GALGAS Book

Version 2.1.3

Pierre Molinaro

December 7, 2010

Contents

Chapter 1

Predefined Types

GALGAS predefines several types. This chapter presents all their features, including their constructors, readers, modifiers, methods, ...

The predefined types are:

- @double type (page 7), floating point numbers;
- @location type (page 10), whose value points out a location in a source file;
- @sint type (page 12), the 32-bit signed integers;
- @sint64 type (page 17), the 64-bit signed integers;
- @uint type (page 21), the 32-bit unsigned integers;
- @uint64 type (page 28), the 64-bit unsigned integers.

1.1 The `@double` Type

The `@double` object values correspond to the C type `double` values. You can initialize an `@double` object from a float constant:

```
@double myDouble := 123.456 ;
```

Note that a `@double` constant is characterized by the occurrence of the decimal point (.)

1.1.1 `sint` Reader

Returns returns the receiver's value in an @sint type (page 12) (32-bit signed integer) object.

```
reader @double sint -> @sint ;
```

Availabilty: available in GALGAS 1.9.9 and later.

Discussion: if receiver?s value is outside `@sint` bounds, a runtime error is raised.

1.1.2 `sint64` Reader

Returns returns the receiver's value in an @sint64 type (page 17) (64-bit signed integer) object.

```
reader @double sint64 -> @sint64 ;
```

Availabilty: available in GALGAS 1.9.9 and later.

Discussion: if receiver?s value is outside `@sint64` bounds, a runtime error is raised.

1.1.3 `string` Reader

Returns returns a decimal string representation of the receiver's value.

```
reader @double string -> @string ;
```

Availabilty: available in GALGAS 1.7.7 and later.

Discussion: this reader never fails.

1.1.4 `uint` Reader

Returns returns the receiver's value in an @uint type (page 21) (32-bit unsigned integer) object.

```
reader @double uint -> @uint ;
```

Availabilty: available in GALGAS 1.9.9 and later.

Discussion: if receiver?s value is outside `@uint` bounds, a runtime error is raised.

1.1.5 `uint64` Reader

Returns returns the receiver's value in an @uint64 type (page 28) (64-bit unsigned integer) object.

```
reader @double uint64 -> @uint64 ;
```

Availabilty: available in GALGAS 1.9.9 and later.

Discussion: if receiver?s value is outside `@uint64` bounds, a runtime error is raised.

1.1.6 Arithmetic Operators

The `@double` type supports the five arithmetic diadic operators:

+	Addition
−	Substraction
*	Multiplication
/	Division
%	Modulo

Theses operators require both arguments to be `@double` objects.

A run-time error is raised if the operation leads to an overflow.
The `@double` type supports the following arithmetic unary operators:

+	No operation
−	Negate

This operator returns the receiver's value (an `@double` object).

1.1.7 Comparison Operators

The `@double` type supports the six comparison operators:

=	Equality
!=	Non Equality
<	Strict Lower Than
<=	Lower or Equal
>	Strict Greater Than
>=	Greater or Equal

Theses operators require both arguments to be `@double` objects, and return a `@bool` object.

1.2 The `@location` Type

An `@location` object value is a location in a source file. Objects of this type are useful for pointing out an error or a warning location.

1.2.1 The `here` Keyword

The `here` keyword indicates the current parsing location is the current source file. Assigning an `@location` object from the `here` keyword is a way for initializing an `@location` object:

```
@location currentLocation := here ;
```

1.2.2 `nowhere` Constructor

Returns an `@location` that does not points out any location.

```
constructor @location nowhere -> @location ;
```

Availabilty: available in GALGAS 2.1.2 and later.

Discussion: The returned object responds `true` to the `isNowhere` reader (page 11).

1.2.3 `column` Reader

Returns an `@uint` value containing the column of the receiver's value.

```
reader @location column -> @uint ;
```

Availabilty: available in GALGAS 1.8.2 and later.

Discussion: this reader raises a run-time error if the receiver's value responds `true` to the `isNowhere` reader (page 11).

1.2.4 `isNowhere` Reader

Returns an `@bool` value indicating whether the receiver'value points out a source location or does not.

```
reader @location isNowhere -> @bool ;
```

Availabilty: available in GALGAS 2.1.2 and later.

Discussion: this reader returns `true` if the receiver's value does not point out an actual location in a text source (i.e. it has been constructed using the nowhere constructor), and `false` if the receiver's value points out an actual location in a text source (i.e. it has been constructed using the `here` keyword.

1.2.5 `line` Reader

Returns an `@uint` value containing the line of the receiver's value.

```
reader @location line -> @uint ;
```

Availabilty: available in GALGAS 1.8.2 and later.

Discussion: this reader raises a run-time error if the receiver's value responds `true` to the `isNowhere` reader (page 11).

1.2.6 `locationIndex` Reader

Returns an `@uint` value containing the the offset from the the beginning of the source of the location defined by receiver's value.

```
reader @location locationIndex -> @uint ;
```

Availabilty: available in GALGAS 1.8.2 and later.

Discussion: this reader raises a run-time error if the receiver's value responds `true` to the `isNowhere` reader (page 11).

1.2.7 `locationString` Reader

returns an `@string` object that contains a string representation of the location defined by receiver's value.

```
reader @location locationString -> @string ;
```

Availabilty: available in GALGAS 1.8.2 and later.

Discussion: this reader raises a run-time error if the receiver's value responds `true` to the `isNowhere` reader (page 11).

1.3 The `@sint` Type

An `@sint` object value is a 32-bit signed integer value. You can initialize an `@sint`' object from an 32-bit signed integer constant:

```
@sint mySignedInteger := 123_456S ;
```

Note that a 32-bit signed integer constant is characterized by the 'S' suffix.

1.3.1 `min` Constructor

Returns an `@sint` object that the minimum value of the 32-bit signed range.

```
constructor @sint min -> @sint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: the returned value is -2^{31}.

1.3.2 `max` Constructor

Returns an `@sint` object that the maximum value of the 32-bit signed range.

```
constructor @sint max -> @sint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: the returned value is $2^{31} - 1$.

1.3.3 `double` Reader

Returns the receiver's value converted in a `@double` object.

```
reader @sint double -> @double ;
```

Availabilty: available in GALGAS 1.9.8 and later.

Discussion: as a 32-bit integer value can always be converted in a `@double` value, this reader never fails.

1.3.4 `sint64` Reader

Returns returns the receiver's value in an @sint64 type (page 17) (64-bit signed integer) object.

```
reader @sint sint64 -> @sint64 ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: as a 32-bit signed value can always be converted in a 64-bit signed value, this reader never fails.

This reader is the only way to convert an @sint type (page 12) object into an @sint64 type (page 17) object.

1.3.5 string Reader

Returns returns a decimal string representation of the receiver's value.

```
reader @sint string -> @string ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see hexString reader (page 23) and xString reader (page 26).

1.3.6 uint Reader

Returns returns the receiver's value in an @uint type (page 21) (32-bit unsigned integer) object.

```
reader @sint uint -> @uint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: an error is raised is receiver's value is negative.

This reader is the only way to convert an @sint type (page 12) object into an @uint type (page 21) object.

1.3.7 uint64 Reader

Returns returns the receiver's value in an @uint64 type (page 28) (64-bit unsigned integer) object.

```
reader @sint uint64 -> @uint64 ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: an error is raised is receiver's value is negative.

This reader is the only way to convert an @sint type (page 12) object into an @uint64 type (page 28) object.

1.3.8 Incrementation and decrementation

The @sint type (page 12) supports incrementation and decrementation instructions.

```
@sint n := ...  ; n ++ ; # Incrementation
@sint p := ...  ; p - ; # Decrementation
```

The incrementation instruction raises an error if receiver's value is equal to $2^{31} - 1$.

The decrementation instruction raises an error if receiver's value is equal to -2^{31}.

Note that incrementation and decrementation are not available within an expression.

1.3.9 Arithmetic Operators

The `@sint` type supports the five arithmetic diadic operators:

+	Addition
−	Substraction
*	Multiplication
/	Division
%	Modulo

Theses operators require both arguments to be `@sint` objects.

A run-time error is raised if the operation leads to a 32-bit signed overflow.

The `@sint` type supports the following arithmetic unary operators:

+	No operation
−	Negate

This operator returns the receiver's value (an `@sint` object). A run-time error is raised if "-" operator is invoked on an object whose value is -2^{31}.

1.3.10 Shift Operators

The `@sint` type supports right and left shift operators:

<<	Left shift
>>	Right shift

Theses operators require the right argument to be `@sint` object, and the left argument to be `@uint` object.

Note the right shift inserts a zero bit in the most significant bit location if the receiver's value is negative, and a one bit otherwise (it is a arithmetic right shift).

The actual amount of the shift is the value of the right-hand operand masked by 31, i.e. the shift distance is always between 0 and 31.

1.3.11 Logical Operators

The `@sint` type supports the three bit-wise logical operators:

&	Bit-wise and
|	Bit-wise or
^	Bit-wise exclusive or

Theses operators require both arguments to be `@sint` objects.

The `@sint` type supports the bit-wise logical unary operator:

~	Bit-wise complementation

This operator returns an `@sint` object.

1.3.12 Comparison Operators

The `@sint` type supports the six comparison operators:

=	Equality
!=	Non Equality
<	Strict Lower Than
<=	Lower or Equal
>	Strict Greater Than
>=	Greater or Equal

Theses operators require both arguments to be `@sint` objects, and return a `@bool` object.

1.4 The `@sint64` Type

An `@sint64` object value is a 64-bit signed integer value. You can initialize an `@sint64`' object from an 64-bit signed integer constant:

```
@sint64 mySignedInteger := 123_456LS ;
```

Note that a 64-bit signed integer constant is characterized by the 'LS' suffix.

1.4.1 `min` Constructor

Returns an `@sint64` object that the minimum value of the 64-bit signed range.

```
constructor @sint64 min -> @sint64 ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: the returned value is -2^{63}.

1.4.2 `max` Constructor

Returns an `@sint64` object that the maximum value of the 32-bit signed range.

```
constructor @sint64 max -> @sint64 ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: the returned value is $2^{63} - 1$.

1.4.3 `double` Reader

Returns the receiver's value converted in a `@double` object.

```
reader @sint64 double -> @double ;
```

Availabilty: available in GALGAS 1.9.8 and later.

Discussion: as a 64-bit integer value can always be converted in a `@double` value, this reader never fails.

1.4.4 `sint` Reader

Returns returns the receiver's value in an @sint type (page 12) (32-bit signed integer) object.

```
reader @sint64 sint -> @sint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: an error is raised is receiver's value is lower than -2^{31} or greater than $2^{31} - 1$.

This reader is the only way to convert an @sint64 type (page 17) object into an @sint type (page 12) object.

1.4.5 `string` Reader

Returns returns a decimal string representation of the receiver's value.

```
reader @sint64 string -> @string ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: this reader never fails.

1.4.6 `uint` Reader

Returns returns the receiver's value in an @uint type (page 21) (32-bit unsigned integer) object.

```
reader @sint64 uint -> @uint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: an error is raised is receiver's value is negative or greater than

$2^{32} - 1$.

This reader is the only way to convert an @sint64 type (page 17) object into an @uint type (page 21) object.

1.4.7 uint64 Reader

Returns returns the receiver's value in an @uint64 type (page 28) (64-bit unsigned integer) object.

```
reader @sint64 uint64 -> @uint64 ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: this reader raises a run-time error if the receiver's value is negative.

This reader is the only way to convert an @sint64 type (page 17) object into an @uint64 type (page 28) object.

1.4.8 Incrementation and decrementation

The @sint64 type (page 17) supports incrementation and decrementation instructions.

```
@sint64 n := ...  ; n ++ ; # Incrementation
@sint64 p := ...  ; p - ; # Decrementation
```

The incrementation instruction raises an error if receiver's value is equal to $2^{63} - 1$.

The decrementation instruction raises an error if receiver's value is equal to -2^{63}.

Note that incrementation and decrementation are not available within an expression.

1.4.9 Arithmetic Operators

The @sint64 type supports the five arithmetic diadic operators:

+	Addition
−	Substraction
*	Multiplication
/	Division
%	Modulo

Theses operators require both arguments to be `@sint64` objects.

A run-time error is raised if the operation leads to a 64-bit signed overflow.

The `@sint` type supports the following arithmetic unary operators:

+	No operation
−	Negate

This operator returns the receiver's value (an `@sint` object). A run-time error is raised if "-" operator is invoked on an object whose value is -2^{63}.

1.4.10 Shift Operators

The `@sint64` type supports right and left shift operators:

<<	Left shift
>>	Right shift

Theses operators require the right argument to be `@sint64` object, and the left argument to be `@uint` object.

Note the right shift inserts a zero bit in the most significant bit location if the receiver's value is negative, and a one bit otherwise (it is a arithmetic right shift).

The actual amount of the shift is the value of the right-hand operand masked by 63, i.e. the shift distance is always between 0 and 63.

1.4.11 Logical Operators

The `@sint64` type supports the three bit-wise logical operators:

&	Bit-wise and
|	Bit-wise or
^	Bit-wise exclusive or

Theses operators require both arguments to be `@sint64` objects.

The `@sint64` type supports the bit-wise logical unary operator:

$\sim$	Bit-wise complementation

This operator returns an `@sint64` object.

1.4.12 Comparison Operators

The `@sint64` type supports the six comparison operators:

=	Equality
! =	Non Equality
<	Strict Lower Than
<=	Lower or Equal
>	Strict Greater Than
>=	Greater or Equal

Theses operators require both arguments to be `@sint64` objects, and return a `@bool` object.

1.5 The `@uint` Type

An `@uint` object value is a 32-bit unsigned integer value. You can initialize an `@uint`' object from an unsigned integer constant:

```
@uint myUnsignedInteger := 123_456 ;
```

Note that a 32-bit unsigned integer constant is characterized by no suffix.

1.5.1 `errorCount` Constructor

Returns an `@uint` object that contains the number of errors.

```
constructor @uint errorCount -> @uint ;
```

Availabilty: available in GALGAS 1.4.9 and later.

Discussion: The returned value is the cumulative count of errors from the beginning of execution.

Example: `@uint x := [@uint errorCount] ;`

1.5.2 `max` Constructor

Returns an `@uint` object that the maximum value of the 32-bit unsigned range.

```
constructor @uint max -> @uint ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: The returned value is $2^{32} - 1$ (4294967295).

1.5.3 `valueWithMask` Constructor

Returns an `@uint` object with bits from *inLowerIndex* to *inUpperIndex* equal to 1.

```
constructor @uint valueWithMask
  ?@uint inLowerIndex
  ?@uint inUpperIndex
  -> @uint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: a run-time error is raised if *inLowerIndex* > *inUpperIndex* or if *inUpperIndex* > *31*.

Example: `@uint x := [@uint valueWithMask !2 !4] ; # x is equal to 28 (11100 in binary)`

1.5.4 `warningCount` Constructor

Returns an `@uint` object that contains the number of warnings.

```
constructor @uint warningCount -> @uint ;
```

Availabilty: available in GALGAS 1.4.9 and later.

Discussion: The returned value is the cumulative count of warnings from the beginning of execution.

1.5.5 `double` Reader

Returns the receiver's value converted in a `@double` object.

```
reader @uint double -> @double ;
```

Availabilty: available in GALGAS 1.9.8 and later.

Discussion: as a 32-bit integer value can always be converted in a `@double` value, this reader never fails.

1.5.6 `hexString` Reader

Returns the an hexadecimal string representation of the receiver value, prefixed by the string "0x".

```
reader @uint hexString -> @string ;
```

Availabilty: available in GALGAS 1.5.2 and later.

Discussion: for getting an hexadecimal representation string without "0x" prefix, see `xString` reader (page 26).

1.5.7 `isUnicodeValueAssigned` Reader

Returns an `@bool` value indicating whether the receiver'value represents an assigned Unicode character.

```
reader @uint isUnicodeValueAssigned -> @bool ;
```

Availabilty: available in GALGAS 1.8.3 and later.

Discussion: it returns `true` if the receiver value represents an assigned Unicode character, `false` and otherwise.

Example:

```
[0xFFFF isUnicodeValueAssigned] # is false, as \uFFFF is not assigned.
[0x41 isUnicodeValueAssigned] # is true, as \u0041 is assigned (LATIN
CAPITAL LETTER A).
```

1.5.8 `lsbIndex` Reader

Returns an `@uint` value of the index of the most significant bit of the receiver value.

```
reader @uint lsbIndex -> @uint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: it raises a run-time error if the receiver value is zero.

Example :

```
@uint value := 192 ; # 192 is 11000000 in binary
@uint x := [value lsbIndex] ; # x is equal to 7
```

The most significant bit of 192 is the 7th bit.

1.5.9 `significantBitCount` Reader

Returns the number of bits needed to express the receiver value.

```
reader @uint significantBitCount -> @uint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: it the receiver value is zero, it returns 0 ; otherwise, it returns the most significant bit index plus one.

Example :

```
@uint value := 145 ; # 145 is 10010001 in binary
@uint x := [value significantBitCount] ; # x is equal to 8
```

1.5.10 `sint` Reader

Returns returns the receiver's value in an @sint type (page 12) (32-bit signed integer) object.

```
reader @uint sint -> @sint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: an error is raised is receiver's value is greater than $2^{31} - 1$.

This reader is the only way to convert an @uint type (page 21) object into an @sint type (page 12) object.

1.5.11 `sint64` Reader

Returns returns the receiver's value in an @sint64 type (page 17) (64-bit signed integer) object.

```
reader @uint sint64 -> @sint64 ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: as a 32-bit unsigned value can always be converted in a 64-bit signed value, this reader never fails.

This reader is the only way to convert an @uint type (page 21) object into an @sint64 type (page 17) object.

1.5.12 `string` Reader

Returns returns a decimal string representation of the receiver's value.

```
reader @uint string -> @string ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see `hexString` reader (page 23) and `xString` reader (page 26).

1.5.13 `uint64` Reader

Returns returns the receiver's value in an @uint64 type (page 28) (64-bit unsigned integer) object.

```
reader @uint uint64 -> @uint64 ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: as a 32-bit unsigned value can always be converted in a 64-bit unsigned value, this reader never fails.

This reader is the only way to convert an @uint type (page 21) object into an @uint64 type (page 28) object.

1.5.14 xString Reader

Returns returns an hexadecimal string representation of the receiver's value (without any prefix).

```
reader @uint xString -> @string ;
```

Availabilty: available in GALGAS 1.9.10 and later.

Discussion: for an decimal string representation of the receiver's value, see the hexString reader (page 23); for a decimal string representation of the receiver's value, see the string reader (page 25).

1.5.15 Incrementation and decrementation

The @uint type (page 21) supports incrementation and decrementation instructions.

```
@uint n := ...  ; n ++ ; # Incrementation
@uint p := ...  ; p - ; # Decrementation
```

The incrementation instruction raises an error if receiver's value is equal to $2^{32} - 1$.

The incrementation instruction raises an error if receiver's value is equal to 0.

Note that incrementation and decrementation are not available within an expression.

1.5.16 Arithmetic Operators

The `@uint` type supports the five arithmetic diadic operators:

+	Addition
−	Substraction
∗	Multiplication
/	Division
%	Modulo

Theses operators require both arguments to be `@uint` objects.

A run-time error is raised if the operation leads to a 32-bit unsigned overflow.

The `@uint` type supports the following arithmetic unary operator:

+	No operation

This operator returns the receiver's value (an `@uint` object).

1.5.17 Shift Operators

The `@uint` type supports right and left shift operators:

<<	Left shift
>>	Right shift

Theses operators require both arguments to be `@uint` objects.

Note the right shift inserts always a zero bit in the most significant bit location (it is a logical right shift).

The actual amount of the shift is the value of the right-hand operand masked by 31, i.e. the shift distance is always between 0 and 31.

1.5.18 Logical Operators

The `@uint` type supports the three bit-wise logical operators:

&	Bit-wise and
|	Bit-wise or
^	Bit-wise exclusive or

Theses operators require both arguments to be `@uint` objects.

The `@uint` type supports the bit-wise logical unary operator:

$\sim$	Bit-wise complementation

This operator returns an `@uint` object.

1.5.19 Comparison Operators

The `@uint` type supports the six comparison operators:

=	Equality
!=	Non Equality
<	Strict Lower Than
<=	Lower or Equal
>	Strict Greater Than
>=	Greater or Equal

Theses operators require both arguments to be `@uint` objects, and return a `@bool` object.

1.6 The `@uint64` Type

An `@uint64` object value is a 64-bit unsigned integer value. You can initialize an `@uint64`' object from a 64-bit unsigned integer constant:

```
@uint64 myUnsignedInteger := 123_456L ;
```

Note the 'L' suffix is required for a 64-bit unsigned integer constant.

1.6.1 `max` Constructor

Returns an `@uint64` object that the maximum value of the 64-bit unsigned range.

```
constructor @uint64 max -> @uint64 ;
```

Availabilty: available in GALGAS 1.3.0 and later.

Discussion: The returned value is $2^{64} - 1$.

1.6.2 uint64BaseValueWithCompressedBitString Constructor

Returns an `@uint64` object computed from a string containing '0', '1' or 'X' characters, replacing all occurrences of 'X' by '0'.

```
constructor @uint64 uint64BaseValueWithCompressedBitString
  ?@string inBitString
  -> @uint64 ;
```

Availabilty: available in GALGAS 1.6.4 and later.

Discussion: the inBitString argument should contain only '0', '1' or 'X' characters. A run time exception is raised if an other character appears.

This constructor considers the *inBitString* argument value as a binary encoding of an integer value. First, it internally replaces all 'X's by '0's, and then converts the resulting string into an integer value that is the one returned by this constructor.

Note that the first character of the *inBitString* argument value corresponds to the most significant bit of the converted value.

Example:

```
@uint64 v [uint64BaseValueWithCompressedBitString !"01XX10"] ;
log v ; # Displays <@uint64:18> ;
```

1.6.3 uint64MaskWithCompressedBitString Constructor

Returns an `@uint64` object computed from a string containing '0', '1' or 'X' characters, replacing all occurrences of '0' by '1' and all occurrences of 'X' by '0'.

```
constructor @uint64 uint64MaskWithCompressedBitString
  ?@string inBitString
  -> @uint64 ;
```

Availabilty: available in GALGAS 1.6.4 and later.

Discussion: the *inBitString* argument should contain only '0', '1' and 'X'

characters. A run time exception is raised if an other character appears.

This constructor considers the *inBitString* argument value as a binary encoding of an integer value. First, it internally replaces all '0's by '1's and all 'X's by '0's, and then converts the resulting string into an integer value that is the one returned by this constructor.

Note that the first '0' or '1' character of the *inBitString* argument value corresponds to the most significant Bit of the converted value.

Example:

```
@uint64 v [uint64MaskWithCompressedBitString !"01XX10"] ;
log v ; # Displays <@uint64:51> ;
```

1.6.4 uint64WithBitString Constructor

Returns an `@uint64` object computed from a string containing '0' or '1' characters.

```
constructor @uint64 uint64WithBitString
  ?@string inBitString
  -> @uint64 ;
```

Availabilty: available in GALGAS 1.6.4 and later.

Discussion: the *inBitString* argument should contain only '0' and '1' characters. A run time exception is raised if an other character appears.

This constructor considers the *inBitString* argument value as a binary encoding of an integer value. It returns an `@uint64` object containing the converted value.

Note that the first '1' character of the *inBitString* argument value corresponds to the most significant bit of the converted value.

Example:

```
@uint64 v [uint64WithBitString !"0101"]] ;
log v ; # Displays <@uint64:5> ;
```

1.6.5 double Reader

Returns the receiver's value converted in a `@double` object.

```
reader @uint64 double -> @double ;
```

Availabilty: available in GALGAS 1.9.8 and later.

Discussion: as a 64-bit integer value can always be converted in a `@double` value, this reader never fails.

1.6.6 `hexString` Reader

Returns the an hexadecimal string representation of the receiver value, prefixed by the string "0x".

```
reader @uint64 hexString -> @string ;
```

Availabilty: available in GALGAS 1.5.2 and later.

Discussion: for getting an hexadecimal representation string without "0x" prefix, see `xString` reader (page 33).

1.6.7 `sint` Reader

Returns returns the receiver's value in an @sint type (page 12) (32-bit signed integer) object.

```
reader @uint64 sint -> @sint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: an error is raised is receiver's value is greater than $2^{31} - 1$.

This reader is the only way to convert an @uint64 type (page 28) object into an @sint type (page 12) object.

1.6.8 `sint64` Reader

Returns returns the receiver's value in an @sint64 type (page 17) (64-bit signed integer) object.

```
reader @uint64 sint64 -> @sint64 ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: an error is raised is receiver's value is greater than $2^{63} - 1$.

This reader is the only way to convert an @uint64 type (page 28) object into an @sint64 type (page 17) object.

1.6.9 `string` Reader

Returns a decimal string representation of the receiver's value.

```
reader @uint64 string -> @string ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see `hexString` reader (page 31) and `xString` reader (page 33).

1.6.10 `uint` Reader

Returns the receiver's value in an @uint type (page 21) (32-bit unsigned integer) object.

```
reader @uint64 uint -> @uint ;
```

Availabilty: available in GALGAS 1.6.12 and later.

Discussion: an error is raised is receiver's value is greater than $2^{32} - 1$.

This reader is the only way to convert an @uint64 type (page 28) object into an @uint type (page 21) object.

1.6.11 `uintSlice` Reader

Returns an @uint type (page 21) value, extracted from a bit slice of the receiver?s value.

```
reader @uint64 uintSlice
  ?@uint inStartBit
  ?@uint inBitCount
  -> @uint ;
```

Availabilty: available in GALGAS 1.6.0 and later.

Discussion: the receiver's value is right shifted by *inStartBit*, and the resulted value is and'ed with a mask equal to $2^{inBitCount} - 1$.

This reader is the only way to convert an @uint type (page 21) object into an @uint64 type (page 28) object.

Example:

```
@uint64 v := 0x1234_5678_9ABC_DEF0L ;
@uint result := [v uintSlice !4 !5] ; # The result value is 0x8_9ABC
```

1.6.12 xString Reader

Returns returns an hexadecimal string representation of the receiver's value (without any prefix).

```
reader @uint64 xString -> @string ;
```

Availabilty: available in GALGAS 1.9.10 and later.

Discussion: for an decimal string representation of the receiver's value, see the hexString reader (page 31); for a decimal string representation of the receiver's value, see the string reader (page 32).

1.6.13 Incrementation and decrementation

The @uint64 type (page 28) supports incrementation and decrementation instructions.

```
@uint64 n := ...  ; n ++ ; # Incrementation
@uint64 p := ...  ; p - ; # Decrementation
```

The incrementation instruction raises an error if receiver's value is equal to $2^{64} - 1$.

The incrementation instruction raises an error if receiver's value is equal to 0.

Note that incrementation and decrementation are not available within an expression.

1.6.14 Arithmetic Operators

The `@uint64` type supports the five arithmetic diadic operators:

+	Addition
−	Substraction
*	Multiplication
/	Division
%	Modulo

Theses operators require both arguments to be `@uint64` objects.

A run-time error is raised if the operation leads to a 64-bit unsigned overflow.

The `@uint64` type supports the following arithmetic unary operator:

+	No operation

This operator returns the receiver's value (an `@uint64` object).

1.6.15 Shift Operators

The `@uint` type supports right and left shift operators:

<<	Left shift
>>	Right shift

Theses operators require the left argument to be `@uint64` object, and the right argument to be `@uint` object.

Note the right shift inserts always a zero bit in the most significant bit location (it is a logical right shift).

The actual amount of the shift is the value of the right-hand operand masked by 63, i.e. the shift distance is always between 0 and 63.

1.6.16 Logical Operators

The `@uint64` type supports the three bit-wise logical diadic operators:

&	Bit-wise and
|	Bit-wise or
^	Bit-wise exclusive or

Theses operators require both arguments to be `@uint64` objects.

The `@uint64` type supports the bit-wise logical unary operator:

$\sim$	Bit-wise complementation

This operator returns an `@uint64` object.

1.6.17 Comparison Operators

The `@uint64` type supports the six comparison operators:

=	Equality
!=	Non Equality
<	Strict Lower Than
<=	Lower or Equal
>	Strict Greater Than
>=	Greater or Equal

Theses operators require both arguments to be `@uint64` objects, and return a `@bool` object.