * sessions.cmd :
 *----------------------------------------------------------------
 * 08-21-93 originally by Patrick J. Mueller
 *----------------------------------------------------------------*/

say "testing the Sessions classes"

if RxFuncQuery("RoxLoadFuncs") then
   do
   rc = RxFuncAdd("RoxLoadFuncs","Rox","RoxLoadFuncs")
   rc = RoxLoadFuncs()
   end

rc = time("r")

rc = RoxLoad("sessions.rox")

Frenchie  = RoxCreate("animal",        "Frenchie",  1, "Grrrrrr")
Rover     = RoxCreate("dog",           "Rover",     1, "Woof")
Fifi      = RoxCreate("littleDog",     "Fifi",      2, "bow wow")
Rex       = RoxCreate("bigDog",        "Rex",       4, "BOW WOW")
HonestBob = RoxCreate("usedCarDealer", "HonestBob", 1, "Buy this deal of a car!")

g = .setMinimumSalary(Rex,30)
g = .setMinimumSalary(Fifi,20)

g = .says(Frenchie)
say

g = .says(Rover)
say

g = .says(Fifi)
g = .scratch(Fifi)
g = .trick(Fifi)
g = .bargain(Fifi)
say

g = .says(Rex)
g = .scratch(Rex)
g = .trick(Rex)
```

```
g = .bargain(Rex)
say

g = .says(HonestBob)
g = .makeSale(HonestBob)
```

C Output of previous samples

Below is a the output of running the 'sessions.cmd' file

```
testing the Sessions classes
Frenchie says:
   Grrrrrr

Rover says:
   Woof

Fifi says:
   bow wow
   bow wow
   Ooooh... what an itch.
   Watch my trick: I can roll over.
   I get 40 dollars a performance.

Rex says:
   BOW WOW
   BOW WOW
   BOW WOW
   BOW WOW
   Ooooh... what an itch.
   Watch my trick: I can fetch the letter carrier.
   I get 60 dollars a performance.

HonestBob says:
   Buy this deal of a car!
   ... and only $500 more if you want the wheels.
```