

Intrinsic Value

Alex Kodat
Sirius Software Inc.



Sirius Software, Inc.

Object-Oriented Programming

- Objects defined by classes contain potentially complex collections of related data
 - Usually maps to “real world” objects and classes of objects
- Rather than calling functions or subroutines, methods are applied to object
 - Syntax is **%object:method** rather than **\$function(%object)** or **call subroutine(%object)**
 - But shared methods don't really apply to objects so they use a syntax where they apply to a class



Pure O-O For Now People

- **Everything** is an object
 - Even strings and numbers
- All function/subroutine/property calls use the O-O %object:method syntax
- Initial driving force for Model 204 was comment on m204-1 from Don Essick



Strings and Numbers as Objects

- It's easy to think of strings and numbers as objects
- But a special kind of object: *immutable objects*
 - Methods cannot modify an instance of an object
 - It would be surprising if %x and %y are strings and, after setting %x = %y, some operation on %x would modify the value of %y
 - This is not surprising for more complex objects
- Even languages that say strings and numbers are objects treat them specially internally
 - For efficiency
 - So strings and numbers as “objects” is largely a fiction
 - But, what the heck, let's pretend strings and numbers are objects in User Language, too



Applying Methods to Strings and Numbers

- Why not?
- Sirius Mods 7.2 supports %string:method and %number:method syntax
- These methods are called “intrinsic methods” and strings and numbers are called “intrinsic classes”
 - Though this is documentation-only – the word “intrinsic” doesn't appear in User Language
 - Maybe “primitive” would have been a better term
 - But we were worried that it sounded pejorative
 - So, naturally, Java has started calling strings and numbers primitive objects



Example of Intrinsic Methods

```
%string = 'Whatever'
```

```
for %i from 1 to %string:length  
    print %string:left(%i)  
end for
```

```
* Will print  
*  
* W  
* Wh  
* Wha  
* What  
* Whate  
* Whatev  
* Whateve  
* Whatever
```



So Why Use Intrinsic Methods?

- Because that's what all the cool kids are doing
- Intrinsic methods (like other methods) operate left to right rather than inside out (like \$functions)
 - So easier to read, write, and understand code
- Brings benefits of other classes to strings and numbers
 - Named variables
 - Exceptions
- This is Sirius's direction
 - All new development will be intrinsic methods rather than \$functions



Example of Intrinsic Methods with Named Parameters

```
%string = 'Whatever'
```

```
print %string:right(12, pad='.')
```

```
* Will print
```

```
*
```

```
* ....Whatever
```



Example of Chained Intrinsic Methods

```
%string = 'F0F1F2F3'
```

```
print %string:right(6):hexToString
```

```
* Will print
```

```
*
```

```
* 123
```



Things To Which Intrinsic Methods Can Apply

- %variables - **%string:right(10)**
- Literals – **'This is a test':right(10)**
- Expression results
 - Method results - **%stringlist:item(5):right(10)**
 - \$function results - **\$time:right(10)**
 - Operator results - **(%x with %y):right(10)**
- Fields - **(rectype):right(10)**
- Image and screen items - **%image:item:right(10)**



Some Caveats

- Field names must be in parentheses before an intrinsic method name
 - Because field names can contain colons and blanks
- If screen/image has same name as a %variable in a context, the %variable must be put inside parentheses (or have a blank separator) to use
 - Imagine image Foo and variable %foo in the same context
 - What does %foo:something mean?
- Print statement flaky with intrinsic methods
 - Well, it's flaky, anyway
 - So use the PrintText, AuditText, and TraceText statements



A Digression – PrintText and Friends

- A shortened form of the Text Data statement
 - With AuditText and TraceText versions
- Useful as a consistent form of the Print, Audit, Trace statements
 - Text always treated as literal
 - Except for parts inside curly braces {} which are treated as expressions
- Useful tilde (~) operator inside curly braces
 - Displays literal of the next expression



Examples of PrintText and AuditText

```
printText It's nice not to have to worry about quotes
```

```
printText %i * %j = {%i * %j}
* The previous can also be written as
printText {~} = {%i * %j}
```

```
auditText {~} = '{%data}', {~} = {%data:stringToHex}
```

```
* Print our multiplication table
for %i from 1 to 10
  for %j from 1 to 10
    * We probably don't want the following:
    printText {~} = {%i * %j}
    * We probably mean to do
    printText {%i} * {%j} = {%i * %j}
  end for
end for
```



Weak Data Typing and Intrinsic Methods

- One of the strengths of User Language is the automatic conversion of strings to numbers and numbers to strings
 - This is quite intuitive
 - Is '3.14' a number or a string?
 - It depends on the context and it's nice for User Language to deal with the conversion
 - Though conversion of non-numbers to zero is (IMO) a mistake
 - Request cancellation would have been better
- But an intrinsic method operates on either a string or a number
 - So numeric and string intrinsic method names cannot have the same name
 - Fortunately, this is rarely a problem



The Implicit Intrinsic Classes

- Longstring
 - All intrinsic string methods have longstring capabilities and semantics
- Float
 - Model 204 Floats have much nicer behavior than Floats in other environments
 - So Floats can be thought of as a generic numeric data type
 - In fact, they're actually (slightly) more efficient to use than Fixed variables, even for integer data!
 - So, for most purposes, use Float for numeric variables
- Fixed
 - Theoretical, currently no Fixed intrinsic methods
 - Will there ever be?



Examples of Weak Data Typing with Intrinsic Methods

```
printText {(%i * %j):right(10, pad='0')}
```

```
printText {(%i with %j):squareRoot}
```



Useful Starter Intrinsic System Methods

- Substring – piece of a string
- Length – length of a string
- Left and Right – left or right piece of string
- PositionOf and PositionIn – find string inside another
- HexToString and StringToHex – Convert from/to hex
- ParseLines – Split string into stringlist
- Base64ToString and StringToBase64 – Convert from/to base64 encoding
- RegexMatch, RegexSplit, and RegexReplace – Regular expression operations on strings
- SquareRoot – So far, the only Float intrinsic method
- Many more coming in Sirius Mods 7.3



Intrinsic Enhancement Methods

- Intrinsic methods written in User Language
 - So you can write User Language methods that operate on strings and numbers
- But we need to talk about enhancement methods, first
 - Which we will in the next presentation



Summary

- Intrinsic methods make it possible to use User Language as a “pure” object-oriented programming language
 - Without any backward compatibility issues
- Intrinsic methods apply to all existing User Language strings and numbers
 - So migration to new syntax is easy
- All the cool kids are doing it
 - You **are** cool... aren't you?



Let's Play Around with Intrinsic Methods a Bit



Sirius Software, Inc.