VHDLwhiz.com

VHDL package: Generic list of protected type

Version:	1.0.1
Date:	March 2, 2022
Product URL:	https://vhdlwhiz.com/product/vhdl-package-generic-list-of-protected-type
Contact email:	jonas@vhdlwhiz.com

This document describes how to use VHDLwhiz's generic list VHDL package to store any data type in the simulator's dynamic memory.

Copyright VHDLwhiz.com

Table of Content

Copyright VHDLwhiz.com

License

The MIT license covers the source code's copyright requirements and terms of use. Refer to the *LICENSE.txt* file in the Zip file for details.

Changelog

These changes refer to the project files, and this document is updated accordingly.

Version	Remarks
1.0.0	Initial release
1.0.1	Adding this user guide to the Zip (no code changes)

Description

The linked-list implementation in this package can store any data type in the simulator's dynamic memory. It mimics the behavior of Python's list class and supports positive and negative indexing.

The list is unidirectional, and you can read from, insert at, or delete any element. But if you use the shorthand `append(data : data_type)` procedure, it will add the new data to the highest index.

In that case, the oldest element is accessible as element number 0, while the newest element is at element -1. The negative indexing makes it easy to read from the list's end, even as it grows.

Consequently, index 1 would refer to the second oldest element and -2 to the second newest. Any element in the list can be indexed from either end.

Refer to the comments above each method prototype for a description of each subprogram, its parameters, and return values.

Compile the list using VHDL-2008 or newer because older language revisions don't support package generics.

Example use cases

You cannot import the *generic_list.vhd* file directly where you want to use it, and that's because it uses package generics that must be mapped to a data type.

Copyright VHDLwhiz.com

First, create a new VHDL file that specifies the data type that the list shall store. Then you can import that VHDL package in your testbench to use it.

To store text strings, for example, first create a new VHDL file named *string_list.vhd* containing the following code:

```
package string_list is new work.generic_list
  generic map(string);
```

Then, import the *string_list.vhd* file in your main testbench and create an instance of it like this:

```
-- Import the string list
use work.string_list.all;

entity generic_list_tb is
end generic_list_tb;

architecture sim of my_tb is

  -- Create the list object
  shared variable list : generic_list;
```

The Zip contains more examples of lists that store other data types.

FIFO behavior

You can achieve a FIFO (first-in, first-out) behavior by using `list.append(data)` to push and `list.get(0)` followed by `list.delete(0)` to fetch the oldest element from the the list:

```
    -- Append new elements to the end
    list.append("Amsterdam");
    list.append("Bangkok");
    list.append("Copenhagen");
    list.append("Damascus");

    -- Pop from the head
    my_var := list.get(0); -- Returns "Amsterdam"
    list.delete(0);
    my_var := list.get(0); -- Returns "Bangkok"
    list.delete(0);
    my_var := list.get(0); -- Returns "Copenhagen"
```


Copyright VHDLwhiz.com

```
    list.delete(0);
    my_var := list.get(0); -- Returns "Damascus"
    list.delete(0);
```

Note that the list protected type doesn't have a `pop()` method like Python's list class. That's because language revisions prior to VHDL-2008 don't have garbage collection. We must delete the elements after using them to prevent memory leaks.

LIFO behavior

To implement a LIFO (last-in, first-out), also known as a *stack*, you can simply read from index -1 to always get the newest element:

```
    -- Append new elements to the end
    list.append("Amsterdam");
    list.append("Bangkok");
    list.append("Copenhagen");
    list.append("Damascus");

    -- Pop from the end
    my_var := list.get(-1); -- Returns "Damascus"
    list.delete(-1);
    my_var := list.get(-1); -- Returns "Copenhagen"
    list.delete(-1);
    my_var := list.get(-1); -- Returns "Bangkok"
    list.delete(-1);
    my_var := list.get(-1); -- Returns "Amsterdam"
    list.delete(-1);
```

Method prototypes

The code listing below shows the declarative region of the *generic_list.vhd* package.

```
package generic_list is

  generic(type data_type);

  type generic_list is protected

    -- Add an item to the end of the list
    --
    -- @param str The data to append
    --
```


Copyright VHDLwhiz.com

```
    procedure append(data : data_type);

    -- Add an item to the list
    --
    -- @param index The list slot to insert <data> at.
    --    A zero or positive index counts from the start of the list.
    --    A negative index counts from the end of the list.
    --    Example:
    --      Insert at the first element: insert(0, my_data)
    --      Insert at the second last element: insert(-1, my_data)
    --
    -- @param data The item to insert at <index>.
    --
    procedure insert(index : integer; data : data_type);

    -- Get an item from the list without deleting it
    --
    -- @param index The list index of the item to get.
    --    Like for insert(), the list index can be negative.
    --    But unlike insert(), insert(-1) returns the last object.
    --
    -- @return The dynamically allocate data_type object
    --
    impure function get(index : integer) return data_type;

    -- Remove an item from the list and free the memory it used
    --
    -- @param index The list index of the object to delete.
    --    The behavor is identical to the get() index parameter.
    --
    procedure delete(index : integer);

    -- Delete all items from the list and free the memory
    procedure clear;

    -- Get the number of items in the list
    --
    -- @return The list's length
    --
    impure function length return integer;

  end protected;

end package;
```


Copyright VHDLwhiz.com

Zip File Content

.

├── **Binary file RW packages - User Manual.pdf**	This document
├── **generic_list_tb.vhd**	Self-checking testbench for the generic list
├── **generic_list.vhd**	The generic list package
├── **How to run.gif**	Screencast showing how to run the testbench
├── **integer_list.vhd**	List of integer type package
├── **LICENSE.txt**	License agreement
├── **project.mpf**	ModelSim/Questa project
├── **real_list.vhd**	List of real type package
├── **run.do**	ModelSim/Questa script for running the testbench
├── **slv8_list.vhd**	List of bytes (std_logic_vector) package
└── **string_list.vhd**	List of string type package

Simulating the design

There is a self-checking testbench in the Zip file (*generic_list_tb.vhd*).

The VHDL testbench should work in any capable VHDL simulator supporting the full VHDL-2008 revision, but the provided *run.do* script only works in ModelSim/Questa.

To run the testbench, open ModelSim/Questa and type in the simulator console:

```
do <path_to_extracted_zip_content>/run.do
runtb
```

Known Issues

The generic list is unsynthesizable and only meant for simulation/testbenches.

Copyright VHDLwhiz.com