### 21.1.1 The Connect and Map operations

The **connect** operation and the **map** operation are used to setup connections to the SUT or between test components.

***Syntactical Structure***

**connect** "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")"

**map** "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")"

[ **param** "(" [ { *ActualPar* [","] }+ ] ")" ]

***Semantic Description***

With both the **connect** operation and the **map** operation, the ports to be connected are identified by the component references of the components to be connected and the names of the ports to be connected.

The operation **mtc** identifies the MTC, the operation **system** identifies the test system interface and the operation **self** identifies the test component in which **self** has been called (see clause 6.2.11). All these operations can be used for identifying and connecting ports.

Both the **connect** and **map** operations can be called from any behaviour definition except for the control part of a module. However before either operation is called, the components to be connected shall have been created and their component references shall be known together with the names of the relevant ports.

Both the **map** and **connect** operations allow the connection of a port to more than one other port. It is not allowed to connect to a mapped port or to map to a connected port.

Applying a **map** or **connect** operation to ports which are already mapped or connected has no effect on the test behaviour or test configuration, i.e. test execution continues as if the operation has not been invoked.

NOTE: Please note that also triMap or tciConnect respectively will not be invoked in such a case.

The **map** operation provides an optional parameter list for configuration purposes. This allows to pass values needed for dynamic runtime configuration. If a parameter list is present, the actual parameters shall conform to the **map** **param** clause of the port type declaration of the system port used.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clauses 5 and 21, the following restrictions apply:

a) For both the **connect** and **map** operations, only consistent connections are allowed.

Assuming the following:

1) ports PORT1 and PORT2 are the ports to be connected;

2) inlist-PORT1 defines the messages or procedures of the in-direction of PORT1;

3) outlist-PORT1defines the messages or procedures of the out-direction of PORT1;

4) inlist-PORT2 defines the messages or procedures of the in-direction of PORT2; and

5) outlist-PORT2 defines the messages or procedures of the out-direction of PORT2.

b) The **connect** operation is allowed if and only if:

outlist-PORT1 ⊆ inlist-PORT2 and outlist-PORT2 ⊆ inlist-PORT1.

c) The **map** operation (assuming PORT2 is the test system interface port) is allowed if and only if:

outlist-PORT1 ⊆ outlist-PORT2 *and* inlist-PORT2 ⊆ inlist-PORT1.

d) In all other cases, the operations shall not be allowed.

e) Since TTCN‑3 allows dynamic configurations and addresses, not all of these consistency checks can be made statically at compile-time. All checks, which could not be made at compile-time, shall be made at runtime and shall lead to a test case error when failing.

f) In addition, the restrictions on allowed and disallowed connections described in clause 9.1 apply.

g) In **map** operations, **param** clauses are optional. If in a **map** operation a **param** clause is present, exactly one of the components referenced by the operation shall be the **system** component reference, the type of the system component shall be known in the context of the operation either via a **system** clause or via a **runs on** clause in a **testcase** without **system** clause, the type of the system port to which the operation is applied shall include a **map param** declaration, and the actual parameters shall conform to the **map** **param** clause of the port type declaration of the system port used.

h) If the type of the component referenced in a connection operation is known (either when the component reference is a variable or value returned from a function or the type is defined in the runs on, mtc or system clause of the calling function), the referenced port declaration shall be present in this component type.

***Examples***

EXAMPLE 1: Simple map and connect

// It is assumed that the ports Port1, Port2, Port3 and PCO1 are properly defined and declared

// in the corresponding port type and component type definitions

:

**var** MyComponentType MyNewPTC;

MyNewPTC := MyComponentType.**create**;

:

**connect**(MyNewPTC:Port1, **mtc**:Port3);

**map**(MyNewPTC:Port2, **system**:PCO1);

:

// In this example a new component of type MyComponentType is created and its reference stored

// in variable MyNewPTC. Afterwards in the connect operation, Port1 of this new component

// is connected with Port3 of the MTC. By means of the map operation, Port2 of the new component

// is then connected to port PCO1 of the test system interface

EXAMPLE 2: Parameterized map

:

**var** MyConfigType MyConfig := { option := 1, lock := false};

:

**map**(**mtc**:Port4, **system**:PCO2) param (MyConfig);

:

// In this example by means of the map operation, Port4 of the MTC is connected to the port PCO2

// of the test system interface, and additionally a parameter containing configuration options

// for the connection is passed.

EXAMPLE 3: Port visibility

**type** **port** P **message** { **inout** **integer**; }  
**type** **component** C1 { **port** P p1; }  
**type** **component** C2 { **port** P p1, p2; }  
  
**testcase** TC **runs on** C1 **system** C1  
{  
  **var** C1 v\_ptc := C2.**create**; // valid assignment, instance of C2 is compatible with C1 type  
  **connect** (**self**:p1, v\_ptc:p1); // valid, p1 is present in C1 type definition  
 **disconnect** (**self**:p1, v\_ptc:p1);  
 **connect** (**self**:p1, v\_ptc:p2); // invalid, although the real instance in v\_ptc is of the  
 // C2 type, the variable itself is of the C1 type making the p2 port invisible to the  
 // mapping operation  
}

### 21.1.2 The Disconnect and Unmap operations

The **disconnect** and **unmap** operations are the opposite operations of **connect** and **map**.

***Syntactical Structure***

**disconnect** [ ( "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")" ) |

( "(" *PortRef* ")" ) |

( "(" *ComponentRef* ":" **all** **port** ")" ) |

( "(" **all** **component** ":" **all** **port** ")" ) ]

**unmap** [ ( "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")"

[ **param** "(" [ { *ActualPar* [","] }+ ] ")" ] ) |

( "(" *PortRef* ")" [ **param** "(" [ { *ActualPar* [","] }+ ] ")" ] ) |

( "(" *ComponentRef* ":" **all** **port** ")" ) |

( "(" **all** **component** ":" **all** **port** ")" ) ]

***Semantic Description***

The **disconnect** and **unmap** operations perform the disconnection (of previously connected) ports of test components and the unmapping of (previously mapped) ports of test components and ports in the test system interface.

Both, the **disconnect** and **unmap** operations can be called from any component if the relevant component references together with the names of the relevant ports are known. A **disconnect** or **unmap** operation has only an effect if the connection or mapping to be removed has been created beforehand.

To ease **disconnect** and **unmap** operations related to all connections and mappings of a component or a port, it is allowed to use **disconnect** and **unmap** operations with one argument only. This one argument specifies one side of the connections to be disconnected or unmapped. The **all port** keyword can be used to denote all ports of a component.

The usage of a **disconnect** or **unmap** operation without any parameters is a shorthand form for using the operation with the parameter **self**:**all port**. It disconnects or unmaps all ports of the component that calls the operation.

The **all component** keyword shall only be used in combination with the **all port** keyword, i.e. **all** **component**:**all** **port**, and shall only be used by the MTC. Furthermore, the **all component**:**all port** argument shall be used as the one and only argument of a **disconnect** or **unmap** operation and it allows to release all connections and mappings of the test configuration.

Similar to the **map** operation, **unmap** provides an optional parameter list for configuration purposes. If a parameter list is present, the actual parameters shall conform to the **unmap** **param** clause of the port type declaration of the system port used. It allows to pass values needed for dynamic runtime configuration.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clauses 5 and 21, the following restrictions apply:

a) In an **unmap** operation, a **param** clause shall only be present if the system port to which the **param** clause belongs to is explicitly referenced.

b) In **unmap** operations, **param** clauses are optional. If in an **unmap** operation a **param** clause is present, exactly one of the components referenced by the operation shall be the **system** component reference, the type of the system component shall be known in the context of the operation either via a **system** clause or via a **runs on** clause in a **testcase** without **system** clause, the type of the system port to which the operation is applied shall include an **unmap param** declaration and the actual parameters shall conform to the **unmap** **param** clause of the port type declaration of the system port used.

c) If the type of the component referenced in a connection operation is known (either when the component reference is a variable or value returned from a function or the type is defined the runs on, mtc or system clause of the calling function), the referenced port declaration shall be present in this component type.

***Examples***

EXAMPLE 1: Disconnect/unmap for specific connections

**connect**(MyNewComponent:Port1, **mtc**:Port3);

**map**(MyNewComponent:Port2, **system**:PCO1);

:

**disconnect**(MyNewComponent:Port1, **mtc**:Port3); // disconnect previously made connection

**unmap**(MyNewComponent:Port2, **system**:PCO1); // unmap previously made mapping

EXAMPLE 2: Disconnect/unmap for a component

**disconnect**(MyNewComponent:Port1); // disconnects all connections of Port1, which  
 // is owned by component MyNewComponent.

**unmap**(MyNewComponent:**all port**); // unmaps all ports of component MyNewComponent

EXAMPLE 3: Disconnect/unmap for "self"

**disconnect**; // is a shorthand form for …

**disconnect**(**self**:**all** **port**); // which disconnects all ports of the component

// that called the operation

:

**unmap**; // is a shorthand form for …

**unmap**(**self**:**all** **port**); // which unmaps all ports of the component

// that called the operation

EXAMPLE 4: Disconnect/unmap for "all component"

**disconnect**(**all component**:**all** **port**); // the MTC disconnects all ports of all

// components in the test configuration.

:

**unmap**(**all component**:**all** **port**); // the MTC unmaps all ports of all

// components in the test configuration.