ETSI ES 203 790 V1.1.1 (2019-01)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 Language Extensions: Object-Oriented Features

**ETSI Standard**

Reference

DES/MTS-203790-00F\_ed111

Keywords

language, TTCN-3

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° 7803/88

***Important notice***

The present document can be downloaded from:
<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2019.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.
**3GPP**TM and **LTE**TM are trademarks of ETSI registered for the benefit of its Members and
of the 3GPP Organizational Partners.
**oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and
of the oneM2M Partners.
**GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Annex B (normative): Standard Collections

## B.1 The TTCN3\_standard\_collections module

The classes and external functions defined in this module provide users with commonly used data structures.

**module** TTCN3\_standard\_collections {

**function** instanceEqual(object element1, object element2) **return boolean** {
 **return** element1 == element2
}

**public** **external function** createLinkedList(**in** equalsFunctionType equalsFunction := instanceEqual) **return** LinkedList;

**public** **external function** createQueue(in equalsFunctionType equalsFunction := instanceEqual)
**return** Queue;

**public external function** createPriorityQueue(**in** comparatorFunctionType comparatorFunction)
**return** PriorityQueue;

**public external function** createStack(**in** equalsFunctionType equalsFunction := instanceEqual)
**return** Stack;

**public external function** createRingBuffer(**in integer** maxSize) **return** RingBuffer;

**public external function** createHashMap(**in** hashFunctionType hashFunction,
 **in** equalsFunctionType equalsFunction) **return** HashMap;

**public external function** createSet(**in** equalsFunctionType equalsFunction := instanceEqual)
**return** Set;

**public type class @abstract** Exception {

}

**type class @abstract** Iterator {

 **function @abstract** hasNext() **return boolean**;

 **function @abstract** next() **return object**;

}

**type class @abstract** Collection {

 **function** size() **return** **integer**;

 **function** contains(**object** element) **exception** Exception **return** **boolean**;

 **function** @abstract iterator() **return** Iterator;

}

**type class @abstract** List **extends** Collection {

 public function @abstract add(**object** element) **exception** Exception;

 public function @abstract remove(object element) **exception** Exception **return** **boolean**;

public function @abstract get(integer index) **exception** Exception **return** **object**;

}

**public type class @abstract** LinkedList **extends** List {

 **public function @abstract** getFirst() **exception** Exception **return object**;

 **public function @abstract** getLast() **exception** Exception **return object**;

 **public function @abstract** removeFirst() **exception** Exception **return object**;

 **public function @abstract** removeLast() **exception** Exception **return object**;

 **public function @abstract** addFirst(**object** element) **exception** Exception;

 **public function @abstract** addLast(**object** element) **exception** Exception;

}

**public type class @abstract** Queue extends Collection {

 **public function @abstract** add(**object** element) **exception** Exception;

 **public function @abstract** remove() **exception** Exception **return object**;

}

**public type function** comparatorFunctionType(**in object** element1, **in object** element2) **exception** Exception **return integer**;

**public type class @abstract** PriorityQueue **extends** Queue {

}

**public type class @abstract** Stack extends Collection {

 **public function @abstract** push(**object** element) **exception** Exception;

 **public function @abstract** pop() **exception** Exception **return object**;

}

**public type class @abstract** RingBuffer extends Collection {

 **public function @abstract** put(**object** element) **exception** Exception;

 **public function @abstract** get() **exception** Exception **return object**;

 **public function @abstract** capacity() **return integer**;

}

**public type function** hashFunctionType(**in object** element) **exception** Exception **return integer**;

public type function equalsFunctionType(in object element1, in object element2) exception Exception return boolean;

**public type class** **@abstract** HashMap {

 **public function @abstract** put(**object** keyElement, **object** valueElement) **exception** Exception;

 **public function @abstract** get(**object** keyElement) **exception** Exception **return object**;

 **public function @abstract** containsKey(**object** keyElement) **exception** Exception **return boolean**;

 **public function @abstract** remove(**object** keyElement) **exception** Exception **return object**;

 **public function @abstract** keySet() **return Set**;

 public function @abstract values() return List;

 **public function @abstract** size() **return integer**;

}

**public type class @abstract** Set extends Collection {

 **public function @abstract** add(**object** element) **exception** Exception **return** boolean;

 **public function @abstract** remove(**object** element) **exception** Exception;

}

}

## B.1.1 The Collection class

The abstract [Collection](#Collection) class represents a data structure that is a collection of elements. It is used as a base class of more specific collection data structures like lists and sets.

External function and class methods :

* size
Returns the number of elements stored in the LinkedList.
* contains

Returns the value true if the given element is contained at least once in the collection.

* iterator

Returns an Iterator object for iterating over the elements of the collection.

## B.1.2 The List class

The abstract [List](#List) class represents a list of elements where each contained element has an index (starting from 0).

External function and class methods :

* add
Adds the given element to the list.
* remove
Tries to remove one instance of the provided element from the List.
Returns **true** if an element was removed, **false** if no elements were removed.
Please note, that a List might contain the same element several times, in which case only one instance will be removed.
* get
Gets the element at the given index from the list, if the index in in the range (0 .. size()-1).

## B.1.3 The LinkedList class

The abstract [LinkedList](#LinkedList) class represents a double linked data structure for storing objects.
A new Instance can be created via the external function **createLinkedList**. It is derived from the abstract List class.

External function and class methods :

* createLinkedList
Factory function for creating a new LinkedList instance.
* getFirst
Returns the first element of the LinkedList if it is not empty.
Raises an exception if the LinkedList is empty.
* getLast
Returns the last element of the LinkedList if it is not empty.
Raises an exception if the LinkedList is empty.
* removeFirst
Removes and returns the first element of the LinkedList if it is not empty.
Raises an exception if the LinkedList is empty.
* removeLast
Removes and returns the last element of the LinkedList if it is not empty.
Raises an exception if the LinkedList is empty.
* addFirst
Adds a new element as the first element of the LinkedList if this is possible.
Raises an exception in case of error, for example: running out of memory.
* addLast
Adds a new element as the last element of the LinkedList if this is possible.
Raises an exception in case of error, for example: running out of memory.
* iterator
Returns an iterator over the elements of this LinkedList.
The elements are iterated from first to last.
* size
Returns the number of elements stored in the LinkedList.

## B.1.4 The Queue class

The abstract [Queue](#Queue) class represents a queue data structure for storing objects. This data structure uses a First In First Out semantics, meaning that the element added first will also be the element removed first. It is derived from the abstract class Collection.
A new Instance can be created via the external function **createQueue**.

External function and class methods :

* createQueue
Factory function for creating a new Queue instance.
* add
Adds an element to the end Queue if this is possible.
Raises an exception in case of error, for example: running out of memory.
* remove
Removes and returns the first element of the Queue if it is not empty.
Raises an exception if the Queue is empty.
* size
Returns the number of elements stored in the Queue.

## B.1.5 The PriorityQueue class

The abstract [PriorityQueue](#PriorityQueue) class represents a priority queue data structure for storing objects. This data structure stores its elements ordered according to the comparator function. This data structure does not allow null elements.
A new Instance can be created via the external function **createPriorityQueue**. It is derived from the class Queue.

External function and class methods :

* createPriorityQueue
Factory function for creating a new PriorityQueue instance.
* comparatorFunctionType
Used to compare the 2 provided elements for their ordering.
Returns a negative integer if the element1 is less than element2.
Returns 0 if the element1 is equivalent to element2.
Return a positive integer if element1 is greater than element2.
Functions of this type can also raise an exception, for example if the object received as one of their actual parameters is not of the expected class.
* add
Adds an element to the PriorityQueue if this is possible. The element will be added before all elements that are greater than the element and after all elements that are smaller than or equal to the element. Thereby it is ensured that always the smallest element first added to the queue is at the head of the queue.
Raises an exception in case of error, for example: running out of memory, or adding a null object.
* remove
Removes and returns the head element of the PriorityQueue if it is not empty. The head element has the lowest priority among the elements of the PriorityQueue.
Raises an exception if the PriorityQueue is empty.
* size
Returns the number of elements stored in the PriorityQueue.

## B.1.6 The Stack class

The abstract [Stack](#Stack) class represents a stack data structure for storing objects. This data structure uses a Last In First Out semantics, meaning that the element added last will also be the element removed first.
A new Instance can be created via the external function **createStack**.

External function and class methods :

* createStack
Factory function for creating a new Stack instance.
* push
Pushes an element onto the Stack if this is possible.
Raises an exception in case of error, for example: running out of memory.
* pop
Removes and returns the element inserted last from the Stack if it is not empty.
Raises an exception if the Stack is empty.
* size
Returns the number of elements stored in the Stack.

## B.1.7 The RingBuffer class

The abstract [RingBuffer](#RingBuffer) class represents a ringbuffer data structure for storing objects. This data structure uses a First In First Out semantics, with a fixed size limit. This means that the element added first will also be the element removed first. An instance of this collection can only accept elements to the maximum amount it is created for.
A new Instance can be created via the external function **createRingBuffer**.

External function and class methods :

* createRingBuffer
Factory function for creating a new RingBuffer instance, with the provided maximum size.
* put
Adds an element to the end of the RingBuffer if this is possible.
Raises an exception in case of error, for example: reching the maximum allowed size of the buffer.
* get
Removes and returns the first element of the RingBuffer if it is not empty.
Raises an exception if the RingBuffer is empty.
* size
Returns the number of elements stored in the RingBuffer.
* capacity
Returns the maximum capacity of the RingBuffer.

## B.1.8 The HashMap class

The abstract [HashMap](#HashMap) class represents a hashmap data structure for storing key-value pairs of objects. This collection can be indexed with the keyElement part of the pair, to receive the valueElement of the pair.
Pleae note that each key has to be unique according to the given **equalsFunction**.

A new Instance can be created via the external function **createHashMap**.
The hash value of the keyElement object can be calculated using the provided **hashFunctionType** function and the equality of two given keyElements can be determined using the provided **equalsFunctionType** function.

External function and class methods :

* createHashMap
Factory function for creating a new HashMap instance, that will use the provided hashFunction for calculating the hash values of the key element objects and an equalsFunction for determinining the equality of keys The two functions need to fulfil the property that for all pairs of objects o1, o2, if equalsFunction(o1,o2) is true then also hashFunction(o1)==hashFunction(o2) is true.
* hashFunctionType
A behaviour type allowing the user of the collection to provide their implementation for calculating the hash value of their key element objects.
Functions of this type will be called with a key element object as their only parameter and must return an integer value that represents the hash value of the object.
Functions of this type can also raise an exception, for example if the object received as their actual parameter is not of the expected class.
* equalsFunctionType
A behaviour type allowing the user of the collection to provide their implementation with an equality relation between key objects insofar that different object instances of the same content can be seen as equal which allows to ensure the uniqueness property for the keys as there can be no two different key instances k1, k2 where equalsFunction(k1.k2) is true.
* put
Adds a new keyElement – valueElement pair to the HashMap.
If the HashMap already contains a pair with the same keyELement, the old pair is removed before inserting the new pair.
Raises an exception in case of error, for example: running out of memory.
* get
Returns the valueElement part of a keyElement – valueElement pair in the HashMap, if such a pair with the provided keyElement object exists in the HashMap..
Raises an exception if the HashMap has no keyElement – valueElement pair with the provided keyElement.
* containsKey
Returns **true** if the HashMap contains a keyElement – valueElement pair with the provided keyElement, **false** otherwise.
Raises an exception in case of error, for example the hashFunction raised an exception.
* remove
Removes a keyElement – valueElement and returns the valueElement part of a keyElement – valueElement pair in the HashMap, if such a pair with the provided keyElement object exists in the HashMap.
Raises an exception in case of error, for example the hashFunction raised an exception.
* keyset
Returns a Set object containg a set of the keyElements of all the keyElement – valueElement pairs in the HashMap.
* values
Returns a List object containing the valueElement objects of all the keyElement-valueElement pairs in the HashMap
* size
Returns the number of pairs stored in the HashMap.

## B.1.9 The Set class

The abstract [Set](#Set) class represents a set data structure for storing objects. This data structure is unordered and contains unique elements.
A new Instance can be created via the external function **createSet**.

External function and class methods :

* createSet
Factory function for creating a new Set instance. It may be passed an equalsFunction to determine equality and ensure uniqueness of the contained set elements. Per default, instance equality is used.
* add
Adds an element to the Set if this is possible.
Returns true if the element could be added, returns false if the element was already present in the set and so was not added (to ensure uniqueness).
Raises an exception in case of error, for example: running out of memory.
* remove
Removes the provided element from the Set if it is present in the set.
Returns true if the element was located in the Set, false otherwise.
Subclasses might raise an exception.
* contains
Returns **true** if the Set contains the element, **false** otherwise.
Subclasses might raise an exception.
* iterator
Returns an iterator over the elements of this Set.
The elements are not iterated in any particular order.
* size
Returns the number of elements stored in the Queue.

## B.1.10 The Exception class

The abstract [Exception](#Exception) class represents a generic exception that can be raised by standard collections.

Please note, that later the list of raised exception can be updated with more specific exceptions.

## B.1.11 The Iterator class

The abstract [Iterator](#Iterator) class represents an iterator over a collection. An instance of the Iterator class allows to iterate over the elements of a collection.

Class methods :

* hasNext
Returns true if the iterated collection still has elements not yet visited by the iterator.
* next
Returns the next element in the collection and steps the iterator for the upcoming collection.
Raises an exception if the collection has no more elements not yet visited.