### 22.2.2 The Receive operation

The **receive** operation is used to receive a message from an incoming message port queue.

***Syntactical Structure***

( *Port* | **any** **port** | **any from** PortArrayRef ) "." **receive**

[ "(" *TemplateInstance* ")" ]

[ **from** *Address* ]

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

 ( "(" { *VariableRef* [ ":=" [ **@decoded** [ "(" *Expression* ")" ] ]

 *FieldOrTypeReference* ][","] } ")" )

 ) ]

 [ **sender** *VariableRef* ]

 [ @**index** **value** *VariableRef* ] ]

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

***Semantic Description***

The **receive** operation is used to receive a message from an incoming message port queue. The message may be specified by referencing a defined template or can be defined as an in-line template.

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

If the match is not successful, the top message shall not be removed from the port queue i.e. if the **receive** operation is used as an alternative of an **alt** statement and it is not successful, the execution of the test case shall continue with the next alternative of the **alt** statement.

**Matching criteria**

The matching criteria are related to the type and value of the message to be received. The type and value of the message to be received are determined by the argument of the **receive** operation, i.e. may either be derived from the defined template or be specified in-line. An optional type field in the matching criteria to the **receive** operation shall be used to avoid any ambiguity of the type of the value being received.

NOTE 2: Encoding attributes also participate in matching in an implicit way, by preventing the decoder to produce an abstract value from the received message encoded in a different way than specified by the attributes.

**Receiving from a specific sender**

In the case of one-to-many connections the **receive** operation may be restricted to a certain communication partner. This restriction shall be denoted using the **from** keyword.

**Storing the received message and parts of the received message**

If the match is successful, the value removed from the port queue and/or parts of this value can be stored in variables or formal parameters. This is denoted by the symbol '->' and the keyword **value**.

When the keyword **value** is followed by a name of a variable or formal parameter, the whole received message shall be stored in the variable or formal parameter. The variable or formal parameter shall be type compatible with the received message.

When the keyword **value** is followed by an assignment list enframed by a pair of parentheses, the whole received message and/or one or more parts of it can be stored. A list element allows storing the value of a referenced field of the template type (right hand side of assignment symbol “:=”) in a variable or formal parameter (left hand side “:=”).. The variable or formal parameter shall be type compatible with the type on the right hand side of the assignment symbol. If the list includes an element that refer to a variable or formal parameter but without any assignment symbol and right hand side the whole message shall be stored to that variable or formal parameter.

When assigning individual fields of a message, 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.

NOTE 3: The model of the behaviour of this implicit decoding is defined in clause B.1.2.9.

NOTE 4: The **@decoded** clause is typically used together with the **decmatch** matching mechanism in the matching part of the receive statement. Since the decoding procedures for assignment and matching are virtually the same, TTCN-3 tools can be optimized in such a way that only one call to the decoder is made when the receiving statement contains both **decmatch** matching mechanism and **@decoded** assignment for the same payload field.

**Storing the sender**

It is also possible to retrieve and store the component reference or address of the sender of a message. This is denoted by the keyword **sender**.

When the message is received on a connected port, only the component reference is stored in the following the **sender** keyword, but the test system shall internally store the component name too, if any (to be used in logging).

**Receive any message**

A **receive** operation with no argument list for the type and value matching criteria of the message to be received shall remove the message on the top of the incoming port queue (if any) if all other matching criteria are fulfilled.

**Receive on any port**

To **receive** a message on any port, use the **any port** keywords.

**Receive on any port from a port array**

To **receive** a message on any port from a specific port array, 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 messages, 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.

**Stand-alone receive**

The **receive** operation can be used as a stand-alone statement in a behaviour description. In this latter case the **receive** operation is considered to be shorthand for an **alt** statement with the **receive** operation as the only alternative.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5 and shown in table 15, the following restrictions apply:

a) When defining the message in-line, the optional type part shall be present whenever the type of the message being received is ambiguous.

b) The **receive** operation shall only be used on message-based ports and the type of the value to be received shall be included in the list of incoming types of the port type definition.

c) No binding of the incoming values to the terms of the expression or to the template shall occur.

d) A message received by *receive any message* shall not be stored, i.e. the **value** clause shall not be present.

e) Type mismatch at storing the received value or parts of the received value and storing the sender shall cause an error.

f) *AddressRef* for retrieving the sending entity shall be of type **address**, **component** or of the type provided in the address declaration of the port type of the port instance referenced in the **receive** operation. No *AddressRef* shall contain the special value **null** at the time of the operation.

g) The *PortArrayRef* shall be a reference to a completely initialized port array.

h) The index redirection shall only be used when the operation is used on an any from port array construct.

i) If the index redirection is used for single-dimensional port arrays, the type of the integer variable shall allow storing the highest index of the respective array.

j) If the index redirection is used for multi-dimensional port arrays, the size of the integer array or record of integer type shall exactly be the same as the dimension of the respective array, and its type shall allow storing the highest index (from all dimensions) of the array.

k) 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 **receive** operation i.e. later evaluation of the lazy or fuzzy variable does not lead to repeated invocation of the **receive** operation.

l) If the **receive** 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.

m) When assigning implicitly decoded message fields (by using the **@decoded** modifier) in cases where the value or template to be matched uses the *MatchDecodedContent* (**decmatch**) matching for the field 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.

***Examples***

EXAMPLE 1: Basic receive

 MyPort.**receive**(MyTemplate(5, MyVar)); // Matches a message that fulfils the conditions

 // defined by template MyTemplate at port MyPort.

 MyPort.**receive**(A<B); // Matches a Boolean value that depends on the outcome of A<B

 MyPort.**receive**(**integer**:MyVar); // Matches an integer value with the value of MyVar

 // at port MyPort

 MyPort.**receive**(MyVar); // Is an alternative to the previous example

EXAMPLE 2: Receiving from a sender, storing the message, parts of the message or the sender

 **type** MyPayloadType **record** {

 **integer** messageId,

 ContentType content

 }

 **type** MyType2 **record** {

 Header header,

 **octetstring** payload

 }

 **template** MyType MyTemplate := {

 messageId := 42,

 content := ?

 }

 ...

 **var** MyPayloadType MyVar;

 **var** **integer** MyMessageIdVar, MyIntegerVar;

 **var** **charstring** MyCharstringVar;

 **var** **address** MyPeer;

 **var** **octetstring** MyVarOne := '00ff'O;

 MyPort.**receive**(**charstring**:"Hello")**from** MyPeer; // Matches charstring "Hello" from MyPeer

 MyPort.**receive**(MyType:?) -> **value** MyVar; // The value of the received message is

 // assigned to MyVar.

 MyPort.**receive**(MyType:?) -> **value** (MyVar, MyMessageIdVar:= messageId)

 // The value of the received message is stored in the variable

 // MyVar and the value of the messageId field of the received

 // message is stored in the variable MyMessageIdVar.

 MyPort.**receive**(anytype:?) -> **value** (MyIntegerVar := integer)

 // If the received value is an integer, it is stored in the variable

 // MyIntegerVar, a test case error otherwise.

 MyPort.**receive**(charstring:?) -> **value** (MyCharstringVar)

 // The received value is stored in the variable MyCharstringVar;

 // Note that it is the same as to write "**value** MyCharstringVar"

 MyPort.**receive**(A<B) -> **sender** MyPeer; // The address of the sender is assigned to MyPeer

 MyPort.**receive**(MyType:{5, MyVarOne}) -> **value** MyVar **sender** MyPeer;

 // The received message value is stored in MyVarTwo and the sender address is stored in MyPeer.

 MyPort.**receive**(MyType2:{header := ?, payload := **decmatch** MyTemplate }) -> **value** (MyVar := **@decoded** payload);

 // The encoded payload field of the received message is decoded and matched with

 // MyTemplate; if the matching is successful the decoded payload is stored in MyVar.

EXAMPLE 3: Receive any message

 MyPort.**receive**; // Removes the top value from MyPort.

 MyPort.**receive** **from** MyPeer; // Removes the top message from MyPort if its sender is
 MyPeer

 MyPort.**receive** -> **sender** MySenderVar; // Removes the top message from MyPort and assigns

 // the sender address to MySenderVar

EXAMPLE 4: Receive on any port

 **any port**.**receive**(MyMessage);

EXAMPLE 5: Receive on any port from a port array

 **type** **port** MyPort **message** { **inout** **integer** }

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

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

 }

 **var** **integer** i[2];

 **any** **from** p.**receive**(MyMessage) -> **@index value** i;

 // checking receiving MyMessage on any port of the port array p and storing the index of the

 // port on which the matching was successful first; if, for example MyMessage is matched first

 // on p[4,2], the content of i will be {4,2}