Objeck Programmer's Guide

Randy Hollines
objeck@gmail.com

May 16, 2014

Abstract

An introduction to the Objeck programming language and it's features. This article is intended to introduce programmers and compiler enthusiasts to the unique features and design of the Objeck programming language. Unless otherwise noted, this article covers functionality that's part of v3.2. For additional information please refer to the general, technical and support websites.

Contents

1 Introduction

The Objeck program language is an object-oriented computer language with functional features. The language was designed to be an easy to use general purpose programming system. The Objeck language allows programmers to quickly create solutions by leveraging existing class libraries and APIs. The syntax for the language was designed with symmetry in mind and enforces the notion that there should one intuitive way to do things. Features include:

- Support for object-oriented programming (all data types are treated as objects)
- Functional support (higher-order functions)
- Unicode support with external UTF-8 encoding/decoding
- Cross platform (support for OS X, Linux and Windows)
- Concurrent runtime JIT support
- Multi-threaded memory garbage collector
- Local optimizations including method inlining
- Support for static libraries
- Command line debugger

2 Getting Started

The Objeck distribution consists of a compiler, virtual machine, debugger and library inspection tool. The compiler executable is named `obc`, while the runtime virtual machine (VM) executable is named `obr`. Here is the world famous "Hello World" program written in the Objeck language:

```
class Hello {
  function : Main(args : String[]) ~ Nil {
    "Hello World!"->PrintLine();
  }
}
```

2.1 Compiling Source

The example below compiles the source program `hello.obs` into the target binary file `hello.obe`. The two output file types that the compiler supports are executables and shared libraries. Shared libraries are binary files that contain all of the metadata needed by the compiler to relink them into executables. Both executables and shared libraries contain enough metadata to support runtime introspection. As a naming convention, executables must end in `*.obe` while shared libraries must end in `*.obl`.

Below is an example of compiling the "Hello World" program:

```
obc -src tests\hello.obs -dest hello.obe
```

Here's a more advanced example of linking in two required class libraries to an executable.

```
obc -src examples\xml_parser.obs -lib collect.obl,xml.obl -dest a.obe
```

Additional compiler options are:

Option	*Description*
`-src`	path to source files, delimited by the '`,`' character
`-lib`	path to library files, delimited by the '`,`' character
`-tar`	target output `exe` for executable and `lib` for library; default is `exe`
`-opt`	optimization level `s0`–`s3` with `s3` being the most aggressive; default is `s0`
`-dest`	output file name
`-alt`	compile code that is written using a C-like syntax
`-debug`	if set, produces debug out for use by the interactive debugger (see below)

2.2 Executing

The command-line example below executes the `hello.obe` executable. Note, for executables all required libraries are statically linked in the target output file. When compiling shared libraries, other shared libraries are not linked into the target output library file.

```
obr hello.obe
```

3 The Basics

Now lets introduce the core features of the Objeck programming language.

Let's first start with a list of keywords that exist in language. These words are reserved for the language and may not be used as identifiers.

Keywords				
—	@parent	—	@self	—
and	As	Bool	break	bundle
Byte	Char	class	critical	do
each	else	enum	false	Float
for	from	function	if	Int
interface	label	method	native	New
or	other	Parent	private	public
return	select	static	true	TypeOf
use	virtual	—	while	xor

In Objeck, all data types (excluding higher-order functions) are treated as objects. Basic objects provide supports for boolean, character, byte, integer and decimal types. These basic objects can be used to create more complex user defined objects. The listing below defines the basic objects that are supported in the language:

Type	*Description*
`Char`	2 or 4 byte character
`Char[]`	character array
`Bool`	boolean value
`Bool[]`	boolean array
`Byte`	1–byte integer
`Byte[]`	byte array
`Int`	4–byte integer
`Int[]`	integer array
`Float`	8–byte decimal
`Float[]`	decimal array
`Function`	4–byte integer pair

As mentioned above, basic types are objects and have associated methods for each basic class type. For example:

```
13->Min(3)->PrintLine();
3.89->Sin()->PrintLine();
-22->Abs()->PrintLine();
Float->Pi()->PrintLine();
```

3.1 Alternative Syntax

The syntax of the language is based on UML which has ties with Pascal. Given the popularity of the C-style languages Objeck supports an alternative C-like syntax. In this mode, operators can be defined using a C-style syntax and statements such as `if`, `while`, etc., do not have to end with a semicolon.

In order to use this feature code must be compiled using the `-alt` compiler option.

```
a = 13;
if(13 != a) {
  for(i = 0; i < a; i+=1) {
    i->PrintLine();
  }
}
```

3.2 Variable Declarations

Variables can be declared for all of the basic types (described above), user defined objects and high-order functions. Variables can be declared anywhere in a program and are bound to traditional block scoping rules. Variable assignments can be made during a declaration or at any other point in a program. Variables may be declared as local, instance or class level scope. Class level variables are declared using the `static` keyword. A class that is derived from another class may access it's parents variables if the parent class is declared in one of the source programs. *If a class is derived from a class declared in a shared library then that class cannot access it's parents variables, unless an accessor method is provided.* Local variables can be declared without specifying their data type, such variables are bound to the type defined by their first right-hand side assignment. Three different declaration styles are shown below:

```
a : Int;
b : Int := 13;
c := 7; # implicit type deceleration
```

Types that are not initialized at declaration time are initialized with the following default values:

Type	*Initialization*
`Char`	'\0'
`Byte`	0
`Int`	0
`Float`	0.0
Array	`Nil`
Object	`Nil`
Function	`Nil`

3.3 Expressions

The Objeck language supports various types of expressions. Some of these expressions include mathematical, logical, array and method calls. The preceding sections describes some of the expressions that are supported in Objeck.

3.3.1 Mathematical and Logical Expressions

The following code example demonstrates two ways to printing the number `42`. The first way invokes the `PrintLine()` method for the literal `42`. The second prints the product of a variable and a literal.

```
class Test {
  function : Main() ~ Nil {
    42->PrintLine();
    eiht := 8;
    (eiht * 7)->PrintLine();
  }
}
```

The following mathematical operators are supported in the Objeck language for integers and decimal types:

- addition (`+`)
- subtraction (`-`)
- multiplication (`*`)
- division (`/`)
- modulus – (`%` - for integer values only)

- shift left – (`<<` - for integer values only)
- shift right – (`>>` - for integer values only)

In addition the following assignment operators are supported:

- addition-equals (`+=`)
- subtraction-equals (`-=`)
- multiplication-equals (`*=`)
- division-equals (`/=`)

The following bitwise operators are also supported for integer types:

- and (`and`)
- or (`or`)
- xor (`xor`)

The [`*`, `/`, `%`] operators have a higher precedence than the [`+`, `-`] operators and are naturally enforced by the language. Operators of the same precedence are evaluated from left-to-right. Logical operations are of lower precedence than mathematical operations. All logical operators are of the same precedence and order is determined via left–to–right evaluation. The [`&`, `|`] logical operators use short-circuit logic; meaning that some expressions may not be executed if evaluation criteria is not satisfied.

The following logical operators are supported in the Objeck language:

- unary not (`!`)
- and (`&`)
- or (`|`)
- equal (`=`)
- not–equal (`<>`)
- less–than (`<`)
- greater–than (`>`)
- less–than–equal (`<=`)
- greater–than–equal (`>=`)

3.3.2 Arrays

Objeck supports single and multi-dimensional arrays. Arrays are allocated dynamically from the system heap. The memory that is allocated for arrays is managed automatically by the garbage collector. All of the basic types described above (as well as user defined types) can be allocated as arrays. The code example below shows how a two-dimensional array of type `Int` is allocated and dereferenced.

```
array := Int->New[2,3];
array[0,2] := 13;
array[1,0] := 7;
```

The size of an array can be obtained by calling the array's `Size()` method. The `Size()` method will return the number of elements in a given array. For a multi-dimensional array the `Size()` method returns an array of sizes for each dimension.

The following example allocates an array of `Widget` objects. An object must implment it's `New` method if it's going to be instnatanced.

```
class Widget {
  New() {
  }
}

class MakeWidgets {
  function : Main(args : String[]) ~ Nil {
    widgets := Widget->New[1000];
    each(i : widgets) {
      widgets[i] := Widget->New();
    };
  }
}
```

3.3.3 Characters and Strings

All characters are Unicode encoded and stored internally in the format of the host operating system. On Windows characters are stored internally as UTF-16 (2-byte) values. On Linux and OS X characters are stored internally as UTF-32 (4-byte) values. On all platforms characters are read and written in UTF-8, which is ASCII compatible.

Methods are provided to convert Unicode character arrays to UTF-8 bytes and vice versa. Character array literals are allocated as `String` objects. The `String` class provides support for advanced string operations (see below).

```
string := "Hello World!";
string->Size()->PrintLine();
strings := ["Hello","World!"];
sizes := strings->Size();
sizes[0]->PrintLine();
sizes[1]->PrintLine();
```

3.3.4 Conditional Expressions

There's also support for ternary conditional expressions. For example the following statement prints the value `false` after two logical comparisons.

```
a := 7;
b := 13;
(((a < 13) ? 10 : 20) >  15)->PrintLine();
```

3.4 Statements

Besides providing support for declaration statements the language has support for conditional and control statements. As with other languages, control statements can be nested in order to provide finer grain logical control. General control statements include `if` and `select` statements. Basic looping statements include `while`, `do/while`, `for` and `each` loops. Note, all statements end with the '`;`' character.

3.4.1 If Statement

An `if` statement is a control statement that executes an associated block of code if it evaluates to `true`. If the evaluation statement does not evaluate to `true` than an `else if` statement may be evaluated (if it exists), otherwise an `else` statement will be executed (if it exists). The example below demonstrates an `if` statement.

```
value := Console->ReadLine()->ToInt();
if(value <> 3) {
  "Not equal to 3"->PrintLine();
```

```
}
else if(value < 13) {
  "Less than 13"->PrintLine();
}
else {
  "Some other number"->PrintLine();
};
```

3.4.2 Select Statement

A `select` statement maps a value to 1 or more labels. Labels are associated to statement blocks. A label may either be a literal or an `enum` value. Multiple labels can be mapped to the same statement block. Below is an example of a `select` statement.

```
select(v) {
   label Color->Red: {
      "Red"->PrintLine();
   }

   label 9:
   label 19: {
      v->PrintLine();
   }

   label 27: {
      (3 * 9)->PrintLine();
   }

   other: {
     "some rather another"->PrintLine();
   }
};
```

3.4.3 While Statement

A `while` statement is a control statement that will continue to execute it's main body as long as it's conditional expression evaluates to `true`. When its conditional expression evaluates to `false` than the loop body will cease to execute.

```
i := 10;
while(i > 0) {
   i->PrintLine();
   i -= 1;
}
```

3.4.4 Do/While Statement

A `do/while` statement is a control statement that will execute it's main body at least once and continue to execute it's main body as long as its conditional expression evaluates to `true`. When it's conditional expression evaluates to `false` than the loop body will cease to execute.

```
i := 10;
do {
   i->PrintLine();
   i -= 1;
}
while(i > 0);
```

3.4.5 For Statements

The `for` statement is another common looping construct. The `for` loop consists of a pre-condition statement followed by an evaluation expression and an update statement.

```
location := "East Bay"->ToCharArray();
for(i := 0; i < location->Size(); i += 1;) {
  location[i]->PrintLine();
}
```

3.4.6 Each Statements

The `each` statement is a specialized version of a `for` statement. The `each` loop consists of a counter variable and a data structure that has a `Size` method, such as arrays and `Vector` classes. The statement iterates thru all elements in the data structure.

```
values := Int->New[3];
values[0] := 5;
values[1] := 1;
```

```
values[2] := 0;

each(i : values) {
  values[i]->PrintLine();
};
```

4 User Defined Types

4.1 Enums

Enums are user defined enumerated types. The main use of an `enum` is to group a class of countable values, for example colors, into a distinct category. Once `enum` values have been defined they may not be assigned or associated to a other `enum` groups or integer classes. The valid operations for enums are as follows:

- assignment (`:=`)
- equal (`=`)
- not–equal (`<>`)

In addition, enum values may be used in `select` statements as conditional tests or labels.

```
enum Color {
   Red,
   Black,
   Green
}

enum Type := -100 {
   TYPES,
   CLASSES,
   INT,
   FLOAT
}
```

4.2 Classes

Classes are user defined types that allow programmers to create specialized data types. Classes are made up of attributes (data) and operations

(methods). Classes are used to encapsulate programming logic and localize information. Operations that are associated to a class may either be at the class level or instance level. Class instances are created by calling an object's `New()` function. Note, an object instance can only be created if one or more `New()` functions have been defined.

4.2.1 Class Inheritance

Classes may be derived from other classes using the `from` keyword. Class inheritance allows classes to share common functionality. The Objeck language supports single class inheritance, meaning that a derived class may only have one parent. The language also supports virtual classes, which assures that derived classes have been defined for all required operations declared in the base class. Virtual classes also allow the programmer to define non-virtual methods that contain program behavior. Virtual classes are dynamically associated with implementation classes at runtime.

```
class Foo {
  @lhs : Int;

  New(lhs : Int) {
    @lhs := lhs;
  }

  method : native : AddTwo(rhs : Int) ~ Int {
    return 2 + rhs;
  }

  method : virtual : AddThree(int rhs) ~ Int;

  method : GetLhs() ~ Int {
    return lhs;
  }
}

class Bar from Foo {
  New(value : Int) {
    Parent(value);
  }

  method : native : AddThree(rhs : Int) ~ Int {
```

```
    return 3 + rhs;
  }

  function : Main() ~ Nil {
    bar : b := Bar->New(31);
    b->AddThree(9)->PrintLine();
  }
}
```

4.2.2 Class Casting and Identification

An object that is inherited from another object may be either upcasted or downcasted. Object casting can be performed using the `As()` operator. Upcasting requires a runtime check, while down casting does not. If cross casting is detected then a compile time error will be generated.

```
values := Vector->New();
values->AddBack(IntHolder->New(2));
values->AddBack(IntHolder->New(4));
values->AddBack(IntHolder->New(8));

total := 0;
each(i : values) {
  total += values->Get(i)->As(IntHolder)->Get();
};
total->PrintLine();
```

To determine if an object instance is of a certain class type (object or interface) it's `TypeOf` method can be invoked. This method will return `true` if the instance is of the same or a derived type, `false` otherwise. This method can be used to check a class instance type before casting it.

```
s := "FooBar";
t := s->TypeOf(String);
t->PrintLine();
```

The class that a given object instance belongs to can found by calling its `GetClassID` method. This method returns an enum that is associated with that instance's class type. This method is generally used to determine if two object instances are of the same or different classes.

4.2.3 Methods and Functions

The Objeck language supports both methods and functions. Functions are public static procedures that may be executed by any class. Methods are operations that may be performed on an object instance. Methods have `public` and `private` qualifiers. Methods that are `private` may only be called from within the same class, while `public` methods may be called from other classes. Note, methods are `private` by default. The Objeck language supports polymorphic methods and functions, meaning that there can be multiple methods with the same name within the same class as long as their declaration arguments vary.

Method and function parameters may also be assigned default values. For example:

```
function : Duplicate(str : String, max : Int := str->Size())
         ~ String { ... }
```

Methods and functions can either be executed in an interpreted or JIT compiled mode. Interpreted execution mimics microprocessor functions in a platform independent manner. JIT execution takes the compiled stack code an produces native machine code. Note, that there is initial overhead involved in the JIT compilation process since it occurs at runtime. In addition, some methods can not be compiled into native machine code but this is a rare case. The keyword `native` is used to JIT compile methods and functions at runtime.

A function or method may be defined as `virtual` meaning that any class that originates from that class must implement all of the class's `virtual` methods or functions. `Virtual` methods are a way to ensure that certain operations are available to a family of classes. If a class declares a `virtual` method then the class become `virtual`, meaning that it cannot be directly instantiated.

Below is an example of declaring a virtual method:

```
method : virtual : public : GetMake() ~ String;
```

4.3 Interfaces

As a modern object-oriented language, Objeck supports interfaces. Interfaces define virtual methods that must be implemented by a given class. A class may implement one or more interfaces. Interface references may be passed, dereferenced and casted in a similar manner to class references. Below is an example of an interface definition:

```
interface Color {
  method : virtual : public : GetColor() ~ String;
}

interface Vehicle {
  method : virtual : SetName(name : String) ~ Nil;
  method : virtual : public : GetNumberOfWheels() ~ Int;
}

class Ufo implements Vehicle, Color {
  method : public : GetColor() ~ String {
    ...
  }
  method : SetName(name : String) ~ Nil {
    ...
  }
  method : public : GetNumberOfWheels() ~ Int {
    ...
  }
}
```

4.3.1 Anonymous Classes

An anonymous class can be used to define inline interface methods. An anonymous class must implement all virtual methods for an interface. Variables that are within the scope of the anonymous class statement can be referenced by the anonymous class if they're passed as parameters the constructor.

```
interface Greetings {
  method : virtual : public : SayHi() ~ Nil;
}

class Hello {
  function : Main(args : String[]) ~ Nil {
    hey := Base->New() implements Greetings {
      New() {}
      method : public : SayHi() ~ Nil {
        "Hey..."->PrintLine();
      }
```

```
};

howdy := Base->New() implements Greetings {
  New() {}
  method : public : SayHi() ~ Nil {
    "Howdy!"->PrintLine();
  }
};
...
```

4.4 Higher-Order Functions

The Objeck language supports the notion of higher-order functions such that a given function may be bound to a variable at runtime. Variables are assigned based upon functional prototypes. Prototypes enforce strong type checking by ensuring that a function's parameters and return type are consistent between assignments. Once a variable is bound, it may be assigned to other variables, passed to other functions/methods, returned from other functions/method or dynamically evoked. Please note, methods are not treated as higher-order constructs only functions.

4.4.1 Assigning and Passing Functions

The following example shows how a function is defined and assigned to a variable:

```
class Foo {
  function : GetSize(s : String) ~ Int {
      return s->Size();
  }
}
....
s1 : (String) ~ Int := Foo-> GetSize(String) ~ Int;
s2 := Foo->GetSize(String) ~ Int;
size1 := EnvokeSize("Hello", s1);
size2 := EnvokeSize("Hello", s2);
```

4.4.2 Envoking Functions

The following example shows how a function variable can be assigned and envoked:

```
...
method : public : EnvokeSize(s : String, f : (String) ~ Int) ~ Int {
  return f(s);
}
...
```

5 Native Shared Library Support

The Objeck Language has the ability to interact with native C/C++ shared libraries via runtime extensions. The APIs allows a programmer to load a shared libraries and invoke native C functions. Data is passed between the two layers via `Object[]`. Basic objects such as `Int` and `Float` types must be wrapped in `IntHolder` and `FloatHolder` classes respectively. Please refer to the examples in the source code distribution for additional information.

A native function signature looks like the following:

```
#ifdef _WIN32
  __declspec(dllexport)
#endif
  void odbc_connect(VMContext& context);
```

The `VMContext` structure contains pointers to the calculation stack as well as utility functions that allow programmers to allocate VM objects and arrays. In addition, this structure provides the ability to invoke Objeck functions and methods..

```
struct VMContext {
  long* data_array;
  long* op_stack;
  long* stack_pos;
  APITools_AllocateArray_Ptr alloc_array;
  APITools_AllocateObject_Ptr alloc_obj;
  APITools_MethodCall_Ptr call_method_by_name;
  APITools_MethodCallId_Ptr call_method_by_id;
};
```

The `lib_api.h` header file also includes helper functions that allow programmers to access and set data that passed into the native C function. A subset of the available functions include the following:

- `APITools_GetFunctionValue`
- `APITools_SetFunctionValue`
- `APITools_GetIntValue`
- `APITools_GetIntAddress`
- `APITools_SetIntValue`
- `APITools_GetFloatValue`
- `APITools_GetFloatAddress`
- `APITools_GetStringValue`
- `APITools_SetStringValue`
- `APITools_CallMethod`
- `APITools_PushInt`
- `APITools_PushFloat`
- `APITools_GetArraySize`
- `APITools_SetIntArrayElement`
- `APITools_GetFloatArrayElement`
- `APITools_SetFloatArrayElement`

6 Debugger

The Objeck compiler toolset contains a simple interactive read-only debugger, which allows programmers to monitor the runtime behavior of their programs. The debugger allows programmers to set breakpoints within methods based upon source line numbers. The debugger can also calculate simple arithmetic expressions involving variables and constants. The following commands are currently supported:

6.1 Debugging Commands

Command	*Description*	*Example*
`[b]reak`	sets a breakpoint	`b` `b hello.obs:10`
`breaks`	shows all breakpoints	
`[d]elete`	deletes a breakpoint	`d hello.obs:10`
`clear`	clears all breakpoints	
`[n]ext`	moves to the next line within the same method/function with debug information	
`[s]tep`	moves to the next line with debug information	
`[j]ump`	jumps out of an existing method/function and moves to the next line with debug information	
`args`	specifies program arguments	`args "Hello World"`
`[r]un`	runs a loaded program	

`[p]rint`	prints the value of an expression, along with metadata	`p locl_ref`
		`p @inst_ref`
		`p Klass→class_ref`

`[l]ist`	lists a range of lines in a source file or the lines near the current breakpoint	`l` `l hello.obs:10`
`[i]nfo`	displays the variables for a class	`i` `i class=Foo`
`stack`	displays the method/function call stack	
`exe`	loads a new executable	`exe "./../test.obe"`
`src`	specifies a new source path	`src "../../"`
`[q]uit`	exits a given debugging session	

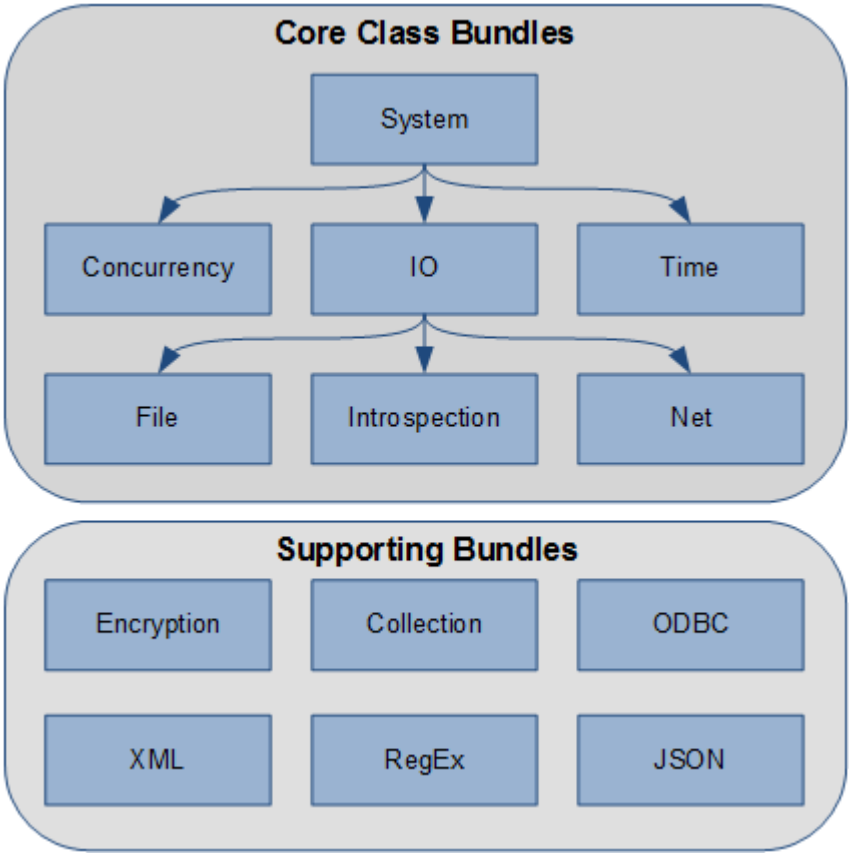
6.2 Starting the Debugger

The source program must be compiled with the `-debug` set. The command line debugger is started up running the `odb` executable. The `-exe` option must be present and specify the path to the executable. The The `-src` option is optional and specifies the path to the program source. Also note, to print instance level variables the path must start with `@self→`.

7 Class Libraries

Objeck includes class libraries that provides access to system resources, such as files and sockets, while also providing support for data structures like vectors, lists, queues, etc. As new classes are added they'll be documented in this section.

The diagram below outlines available class bundles:



7.1 System

Class support for basic data types. These classes are located in the default 'System' bundle.

7.1.1 Base

Base class for all objects.

- `GetClass` - returns metadata about a class
  - method : public : GetClass() ~Class
- `GetClassID` - returns the class ID
  - method : public : native : GetClassID() ~ClassID
- `GetInstanceID` - returns a unique instance ID
  - method : public : native : GetInstanceID() ~Int

7.1.2 Bool

- `Print` - prints the current value
  - method : native : Print() ~Nil
- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil
- `ToString` - converts the current value to a `String` object instance
  - method : native : ToString() ~String

7.1.3 Char

- `ToUpper` - coverts a character to a uppercase character
  - method : public : native : ToUpper() ~Char
- `ToLower` - coverts a character to a lowercase character
  - method : public : native : ToLower() ~Char
- `IsDigit` - determines if the character is a digit (in the range of `0-9`)

  - method : native : IsDigit() ~Bool

- `IsChar` - determines if the character is a alpha (in the range of `A-Z` or `a-z`)
  - method : native : IsChar() ~Bool

- `Min` - returns the smallest of the two numbers; returns the same number if they are equal
  - method : native : Min(r : Char) ~Char

- `Max` - returns the largest of the two numbers; returns the same number if they are equal
  - method : native : Max(r : Char) ~Char

- `Print` - prints the current value
  - method : native : Print() ~Nil

- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil

- `ToString` - converts the current value to a `String` object instance
  - method : native : ToString() ~String

- `ToBytes` - converts `Char` array to a UTF-8 `Byte` stream (`Char[]` only)
  - method : native : ToBytes() ~Byte[]

7.1.4 Byte/Int

- `Min` - returns the smallest of the two numbers; returns the same number if they are equal
  - method : native : Min(r : Int) ~Int

- `Max` - returns the largest of the two numbers; returns the same number if they are equal
  - method : native : Max(r : Int) ~Int

- `MaxSize` - maximum size of an integer

  - method : native : MaxSize() ~Int

- `Abs` - returns the absolute value of the current number
  - method : native : Abs() ~Int
- `Print` - prints the current value
  - method : native : Print() ~Nil
- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil
- `ToHexString` - converts the current value to a hexadecimal `String`
  - method : native : ToHexString() ~String
- `ToBinaryString` - converts the current value to a binary `String`
  - method : native : ToBinaryString() ~String
- `ToString` - converts the current value to a decimal `String`
  - method : native : ToString() ~String
- `ToUnicode` - converts a UTF-8 `Byte` array to `Char` array (`Byte[]` only)
  - method : native : ToUnicode() ~Char[]

7.1.5 Float

- `Min` - returns the smallest of the two numbers; returns the same number if they are equal
  - method : native : Min(r : Float) ~Float
- `Max` - returns the largest of the two numbers; returns the same number if they are equal
  - method : native : Max(r : Float) ~Float
- `Abs` - returns the absolute value of the current number
  - method : native : Abs() ~Float
- `Floor` - returns the floor of the current number

  - method : Floor() ˜Float

- `Ceiling` - returns the ceiling of the current number
  - method : Ceiling() ˜Float
- `Sin` - returns the sine of the radian value
  - method : Sin() ˜Float
- `Cos` - returns the cosine of the radian value
  - method : Cos() ˜Float
- `Tan` - returns the tangent of the radian value
  - method : Tan() ˜Float
- `ArcSin` - returns the arc sine of the radian value
  - method : ArcSin() ˜Float
- `ArcCos` - returns the arc cosine of the radian value
  - method : ArcCos() ˜Float
- `ArcTan` - returns the arc tangent of the radian value
  - method : ArcTan() ˜Float
- `ToRadians` - converts degrees to radians
  - method : ToRadians() ˜Float
- `ToDegrees` - converts radians to degrees
  - method : ToDegrees() ˜Float
- `Log` - returns the natural log of the radian value
  - method : Log() ˜Float
- `Pi` - returns the value of Pi
  - function : Pi() ˜Float
- `E` - returns the value of E

  - function : E() ~Float

- `Power` - returns the exponential power value
  - method : Power(v : Float) ~Float
- `SquareRoot` - returns the square root
  - method : SquareRoot() ~Float
- `Random` - returns a puedo-random number between 0.0 and 1.0
  - function : Random() ~Float
- `Print` - prints the current value
  - method : native : Print() ~Nil
- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil
- `ToString` - converts the current value to a `String` object instance
  - method : native : ToString() ~String

7.1.6 String

- `New`
  - New()
  - New(s : String)
  - New(a : Char[])
  - New(a : Char[], offset : Int, max : Int)
  - New(a : Byte[])
  - New(a : Byte[], offset : Int, max : Int)
- `Append` - Appends a `String`, `Char[]`, `Char`, `Int` or `Float` to the current String instance
  - method : public : native : Append(s : String) ~Nil
  - method : public : native : Append(c : Char) ~Nil
  - method : public : native : Append(b : Byte) ~Nil

  - method : public : native : Append(i : Int) ~Nil
  - method : public : native : Append(f : Float) ~Nil
  - method : public : native : Append(a : Char[]) ~Nil
  - method : public : native : Append(a : Char[], offset : Int, max : Int) ~Nil
  - method : public : native : Append(a : Byte[]) ~Nil
  - method : public : native : Append(a : Byte[], offset : Int, max : Int) ~Nil

- `Find` - returns the index of the first occurrence of a given Character; -1 if not found
  - method : public : native : Find(c : Char) ~Int
  - method : public : native : Find(offset : Int, c : Char) ~Int

- `Find` - returns the index of the first occurrence of a given String; -1 if not found
  - method : public : native : Find(s : String) ~Int
  - method : public : native : Find(offfset : Int, s : String) ~Int

- `FindAll` - returns an array of found indices of the given String
  - method : public : native : FindAll(s : String) ~Int[]

- `Replace` - find and replace a single string instance.
  - method : public : native : Replace(find : String, replace : String) ~String

- `ReplaceAll` - find and replace all string instances.
  - method : public : native : ReplaceAll(find : String, replace : String) ~String

- `Size` - returns the size of the String
  - method : public : native : Size() ~Int

- `IsEmpty` - returns true if the string is empty, false otherwise
  - method : public : native : IsEmpty() ~Bool

- `Get` - returns the Character at the given index or -1 if not found
  - method : public : native : Get(i : Int) ~Char
- `ToCharArray` - converts a string to a `Char[]`
  - method : public : native : ToCharArray() ~Char[]
- `ToByteArray` - converts a string to a `Byte[]`
  - method : public : native : ToByteArray() ~Byte[]
- `ToInt` - converts a string to a `Int`
  - method : public : native : ToInt() ~Int
- `ToFloat` - converts a string to a `Float`
  - method : public : native : ToFloat() ~Int
- `SubString` - creates a new string that contains a subset of the string's contents
  - method : public : native : SubString(offset : Int) ~String
  - method : public : native : SubString(offset : Int, length : Int) ~String
- `Trim` - removes all leading and trailing whitespace
  - method : public : native : Trim() ~String
- `Pop` - removes the last character from a string
  - method : public : native : Pop() ~Char
- `StartsWith` - returns true if `String` starts with matching pattern; returns false otherwise
  - method : public : native : StartsWith() ~Bool
- `EndsWith` - returns true if `String` ends with matching pattern; returns false otherwise
  - method : public : native : EndsWith() ~Bool
- `ToUpper` - coverts all lowercase characters to uppercase characters

  - method : public : native : ToUpper() ~String

- `ToLower` - coverts all uppercase characters to lowercase characters
  - method : public : native : ToLower() ~String
- `Reverse` - returns a string with reversed characters
  - method : public : native : Reverse() ~String
- `Equals` - compares two string returns `true` if they are equal
  - method : public : Equals(rhs : String) ~Bool
- `Compare` - compares two string returns 0 if they are equal
  - method : public : native : Compare(rhs : Compare) ~Int
- `Split` - breaks a string into tokens based upon a given delimiter
  - method : public : native : Split(delim : String) ~String[]
- `Print` - prints the current value
  - method : native : Print() ~Nil
- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil

7.1.7 Bool

- `Print` - prints the current value
  - method : native : Print() ~Nil
- `PrintLine` - prints the current value along with a line return
  - method : native : PrintLine() ~Nil
- `ToString` - converts the current value to a `String` object instance
  - method : native : ToString() ~String

7.1.8 IntHolder

Holds an interger type.

- `New`
  - New(value : Int)
- `Get` - returns a value
  - method : public : Get() ~Int
- `Compare` - returns a value
  - method : public : Compare(rhs : Compare) ~Int
- `HashID` - returns a unique value for the given class type
  - method : public : HashID() ~Int

7.1.9 IntArrayHolder

Holds an interger array.

- `New`
  - New(value : Int[])
- `Get` - returns a value
  - method : public : Get() ~Int[]

7.1.10 FloatHolder

Holds a float type.

- `New`
  - New(value : Float)
- `Get` - returns a value
  - method : public : Get() ~Float
- `Compare` - returns a value
  - method : public : Compare(rhs : Compare) ~Int
- `HashID` - returns a unique value for the given class type
  - method : public : HashID() ~Int

7.1.11 FloatArrayHolder

Holds a float array.

- `New`
  - New(value : Float[])
- `Get` - returns a value
  - method : public : Get() ~Float[]

7.1.12 ByteArrayHolder

Holds a byte array.

- `New`
  - New(value : Byte[])
- `Get` - returns a value
  - method : public : Get() ~Byte[]

7.1.13 CharArrayHolder

Holds a character array.

- `New`
  - New(value : Char[])
- `Get` - returns a value
  - method : public : Get() ~Char[]

7.1.14 Runtime

The Runtime class provides information about the runtime VM environment.

- `GetPlatform` - returns a string containing information about the runtime environment
  - function : GetPlatform() ~String
- `GetProperty` - returns the value for a given property

  - function : GetProperty() ~String

- `SetProperty` - sets the key/value pair for a given property. Properties can also be set via the 'config.prop' configuration file, which must be located in the same directory as the VM.
  - function : SetProperty(k : String, v : String) ~Nil

- `GetDate` - returns the current time
  - function : public : GetDate() ~Date

- `Exit` - exits the current running program
  - function : public : Exit() ~Nil

- `Copy` - copies a block of memory from one array to another
  - function : Copy(dest : Char[], dest_offset : Int, src : Char[], src_offset : Int, len : Int) ~Bool
  - function : Copy(dest : Int[], dest_offset : Int, src : Int[], src_offset : Int, len : Int) ~Bool
  - function : Copy(dest : Float[], dest_offset : Int, src : Float[], src_offset : Int, len : Int) ~Bool
  - function : Copy(dest : Base[], dest_offset : Int, src : Base[], src_offset : Int, len : Int) ~Bool
  - function : Copy(dest : Compare[], dest_offset : Int, src : Compare[], src_offset : Int, len : Int) ~Bool

7.2 System.Introspection

Provides the ability to access program metadata at runtime.

7.2.1 Class

- `New`
  - New(c : String)

- `IsLoaded` - returns true if this class has been loaded, false otherwise
  - method : public : IsLoaded() ~Bool

- `Instance` - creates a new class instance and calls default constructor

  - function : Instacne(name : String) ˜Base

- `GetName` - returns the name of the class
  - method : public : GetName() ˜String
- `GetMethods` - returns an array of methods
  - method : public : GetMethods() ˜GetMethods[]
- `GetMethodNumber` - returns the number of methods in the class
  - method : public : GetMethodNumber() ˜Int

7.2.2 Method

- `GetClass` - returns class assoicated with the method
  - method : public : GetClass() ˜Class
- `GetName` - returns the pretty name of the method
  - method : public : GetName() ˜String
- `GetParameters` - returns the datatypes of the parameters
  - method : public : GetParameters() ˜DataType[]
- `GetReturn` - returns the datatype of the return
  - method : public : GetReturn() ˜DataType

7.2.3 DataType

- `GetType` - returns the type ID
  - method : public : GetType() ˜TypeId
- `GetDimension` - returns dimensions of the datatype
  - method : public : GetDimension() ˜Int
- `GetClassName` - returns string name of non-basic objects, Nil otherwise
  - method : public : GetClassName() ˜String

7.2.4 TypeId

- BOOL
- BYTE
- CHAR
- INT
- FLOAT
- CLASS
- FUNC

7.3 System.IO

This bundle supports console I/O, object serialization and defines general I/O interfaces.

7.3.1 InputStream

The InputStream interface defines input operations.

- `ReadByte` - reads a byte
  - method : public : ReadByte() ~Byte
- `ReadBuffer` - reads n number of bytes
  - method : public : ReadBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int
- `ReadString` - reads a line
  - method : public : ReadString() ~String

7.3.2 OutputStream

The OutputStream interface defines output operations.

- `WriteByte` - writes a byte
  - method : public : WriteByte(b : Int) ~Bool
- `WriteBuffer` - writes n number of bytes

  - method : public : WriteBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int

- `WriteString` - writes a string
  - method : public : WriteString(s : String) ~Nil

- `Flush` - flushes buffer
  - method : public : Flush() ~Nil

7.3.3 Console

The Console class allows programmers to read and write information to the system console. The class supports the following operations:

- `Print` - prints all basic types including `String` and `Char[]` to standard out.
  - function : Print(t : type) ~ConsoleIO

- `PrintLine` - prints all basic types including `String` and `Char[]` to standard out followed by a newline.
  - function : PrintLine(t : type) ~ConsoleIO

- `WriteBuffer` - writes a buffer to standard out.
  - function : public : WriteBuffer(offset : Int, num : Int, buffer : Byte[]) ~Bool
  - function : public : WriteBuffer(offset : Int, num : Int, buffer : Char[]) ~Bool

- `ReadString` - reads in a line of text as a `Char[]` from standard in.
  - method : public : ReadString() ~String

7.3.4 Serializer

The Serializer class allows datatypes to be deflated into a byte array for storage or transmission. This class is in the 'IO' bundle. The class supports the following operations:

- `New`

  - New()

- `Add` - deflates a datatype
  - method : public : Write(b : Bool) ~Nil
  - method : public : Write(i : Int) ~Nil
  - method : public : Write(f : Float) ~Nil
  - method : public : Write(o : Base) ~Nil
  - method : public : Write(b : Bool[]) ~Nil
  - method : public : Write(b : Byte[]) ~Nil
  - method : public : Write(c : Charl[]) ~Nil
  - method : public : Write(i : Int[]) ~Nil
  - method : public : Write(f : Float[]) ~Nil
- `Serialize` - returns a byte stream of serialized data
  - method : public : Serialize() ~Byte[]

7.3.5 Deserializer

The Deserializer class allows datatypes to be inflated from a byte array. This class is in the 'IO' bundle. The class supports the following operations:

- `New`
  - New(b : Byte[])
- `ReadBool` - reads a boolean value
  - method : public : ReadBool() ~Bool
- `ReadInt` - reads an integer value
  - method : public : ReadInt() ~Int
- `ReadFloat` - reads a float value

  - method : public : ReadFloat() ~Float

- `ReadObject` - reads an object
  - method : public : ReadObject() ~Float
- `ReadBoolArray` - reads a boolean array
  - method : public : ReadBoolArray() ~Bool[]
- `ReadByteArray` - reads a byte array
  - method : public : ReadByteArray() ~Byte[]
- `ReadCharArray` - reads a character array
  - method : public : ReadCharArray() ~Char[]
- `ReadIntArray` - reads an integer array
  - method : public : ReadIntArray() ~Byte[]
- `ReadFloatArray` - reads a float array
  - method : public : ReadFloatArray() ~Float[]

7.4 System.IO.File

Provides support for file and directory operations.

7.4.1 File

The File class allows programmers to access files.

- `New`
  - New(name : String)
- `IsOpen` - returns true if file is open.
  - method : public : IsOpen() ~Bool
- `IsEOF` - returns true if the file pointer is at the EOF.
  - method : public : IsEOF() ~Bool
- `Seek` - seeks to a position in a file.

  - method : public : Seek(p : Int) ~Bool

- `Rewind` - moves the file pointer to the beginning of a file.
  - method : public : Rewind() ~Nil

- `Size` - returns the size of the file.
  - function : Size(name : String) ~Int

- `Remove` - deletes a file.
  - function : Remove(n : String) ~Bool

- `Exists` - returns true if the file exists.
  - function : Exists(n : String) ~Bool

- `CreateTime` - returns the time in which the file was created
  - function : CreateTime(n : String) ~Date
  - function : CreateTime(n : String, gmt : Bool) ~Date

- `ModifiedTime` - returns the time in which the file was last modifed
  - function : ModifiedTime(n : String) ~Date
  - function : ModifiedTime(n : String, gmt : Bool) ~Date

- `AccessedTime` - returns the time in which the file was last modifed
  - function : AccessedTime(n : String) ~Date
  - function : AccessedTime(n : String, gmt : Bool) ~Date

- `Rename` - renames a file
  - function : Rename(from : String, to : String) ~Bool

7.4.2 FileReader

The FileReader is inherited from the File class implements InputStream and allows programmers read files. The class supports the following operations:

- `New`
  - New(name : String)

- `Close` - closes a file.
  - method : public : Close() ~Nil
- `ReadByte` - reads a byte from a file.
  - method : public : ReadByte() ~Byte
- `ReadBinaryFile` - reads an entire file into a byte array.
  - function : public : ReadBinaryFile(file : String) ~Byte[]
- `ReadBuffer` - reads n number of bytes from a file.
  - method : public : ReadBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int
- `ReadString` - reads a line from a file.
  - method : public : ReadString() ~String
- `ReadFile` - reads an entire file.
  - method : public : ReadFile() ~String

7.4.3 FileWriter

The FileReader is inherited from the File class implements OutputStream and allows programmers to write to files. The class supports the following operations:

- `Close` - closes a file.
  - method : public : Close() ~Nil
- `Flush` - flushes a file.
  - method : public : Flush() ~Nil
- `WriteByte` - writes a byte to a file.
  - method : public : WriteByte(b : Int) ~Bool
- `WriteBuffer` - writes n number of bytes to a file.
  - method : public : WriteBuffer(buffer : Byte[]) ~Int

  - method : public : WriteBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int

- `WriteString` - writes a string to a file.
  - method : public : WriteString(s : String) ~Nil

7.4.4 Directory

The Directory class allows programmers manipulate file system directories. The class supports the following operations:

- `Create` - creates a new directory.
  - function : Create(n : String) ~Bool
- `Exists` - returns true if the directory exists.
  - function : Exists(n : String) ~Bool
- `List` - returns vector of file and directory names.
  - function : List(n : String) ~String[]

7.5 System.IO.Net

Provides support for TCP/IP client and server sockets. The TCPSecureSocket class provides SSL support for TCP/IP connections.

7.5.1 TCPSocket/TCPSecureSocket

The TCPSocket class implements the InputStream and OuptStream interfaces and allows programmers to connect to TCP/IP socket servers. The class supports the following operations:

- `New`
  - New(address : String, port : Int)
- `IsOpen` - returns true if the socket is connected.
  - method : public : IsOpen() ~Bool
- `GetAddress` - return the socket address
  - method : public : GetAddress() ~String

- `GetPort` - return the socket port
  - method : public : GetPort() ~Int
- `Close` - closes a connected socket.
  - method : public : Close() ~Nil
- `WriteByte` - writes a byte to a socket.
  - method : public : WriteByte(b : Int) ~Bool
- `WriteBuffer` - writes n number of bytes to a socket.
  - method : public : WriteBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int
- `WriteString` - writes a string to a socket.
  - method : public : WriteString(s : String) ~Nil
- `ReadByte` - reads a byte from a socket.
  - method : public : ReadByte() ~Byte
- `ReadBuffer` - reads n number of bytes from a socket.
  - method : public : ReadBuffer(offset : Int, num : Int, buffer : Byte[]) ~Int
- `ReadString` - reads a line from a socket.
  - method : public : ReadString() ~String
- `HostName` - returns the host name (for TCPSocket only)
  - function : HostName() ~String

7.5.2 TCPSocketServer

The TCPSocket class allows programmers develop TCP/IP socket servers. The class supports the following operations:

- `New`
  - New(port : Int)

- `Listen` - listens for client connections
  - method : public : Listen(backlog : Int) ~Bool
- `Accept` - listens for client connections
  - method : public : Accept() ~TCPSocket
- `Close` - closes a connected socket.
  - method : public : Close() ~Nil

7.6 System.Time

Provides time and date operations.

7.6.1 Date

The Date class allows programmers gain access to the current system time. This class is in the 'Time' bundle. The class supports the following operations:

- `New`
  - New()
  - New(gmt : Bool)
  - New(day : Int, month : Int, year : Int, gmt : Bool)
  - New(day : Int, month : Int, year : Int, hours : Int, mins : Int, secs : Int, gmt : Bool)
- `GetDay` - return the current day as an `Int`.
  - method : public : GetDay() ~Int
- `GetDayName` - return the current day of the week as a `String`.
  - method : public : GetNameDay() ~String
- `GetMonth` - return the current month as an `Int`.
  - method : public : GetMonth() ~Int
- `GetMonthName` - return the current month as a `String`.
  - method : public : GetMonthName() ~String

- `GetYear` - return the current year as an `Int`.
  - method : public : GetYear() ~Int
- `GetHours` - return the current hour as an `Int`.
  - method : public : GetHours() ~Int
- `GetMinutes` - return the current minutes as an `Int`.
  - method : public : GetMinutes() ~Int
- `GetSeconds` - return the current seconds as an `Int`.
  - method : public : GetSeconds() ~Int
- `IsSavingsTime` - return true if daylights saving time, false otherwise
  - method : public : IsSavingsTime() ~Bool
- `Add` - adds (or subtracts) time from a date
  - method : public : AddDays(value : Int) ~Nil
  - method : public : AddHours(value : Int) ~Nil
  - method : public : AddMinutes(value : Int) ~Nil
  - method : public : AddSeconds(value : Int) ~Nil
- `ToString` - formats a date into a `String`.
  - method : public : ToString() ~String

7.6.2 Timer

The Time class allows programmers perform simple timings. This class is in the 'Time' bundle. The class supports the following operations:

- `New`
  - New()
- `Start` - Starts the timing
  - method : public : Start() ~Nil
- `End` - Ends the timing

  - method : public : End() ~Nil

- `End` - Returns the elapsed time in milliseconds
  - method : public : GetElapsedTime() ~Int

7.7 System.Concurrency

Support for system concurrency. These classes are located in the default 'System.Concurrency' bundle.

7.7.1 Thread

The Thread class allows programmers to invoke a method in a separately running system thread. This class is in the 'System.Concurrency' bundle.

- `New`
  - New(name : String)
- `Execute` - Execute a `Thread()` by calling the thread's `Run()` method in a separate thread.
  - method : public : Execute(param : System.Base) ~Nil
- `Sleep` - Sleeps a given thread for a number of milliseconds.
  - function : Sleep(t : Int) ~Nil
- `Join` - Joins a thread instance with the creating thread
  - method : public : Join() ~Nil
- `GetName` - Returns the name of the thread
  - method : public : GetName() ~String
- `GetRunID` - Returns running thread ID
  - method : public : GetName() ~String
- `Run` - Method that is executed by the new thread
  - method : virtual : public : Run(param : System.Base) ~Nil

7.7.2 ThreadMutex

The ThreadMutex class is used for thread synchronization i.e. locking and unlocking running threads.

- `New`
  - New(name : String)
- `GetName` - Returns the name of the thread
  - method : public : GetName() ~String

7.8 Collection

Provides support for basic data structures such as vectors, lists, hashes, maps, etc. This bundle is included in the 'collect.obl' file.

7.8.1 Compare

Interface for all objects that use comparative algorithms.

- `Compare` - compares two object instances; 0 if equal, less-then 0 if less and greater-then 0 if greater.
  - method : virtual : public : Compare(rhs : System.Compare) ~Int
- `HashID` - returns a unique value for the given class type
  - method : virtual : public : HashID() ~Int

7.8.2 BasicCompare

Basic implementation of `Compare` methods.

- `Compare` - returns true if argument is the same instance, false otherwise.
  - method : public : Compare(rhs : System.Compare) ~Int
- `HashID` - hash value based upon instance ID
  - method : public : HashID() ~Int

7.8.3 Queue/IntQueue/FloatQueue

The Queue class support the concept of a growing queue and based upon a FIFO priority. There are two specialized version of this class: `IntQueue` and `FloatQueue`. The `IntQueue` and `FloatQueue` classes use `Int` and `Float` types respectively instead of `Base` type. The general class supports the following operations:

- `New`
- `Add` - adds a new value to the back of the list
  - method : public Add(value : Base) ~Nil
- `Remove` - removes the first element in the list
  - method : public : Remove() ~Base
- `Head` - returns the first element in the list
  - method : public : Head() ~Base
- `Empty` - clears the list
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise
  - method : public : native : IsEmpty() ~Bool
- `Size` - returns the size
  - method : public : Size() ~Int

7.8.4 List/IntList/FloatList

The List class allows values to be added, removed and deleted from a list. There are two specialized version of this class: `IntList` and `FloatList`. The `IntList` and `FloatList` classes use `Int` and `Float` types respectively instead of `Compare` type. The general class supports the following operations:

- `New`
- `AddBack` - adds a new value to the back of the list
  - method : public : native : AddBack(value : Compare) ~Nil

- `RemoveBack` - removes the last element in the list
  - method : public : RemoveBack() ~Nil
- `AddFront` - adds a new value to the front of the list
  - method : public : native : AddFront(value : Compare) ~Nil
- `RemoveFront` - removes the first element in the list
  - method : public : RemoveFront() ~Nil
- `Find` - finds an element in the list
  - method : public : Find(value : Compare) ~Bool
- `Has` - returns true if item is in list
  - method : public : Has(value : Compare) ~Bool
- `Insert` - inserts a new value in the position pointed to the cursor
  - method : public : Insert(value : Compare) ~Bool
- `Remove` - removes the last element in the list
  - method : public : Remove() ~Nil
- `Insert` - inserts an element into the current cursor position
  - method : public : Insert(value : Compare) ~Bool
- `Next` - advances the internal cursor by one element
  - method : public : Next() ~Nil
- `Previous` - retreats the internal cursor by one element
  - method : public : Previous() ~Nil
- `Get` - returns the value of the element pointed to by the cursor
  - method : public : Get() ~Compare
- `Forward` - moves the cursor to the end of the list
  - method : public : Forward() ~Nil

- `Rewind` - moves the cursor to the start of the list
  - method : public : Rewind() ~Nil
- `IsFront` - returns true if cursor is at the front of the list
  - method : public : IsFront() ~Bool
- `IsBack` - returns true if cursor is at the back of the list
  - method : public : IsBack() ~Bool
- `IsEnd` - returns true if cursor is at the end of the list
  - method : public : IsEnd() ~Bool
- `Front` - returns the item at the front of the list
  - method : public : Front() ~Compare
- `Back` - returns the item at the back of the list
  - method : public : Back() ~Compare
- `Empty` - clears the list
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise
  - method : public : native : IsEmpty() ~Bool
- `Size` - returns the stack size
  - method : public : Size() ~Int

7.8.5 Stack/IntStack/FloatStack

The Stack class support the concept of a growing stack based upon a FILO priority. There are two specialized version of this class: `IntStack` and `FloatStack`. The `IntStack` and `FloatStack` classes use `Int` and `Float` types respectively instead of `Base` type. The general class supports the following operations:

- `New`
- `Push` - pushes a new value onto the stack

  - method : public : Push(value : Base) ~Nil

- `Pop` - pops a values from the top of the stack
  - method : public : Pop() ~Base
- `Top` - retrieves the top value from the stack (without popping the stack)
  - method : public : Top() ~Base
- `IsEmpty` - returns true if stack is empty
  - method : public : IsEmpty() ~Bool
- `Empty` - clears the stack
  - method : public : Emtpy() ~Nil
- `Size` - returns the stack size
  - method : public : Size() ~Int

7.8.6 Vector/CompareVector/IntVector/FloatVector

The Vector class support the concept of a growing array. There are three specialized version of this class: `CompareVector`, `IntVector` and `FloatVector`. The `IntVector`, `FloatVector` and `CompareVector` classes use `Int`, `Float` and `Compare` types respectively instead of `Base` type. The general class supports the following operations:

- `New`
  - New()
  - New(values : Base[])
  - New(values : Vector)
- `AddBack` - adds a new value to the back of the vector
  - method : public : AddBack(value : Base) ~Nil
- `RemoveBack` - removes the last element in the vector
  - method : public : RemoveBack() ~Base
- `Remove` - removes the indexed item from the vector

  - method : public : Remove(index : Int) ~Base

- `Get` - returns the value of the element pointed to by the cursor
  - method : public : Get(index : Int) ~Base
- `Set` - replaces the list value based upon the given index
  - method : public : Set(value : Base, index : Int) ~Bool
- `Size` - returns the size of the list
  - method : public : Size() ~Int
- `Empty` - clears the vector
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise
  - method : public : native : IsEmpty() ~Bool
- `ToArray` - returns an array of elements
  - method : public : ToArray() ~Base
- `Sort` - sorts a vector of values using a merge sort algorithm in $O(n \log_2 n)$ time.
  - method : public : Sort() ~Nil
- `BinarySearch` - performs a binary search for an element in a *sorted* vector using an iterative binary search algorithm in $O(\log_2 n)$ time. Returns the index of element is returned, -1 otherwise.
  - method : public : BinarySearch(v : Compare) ~Int
- `Find` - search for an element in the array and reutrn the index of the element found, -1 otherwise.
  - method : public : Find(v : Compare) ~Int
- `Min` - returns the smallest value in the vector
  - method : public : native : Min() ~Int/Float
- `Max` - returns the largest value in the vector

  - method : public : native : Max() ~Int/Float

- `Average` - returns the calculated average for all values in the list
  - method : public : native : Average() ~Int/Float

- `Filter` - applys the result of the passed in function to each element in the array
  - method : public : Filter(func : (Int/Float/Compare) ~ Bool) ~IntVector/FloatVector/CompareVector

- `Apply` - applys the result of the passed in function to each element in the array
  - method : public : Apply(func : (Int/Float) ~ Int/Float), IntVector/FloatVector

7.8.7 Map/IntMap/FloatMap/StringMap

The Map class supports the concept of an associative array with key/value pairs. The class implements a balance binary tree algorithm such that inserts, deletes and searches are $O(\log_2 n)$.

- `New`

- `Insert` - adds a new value to the tree
  - method : public : Insert(key : Compare, value : Base) ~Nil

- `Remove` - removes a value from the tree
  - method : public : Remove(key : Compare) ~Nil

- `Find` - searches for a value based upon a key
  - method : public : Find(key : Compare) ~Base

- `Has` - returns true if item is in map
  - method : public : Has(value : Compare) ~Bool

- `GetKeys` - returns a vector of keys
  - method : public : GetKeys() ~Vector

- `GetValues` - returns a vector of values
  - method : public : GetValues() ~Vector
- `Empty` - clears the map
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise
  - method : public : native : IsEmpty() ~Bool
- `Size` - returns the size of the map
  - method : public : Size() ~Int

7.8.8 Set/IntSet/FloatSet/StringSet

The Set class supports the concept of an associative array with keys. The class implements a balance binary tree algorithm such that inserts, deletes and searches are $O(\log_2 n)$.

- `New`
  - New()
- `Insert` - adds a new key to the tree
  - method : public : Insert(key : Compare) ~Nil
- `Remove` - removes a value from the tree
  - method : public : Remove(key : Compare) ~Nil
- `Has` - returns true if item is in map
  - method : public : Has(value : Compare) ~Bool
- `GetKeys` - returns a vector of keys
  - method : public : GetKeys() ~Vector
- `Empty` - clears the map
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise

  - method : public : native : IsEmpty() ~Bool

- `Size` - returns the size of the map
  - method : public : Size() ~Int

7.8.9 Hash/StringHash

The Hash class supports the concept of a hash table with key/value pairs. The class implements a hashing algorithm that is optimized for pairs of 256 elements or less (consider using the `Map` class for larger sets):

- `Insert` - adds a new value to the hash table
  - method : public : Insert(key : Compare, value : Base) ~Nil
- `Remove` - removes a value from the hash table
  - method : public : Remove(key : Compare) ~Nil
- `Find` - searches for a value based upon a key
  - method : public : Find(key : Compare) ~Base
- `Has` - returns true if item is in hash
  - method : public : Has(value : Compare) ~Bool
- `GetKeys` - returns a vector of keys
  - method : public : GetKeys() ~Vector
- `GetValues` - returns a vector of values
  - method : public : GetValues() ~Vector
- `Empty` - clears the hash table
  - method : public : Emtpy() ~Nil
- `IsEmpty` - returns true if empty, false otherwise
  - method : public : native : IsEmpty() ~Bool
- `Size` - returns the size of the Hash
  - method : public : Size() ~Int

7.9 HTTP

These classes provide support for the HTTP and HTTPS web protocols. This bundle is included in the 'collect.obl' file.

7.9.1 HttpClient/HttpsClient

The HttpClient class allows programmers access web servers via HTTP.

- `New`
  - New();
- `Post` - Performs a HTTP(S) POST to the connected website, returns true is successful
  - method : public : Post(url : String, content : String) ~Vector
  - method : public : Post(url : String, content_type: String, content : String) ~Vector
- `Get` - Performs a HTTP(S) GET from the connected website, returns a `Vector` of strings
  - method : public : Get(url : String) ~Vector
  - method : public : Get(url : String, content_type: String) ~Vector
- `GetHeaders` - Returns a `Map` of headers information. HTTP header information is populated after a GET and POST requests.
  - method : GetHeaders() ~Map
- `GetCookies` - Gets a list of cookies. The cookie list can be cleared using this method.
  - method : GetCookies() ~List
- `CookiesEnabled` - Enables or disables cookie support.
  - method : CookiesEnabled(state : Bool) ~List
- `SetCookie` - Sets a cookie.
  - method : SetCookie(cookie : String) ~Nil

7.10 JSON

These bundle provides for JSON parsing and document creation. This bundle is included in the 'json.obl' file.

7.10.1 JSONParser

The JSONParser class allows programmers to parse and search JSON documents.

- `New`
  - New(buffer : String)
- `Parse` - Parses a JSON stream into a graph of elements. Returns root instance if successful; Nil otherwise.
  - method : public : Parse() ~JSONElement

7.10.2 JSONElement

JSON element that contains a root value or references to nested elements (i.e. array and objects).

- `GetType` - Returns the JSON element type.
  - method : public : GetType() ~JSONType
- `GetValue` - Returns a JSON element value.
  - method : public : GetValue() ~String
- `Get` - Returns a JSON element by index for array element types.
  - method : public : Get(index : Int) ~JSONElement
- `Get` - Returns a JSON element by name for object elemet members.
  - method : public : Get(key : String) ~JSONElement
- `GetKeys` - Returns a list of element keys corresponding to object elemet members.
  - method : public : GetKeys() ~Vector
- `Size` - Returns number of sub-elements

  - method : public : Size() ~Int

- `ToString` - serialize object into a String
  - method : public : ToString() ~String

7.10.3 JSONType

- STRING
- NUMBER
- TRUE
- FALSE
- ARRAY
- OBJECT

7.11 XML

These bundle provides support for XML parsing and document creation. This bundle is included in the 'xml.obl' file.

7.11.1 XMLParser

The XMLParser class allows programmers to parse and search XML documents using a simplified DOM structure. This class in the 'XML' bundle:

- `New`
  - New(buffer : Char[])
  - New(buffer : String)
- `ToString` - serializes the DOM into a string
  - method : public : ToString() ~String
- `Parse` - Parses an XML stream into a DOM structure. Returns true if successful; false otherwise.
  - method : public : Parse() ~Bool
- `GetVersion` - returns the XML document version

  - method : public : GetVersion() ˜String

- `GetEncoding` - returns the XML character encoding
  - method : public : GetEncoding() ˜String
- `GetRoot` - Returns the root node of the DOM.
  - method : public : GetRoot() ˜XmlElement
- `FindElements` - returns all elements that match the given query string. The following XPath expressions are supported num, attrib_name first() and last().
  - method : public : FindElements(path : String) ˜Vector
- `GetError` - Returns the last parsing error; if one occurred.
  - method : public : Parse() ˜Bool

7.11.2 XmlBuilder

The XmlBuilder class allows a programmer to dynamically build an XML DOM. Once the DOM is build it may be traversed and/or serialized into a string. This class in the 'XML' bundle:

- `New`
  - New(root : String)
  - New(root : String, version : String, encoding : String)
- `GetRoot` - Returns the root element.
  - method : public : GetRoot() ˜XmlElement
- `ToString` - Serializes an element into a string
  - method : public : ToString() ˜String
- `GetVersion` - Returns the version name.
  - method : public : GetVersion() ˜String
- `SetVersion` - Sets the version name.
  - method : public : SetVersion(name : String) ˜Nil

- `GetEncoding` - Returns the encoding name.
  - method : public : GetEncoding() ˜String
- `SetEncoding` - Sets the encoding name.
  - method : public : SetEncoding(name : String) ˜Nil

7.11.3 XmlElement

The XmlElement class represents a canonical XML node within a DOM. This class in the 'XML' bundle:

- `New`
  - New(type : XmlElementType, name : String)
  - New(type : XmlElementType, name : String, content : String)
  - New(type : XmlElementType, name : String, attribs : String-Hash, content : String)
- `GetName` - Returns the element name.
  - method : public : GetName() ˜String
- `GetNamespace` - Returns the element namespace prefix.
  - method : public : GetNamespace() ˜String
- `SetName` - Sets the element name.
  - method : public : SetName(name : String) ˜Nil
- `SetNamespace` - Sets the element namespace.
  - method : public : SetNamespace(namespace : String) ˜Nil
- `GetRoot` - Returns the root element.
  - method : public : GetRoot() ˜XmlElement
- `GetType` - Returns the element type.
  - method : public : GetType() ˜XmlElementType
- `SetType` - Sets the element type.

  - method : public : SetType(type : XmlElementType) ~Nil

- `GetParent` - Gets the parent element
  - method : public : GetParent() ~XmlElement
- `SetParent` - Sets the parent element
  - method : public : SetParent(parent : XmlElement) ~Nil
- `GetContent` - Returns the XML content between two tags.
  - method : public : GetContent() ~String
- `SetContent` - Sets the XML content between two tags.
  - method : public : SetContent(content : String) ~Nil
- `GetChildCount` - Returns number of logical child elements.
  - method : public : GetChildCount() ~Int
- `FindElements` - returns all elements that match the given query string. The following XPath expressions are supported num, attrib_name, first() and last(). Note: current node name must be part of the query.
  - method : public : FindElements(path : String) ~Vector
- `GetChildren` - Returns all child elements.
  - method : public : GetChildren() ~Vector
- `GetChildren` - Returns all child elements that match the given name.
  - method : public : GetChildren(name : String) ~Vector
- `GetChild` - Returns a child element.
  - method : public : GetChild(i : Int) ~XmlElement
- `AddChild` - adds a child element.
  - method : public : AddChild(child : XmlElement) ~Nil
- `GetAttribute` - Returns an attribute.
  - method : public : GetAttribute(name : String) ~XmlAttribute

- `SetAttribute` - Sets an attribute.
  - method : public : SetAttribute(attrib : XmlAttribute), Nil
- `ToString` - Serializes an element into a string
  - method : public : ToString() ~String
- `DecodeString` - Decodes an XML encoded string
  - function : DecodeString(in : String) ~String
- `EncodeString` - Encodes an XML encoded string
  - function : EncodeString(in : String) ~String

7.11.4 XmlAttribute

The XmlAttribute class represents an XML attribute entity.

- `New`
  - New(name : String, value : String)
- `GetName` - Returns the attribute name.
  - method : public : GetName() ~String
- `SetName` - Sets the attribute name.
  - method : public : SetName(name : String) ~Nil
- `GetNamespace` - Returns the attribute namespace prefix.
  - method : public : GetNamespace() ~String
- `SetNamespace` - Sets the attribute namespace.
  - method : public : SetNamespace(namespace : String) ~Nil
- `GetValue` - Return the value
  - method : public : GetValue() ~String
- `SetValue` - Sets the value
  - method : public : SetValue(value : String) ~Nil

7.12 ODBC

Provides support for database access. This bundle is included in the 'odbc.obl' file.

7.12.1 Connection

The Connection class allow programmers to connect to ODBC sources.

- `New` - Establishes a database connection
  - New(ds : String, username : String, password : String)
- `IsOpen` - Returns true of the connection is open, false otherwise
  - method : public : IsOpen() ~Bool
- `Close` - Closes a database connection
  - method : public : Close() ~Nil
- `Update` - Executes an SQL update statement returning status code
  - method : public : Update(sql : String) ~Int
- `Select` - Executes an SQL select statement returning a result set
  - method : public : Select(sql : String) ~ResultSet
- `CreateParameterStatement` - Creates a prepared statements, which is used to pass parameters into SQL statements.
  - method : public : CreateParameterStatement(sql : String) ~ParameterStatement

7.12.2 ParameterStatement

The ParameterStatement class is used to set parameters for SQL queries.

- `Set` - Sets the values for various SQL data types. Returns true if the value has been set, false otherwise.
  - method : public : SetBit(pos : Int, value : Bool) ~Bool
  - method : public : SetSmallInt(pos : Int, value : Int) ~Bool
  - method : public : SetInt(pos : Int, value : Int) ~Bool

  - method : public : SetDouble(pos : Int, value : Float) ~Bool
  - method : public : SetReal(pos : Int, value : Float) ~Bool
  - method : public : SetVarchar(pos : Int, value : String) ~Bool
  - method : public : SetDate(pos : Int, value : Date) ~Bool
  - method : public : SetTimestamp(pos : Int, value : Timestamp) ~

- `Update` - Executes a SQL update statement that returns a status code
  - method : public : Update() ~Int
- `Select` - Executes a SQL select statement that returns a result set.
  - method : public : Update() ~Int
- `Close` - Closes the statement
  - method : public : Close() ~Nil

7.12.3 ResultSet

The ResultSet class holds the results of a query statement. A ResultSet contains 0 or more rows with column values that may be accessed via index or by name. Column values may be 'null'.

- `Next` - Moves the result set cursor to the next row.
  - method : public : Next() ~Bool
- `IsNull` - Returns true if the accessed column value is 'null'.
  - method : public : IsNull() ~Bool
- `Get` - Returns the column value for the given row.
  - method : public : GetInt(column : Int) ~Int
  - method : public : GetSmallInt(column : Int) ~Int
  - method : public : GetBit(column : Int) ~Bool
  - method : public : GetDouble(column : Int) ~Float
  - method : public : GetReal(column : Int) ~Float
  - method : public : GetVarchar(column : Int) ~String

  - method : public : GetVarchar(name : String) ~String
  - method : public : GetDate(column : Int) ~ODBC.Date
  - method : public : GetTimestamp(column : Int) ~Timestamp

- `Close` - Closes the result set.
  - method : public : Close() ~Nil

7.12.4 Date

The ODBC.Date class holds the values for a SQL date.

- `Get` - Gets various data values
  - method : public : GetYear() ~Int
  - method : public : SetYear(year : Int) ~Nil
  - method : public : GetMonth() ~Int
  - method : public : SetMonth(month : Int) ~Nil
  - method : public : GetDay() ~Int
  - method : public : SetDay(day : Int) ~Nil

7.12.5 Timestamp

The ODBC.Timestamp class holds the values for a SQL timestamp.

- `Get` - Gets various data values
  - method : public : GetYear() Int
  - method : public : SetYear(year : Int) Nil
  - method : public : GetMonth() Int
  - method : public : SetMonth(month : Int) Nil
  - method : public : GetDay() Int
  - method : public : SetDay(day : Int) Nil
  - method : public : GetHours() Int
  - method : public : SetHours(hours : Int) Nil
  - method : public : GetMinute() Int
  - method : public : SetMinute(minute : Int) Nil
  - method : public : GetSeconds() Int
  - method : public : SetSeconds(second : Int) Nil
  - method : public : GetFraction() Int

7.13 RegEx

Support for regular expressions. This bundle is included in the 'regex.obl' file.

7.13.1 RegEx

The RegEx class provides support for regular expressions parsing.

- *Expressions*
  - `\d \w \s` digit, word or white space
  - `.` match any 1
  - `?` optional 1
  - `*` match 0 or more
  - `+` match 1 or more
  - `a{2,4}` range
  - `|` or
  - `(abc)` grouping
  - `[abc]`, `[0-9]` classes
  - `^` or `$` starts with and ends wtih anchors
- `New` - Compiles the regular expression
  - New(pattern : String)
- `MatchExact` - Attempts to match the exact input string using supplied pattern. Returns true if successful, false otherwise.
  - method : public : MatchExact(input : String) ~Bool
- `Match` - Attempts to best match the input string using supplied pattern. Returns the index of the best match.
  - method : public : Match(input : String) ~Int
- `FindFirst` - Searches the input string for the best first match.
  - method : public : FindFirst(input : String) ~String
- `Find` - Searches the input string for all the matches. Returns a Vector of string matches.

  - method : public : Find(input : String) ~Vector

- `ReplaceFirst` - Searches and replaces the first best best match.
  - method : public : ReplaceFirst(input : String, replace : String) ~String

- `ReplaceAll` - Searches and replaces all best best matchs.
  - method : public : ReplaceAll(input : String, replace : String) ~String

7.14 Encryption

Support for encryption. This bundle is included in the 'encrypt.obl' file.

7.14.1 Hash

The Hash class provides support for one-way hashing.

- `SHA256` - SHA-256 hash
  - function : SHA256(input : Byte[]) ~Byte[]

- `RIPEMD160` - RIPEMD-160 hash
  - function : RIPEMD160(input : Byte[]) ~Byte[]

- `MD5` - MD5 hash
  - function : MD5(input : Byte[]) ~Byte[]

7.14.2 Encrypt

Provides support for encryption.

- `SHA256` - AES-256 encryption without "salt"
  - function : AES256(key : Byte[], input : Byte[]) ~Byte[]

- `Base64` - Base64 string encoding
  - function : Base64(input : String) ~String

7.14.3 Decrypt

Provides support for encryption.

- `SHA256` - AES-256 decryption without "salt"
  - function : AES256(key : Byte[], input : Byte[]) ˜Byte[]
- `Base64` - Base64 string decoding
  - function : Base64(input : String) ˜String

8 Examples

8.1 Prime Numbers

Demonstrates basic language features such as arithmetic and logical operations.

```
class FindPrime {
  function : Main() ~ Nil {
    Run(1000000);
  }

  function : native : Run(topCandidate : Int)~ Nil {
    candidate : Int := 2;
    while(candidate <= topCandidate) {
      trialDivisor : Int := 2;
      prime : Int := 1;

      found : Bool := true;
      while(trialDivisor * trialDivisor <= candidate & found) {
        if(candidate % trialDivisor = 0) {
          prime := 0;
          found := false;
        }
        else {
          trialDivisor := trialDivisor + 1;
        };
      };

      if(found) {
        candidate->PrintLine();
      };
      candidate := candidate + 1;
    };
  }
}
```

8.2 Arrays

Demonstrates the use of arrays.

```
class Transpose {
  function : Main(args : String[]) ~ Nil {
    input := [
      [1, 1, 1, 1]
      [2, 4, 8, 16]
      [3, 9, 27, 81]
      [4, 16, 64, 256]
      [5, 25, 125, 625]
    ];
    dim := input->Size();

    output := Int->New[dim[0],dim[1]];
    for(i := 0; i < dim[0]; i+=1;) {
      for(j := 0; j < dim[1]; j+=1;) {
        output[i,j] := input[i,j];
      };
    };
```

```
    Print(output);
  }

  function : Print(matrix : Int[,]) ~ Nil {
    dim := matrix->Size();
    for(i := 0; i < dim[0]; i+=1;) {
      for(j := 0; j < dim[1]; j+=1;) {
        IO.Console->Print(matrix[i,j])->Print('\t');
      };
      '\n'->Print();
    };
  }
}
```

8.3 Simple HTTP client

Demonstrates HTTP access.

```
use Collection;

class HttpTest {
  client := HttpClient->New();
    # enable cookies
    client->CookiesEnabled(true);
    # request creates a cookie
    lines := client->Get("http://www.rexswain.com/cgi-bin/cookie.cgi?create");
    each(i : lines) {
      line := lines->Get(i)->As(String)->PrintLine();
    };
    # request sends back cookie
    lines := client->Get("http://www.rexswain.com/cgi-bin/cookie.cgi");
    each(i : lines) {
      line := lines->Get(i)->As(String)->PrintLine();
    };
  }
}
```

8.4 XML Parsing and Querying

Demonstrates simple XML parsing.

```
use System.IO;
use XML;

class Test {
  function : Main(args : String[]) ~ Nil {
    in := String->New();
    in->Append("<inventory title=\"OmniCorp Store #45x10^3\">");
    in->Append("<section name=\"health\">");
    in->Append("<item upc=\"123456789\" stock=\"12\">");
    in->Append("<name>Invisibility Cream</name>");
    in->Append("<price>14.50</price>");
    in->Append("<description>Makes you invisible</description>");
    in->Append("</item>");
    in->Append("<item upc=\"445322344\" stock=\"18\">");
    in->Append("<name>Levitation Salve</name>");
```

```
    in->Append("<price>23.99</price>");
    in->Append("<description>Levitate yourself for up to 3 hours per application</description>");
    in->Append("</item>");
    in->Append("</section>");
    in->Append("<section name=\"food\">");
    in->Append("<item upc=\"485672034\" stock=\"653\">");
    in->Append("<name>Blork and Freen Instameal</name>");
    in->Append("<price>4.95</price>");
    in->Append("<description>A tasty meal in a tablet; just add water</description>");
    in->Append("</item>");
    in->Append("<item upc=\"132957764\" stock=\"44\">");
    in->Append("<name>Grob winglets</name>");
    in->Append("<price>3.56</price>");
    in->Append("<description>Tender winglets of Grob. Just add water</description>");
    in->Append("</item>");
    in->Append("</section>");
    in->Append("</inventory>");

    parser := XmlParser->New(in);
    if(parser->Parse()) {
      # get first item
      results := parser->FindElements("/inventory/section[1]/item[1]");
      if(results <> Nil) {
        Console->Print("items: ")->PrintLine(results->Size());
      };
      # get all prices
      results := parser->FindElements("/inventory/section/item/price");
      if(results <> Nil) {
        each(i : results) {
          element := results->Get(i)->As(XmlElement);
          element->GetContent()->PrintLine();
        };
      };
      # get names
      results := parser->FindElements("/inventory/section/item/name");
      if(results <> Nil) {
        Console->Print("names: ")->PrintLine(results->Size());
      };
    };
  }
}
```

8.5 Echo Server

Demonstrates usage of server sockets and threads.

```
use System.IO.Net;
use System.Concurrency;

bundle Default {
  class SocketServer {
    id : static : Int;

    function : Main(args : String[]) ~ Nil {
      server := TCPSocketServer->New(12321);
      if(server->Listen(5)) {
        while(true) {
          client := server->Accept();
          service := Service->New(id->ToString());
```

```
        service->Execute(client);
        id += 1;
      };
    };
    server->Close();
  }
 }

 class Service from Thread {
  New(name : String) {
   Parent(name);
  }

  method : public : Run(param : Base) ~ Nil {
   client := param->As(TCPSocket);
   line := client->ReadString();
   while(line->Size() > 0) {
    line->PrintLine();
    line := client->ReadString();
   };
  }
 }
}
```

9 Appendix A: Sample Debugging Session

9.1 Source for Example

```
bundle Default {
  class Bar {
    v1 : Float;
    v2 : Int;

    New() {
      v1 := 2.31;
      v2 := 26;
    }
  }

  class Foo {
    bar : Bar;
    value : Int;

    New(v : Int) {
      value := v;
    }

    method : public : Get() ~ Int {
      return value;
    }

    method : public : SetBar() ~ Nil {
      bar := Bar->New();
    }
  }

  class Test {
    function : Main(args : System.String[]) ~ Nil {
      d : Float := 11.12;
      z := Int->New[5,6];
      z[2,3] := 27;

      f := Foo->New(24);
      f->SetBar();
```

```
      v := f->Get();
    }
  }
}
```

The sample file is named `debug.obs`.

9.2 Compiling the Source and Starting the Debugger

```
obc -src test_src\debug.obs -dest a.obe -debug
obd -exe ..\..\compiler\a.obe -src ..\..\compiler\test_src

-------------------------------------
Objeck v1.0.0 - Interactive Debugger
-------------------------------------
loaded executable: file='../../compiler/a.obe'
source files: path='../../compiler/test_src/'
```

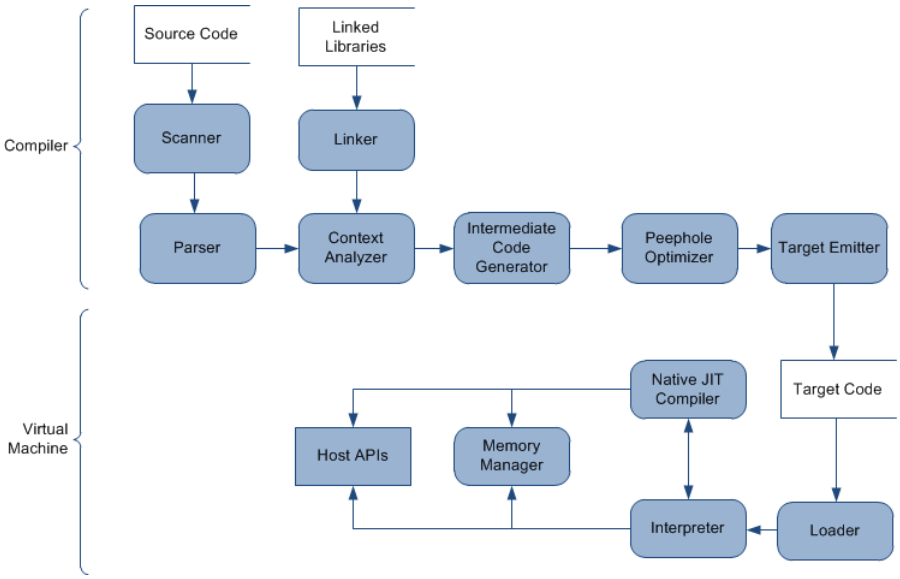
9.3 Setting a Breakpoint and Running the Program

```
> b debug.obs:31
added break point: file='debug.obs:31'
> r
break: file='debug.obs:31', method='Test->Main(..)'
> l
  List
     26:  }
     27:  }
     28:
     29:  class Test {
     30:  function : Main(args : System.String[]) ~ Nil {
=>   31:  d : Float := 11.12;
     32:  z := Int->New[5,6];
     33:  z[2,3] := 27;
     34:
     35:  f := Foo->New(24);
     36:  f->SetBar();
> n
break: file='debug2.obs:32', method='Test->Main(..)'
```

9.4 Printing a Value

```
> p d
print: type=Float, value=11.12
> b debug.obs:37
added break point: file='debug.obs:37'
> c
break: file='debug2.obs:37', method='Test->Main(..)'
> p z
print: type=Int[], value=2197556(0x218834), dimension=2, size=30
> p z[2,3]
print: type=Int[], value=27(0x1b)
>  p f->value
print: type=Int, value=24
> p f->bar
print: type=Bar, value=0x218864
> p f->bar->v1
print: type=Float, value=2.31
> q
> p f->bar->v1 * 3.5
print: type=Float, value=8.085
goodbye.
```

10 Appendix B: Compiler and VM Design



The following section gives a brief overview of the major architectural components the comprise the Objeck language compiler and virtual machine.

10.1 Compiler

The language compiler is written in C++ and makes heavy use of the C++ STL for portability across platforms. As mentioned in the introduction, the compiler accepts source files and shared libraries as inputs and produces either executables or shared libraries. Note, the compiler has two modes of operation: `User Mode` compiles traditional end-user programs, while `System Mode` compiles system libraries and processes special system language directives.

10.1.1 Scanner and Parser

The scanner component reads source files and parses the text into tokens. The scanner works in conjugation with the *LL(k)* parser by providing k lookahead tokens for parsing. Note, the scanner can only scan system language directives while in `System Mode`. The source parser is a recursive-decent parser that generates an abstract parser tree, which is passed to the Contextual Analyser for validation.

10.1.2 Contextual Analyser

The Contextual Analyzer is responsible for ensuring that a source program is valid. In addition, the context analyzer also creates relationships between contextually resolved entities (i.e. methods $\longleftrightarrow$ method calls). The analyzer accepts an abstract parser tree and shared libraries as input and produces a decorated parse tree as output. The decorated parse tree is then passed to the Intermediate Code Generator for the production of VM instructions.

10.1.3 Intermediate Code Generator and Optimzier

The Intermediate Code Generator accpets a decorated parse and produces a flat list of VM stack instructions. These instruction lists are then passed to the Optimizer for basic block optimizations (constant folding, strength reduction, instruction simplification and method inlining).

10.1.4 Target Emitter

Finally, the improved intermediate code is passed to code emitter component, which writes it to a file.

10.2 Virtual Machine

The language VM is written in C/C++ and was designed to be highly portable. The VM makes heavy use of operating system specific APIs (i.e. WIN32 and POSIX) but does so in an abstracted manner. The JIT compiler is targeted to produce machine code for the IA-32 and AMD64 hardware architectures.

10.2.1 Loader

The loader component allows the VM to read target code structures such as classes, methods and VM instructions. The loader create an in-memory representation of this information, which is used by the VM interpreter and JIT compiler. In addition, the loader processes command-line parameters that are passed into the VM prior to execution.

10.2.2 Interpreter

The Interpreter executes stack based VM instructions (listed below) and manages two primary stacks: the execution stack and call stack. The execution stack is used to manage the data that is needed for VM calculations.

The call stack is used to manage function/method calls and the states between those calls.

10.2.3 JIT Compiler

The JIT compiler translates stack based VM instructions into processor specific machine code (i.e. IA-32 and AMD64). The JIT compiler is evoked by the interpreter and methods are translated into machine code and cached for subsequent calls.

10.2.4 Memory Manager

The Memory Manager component allows the runtime system to manage the user allocation/deallocation of heap memory. The memory mangers implements a multi-thread "mark and sweep" algorithm. The marking stage of the process is multi-thread, such that, each root in scanned in a separate thread. The sweeping stage is done in a single thread since runtime structures are modified.

11 Appendix C: VM Instruction Set

The appendix below lists the types of stack instructions that are executed by the Objeck VM. The VM was designed to be portable and language independent. Early development versions of the VM included an inline assembler, which may be re-added in a future release.

Stack Operators		
Mnemonic	*Opcode(s)*	*Description*
LOAD_INT_LIT	4-byte integer	pushes integer onto stack
LOAD_FLOAT_LIT	8-byte float	pushes float onto stack
LOAD_INT_VAR	variable index	pushes integer onto stack
LOAD_FLOAT_VAR	variable index	pushes float onto stack
LOAD_FUNC_VAR	variable index	pushes float onto stack
LOAD_SELF	n/a	pushes self integer on stack
STOR_INT_VAR	variable index	pops integer from stack and saves to index location
STOR_FLOAT_VAR	variable index	pops float from stack and saves to index location
STOR_FUNC_VAR	variable index	pops float from stack and saves to index location
COPY_INT_VAR	variable index	copies an integer from stack and saves to index location
COPY_FLOAT_VAR	variable index	copies a float from stack and saves to index location
LOAD_BYTE_ARY_ELM	array dimension	pushes byte onto stack; assumes array address was pushed prior
LOAD_INT_ARY_ELM	array dimension	pushes integer onto stack; assumes array address was pushed prior
LOAD_FLOAT_ARY_ELM	array dimension	pushes float onto stack; assumes array address was pushed prior
LOAD_ARY_SIZE	n/a	pushes array size as integer onto stack; assumes array address was pushed prior
STOR_BYTE_ARY_ELM	variable index	stores byte at index location; assumes array address was pushed prior
STOR_INT_ARY_ELM	variable index	stores integer at index location ; assumes array address was pushed prior
STOR_FLOAT_ARY_ELM	variable index	stores float at index location; assumes array address was pushed prior

Logical Operators		
Mnemonic	*Opcode(s)*	*Description*
EQL_INT	n/a	pops top two integer values and pushes result of equal operation
NEQL_INT	n/a	pops top two integer values and pushes result of not-equal operation
LES_INT	n/a	pops top two integer values and pushes result of less-than operation
GTR_INT	n/a	pops top two integer values and pushes result of greater-than operation
LES_EQL_INT	n/a	pops top two integer values and pushes result of less-than-equal operation
GTR_EQL_INT	n/a	pops top two integer values and pushes result of greater-than-equal operation
EQL_FLOAT	n/a	pops top two floats values and pushes result of equal operation
NEQL_FLOAT	n/a	pops top two floats values and pushes result of not-equal operation
LES_FLOAT	n/a	pops top two floats values and pushes result of less-than operation
GTR_FLOAT	n/a	pops top two floats values and pushes result of greater-than operation
LES_EQL_FLOAT	n/a	pops top two floats values and pushes result of less-than-equal operation
GTR_EQL_FLOAT	n/a	pops top two floats values and pushes result of greater-than-equal operation

Logical Operators		
Mnemonic	*Opcode(s)*	*Description*
AND_INT	n/a	pops top two integer values and pushes result of add operation
OR_INT	n/a	pops top two integer values and pushes result of or operation

Mathematical Operators		
Mnemonic	*Opcode(s)*	*Description*
ADD_INT	n/a	pops top two integer values and pushes result of add operation
SUB_INT	n/a	pops top two integer values and pushes result of subtract operation
MUL_INT	n/a	pops top two integer values and pushes result of multiply operation
DIV_INT	n/a	pops top two integer values and pushes result of divide operation
SHL_INT	n/a	pops top two floats values and pushes result of shift left operation
SHR_INT	n/a	pops top two floats values and pushes result of shift right operation
MOD_INT	n/a	pops top two integer values and pushes result of modulus operation
ADD_FLOAT	n/a	pops top two floats values and pushes result of greater-than-equal operation
SUB_FLOAT	n/a	pops top two floats values and pushes result of subtract operation
MUL_FLOAT	n/a	pops top two floats values and pushes result of multiply operation
DIV_FLOAT	n/a	pops top two floats values and pushes result of divide operation
I2F	n/a	pop top integer and pushes result of float cast
F2I	n/a	pop top float and pushes result of integer cast

Methods		
Mnemonic	*Opcode(s)*	*Description*
SWAP_INT	n/a	swaps the top two integer values on the stack
POP_INT	n/a	control pop of an integer from the stack
POP_FLOAT	n/a	control pop of a float from the stack
RTRN	n/a	exits existing method returning control to callee
MTHD_CALL	integer values for class id and method id	synchronous call to given method releasing control
DYN_MTHD_CALL	pops integer values for class id and method id	dynamic synchronous call to given method releasing control
ASYNC_MTHD_CALL	integer values for class id and method id; pushes new thread id	asynchronous call to given method
ASYNC_JOIN	thread id	waits for identified thread to end execution
LBL	label id	identifies a jump label

Objects and Memory Operations		
Mnemonic	*Opcode(s)*	*Description*
JMP	label id and conditional context (1=true, 0=unconditional, -1=false)	jump to label id
NEW_BYTE_ARY	array dimension	pushes address of new byte array
NEW_INT_ARY	array dimension	pushes address of new integer array
NEW_FLOAT_ARY	array dimension	pushes address of new float array
NEW_OBJ_INST	integer value for class id	pushes address of new class instance
OBJ_TYPE_OF	integer value of "check" class	performs runtime object instance check
OBJ_INST_CAST	integer values for "from" class and "to" class	performs runtime class cast check (note: only required for up casting)
CPY_BYTE_ARY	destination of source array, offset of destination array, address of source array, offset of source array, number of	copies elements from one array to another
CPY_INT_ARY	destination of source array, offset of destination array, address of source array, offset of source array, number of	copies elements from one array to another
CPY_FLOAT_ARY	destination of source array, offset of destination array, address of source array, offset of source array, number of	copies elements from one array to another

Traps and Threads		
Mnemonic	*Opcode(s)*	*Description*
THREAD_CREATE	n/a	creates an new thread instance (calculation stack and stack pointer)
THREAD_WAIT	n/a	waits for worker threads to stop execution
CRITICAL_START	n/a	creates a mutex such that only one thread can execute in a given section
CRITICAL_END	n/a	releases a system mutex
TRAP	integer value for trap id	calls runtime subroutine releasing control
TRAP_RTRN	integer value for trap id and number of arguments	calls runtime subroutine releasing control and then processes an integer return value
LIB_NEW_OBJ_INST	n/a	symbolic library link for a new object instance
LIB_MTHD_CALL	n/a	symbolic library link for a method call
LIB_OBJ_INST_CAST	n/a	symbolic library link for an object cast