FreezeScript Descriptor Expression Language

An expression language is provided for use in FreezeScript descriptors.

On this page:

- Operators in FreezeScript
- Literals in FreezeScript
- Symbols in FreezeScript
- The nil Keyword in FreezeScript
- Accessing Elements in FreezeScript
- Reserved Keywords in FreezeScript
- Implicit Data Members in FreezeScript
- Calling Functions in FreezeScript
  - String Member Functions
  - Dictionary Member Functions
  - Object Member Functions
  - Global Functions

Operators in FreezeScript

The language supports the usual complement of operators: `and`, `or`, `not`, {+}, `-`, `/`, `*`, `%`, <, >, `==`, `!=`, <=, >=, `(`, `)`. Note that the < character must be escaped as `<` in order to comply with XML syntax restrictions.

Literals in FreezeScript

Literal values can be specified for integer, floating point, boolean, and string. The expression language supports the same syntax for literal values as that of Slice, with one exception: string literals must be enclosed in single quotes.

Symbols in FreezeScript

Certain descriptors introduce symbols that can be used in expressions. These symbols must comply with the naming rules for Slice identifiers (i.e., a leading letter followed by zero or more alphanumeric characters). Data members are accessed using dotted notation, such as `value.memberA.memberB`.

Expressions can refer to Slice constants and enumerators using scoped names. In a `transformdb` descriptor, there are two sets of Slice definitions, therefore the expression must indicate which set of definitions it is accessing by prefixing the scoped name with `::Old` or `::New`. For example, the expression `old.fruitMember == ::Old::Pear` evaluates to `true` if the data member `fruitMember` has the enumerated value `Pear`. In `dumpdb`, only one set of Slice definitions is present and therefore the constant or enumerator can be identified without any special prefix.

The `nil` Keyword in FreezeScript

The keyword `nil` represents a nil value of type `Object`. This keyword can be used in expressions to test for a nil object value, and can also be used to set an object value to nil.

Accessing Elements in FreezeScript

Dictionary and sequence elements are accessed using array notation, such as `userMap['joe']` or `stringSeq[5]`. An error occurs if an expression attempts to access a dictionary or sequence element that does not exist. For dictionaries, the recommended practice is to check for the presence of a key before accessing it:

XML

```
<if test="userMap.containsKey('joe') and userMap['joe'].active">
```

(This example shows that you can also call functions in FreezeScript.)

Similarly, expressions involving sequences should check the length of the sequence:

XML

```
<if test="stringSeq.length > 5 and stringSeq[5] == 'fruit'">
```

The `length` member is an implicit data member.

Reserved Keywords in FreezeScript

The following keywords are reserved: `and`, `or`, `not`, `true`, `false`, `nil`.

Implicit Data Members in FreezeScript

Certain Slice types support implicit data members:

- Dictionary and sequence instances have a member `length` representing the number of elements.
- Object instances have a member `ice_id` denoting the actual type of the object.

Calling Functions in FreezeScript

The expression language supports two forms of function invocation: member functions and global functions. A member function is invoked on a particular data value, whereas global functions are not bound to a data value. For instance, here is an expression that invokes the `find` member function of a `string` value:

```
old.stringValue.find('theSubstring') != -1
```

And here is an example that invokes the global function `stringToIdentity`:

```
stringToIdentity(old.stringValue)
```

If a function takes multiple arguments, the arguments must be separated with commas.

String Member Functions

The `string` data type supports the following member functions:

- `int find(string match[, int start])`
  Returns the index of the substring, or `-1` if not found. A starting position can optionally be supplied.
- `string replace(int start, int len, string str)`
  Replaces a given portion of the string with a new substring, and returns the modified string.
- `string substr(int start[, int len])`
  Returns a substring beginning at the given start position. If the optional length argument is supplied, the substring contains at most `len` characters, otherwise the substring contains the remainder of the string.

Dictionary Member Functions

The `dictionary` data type supports the following member function:

- `bool containsKey(key)`
  Returns `true` if the dictionary contains an element with the given key, or `false` otherwise. The `key` argument must have a value that is compatible with the dictionary's key type.

Object Member Functions

Object instances support the following member function:

- `bool ice_isA(string id)`
  Returns `true` if the object implements the given interface type, or `false` otherwise. This function cannot be invoked on a nil object.

Global Functions

The following global functions are provided:

- `string generateUUID()`
  Returns a new UUID.
- `string identityToString(Ice::Identity id)`
  Converts an identity into its string representation.
- `string lowercase(string str)`
  Returns a new string converted to lowercase.
- `string proxyToString(Ice::ObjectPrx prx)`
  Returns the string representation of the given proxy.
- `Ice::Identity stringToIdentity(string str)`
  Converts a string into an `Ice::Identity`.
- `Ice::ObjectPrx stringToProxy(string str)`
  Converts a string into a proxy.
- `string typeOf(val)`
  Returns the formal Slice type of the argument.

See Also

- Constants and Literals