GALGAS Book

Version 2.1.4

Pierre Molinaro

December 8, 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:

- @char type (page 7), Unicode characters;
- @double type (page 22), floating point numbers;
- @location type (page 25), whose value points out a location in a source file;
- @sint type (page 27), the 32-bit signed integers;
- @sint64 type (page 32), the 64-bit signed integers;
- @stringset type (page 36), set of `@string` objects;
- @uint type (page 39), the 32-bit unsigned integers;
- @uint64 type (page 46), the 64-bit unsigned integers.

1.1 The `@char` Type

An `@char` object value is an Unicode character. You can initialize an `@char` object from a character constant:

```
@char myCharacter := 'A';
```

You have several ways for writing a literal character constant. In any case, it should define an assigned Unicode character. A compile-time error is raised if it does not.

A literal character constant is a single character or an escape sequence enclosed by single quotes (''').

For an ASCII printable character:

```
@char myCharacter := 'a';
```

If you want to get ASCII source text file, any character that does not correspond to an ASCII printable character should be expressed with an escape sequence.

Otherwise, for any printable Unicode character, you can write it directly, without escape sequence, provided your text file encoding supports this character:

```
@char myCharacter := 'æ';
```

The following escape sequences are defined (they begin with a '\').

Character Constant	Meaning
'\f'	A Form Feed Character
'\n'	A New Line Character
'\r'	A Carriage Return Character
'\v'	A Vertical Tabulation Character
'\\'	A back slash Character
'\0'	A Nul Character
'\''	A Single Quote Character

Character Constant	Meaning
'\uABCD'	An Unicode Character

Where *ABCD* is a four digit hexadecimal number that represents an assigned Unicode point code. For example:

```
@char myCharacter := '\u03A0'; # The 'Σ' character
```

Note: an unassigned point code raises a compile-time error:

```
@char myCharacter := '\uFFFF'; # The \uFFFF point code is not assigned
```

Character Constant	Meaning
'\Uabcdxyzt'	An Unicode Character

Where *abcdxyzt* is a eight digit hexadecimal number that represents an assigned Unicode point code. For example:

```
@char myCharacter := '\U0010170'; # The 'GREEK ACROPHONIC NAXIAN
FIVE HUNDRED' character
```

Note: an unassigned point code raises a compile-time error:

```
@char myCharacter := '\U0000FFFF'; # Raises a compile-time error:
\U0000FFFF is not assigned.
```

Any point code beyond \U0010FFFF is invalid and not assigned.

Character Constant	Meaning
`&html-seq;`	A defined HTML sequence

Where *html-seq* is a defined sequence. GALGAS accepts 260 sequences:

Literal Constant	Code Point	Character
'\Æ'	u00C6	Æ
'\Á'	u00C1	Á
'\Ă'	u0102	?
'\Â'	u00C2	Â
'\À'	u00C0	À
'\Ā'	u0100	?
'\Ą'	u0104	?
'\Å'	u00C5	Å
'\Ã'	u00C3	Ã
'\Ä'	u00C4	Ä
'\Ć'	u0106	?
'\Č'	u010C	?
'\Ç'	u00C7	Ç
'\Ĉ'	u0108	?
'\Ċ'	u010A	?
'\Ď'	u010E	?
'\Đ'	u0110	?
'\Ŋ'	u014A	?
'\Ð'	u00D0	Đ
'\É'	u00C9	É
'\Ě'	u011A	?
'\Ê'	u00CA	Ê
'\Ė'	u0116	?
'\È'	u00C8	È
'\Ē'	u0112	?
'\Ę'	u0118	?
'\Ë'	u00CB	Ë
'\Ğ'	u011E	?
'\Ģ'	u0122	?

'\Ĝ'	u011C	?
'\Ġ'	u0120	?
'\Ĥ'	u0124	?
'\Ħ'	u0126	?
'\Ĳ'	u0132	?
'\Í'	u00CD	Í
'\Î'	u00CE	Î
'\İ'	u0130	?
'\Ì'	u00CC	Ì
'\Ī'	u012A	?
'\Į'	u012E	?
'\Ĩ'	u0128	?
'\Ï'	u00CF	Ï
'\Ĵ'	u0134	?
'\Ķ'	u0136	?
'\Ĺ'	u0139	?
'\Ľ'	u013D	?
'\Ļ'	u013B	?
'\Ŀ'	u013F	?
'\Ł'	u0141	?
'\Ń'	u0143	?
'\Ň'	u0147	?
'\Ņ'	u0145	?
'\Ñ'	u00D1	Ñ
'\Œ'	u0152	?
'\Ó'	u00D3	Ó
'\Ô'	u00D4	Ô
'\Ő'	u0150	?
'\Ò'	u00D2	Ò
'\Ō'	u014C	?
'\Ø'	u00D8	Ø
'\Õ'	u00D5	Õ
'\Ö'	u00D6	Ö
'\Ŕ'	u0154	?
'\Ř'	u0158	?
'\Ŗ'	u0156	?
'\Ś'	u015A	?
'\Š'	u0160	?

'\Ş'	u015E	?
'\Ŝ'	u015C	?
'\Þ'	u00DE	Þ
'\Ť'	u0164	?
'\Ţ'	u0162	?
'\Ŧ'	u0166	?
'\Ú'	u00DA	Ú
'\Ŭ'	u016C	?
'\Û'	u00DB	Û
'\Ű'	u0170	?
'\Ù'	u00D9	Ù
'\Ū'	u016A	?
'\Ų'	u0172	?
'\Ů'	u016E	?
'\Ũ'	u0168	?
'\Ü'	u00DC	Ü
'\Ŵ'	u0174	?
'\Ý'	u00DD	Ý
'\Ŷ'	u0176	?
'\Ÿ'	u0178	?
'\Ź'	u0179	?
'\Ž'	u017D	?
'\Ż'	u017B	?
'\á'	u00E1	á
'\ă'	u0103	?
'\â'	u00E2	â
'\æ'	u00E6	æ
'\à'	u00E0	à
'\ā'	u0101	?
'\&'	u0026	&
'\ą'	u0105	?
'\''	u02BC	?
'\å'	u00E5	å
'*'	u002A	*
'\ã'	u00E3	ã
'\ä'	u00E4	ä
'\¦'	u00A6	ę
'\\'	u005C	

'\ć'	u0107	?
'\č'	u010D	?
'\ç'	u00E7	ç
'\ĉ'	u0109	?
'\ċ'	u010B	?
'\¢'	u00A2	ć
'\:'	u003A	:
'\,'	u002C	,
'\@'	u0040	@
'\©'	u00A9	ľ
'\¤'	u00A4	ď
'\↓'	u2193	?
'\ď'	u010F	?
'\°'	u00B0	ř
'\÷'	u00F7	œ
'\$'	u0024	$
'\đ'	u0111	?
'\é'	u00E9	é
'\ě'	u011B	?
'\ê'	u00EA	ê
'\ė'	u0117	?
'\è'	u00E8	è
'\ē'	u0113	?
'\ŋ'	u014B	?
'\ę'	u0119	?
'\='	u003D	=
'\ð'	u00F0	ð
'\ë'	u00EB	ë
'\!'	u0021	!
'\½'	u00BD	¡
'\¼'	u00BC	ij
'\⅛'	u215B	?
'\¾'	u00BE	¿
'\⅜'	u215C	?
'\⅝'	u215D	?
'\⅞'	u215E	?
'\ǵ'	u01F5	?
'\ğ'	u011F	?

'\&gcedil;'	u0123	?
'\ĝ'	u011D	?
'\ġ'	u0121	?
'\>'	u003E	>
'\½'	u00BD	¡
'\ĥ'	u0125	?
'\―'	u2015	?
'\ħ'	u0127	?
'\‐'	u002D	-
'\í'	u00ED	í
'\î'	u00EE	î
'\¡'	u00A1	ą
'\ì'	u00EC	ì
'\ĳ'	u0133	?
'\ī'	u012B	?
'\ı'	u0131	?
'\į'	u012F	?
'\¿'	u00BF	£
'\ĩ'	u0129	?
'\ï'	u00EF	ï
'\ĵ'	u0135	?
'\ķ'	u0137	?
'\ĸ'	u0138	?
'\ĺ'	u013A	?
'\«'	u00AB	ń
'\←'	u2190	?
'\ľ'	u013E	?
'\ļ'	u013C	?
'\{'	u007B	{
'\“'	u201C	?
'\ŀ'	u0140	?
'_'	u005F	_
'\('	u0028	(
'\['	u005B	[
'\‘'	u2018	?
'\ł'	u0142	?
'\<'	u003C	<
'\µ'	u00B5	ţ

'\·'	u00B7	ů
'\ń'	u0144	?
'\ŉ'	u0149	?
'\ '	u00A0	
'\ň'	u0148	?
'\ņ'	u0146	?
'\¬'	u00AC	ň
'\ñ'	u00F1	ñ
'\#'	u0023	#
'\ó'	u00F3	ó
'\ô'	u00F4	ô
'\ő'	u0151	?
'\œ'	u0153	?
'\ò'	u00F2	ò
'\Ω'	u2126	?
'\ō'	u014D	?
'\ª'	u00AA	ł
'\º'	u00BA	ž
'\ø'	u00F8	ø
'\õ'	u00F5	õ
'\ö'	u00F6	ö
'\¶'	u00B6	ű
'\%'	u0025	%
'\.'	u002E	.
'\+'	u002B	+
'\±'	u00B1	ś
'\£'	u00A3	č
'\?'	u003F	?
'\"'	u0022	"
'\ŕ'	u0155	?
'\»'	u00BB	ż
'\→'	u2192	?
'\ř'	u0159	?
'\ŗ'	u0157	?
'\}'	u007D	}
'\”'	u201D	?
'\®'	u00AE	ő
'\)'	u0029	)

'\]'	u005D	]
'\’'	u2019	?
'\ś'	u015B	?
'\š'	u0161	?
'\ş'	u015F	?
'\ŝ'	u015D	?
'\§'	u00A7	ğ
'\;'	u003B	;
'\'	u00AD	ŋ
'\/'	u002F	/
'\♪'	u266A	?
'\¹'	u00B9	ź
'\²'	u00B2	š
'\³'	u00B3	ş
'\ß'	u00DF	SS
'\ť'	u0165	?
'\ţ'	u0163	?
'\þ'	u00FE	þ
'\×'	u00D7	Œ
'\™'	u2122	?
'\ŧ'	u0167	?
'\ú'	u00FA	ú
'\↑'	u2191	?
'\ŭ'	u016D	?
'\û'	u00FB	û
'\ű'	u0171	?
'\ù'	u00F9	ù
'\ū'	u016B	?
'\ų'	u0173	?
'\ů'	u016F	?
'\ũ'	u0169	?
'\ü'	u00FC	ü
'|'	u007C	|
'\ŵ'	u0175	?
'\ý'	u00FD	ý
'\ŷ'	u0177	?
'\¥'	u00A5	ě
'\ÿ'	u00FF	ß

'\ź'	u017A	?
'\ž'	u017E	?
'\ż'	u017C	?

1.1.1 replacementCharacter Constructor

Returns an @char object corresponding to Unicode replacement character ('\uFFFD.

```
constructor @char replacementCharacter -> @char ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion:

1.1.2 unicodeCharacterWithUnsigned Constructor

Returns an @char object from an Unicode code point.

```
constructor @char unicodeCharacterWithUnsigned
  ?@uint inValue
  -> @char ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: A run-time error is raised if the *inValue* value does not represent an assigned Unicode value. You can check if an @uint value represents an assigned Unicode value with the isUnicodeValueAssigned reader (page 42).

1.1.3 isalnum Reader

Returns an @bool value indicating whether the receiver'value represents an ASCII letter or an ASCII digit.

```
reader @char isalnum -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns true if the receiver'value represents an ASCII letter

or an ASCII digit (between `'A'` and `'Z'`, or between `'a'` and `'z'`, or between `'0'` and `'9'`), and `false` otherwise.

1.1.4 `isalpha` Reader

Returns an `@bool` value indicating whether the receiver'value represents an ASCII letter.

```
reader @char isalpha -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns `true` if the receiver'value represents an ASCII letter (between `'A'` and `'Z'`, or between `'a'` and `'z'`), and `false` otherwise.

1.1.5 `iscntrl` Reader

Returns an `@bool` value indicating whether the receiver'value represents an ASCII control character.

```
reader @char iscntrl -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns `true` if the receiver'value represents an ASCII control character (strictly before the *SPACE* character), and `false` otherwise.

1.1.6 `isdigit` Reader

Returns an `@bool` value indicating whether the receiver'value represents an ASCII digit.

```
reader @char isdigit -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns `true` if the receiver'value represents an ASCII digit

(between '0' and '9'), and false otherwise.

1.1.7 islower Reader

Returns an @bool value indicating whether the receiver'value represents an ASCII lowercase ASCII letter.

```
reader @char islower -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns true if the receiver'value represents an ASCII lowercase letter (between 'a' and 'z'), and false otherwise.

1.1.8 isUnicodeCommand Reader

Returns an @bool value indicating whether the receiver'value represents an Unicode command.

```
reader @char isUnicodeCommand -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: returns true if the receiver'value represents an Unicode command, and false otherwise.

1.1.9 isUnicodeLetter Reader

Returns an @bool value indicating whether the receiver'value represents an Unicode letter.

```
reader @char isUnicodeLetter -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: returns true if the receiver'value represents an Unicode letter, and false otherwise.

1.1.10 isUnicodeMark Reader

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

```
reader @char isUnicodeMark -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

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

1.1.11 isUnicodePunctuation Reader

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

```
reader @char isUnicodePunctuation -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

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

1.1.12 isUnicodeSeparator Reader

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

```
reader @char isUnicodeSeparator -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

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

1.1.13 isUnicodeSymbol Reader

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

```
reader @char isUnicodeSymbol -> @bool ;
```

Availability: available in GALGAS 1.8.3 and later.

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

1.1.14 isupper Reader

Returns an `@bool` value indicating whether the receiver'value represents an ASCII uppercase ASCII letter.

```
reader @char isupper -> @bool ;
```

Availability: available in GALGAS 1.7.2 and later.

Discussion: returns `true` if the receiver'value represents an ASCII uppercase letter (between `'A'` and `'Z'`), and `false` otherwise.

1.1.15 string Reader

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

```
reader @char string -> @string ;
```

Availability: available in GALGAS 1.5.5 and later.

Discussion: returns a one character `@string` object, containing the receiver's value.

1.1.16 uint Reader

Returns an `@uint` object representing the Unicode code point of the receiver's value.

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

Availability: available in GALGAS 1.7.7 and later.

Discussion:

1.1.17 unicodeName Reader

Returns the unicode name of the receiver's value.

```
reader @char unicodeName -> @string ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: for an decimal string representation of the receiver's value, see the `hexString` reader (page 41); for a decimal string representation of the receiver's value, see the `string` reader (page 43).

Example: `['Æ'unicodeName] returns "LATIN CAPITAL LETTER AE "`

1.1.18 unicodeToLower Reader

Returns the lowercase character corresponding to the receiver's value.

```
reader @char unicodeToLower -> @char ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: if the receiver's value is an Unicode uppercase character, this reader returns the corresponding lowercase character. Otherwise, it returns the receiver's value.

Example:
`['Æ'unicodeToLower] returns 'æ'`

```
['æ'unicodeToLower] returns 'æ'
```

1.1.19 unicodeToUpper Reader

Returns the uppercase character corresponding to the receiver's value.

```
reader @char unicodeToUpper -> @char ;
```

Availability: available in GALGAS 1.8.3 and later.

Discussion: if the receiver's value is an Unicode lowercase character, this reader returns the corresponding uppercase character. Otherwise, it returns the receiver's value.

Example:

```
['Æ'unicodeToUpper] returns 'Æ'
['æ'unicodeToUpper] returns 'Æ'
```

1.1.20 Comparison Operators

The `@char` 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 `@char` objects, and return a `@bool` object. Comparison is done by comparing of the Unicode code point's value.

1.2 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 (.)

1.2.1 `sint` Reader

Returns the receiver's value in an @sint type (page 27) (32-bit signed integer) object.

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

Availability: available in GALGAS 1.9.9 and later.

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

1.2.2 `sint64` Reader

Returns the receiver's value in an @sint64 type (page 32) (64-bit signed integer) object.

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

Availability: available in GALGAS 1.9.9 and later.

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

1.2.3 `string` Reader

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

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

Availability: available in GALGAS 1.7.7 and later.

Discussion: this reader never fails.

1.2.4 uint Reader

Returns the receiver's value in an @uint type (page 39) (32-bit unsigned integer) object.

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

Availability: available in GALGAS 1.9.9 and later.

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

1.2.5 uint64 Reader

Returns the receiver's value in an @uint64 type (page 46) (64-bit unsigned integer) object.

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

Availability: available in GALGAS 1.9.9 and later.

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

1.2.6 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).

1.2.7 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.

1.3 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.

1.3.1 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 ;
```

1.3.2 `nowhere` Constructor

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

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

Availability: available in GALGAS 2.1.2 and later.

Discussion: The returned object responds `true` to the `isNowhere` reader

(page 26).

1.3.3 column Reader

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

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

Availability: 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 26).

1.3.4 isNowhere Reader

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

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

Availability: 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.

1.3.5 line Reader

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

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

Availability: 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 26).

1.3.6 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 ;
```

Availability: 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 26).

1.3.7 locationString Reader

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

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

Availability: 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 26).

1.4 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.

1.4.1 min Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.4.2 `max` Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.4.3 `double` Reader

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

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

Availability: 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.

1.4.4 `sint64` Reader

Returns the receiver's value in an @sint64 type (page 32) (64-bit signed integer) object.

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

Availability: 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 27) object into an @sint64 type (page 32) object.

1.4.5 string Reader

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

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

Availability: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see hexString reader (page 41) and xString reader (page 44).

1.4.6 uint Reader

Returns the receiver's value in an @uint type (page 39) (32-bit unsigned integer) object.

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

Availability: 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 27) object into an @uint type (page 39) object.

1.4.7 uint64 Reader

Returns the receiver's value in an @uint64 type (page 46) (64-bit unsigned integer) object.

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

Availability: 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 27) object into an @uint64 type (page 46) object.

1.4.8 Incrementation and decrementation

The @sint type (page 27) 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.

1.4.9 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}.

1.4.10 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.

1.4.11 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.

1.4.12 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.

1.5 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.

1.5.1 `min` Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.5.2 `max` Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.5.3 `double` Reader

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

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

Availability: 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.

1.5.4 `sint` Reader

Returns the receiver's value in an @sint type (page 27) (32-bit signed integer) object.

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

Availability: 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 32) object into an @sint type (page 27) object.

1.5.5 `string` Reader

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

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

Availability: available in GALGAS 1.6.12 and later.

Discussion: this reader never fails.

1.5.6 uint Reader

Returns the receiver's value in an @uint type (page 39) (32-bit unsigned integer) object.

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

Availability: 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 32) object into an @uint type (page 39) object.

1.5.7 uint64 Reader

Returns the receiver's value in an @uint64 type (page 46) (64-bit unsigned integer) object.

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

Availability: 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 32) object into an @uint64 type (page 46) object.

1.5.8 Incrementation and decrementation

The @sint64 type (page 32) 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.

1.5.9 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}.

1.5.10 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.

1.5.11 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:

~	Bit-wise complementation

This operator returns an `@sint64` object.

1.5.12 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.

1.6 The `@stringset` Type

An `@stringset` object value is a set of `@string` values.

1.6.1 `emptySet` Constructor

Creates and returns an empty `@stringset` object.

```
constructor @stringset emptySet -> @stringset ;
```

Availability: available in GALGAS 1.3.0 and later.

Discussion:

1.6.2 setWithString Constructor

Creates and returns an `@stringset` object that contains the value of the *inString* argument object.

```
constructor @stringset setWithString
  ?@string inString
  -> @stringset ;
```

Availability: available in GALGAS 1.3.0 and later.

Discussion:

1.6.3 count Reader

Returns the number of strings in the set.

```
reader @stringset count -> @uint ;
```

Availability: available in GALGAS 1.3.0 and later.

Discussion:

1.6.4 hasKey Reader

Returns a boolean value that indicates whether the value of *inString* argument is present in the set.

```
reader @stringset hasKey
  ?@string inString
  -> @bool ;
```

Availability: available in GALGAS 1.3.0 and later.

Discussion: returns `true` if the value of *inString* argument is present in the set, `false` otherwise.

1.6.5 removeKey Modifier

Removes the value of *inString* argument from the receiver's value.

```
modifier @stringset removeKey
  ?@string inString
```

Availability: available in GALGAS 1.3.0 and later.

Discussion: if the receiver's value does not contain the value of *inString* argument, this modifier leaves the receiver's value unchanged.

1.6.6 the += Operator

The += operator adds a string value to the receiver. If the receiver's value already contains the added value, this operator has no effect.

Example:

```
@string aString := ...  ;
@stringset aStringSet := ...  ;
aStringSet += !aString ;
```

1.6.7 the & Operator

The & operator returns the intersection of its operand values.

Example:

```
@stringset s1 := ...  ;
@stringset s2 := ...  ;
@stringset s := s1 & s2 ; # s is the intersection of s1 and s2
```

1.6.8 the | Operator

The | operator returns the union of its operand values.

Example:

```
@stringset s1 := ...  ;
@stringset s2 := ...  ;
@stringset s := s1 |s2 ; # s is the union of s1 and s2
```

1.6.9 the − Operator

The − operator returns the difference of its operand values.

Example:

```
@stringset s1 := ...  ;
@stringset s2 := ...  ;
@stringset s := s1 - s2 ; # s is the difference of s1 and s2
```

1.6.10 Enumerating `@stringset` objects

The `foreach` instruction can be used for enumerating `@stringset` values; enumeration is performed in the ascending order, or in the reverse alphabetical order using the '>' qualifier.

```
   @stringset s := ...  ;
foreach s do
# the key constant has the value of current entry of s stringset
end foreach ;
```

1.6.11 Comparison Operators

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

=	Equality
!=	Non Equality
<	Strict Inclusion
<=	Inclusion or Equality
>	Strict Greater
>=	Greater or Equality

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

1.7 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.

1.7.1 errorCount Constructor

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

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

Availability: 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] ;`

1.7.2 max Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.7.3 valueWithMask Constructor

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

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

Availability: 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)`

1.7.4 warningCount Constructor

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

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

Availability: available in GALGAS 1.4.9 and later.

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

1.7.5 double Reader

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

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

Availability: 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.

1.7.6 hexString Reader

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

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

Availability: available in GALGAS 1.5.2 and later.

Discussion: for getting an hexadecimal representation string without "0x" prefix, see `xString` reader (page 44).

1.7.7 isUnicodeValueAssigned Reader

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

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

Availability: 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).
```

1.7.8 lsbIndex Reader

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

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

Availability: 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.

1.7.9 significantBitCount Reader

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

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

Availability: 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
```

1.7.10 `sint` Reader

Returns the receiver's value in an @sint type (page 27) (32-bit signed integer) object.

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

Availability: 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 39) object into an @sint type (page 27) object.

1.7.11 `sint64` Reader

Returns the receiver's value in an @sint64 type (page 32) (64-bit signed integer) object.

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

Availability: 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 39) object into an @sint64 type (page 32) object.

1.7.12 `string` Reader

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

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

Availability: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see `hexString` reader (page 41) and `xString` reader (page 44).

1.7.13 `uint64` Reader

Returns the receiver's value in an @uint64 type (page 46) (64-bit unsigned integer) object.

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

Availability: 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 39) object into an @uint64 type (page 46) object.

1.7.14 `xString` Reader

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

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

Availability: available in GALGAS 1.9.10 and later.

Discussion: for an decimal string representation of the receiver's value, see the `hexString` reader (page 41); for a decimal string representation of the receiver's value, see the `string` reader (page 43).

1.7.15 Incrementation and decrementation

The @uint type (page 39) 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.

1.7.16 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).

1.7.17 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.

1.7.18 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:

~	Bit-wise complementation

This operator returns an `@uint` object.

1.7.19 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.

1.8 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.

1.8.1 max Constructor

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

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

Availability: available in GALGAS 1.3.0 and later.

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

1.8.2 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 ;
```

Availability: 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> ;
```

1.8.3 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 ;
```

Availability: 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> ;
```

1.8.4 uint64WithBitString Constructor

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

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

Availability: 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> ;
```

1.8.5 `double` Reader

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

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

Availability: 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.

1.8.6 `hexString` Reader

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

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

Availability: available in GALGAS 1.5.2 and later.

Discussion: for getting an hexadecimal representation string without "0x" prefix, see `xString` reader (page 51).

1.8.7 `sint` Reader

Returns the receiver's value in an @sint type (page 27) (32-bit signed integer) object.

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

Availability: 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 46) object into an @sint type (page 27) object.

1.8.8 `sint64` Reader

Returns the receiver's value in an @sint64 type (page 32) (64-bit signed integer) object.

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

Availability: 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 46) object into an @sint64 type (page 32) object.

1.8.9 `string` Reader

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

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

Availability: available in GALGAS 1.6.12 and later.

Discussion: for an hexadecimal string representation of the receiver's value, see `hexString` reader (page 49) and `xString` reader (page 51).

1.8.10 `uint` Reader

Returns the receiver's value in an @uint type (page 39) (32-bit unsigned integer) object.

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

Availability: 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 46) object into an @uint type (page 39) object.

1.8.11 uintSlice Reader

Returns an @uint type (page 39) value, extracted from a bit slice of the receiver?s value.

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

Availability: 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 39) object into an @uint64 type (page 46) object.

Example:

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

1.8.12 xString Reader

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

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

Availability: available in GALGAS 1.9.10 and later.

Discussion: for an decimal string representation of the receiver's value, see the hexString reader (page 49); for a decimal string representation of the receiver's value, see the string reader (page 50).

1.8.13 Incrementation and decrementation

The @uint64 type (page 46) 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.

1.8.14 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).

1.8.15 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.

1.8.16 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:

~	Bit-wise complementation

This operator returns an `@uint64` object.

1.8.17 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.