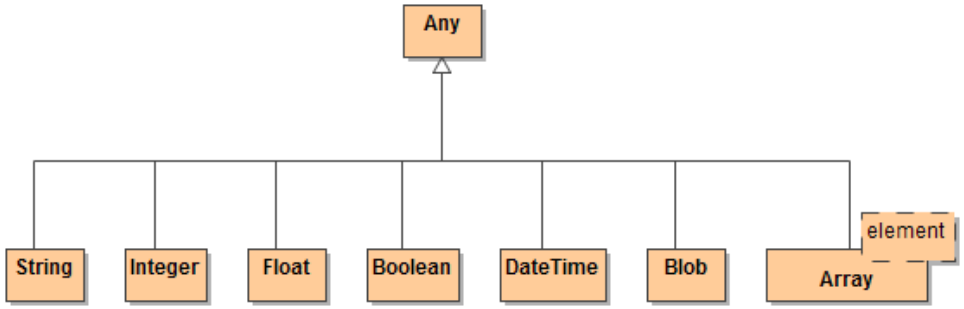
Base Types

The Bridge supports seven base types:

- **Array**
- **Blob**
- **Boolean**
- **DateTime**
- **Float**
- **Integer**
- **String**

Figure: Bridge Base Types (Operations are Suppressed)



All base types are derived from the abstract **Any** type. If no base type is specified for an attribute, it gets the default base type **String**.

All built-in base types of the Bridge are located in a dedicated package **Base Types** that is provided with the Builder. This package is added as a module and has assigned the stereotype <<Repository>>. Defining a package as a repository package makes it recognizable to the xUML Model Compiler.

Most of these base types are only able to hold one single piece of information, like text in a string, true or false in a boolean, or binary data in a blob. UML does not provide definitions like lexical representations, value ranges, etc. of Bridge base types. Thus, we follow the widespread XML Schema definitions of base types as provided by the W3C consortium. The W3C defines the exact definition of valid values for the base types displayed above. The following table shows a short description of its types and example values:

Base Type	Type Definition	Example Values	Operations / Functions / Operators
Blob	A blob represents base64-encoded arbitrary binary data.	e.g. "`YWJjZGVmZ2g=`" for the encoded value "abcdefgh"	• blobLength() Operation • concatBlob() Operation • convertBinaryToInteger () Operation • convertToHex() Operation • convertToString() Operation for Blobs • subblob() Operation • transcodeToString() Operation • xmlToClass() Operation for Blobs
Boolean	Boolean values represent binary-valued logic (true, false).	`true, false, 0, 1`	• Logical Operators • Boolean Operators • like Operator • unlike Operator • not() Operation of Boolean Expressions • condition () Statement • convertToInteger() Operation • convertToString() Operation for Booleans • set Assignment Statement with Boolean Constants • set Assignment Statement with Boolean Expressions • UML Example for Boolean Operations

DateTime	`CCYY-MM-DDThh:mm:ss[.sss][Z|[+|-]hh:mm]` Whereas "Z" stands for the time zone: Coordinated Universal Time (UTC). The W3C value space of dateTime is closely related to the dates and times described in ISO 8601.	`2004-12-01T00:00:00.0.Z` This value stands for the 1st December of the year 2004 at midnight.	• add() / subtract() Operation • convertDateTimeToStructure() Operation • convertDateTimeToTimeTicks() Operation • convertStructureToDateTime() Operation • convertToString() Operation for DateTime Objects • difference() Operation • printDateTimeExpression() Operation • printLocalDateTimeExpression() Operation • currentDateTime() Function • currentLocalDateTime() Function • currentTimeTicks() Function • getTimestamp() Function
Float	A float corresponds to the IEEE single-precision 32-bit floating-point type. Lexical representation: `[white spaces] [+|-][nnn][.nnn][e|E[+|-]nnn]` (where `white spaces` are any tab or space character; `nnn` may be any number of digits)	`1.234, 1e-5`	• absolute() Operation for Float Values • ceil() Operation • convertToInteger() for Float Values • convertToString() for Float Values • floor() Operation • power() Operation for Float Values • printFloatExpression() Operation • random() Operation returning a Float Value • round() operation
Integer	The W3C defines integers as "decimal", which represents arbitrary precision decimal numbers.	`5`	• absolute() Operation for Integer Values • convertTimeTicksToDateTime() Operation • convertToFloat() Operation for Integer Values • convertToString() Operation for Integer Values • modulo() Operation • power() Operation for Integer Values • printIntegerExpression() Operation • random() Operation returning an Integer Value

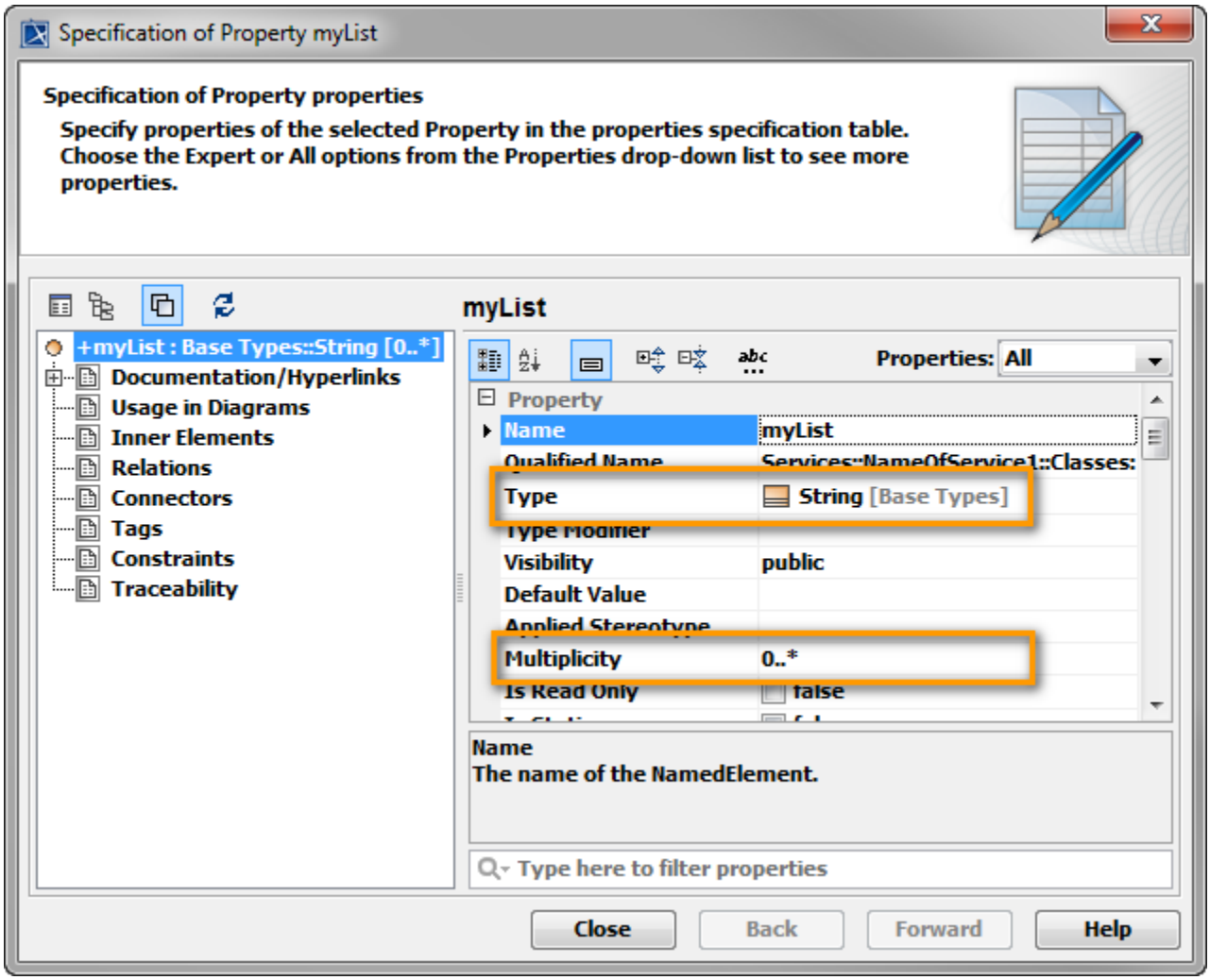
String	A string is a set of finite-length sequences of a character set (the Bridge uses UTF-8 internally).	`Hello World!`	• capture() Operation • concat() Operation • convertBase64ToBlob() Operation • convertDurationToDateTime() Operation • convertHexToBlob() Operation • convertToBoolean Operation() • convertToDateTime() Operation • convertToFloat() Operation for Strings • convertToInteger() Operation for Strings • endsWith() Operation • escapeURI() Operation • extendedJSONToClass() Operation • findPattern() Operation • findPatterns() Operation • findString() Operation • jsonToClass() Operation • match() Operation • normalizeSpaces() Operation • padLeft() Operation • padRight() Operation • parseDateTimeExpression() Operation • parseFloatExpression() Operation • parseIntegerExpression() Operation • parseLocalDateTimeExpression() Operation • removeAccents() Operation • replace() Operation • split() Operation • startsWith() Operation • stringLength() Operation • substring() Operation • substringAfter() Operation • substringBefore() Operation • toASCII() Operation • toLower() Operation • toUpper() Operation • transcodeToBlob() Operation • transliterate() Operation • unescapeURI() Operation • xmlToClass() Operation for Strings

Only the base type **Array** can store multiple pieces of information like multiple array elements. If you wish to associate several bits of information, you have to define a complex type that combines a number of independent base types and possibly other complex types. Such complex types are modeled as classes in UML.
For example, an **Animal** type might contain a string **name**, two DateTimes **birth** and **lastFed** and an integer **weight**. The implementation of **Animal** would be a class with several attributes of base type **String**, **DateTime**, and **Integer**.
The base type **Array** is an exception from the rule that base types do not have any sub-structure. Arrays allow multiple elements to be stored, but the array elements must be all of the same type. For example, an array may not contain string and integer elements, but only elements, which are of one of these types. Arrays may also contain complex types.

> Arrays of arrays are not supported by the Bridge.

Arrays are characterized by multiplicity and type.

Figure: Definition of Arrays



Definition of attributes without any type is allowed. They will be treated as strings internally.

Instances of classes are the transportation medium for data inside the **xUML Runtime**. In UML, they are modeled as so-called object nodes. In addition, these nodes form the interface from the service to its clients: object nodes are used in input and/or output messages for all Bridge-based services.