ETSI ES 201 873-1 V4.10.1 (2018-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 1: TTCN‑3 Core Language

**ETSI Standard**

Reference

RES/MTS-201873-1v4A1

Keywords

language, methodology, testing, 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 2018.

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 protected for the benefit of its Members.
**GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

### 22.3.6 The Catch operation

The **catch** operation is used to catch exceptions.

***Syntactical Structure***

( *ObjectReference* | **any** **port** | **any from** PortArrayRef ) "." **catch**

[ "(" ( *Signature* [ "," *TemplateInstance* ] ) | *TimeoutKeyword* ")" ]

[ **from** *Address* ]

[ "->" [ **value** ( ValueRef|

 ( "(" { ValueRef[ ":=" [ **@decoded** [ "(" *Expression* ")" ] ] *FieldOrTypeReference* ][","] } ")" )

 ) ]

 [ **sender** ValueRef]

 [ **@index** **value** ValueRef] ]

NOTE 1: *Address* may be an *AddressRef*, a list of *AddressRef*-s or "**any component**".

***Semantic Description***

The **catch** operation is used to catch exceptions raised by a test component or the SUT as a reaction to a procedure call. Exceptions are specified as types and thus, can be treated like messages, e.g. templates can be used to distinguish between different values of the same exception type. If a *Signature* is given in the parameter list, it is possible to omit the *TemplateInstance* part if the **catch** operation shall match any exception value of any of the exception types declared in the definition of the referenced *Signature*.

The **catch** operation removes the top exception from the associated incoming port queue if, and only if, that top exception satisfies all the matching criteria associated with the **catch** operation.

A **catch** operation may be restricted to a certain communication partner in case of one-to-many connections. This restriction shall be denoted by using the **from** keyword followed by a specification of an address or component reference, a list of address or component references or **any component**.

NOTE 2: The one-to-one connection is considered to be a simple case of the one-to-many connections and allows the usage of the **from**-clause.

The (optional) redirection part of the **catch** operation comprises of storing the exception value and/or one or more parts of it and the retrieval of the address of the calling component. The keyword **value** is used to retrieve the value of an exception and/or the parts of it and the keyword **sender** is used when it is required to retrieve the address of the sender.

When assigning individual fields of an exception, encoded payload fields can be decoded prior to assignment using the **@decoded** modifier. In this case, the referenced field on the right hand sided of the assignment shall be one of the **bitstring**, **hexstring**, **octetstring**, **charstring** or **universal** **charstring** types. It shall be decoded into a value of the same type as the variable on the left hand side of the assignment. Failure of this decoding shall cause a test case error. In case the referenced field is of the **universal** **charstring** type, the **@decoded** clause can contain an optional parameter defining the encoding format. The parameter shall be of the **charstring** type and it shall contain one of the strings allowed for the **decvalue\_unichar** function (specified in clause C.5.4). Any other value shall cause an error. In case the referenced field is not a **universal** **charstring**, the optional parameter shall not be present.

The **catch** operation may be part of the response and exception handling part of a **call** operation or be used to determine an alternative in an **alt** statement. If the **catch** operation is used in the accepting part of a **call** operation, the information about port name and signature reference to indicate the procedure that raised the exception is redundant, because this information follows from the **call** operation. However, for readability reasons (e.g. in case of complex **call** statements) this information shall be repeated.

**The Timeout exception**

There is one special **timeout** exception that can be caught by the **catch** operation. The **timeout** exception is an emergency exit for cases where a called procedure neither replies nor raises an exception within a predetermined time (see clause 22.3.1).

**Catch any exception**

A **catch** operation with no argument list allows any valid exception to be caught. The most general case is without using the **from** keyword. *CatchAnyException* will also catch the **timeout** exception.

**Catch any exception for specific signature**

A **catch** operation using only a *Signature* reference in the argument list allows any valid exception for that signature to be caught.

**Catch on any port**

To **catch** an exception on any port use the **any** keyword.

**Catch on any port from a port array**

To **catch** an exception on any port from a specific port array, indices use the **any from** *PortArrayRef*syntax where PortArrayRefshallbe areference to a port array identifier**.** It is also possible to store the index of a port in a single-dimensional port array at which the operation was successful to a variable of type integer or, in case of multi‑dimensional port arrays the index of the successful port to an integer array or record of integer variable. When checking the port array for matching exceptions, the port indices to be checked are iterated from lowest to highest. If the port array is multi-dimensional, then the ports are iterated over from innermost to outermost array dimension from lowest to highest index for each dimension, e.g. [0][0], [0][1], [1][0], [1][1]. The first port which matches all the criteria will cause the operation to be successful even if other ports in the array would also meet the criteria.

The catch on any port from a port array operation cannot be used to catch a call timeout.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5 and shown in table 16, the following restrictions apply:

1. The catch operation shall only be used at procedure-based ports. The type of the caught exception shall be specified in the signature of the procedure that raised the exception.
2. The type definition of the port shall include in its out or inout list the name of the procedure to which the exception belongs.
3. No binding of the incoming values to the terms of the expression or to the template shall occur. The assignment of the exception values to variables shall be made in the assignment part of the catch operation.
4. Catching timeout exceptions shall be restricted to the exception handling part of a call. No further matching criteria (including a from part) and no assignment part is allowed for a catch operation that handles a timeout exception.
5. Exception values accepted by catch any exception shall not be assigned to a variable, i.e. the value clause shall not be present.
6. If CatchAnyException is used in the response and exception handling part of a call operation, it shall only treat exceptions raised by the procedure invoked by the call operation.
7. All AddressRef items in the from clause and all ValueRef items in the sender clause shall be of type address, component or of the address type bound to the port type (see clause 6.2.9) of the port instance referenced in the catch operation. No AddressRef in the from clause shall contain the special value null at the time of the operation.
8. The PortArrayRef shall be a reference to a completely initialized port array.
9. The index redirection shall only be used when the operation is used on an any from port array construct.
10. If the index redirection is used for single-dimensional arrays, the type of the integer variable shall allow storing the highest index of the respective port array.
11. If the index redirection is used for multi-dimensional arrays, the size of the integer array or record of integer type shall exactly be the same as the dimension of the respective port array, and the its type shall allow storing the highest index (from all dimensions) of the port array.
12. If a variable referenced in the value, sender or @index clause is a lazy or fuzzy variable, the expression assigned to this variable is equal to the result produced by the catch operation, i.e. later evaluation of the lazy or fuzzy variable does not lead to repeated invocation of the catch operation.
13. If the catch operation contains both from and sender clause, the variable or parameter referenced in the sender clause shall be type compatible with the template in the from clause. If the operation contains a sender clause but no from clause, the sender shall be type compatible with the variable or parameter referenced in the sender clause.

NOTE 3: An error due to a type mismatch may happen if the types in the receive part are not compatible to the types in the assignment part or, if the from clause is missing, but the type of the sender can be determined and it is not type compatible with the type in the sender clause.

1. When assigning implicitly decoded exception fields (by using the @decoded modifier) in cases where the value or template to be matched uses the MatchDecodedContent (decmatch) matching for the parameter to be stored, the type of the template in the MatchDecodedContent matching shall be type-compatible to the type of the variable the decoded field is stored into.
2. The referenced value associated with Ref or the return type associated with FunctionInstance followed by the catch keyword, shall be of a port type.
3. If no *TemplateInstance* is provided in the parameter list, then also no **value** clause shall be present in the redirection part.

***Examples***

EXAMPLE 1: Basic catch

 myPort.**catch**(MyProc, **integer:** v\_myVar); // Catches an integer exception of value

 // v\_myVar raised by MyProc at port myPort.

 myPort.**catch**(MyProc, v\_myVar); // Is an alternative to the previous example.

 myPort.**catch**(MyProc, v\_a<v\_b); // Catches a boolean exception

 myPort.**catch**(MyProc, MyType:{5, v\_myVar}); // In-line template definition of an exception value.

 myPort.**catch**(MyProc, **charstring**:"Hello")**from** myPeer; // Catches "Hello" exception from myPeer

EXAMPLE 2: Catch with storing value and/or sender in variables

 myPort.**catch**(MyProc, MyType:?) **from** myPartner -> **value** v\_myVar;

 // Catches an exception from myPartner and assigns its value to v\_myVar.

 myPort.**catch**(MyProc, s\_myTemplate(5)) -> **value** v\_myVarTwo **sender** myPeer;

 // Catches an exception, assigns its value to v\_myVarTwo and retrieves the

 // address of the sender.

 myPort.**catch**(MyProc, s\_myTemplate(5)) -> **value** (v\_myVarThree:= f1)

 **sender** myPeer;

 // Catches an exception, assigns the value of its field f1 to v\_myVarThree and retrieves the

 // address of the sender.

 // Handling encoded exception payload:

 **type** MyException **record** {

 **...**

 }

 **type** CommonException **record** {

 **integer** exceptionId,

 **octetstring** payload

 }

 **signature** S() **exception** (CommonException);

 ...

 **var** MyException v\_myVar;

 myPort.**catch** (S, CommonException:{exceptionId := 25, payload := **decmatch** MyException:? }) -> **value** (v\_myVar := **@decoded** payload);

 // The encoded payload field of the caught exception is decoded and matched with m\_excTemplate;

 // if the matching is successful the decoded payload is stored in v\_myVar.

EXAMPLE 3: The Timeout exception

 myPort.**call**(MyProc:{5, v\_myVar}, 20E-3) {

 [] myPort.**getreply**(MyProc:{?, ?}) { }

 [] myPort.**catch**(**timeout**) { // timeout exception after 20ms

 **setverdict**(**fail**);

 **stop**;

 }

 }

EXAMPLE 4: Catch any exception

 myPort.**catch**;

 myPort.**catch** **from** myPartner;

 myPort.**catch** -> **sender** v\_mySenderVar;

 myPort.catch(MyProc); // catch any exception raised by procedure MyProc

EXAMPLE 5: Catch on any port

 **any port**.**catch;**

EXAMPLE 6: Catch on any port from port array

 **type** **port** MyPort **procedure** { **inout** MyProc }

 **type** **component** MyComponent {

 **port** MyPort p[10][10];

 }

 **var** **integer** v\_i[2];

 **any** **from** p.**catch**(MyProc, MyType:?) -> **@index** **value** v\_i;

 // Catching an incoming exception of type MyType on any port in the port array p and

 // storing the index of the port on which the matching was successful first

#### A.1.6.4.2 Port operations

CommunicationStatements ::= [SendStatement](#TSendStatement) |

 [CallStatement](#TCallStatement) |

 [ReplyStatement](#TReplyStatement) |

 [RaiseStatement](#TRaiseStatement) |

 [ReceiveStatement](#TReceiveStatement) |

 [TriggerStatement](#TTriggerStatement) |

 [GetCallStatement](#TGetCallStatement) |

 [GetReplyStatement](#TGetReplyStatement) |

 [CatchStatement](#TCatchStatement) |

 [CheckStatement](#TCheckStatement) |

 [ClearStatement](#TClearStatement) |

 [StartStatement](#TStartStatement) |

 [StopStatement](#TStopStatement) |

 [HaltStatement](#THaltStatement) |

 [CheckStateStatement](#TCheckStateStatement)

SendStatement ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) [Dot](#TDot) [PortSendOp](#TPortSendOp)

PortSendOp ::= [SendOpKeyword](#TSendOpKeyword) "(" [TemplateInstance](#TTemplateInstance)")" [[ToClause](#TToClause)]

SendOpKeyword ::= "send"

ToClause ::= [ToKeyword](#TToKeyword) ([TemplateInstance](#TTemplateInstance)|

 [AddressRefList](#TAddressRefList) |

 [AllKeyword](#TAllKeyword) [ComponentKeyword](#TComponentKeyword)

 )

AddressRefList ::= "(" [TemplateInstance](#TTemplateInstance){"," [TemplateInstance](#TTemplateInstance)} ")"

ToKeyword ::= "to"

CallStatement ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) [Dot](#TDot) [PortCallOp](#TPortCallOp) [[PortCallBody](#TPortCallBody)]

PortCallOp ::= [CallOpKeyword](#TCallOpKeyword) "(" [CallParameters](#TCallParameters) ")" [[ToClause](#TToClause)]

CallOpKeyword ::= "call"

CallParameters ::= [TemplateInstance](#TTemplateInstance) ["," [CallTimerValue](#TCallTimerValue)]

CallTimerValue ::= [Expression](#TExpression) | [NowaitKeyword](#TNowaitKeyword)

NowaitKeyword ::= "nowait"

PortCallBody ::= "{" [CallBodyStatementList](#TCallBodyStatementList) "}"

CallBodyStatementList ::= {[CallBodyStatement](#TCallBodyStatement) [[SemiColon](#TSemiColon)]}+

CallBodyStatement ::= [CallBodyGuard](#TCallBodyGuard) [StatementBlock](#TStatementBlock)

CallBodyGuard ::= [AltGuardChar](#TAltGuardChar) [CallBodyOps](#TCallBodyOps)

CallBodyOps ::= [GetReplyStatement](#TGetReplyStatement) | [CatchStatement](#TCatchStatement)

ReplyStatement ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) [Dot](#TDot) [PortReplyOp](#TPortReplyOp)

PortReplyOp ::= [ReplyKeyword](#TReplyKeyword) "(" [TemplateInstance](#TTemplateInstance) [[ReplyValue](#TReplyValue)] ")" [[ToClause](#TToClause)]

ReplyKeyword ::= "reply"

ReplyValue ::= [ValueKeyword](#TValueKeyword) [TemplateBody](#TTemplateBody)

/\* STATIC SEMANTICS - TemplateBody shall be type compatible with the return type. It shall evaluate to a value or template (literal or template instance) conforming to the template(value) restriction. \*/

RaiseStatement ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) [Dot](#TDot) [PortRaiseOp](#TPortRaiseOp)

PortRaiseOp ::= [RaiseKeyword](#TRaiseKeyword) "(" [Signature](#TSignature) "," [TemplateInstance](#TTemplateInstance)")"

 [[ToClause](#TToClause)]

RaiseKeyword ::= "raise"

ReceiveStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortReceiveOp](#TPortReceiveOp)

PortOrAny ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) | ([AnyKeyword](#TAnyKeyword) ([PortKeyword](#TPortKeyword) | [FromKeyword](#TFromKeyword) ValueRef))

PortReceiveOp ::= [ReceiveOpKeyword](#TReceiveOpKeyword) ["("[TemplateInstance](#TTemplateInstance)")"] [[FromClause](#TFromClause)] [[PortRedirect](#TPortRedirect)]

ReceiveOpKeyword ::= "receive"

FromClause ::= [FromKeyword](#TFromKeyword) ([TemplateInstance](#TTemplateInstance) |

 [AddressRefList](#TAddressRefList) |

 [AnyKeyword](#TAnyKeyword) [ComponentKeyword](#TComponentKeyword)

 )

FromKeyword ::= "from"

PortRedirect ::= [PortRedirectSymbol](#TPortRedirectSymbol) (([ValueSpec](#TValueSpec) [[SenderSpec](#TSenderSpec)] [[IndexSpec](#TIndexSpec)]) |

 ([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

 [IndexSpec](#TIndexSpec)

 )

PortRedirectSymbol ::= "->"

ValueSpec ::= [ValueKeyword](#TValueKeyword) (ValueRef | ("(" [SingleValueSpec](#TSingleValueSpec) {"," [SingleValueSpec](#TSingleValueSpec)} ")"))

SingleValueSpec ::= ValueRef [[AssignmentChar](#TAssignmentChar) [ DecodedModifier ["(" [Expression] ")"] ]

 [FieldReference](#TFieldReference) [ExtendedFieldReference](#TExtendedFieldReference)]

/\*STATIC SEMANTICS – FieldReference shall not be ParRef and ExtendedFieldReference shall not be TypeDefIdentifier\*/

ValueKeyword ::= "value"

SenderSpec ::= [SenderKeyword](#TSenderKeyword) ValueRef

SenderKeyword ::= "sender"

TriggerStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortTriggerOp](#TPortTriggerOp)

PortTriggerOp ::= [TriggerOpKeyword](#TTriggerOpKeyword) ["(" [TemplateInstance](#TInLineTemplate)  ")"] [[FromClause](#TFromClause)]

 [[PortRedirect](#TPortRedirect)]

TriggerOpKeyword ::= "trigger"

GetCallStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortGetCallOp](#TPortGetCallOp)

PortGetCallOp ::= [GetCallOpKeyword](#TGetCallOpKeyword) ["(" [TemplateInstance](#TTemplateInstance) ")"] [[FromClause](#TFromClause)]

 [[PortRedirectWithParam](#TPortRedirectWithParam)]

GetCallOpKeyword ::= "getcall"

PortRedirectWithParam ::= [PortRedirectSymbol](#TPortRedirectSymbol) [RedirectWithParamSpec](#TRedirectWithParamSpec)

RedirectWithParamSpec ::= ([ParamSpec](#TParamSpec) [[SenderSpec](#TSenderSpec)] [[IndexSpec](#TIndexSpec)]) |

 ([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

 [IndexSpec](#TIndexSpec)

ParamSpec ::= [ParamKeyword](#TParamKeyword) [ParamAssignmentList](#TParamAssignmentList)

ParamKeyword ::= "param"

ParamAssignmentList ::= "(" ([AssignmentList](#TAssignmentList) | [VariableList](#TVariableList)) ")"

AssignmentList ::= [VariableAssignment](#TVariableAssignment) {"," [VariableAssignment](#TVariableAssignment)}

VariableAssignment ::= ValueRef [AssignmentChar](#TAssignmentChar) [ [DecodedModifier](#TDecodedModifier) ["(" [Expression](#TExpression)] ")"]

 [Identifier](#TIdentifier)

VariableList ::= [VariableEntry](#TVariableEntry) {"," [VariableEntry](#TVariableEntry)}

VariableEntry ::= ValueRef | [Minus](#TMinus)

GetReplyStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortGetReplyOp](#TPortGetReplyOp)

PortGetReplyOp ::= [GetReplyOpKeyword](#TGetReplyOpKeyword) ["(" [TemplateInstance](#TTemplateInstance) [[ValueMatchSpec](#TValueMatchSpec)]

 ")"] [[FromClause](#TFromClause)] [[PortRedirectWithValueAndParam](#TPortRedirectWithValueAndParam)]

PortRedirectWithValueAndParam ::= [PortRedirectSymbol](#TPortRedirectSymbol) [RedirectWithValueAndParamSpec](#TRedirectWithValueAndParamSpec)

RedirectWithValueAndParamSpec ::= ([ValueSpec](#TValueSpec) [[ParamSpec](#TParamSpec)] [[SenderSpec](#TSenderSpec)]

 [[IndexSpec](#TIndexSpec)]) | [RedirectWithParamSpec](#TRedirectWithParamSpec)

GetReplyOpKeyword ::= "getreply"

ValueMatchSpec ::= [ValueKeyword](#TValueKeyword) [TemplateInstance](#TTemplateInstance)

CheckStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortCheckOp](#TPortCheckOp)

PortCheckOp ::= [CheckOpKeyword](#TCheckOpKeyword) ["(" [CheckParameter](#TCheckParameter) ")"]

CheckOpKeyword ::= "check"

CheckParameter ::= [CheckPortOpsPresent](#TCheckPortOpsPresent) |

 [FromClausePresent](#TFromClausePresent) |

 [RedirectPresent](#TRedirectPresent)

FromClausePresent ::= [FromClause](#TFromClause) [[PortRedirectSymbol](#TPortRedirectSymbol) (([SenderSpec](#TSenderSpec)

 [[IndexSpec](#TIndexSpec)]) |

 [IndexSpec](#TIndexSpec))]

RedirectPresent ::= [PortRedirectSymbol](#TPortRedirectSymbol) (([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

 [IndexSpec](#TIndexSpec))

CheckPortOpsPresent ::= [PortReceiveOp](#TPortReceiveOp) |

 [PortGetCallOp](#TPortGetCallOp) |

 [PortGetReplyOp](#TPortGetReplyOp) |

 [PortCatchOp](#TPortCatchOp)

CatchStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortCatchOp](#TPortCatchOp)

PortCatchOp ::= [CatchOpKeyword](#TCatchOpKeyword) ["(" [CatchOpParameter](#TCatchOpParameter) ")"] [[FromClause](#TFromClause)] [[PortRedirect](#TPortRedirect)]

CatchOpKeyword ::= "catch"

CatchOpParameter ::= [Signature](#TSignature) [ "," [TemplateInstance](#TTemplateInstance) ] | [TimeoutKeyword](#TTimeoutKeyword)

ClearStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [ClearOpKeyword](#TClearOpKeyword)

PortOrAll ::= [ObjectReference](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7707-v5.docx#TObjectReference) | [AllKeyword](#TAllKeyword) [PortKeyword](#TPortKeyword)

ClearOpKeyword ::= "clear"

StartStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [StartKeyword](#TStartKeyword)

StopStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [StopKeyword](#TStopKeyword)

StopKeyword ::= "stop"

HaltStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [HaltKeyword](#THaltKeyword)

HaltKeyword ::= "halt"

AnyKeyword ::= "any"

CheckStateStatement ::= [PortOrAllAny](#TPortOrAllAny) [Dot](#TDot) [CheckStateKeyword](#TCheckStateKeyword) "(" [SingleExpression](#TSingleExpression)

 ")"

PortOrAllAny ::= [PortOrAll](#TPortOrAll) | [AnyKeyword](#TAnyKeyword) [PortKeyword](#TPortKeyword)

CheckStateKeyword ::= "checkstate"