Prepared (also subject responsible if other)
ETH/RUST Csaba Koppany +36 1 437 7930

No.
ETH/R-04:000010 Uen

Approved | Checked
ETH/RUS [György Réthy]ETH/RUS | 2014-10-09 | A

TTCN-3 Naming convention

Contents

Prepared (also subject responsible if other)
ETH/RUST Csaba Koppany +36 1 437 7930
No.
ETH/R-04:000010 Uen
Approved | Checked
ETH/RUS [György Réthy]ETH/RUS
2014-10-09 A

1 INTRODUCTION

1.1 Application

This document is applicable to all TTCN-3 development projects. In special cases projects can deviate from this convention only if they do not want to use already developed TTCN-3 code or parts of code, and does not want to make their code available for reuse.

When a project is using ASN.1 modules from the standard this naming convention is not applicable for those ASN.1 modules. This naming convention should be followed only if the ASN.1 module from the standard need to be modified, and only applicable for that modifications.

1.2 Revision Information

Date	Rev	Characteristics	Prepared
2003-12-11	PA1	Draft version, the style, and the examples need to be checked	ETHCKY
2003-12-16	PA2	Editorial correction (styles etc.)	ETHCKY
2003-12-18	PA3	*ST postfix and par_ prefix changed to SCT and pl_, meaning of vl_, vc_*, vd_ prefixes changed, new prefixes vlc_, vld_ introduced, prefixes vv_, ts_ and asd_ deleted. Several editorial and minor technical changes	ETHGRY
2004-01-06	PA4	Adding signatures (new clause 6.3 and additions to clause 10); editorial changes	ETHGRY
2004-01-06	PA5	Adding new clause 3 (shifting main clause numbers accordingly); editorial changes	ETHGRY

1.3 Purpose of the Document

This document contains the naming convention of TTCN-3 test suite writing. Following this naming convention makes the TTCN-3 code more readable and reusable and code development more efficient in projects.

2 References

[1] ETH/R-04:000011 Uen Resolving Naming Conflicts When Using TTCN-3

3 Design Philosophy

Vital targets of a good naming convention should be at least:

a) Increasing coding efficiency
by allowing mechanically forming names used in the code, without the need of thinking too much what should be the good name in the given situation.

b) Increasing the quality of the code
by allowing a basic level of static checking of the code; in addition, the test suite writer is thinking about the behaviour of the code instead of thinking on the name.

c) Increasing efficiency of code modification and maintenance
unique naming convention makes easier for other test suite writers to understand the code quickly.

To fulfill the above targets the present document applies the following design rules:

1) Each name should identify the kind, scope and place of definition of the item it identifies; in most cases this is done by using prefixes.

2) Prefixes shall be unique, clear (identify the kind self-evidently) and minimal length to allow efficient writing.

3) Length of prefixes and postfixes (and also names) should relate to the frequency of use of the given language element; e.g. variable, template etc. prefixes should be short but port type names may be longer as these language elements are not written many times.

The most evident example for the above is the prefixing scheme for constants, variables, timers and formal parameters. There are three scopes in which these language elements can be defined:
- global definitions (constants only)
- component scope
- local scope (testcases, functions, altsteps, block of statements and the module control part)

The first letter of the prefix identifies the kind of the TTCN-3 language element (“c” for constants, “v” for variables, “T” for timers and “p” for formal parameters). The second letter identifies the scope (and hence the place of definition) of the item (missing second letter – component scope, “g” for the global scope and “l” for the local scope). For some specific cases a third letter of the prefix identifies a specific use (“c” for component references, “d” for default references). Summary of different prefixes is given in Table 2 (see clause 15).

4 Notational Conventions

This convention is in line with ES 201 873-1 v2.2.1 and ETR 141; in particular, templates will reference the name of the type declaration they are based on when applicable, and will be identified additionally by a prefix and an optional explanatory part. To be in line with ASN.1 naming rules, all prefixes to type names are proposed to write in uppercase letters and all other prefixes are proposed to be written in lowercase letters.

Following general rules apply:

1 Parts of the names written between <…> and in *italic* font shall be substituted by the relevant name of the *protocol*, identifier, chosen name etc.

2 Parts of names in square brackets are optional.

3 Items within brackets separated by a vertical line ("|") identify alternative fragments of the name; the alternative to be used shall be chosen depending on the context the name is used for.

4 Except underscores given in the naming convention, text of the name (e.g. <*descriptive name*>) may be either of the `"ThisIsTheNameOfAType"` style or the `"This_is_the_name_of_a_type"` style. When choosing names and one of the styles the following shall be kept in mind:
- preserve readability as much as possible
- short names are more effective to write but they may became too cryptic hence not readable
- long names are more descriptive but they may became too verbose hence not effective
- use names which allows more effective writing (e.g. by not using uppercase letters when possible)
- keep one unique style throughout the test suite (bearing in mind, that in ASN.1 modules the use of names without hyphens is quite widespread).

5 The optional <*PROTOCOL*> part shall be included in the name when the object is closely related to the *PROTOCOL* (e.g. PICS, some PIXIT parameters). It is necessary to be unambiguous or improves comprehension significantly (e.g. no need to think about *PROTOCOL* stacks on all used interfaces during reading). The protocol names shall be written in uppercase letters.

6 The <*type reference*> shall reference the relevant type declaration by a reasonably shortened name (type identifiers used in ASN.1 are tends to be long).

7 Generally, names should be kept reasonably short (e.g. the SAP type should not be included into the name of a PCO declaration if only a single PCO exists on the given layers boundary).

5 Module Identifiers

In general TTCN-3 and ASN.1 module identifiers shall be used as the filename of the given module as well.

5.1 Identifiers of TTCN Modules

[<*TestObject*>_][<*PROTOCOL*>_]<*descriptive name*>[<*objid*>]

where <objid> is an object identifier from the node:
`{ itu_t(0) identified_organization (4) etsi(0) reserved(127) etsi_identified-organization(0) ericsson(5) testing (0) …}`

The following categories are identified as part of the <descriptive name>:

- _Types : Modules containing type definitions
- _Templates : Modules containing template definitions
- _Functions : Modules containing functions and altsteps
- _Testcases : Modules containing testcases
- _Config : Modules containing configuration related parameters, types (system component type, parameter types, etc.) and configuration functions
- _PortType : Modules containing port type definition(s)
- _ComponentType: Modules Component type definition(s)
- _Parameters: Modules containing module parameter(s)
- _Mapping: Modules containing all definitions (port type(s), mapping component type, declarations of external EncDec functions, mapping component behaviour function and functions and altsteps called by the behaviour function directly or indirectly) used only by a mapping component
- _Signatures: Modules containing procedure signature definition(s).

If the module fit into these categories, these names shall be used as the descriptive name or last part of the descriptive name.

5.2 Identifiers of ASN.1 Modules

In TTCN-3 the ASN.1 modules of the *protocol* standard is directly usable. In this case the module name and object identifier used in the *protocol* specification shall be used.

However for testing often modifications and additions to the *protocol* ASN.1 module is necessary. In this case the new or changed ASN.1 module shall have its own name and object identifier:

<*PROTOCOL*>-(PDU|Type|Constant)-Defs<*{objid value}*>

where <objid> is an object identifier from the node:
`{ itu-t identified-organization etsi(0) reserved(127) etsi-identified-organization(0) ericsson(5) testing (0) …}`

Allocated ASN.1 module object ids are given at http://ttcn.ericsson.se/standardization/EriNodeInETSI_Assigned_OIDs.pdf under menu point "**Ericsson node in ETSI**". Also here a new node can be obtained. Object identifier allocation below the obtained node is the responsibility of the new node owner.

5.3 Referencing Names at Import

Modular test suites requires to import definitions made in one module to be imported into (an)other module(s). This is true for both TTCN and ASN.1 modules.

TTCN-3 does not allow using the ASN.1 syntax inside TTCN modules. Therefore all ASN.1 declarations used shall be defined in ASN.1 modules and imported into TTCN. Type definitions imported from ASN.1 shall have the same name as in the ASN.1 module but all occurrences of a dash ("-") are changed to an underscore ("_").

Note 1: *ASN.1 allow name clashes but complete sets of ASN.1 modules (e.g. related to a single protocol) normally free of name conflicts. Hence all definitions can be imported from ASN.1 modules by using the TTCN all keyword. In this case the ASN.1 module name shall be referenced in the TTCN-3 import statement by changing dashes to underscore. For all other ASN.1 names it will be done by the tool automatically. When individual ASN.1 definitions are imported or excluded from import, ASN.1 names with dash changed to underscore shall be used to reference the given item.*

Note 2: *This rule applies to types and values imported. Imported ASN.1 values can be used as global constants within TTCN-3.*

The module name in the import statement should contain the object identifier value of the module. This ensures that always the correct modules are used with the given TTCN-3 test suite.

Note 3: *The Titan test tool supports the import all definitions TTCN-3 option only and not import individual definitions or group of definitions.*

Example:

```
import from L3_Types all;

import from QAAL2_PDUDef.objid { 0 4 0 127 0 5 0 5 0 5 3 1 0 }      all;
                                                    // note, this statement imports all PDU definitions
```

6 Module Parameter Declarations

tsp[(c|x)]_[<*PROTOCOL*>_]<*descriptive name*>

Test suites may be parameterized by run-time constant values. These parameters shall be declared as modulepar definitions in different TTCN-3 modules. However, modulepar are global definitions, can be imported into any other module of a test suite, hence their effect may be wider than just the module in which defined. For that reason and also for compatibility with TTCN-2 test suite parameters' prefix, the tsp_ prefix is selected. The use of the "c" or "x" modifiers are optional.

Note: When used, "c" or "x" identifies if the given parameter is a PICS or a PIXIT item respectively; "tsp" alone identifies test suite parameters, which are nor PICS neither PIXIT items (or can not be categorized).

Example:

tsp_RRC_IntProtUsed // *test configuration* specific parameter in RNC
// or UE testing;

tspc_RRC_T300 // *protocol*-specific PICS value when testing RNC;

tspx_IMSI // generic parameter, not specific for one *protocol*;
// note, that PICS/PIXIT documents are *protocol*-specific, at run-time a generic
// PIXIT value will contain the actual PIXIT value drawn from the filled-up
// document for the *protocol* being tested

7 Type And Signature Definitions

TTCN-3 does not make a functional distinction between types (e.g. ASPs, PDUs, CMs, structured types etc.). This functional distinction has to be done by the test suite writer by using an appropriate naming scheme (and by listing types which can be sent/received at port type declarations).

7.1 ASN.1 Type Definitions

TTCN-3 does not allow to use the ASN.1 syntax inside TTCN modules. Therefore all ASN.1 declarations shall be defined in ASN.1 modules and imported into TTCN. Rules on names at import are given in clause 5.3.

When a TTCN-3 keyword is used within an ASN.1 module, it shall be changed in ASN.1 according to the rules defined in [1] (only field names and value definitions can conflict with TTCN-3 keywords).

When naming conflict(s) occur within an ASN.1 module or between definitions of different ASN.1 modules, it (they) shall be resolved according to rules in [1] before importing ASN.1 definitions into TTCN-3 module(s).

7.2 TTCN Type Definitions

The TTCN-3 language does not make a functional distinction between types used for different purposes. Such distinction may not be appropriate or possible in some cases (e.g. in some IP testing scenarios) but its use is anticipated for names of types related to a protocol.

If names of protocol-related type definitions are wished to be classified based on their use, one of the naming schemes in clauses 7.2.2 to 7.2.5 shall be applied.

Note: The distinction between abstract service primitives (ASPs), protocol data units (PDUs) and co-ordination messages (CMs) is the way of using the relevant type definitions in the test scenario. ASPs are internal signals of the test system and used for inter-layer communication, PDUs are the messages sent from one protocol entity to its peer within the tested system and CMs are messages sent between TTCN test components.

7.2.1 Subtype and Structured Type Definitions

<Descriptive name>

To keep a consistent style, it is anticipated that names of TTCN-3 types start with a uppercase letter. To be consistent with ASN.1 language rules, names of TTCN-3 types shall start with a uppercase letter in TTCN-3 module(s) which import ASN.1 definition(s) and recommended to keep this rule all over the test suite.

Example:

MSCparameters // type definition not related to a given protocol

7.2.2 PDU Type Definitions

Protocol data units are messages sent to the peer entity of the protocol(s) actually being tested.

PDU_[<*PROTOCOL*>_]<*descriptive name*>

Note1: In many of cases PDU type definitions are imported from ASN.1 modules and therefore the name imported from ASN.1 shall be used within TTCN-3. Names in standard ASN.1 modules shall NOT be changed. See also clause 5.3.

Note2: Examples show TTCN-3 names, in case of ASN.1 names a dash shall be used where an underscore is shown in this clause.

Example:

PDU_QAAL2_BLC_BlockConfirm

7.2.3 ASP Type Definitions

Abstract service primitives are means of the internal communication inside a system. Primitives of the underlying layer (service provider) carry the PDUs of the tested protocol and/or control the underlying transport connectivity.

ASP_[<*service provider*>_]<*descriptive name*>

Note1: *in some cases ASP type definitions are imported from ASN.1 modules and therefore the name imported from ASN.1 shall be used within TTCN-3. Names in standard ASN.1 modules shall NOT be changed. See also clause 5.3.*

Note2: *Examples show TTCN-3 names, in case of ASN.1 names a dash shall be used where an underscore is shown in this clause.*

Example:
ASP_RLC_AMDataReq

7.2.4 CM Type Definitions

Coordination messages are means of internal communication between test components.

CM_[<*component*>[_<*component*>]]_<*descriptive name*>

Note: *Examples show TTCN-3 names, in case of ASN.1 names a dash shall be used where an underscore is shown in this clause.*

Example:
CM_mtc_Initial_UE_message_control

7.2.5 Information Elements, Parameters, Fields

[<*PROTOCOL*>_](IE|PARAM|FIELD)_<*Descriptive name*>

The name of the *protocol* shall be given for types used as building blocks of messages. It is useful to distinguish between information elements, parameters etc.

Note1: *In many cases IE, parameter, field etc. type definitions are imported from ASN.1 modules and therefore the name imported from ASN.1 shall be used within TTCN-3. Names in standard ASN.1 modules shall NOT be changed. See also clause 5.3.*

Note2: *Examples show TTCN-3 names, in case of ASN.1 names a dash shall be used where an underscore is shown in this clause.*

Example:
qaal2_param_Cause

7.2.6 Component Type Definitions

[<*simulated object*>_]
(<*PROTOCOL*>|<*function*>|(<PROTOCOL>|MTC)_<function>|MTC)
_CT

The postfix "_CT" shall be used for component type definitions instantiated as the MTC and/or as the PTC(s), and also for components types instantiated both in the role of test component(s) and the test system interface. Component type definitions instantiated only as the MTC shall not use the <simulated object> part and shall use "MTC" in the descriptive part of the name.

Example:

```
MTC_CT                  //component type used for the MTC, when no other
                        //component type is used as MTC in the test suite
                        //and the MTC behaviour does not relate to any
                        //node or protocol (i.e. test case control and
                        //supervision only)

MTC_4IMSCHO_CT          //component type used for the MTC when testing
                        //the inter-MSC handover functionality

RNC_CN_CT               // component type definition for
                        //the component emulating CN in the
                        // RNC test suite

RANAP_CT                //component type used to carry out the RANAP
                        protocol behaviour

RANAP_mapping_CT        //component type used for the mapping component
                        //below the RANAP behaviour component
```

7.2.7 Test System Interface Component Type Definitions

(*<Test Object>*|*<Test Object>*_*<configuration>*)_SCT

The postfix "SCT" shall be used for component types which are used exclusively in the role of the test system interface.

Example:

```
SGSN_SCT                // component type definition used for the TSI in the
                        // SCSN test suite.

RNC_cfg3_SCT            // component type definition used for the TSI in the
                        // test suite, and using 'cfg3' configuration in the RNC
                        //test suite

MSC_4IMSCHO_CT          //component type used for the TSI and for other test
                        // component(s) when testing the inter-MSC handover
                        // functionality in the MSC test suite
```

7.2.8 Port Type Definitions

TTCN-3 does not make a functional distinction of ports used for test co-ordination and ports used to reach the IUT/SUT. Such functional distinction – when necessary - may be done by the test suite writer by using an appropriate naming scheme.

Note: The distinction between PCOs and CPs only depends on the use of the given port in the test scenario. When the same port

type definition can be used for PCO ports and CP ports, no distinction is necessary in their names.

<*service provider*>[_<*SAP type*>]asp[_<*PROTOCOL*>][_SP]_PT
or
<*PROTOCOL*>*msg*[_SP]_PT

This name form is usable for port types through which the IUT/SUT or other (non-TTCN-3) elements of the test environment is accessed (e.g. SW or HW test equipment in target test environment) or between test components when carrying protocol data like ASPs, PDUs, datagrams etc. (functionally equivalent to TTCN-2 PCOs). The ”asp” name form shall be used when transport protocol ASPs are passed via port instances of the given type (i.e. between entities of different protocols) and the ”msg” option shall be used when messages of the tested protocol are passed through (i.e. entities at both end of the connection handle the same protocol).

For symmetrical ports (when the in and out lists of the port type definition are identical) the “_SP” postfix shall not be used. For asymmetrical ports the “_SP_PT” postfix shall be used for port types at the service provider (lower) side and the “_PT” postfix shall be used for port types at the service user (upper) side of the port connection.

<*component*>[_<*component*>]_PT

This name form is usable for port types used to send/receive coordination messages between test components (functionally equivalent to TTCN-2 CPs).

Example:

```
RLC_AMasp_RRC_PT      // PCO for an RRC test component

MTC_PT                // Coordination point between the MTC and any other
                      // PTC (may be point-to-multipoint)

MTC_BOIP_PT           // Coordination point between the MTC and
                      // the BOIP PTC

SCCPasp_RANAP_PT      // A port type used in the SCCP user component
                      // (RANAP PTC) and through which SCCP ASPs
                      // carrying RANAP PDUs are passed (“upper” end
                      // of the port connection)

SCCPasp_RANAP_SP_PT   // A port type used in the SCCP service provider (SP)
                      // component and through which SCCP ASPs carrying
                      // RANAP PDUs are passed(“lower” end of the port
                      // connection)

TCPmsg_PT             // port type via through TCP messages are passed
                      // (e.g. from a TCP PTC to a TCP test port)
```

7.3 TTCN-3 Signature Definitions

S_[<API>|<*interface*>_]<*descriptive name*>

Signatures are syntactically similar to external functions (contain the remote procedure prototype) while semantically are similar to type definitions (used for template definitions).

Example:

```
S_RPC_RestartCP (charstring pl_cpName);
                                    //Signature to be used on the RPC interface

S_RPC_waitForStart ( charstring pl_cpName );
                                    //Signature to be used on the RPC interface
```

8 Constant Definitions

Note: Normally protocol related constants should be declared in the module containing PDU and type definitions (and just be imported in cases when declared in ASN.1 modules). For non-protocol-specific constants or if not this solution is chosen, it is more efficient to declare constants in test components. These constants are seen by all test cases,functions and altsteps with an appropriate "runs on" clause.

8.1 Constant Definitions in the Declarations Part

cg_[<*PROTOCOL*>_]<*descriptive name*>

Global constants defined in the definitions part of a TTCN-3 module.

Note: functionally identical to TTCN-2 test suite constants.

Example:

```
cg_GMM_AttachCompleteID     //a constant containing a given message type ID
```

8.2 Constant Declarations in Component Types

c_[(<*PROTOCOL*>|<*component*>)_]<*descriptive name*>

Constants declared within a component type definition.

The *protocol* name shall only be used for constants of *protocol* related data (i.e. received call reference values etc.), the component name shall only be used for non-*protocol* related constants utilised in a specific test component.

Example:

```
c_LAPm_N201                 // constant used to store a L2 protocol value;
```

8.3 Other Local Constant Declarations

Constants declared in testcases , functions, altsteps, block of statements or in the control part:

cl_[<*PROTOCOL*>_]<*descriptive name*>

9 Variable Declarations

Note: To save time and effort protocol related variables should be declared in the test component type definition. Names of these variables are visible in all testcases, functions and altsteps referring to the given component type in their "runs on" clauses and same instances of such variables will be used by testcases, functions and altsteps called on the same component instance. On the contrary, e.g. test case variables are not seen by test steps called by the test case unless passed as a parameter.

9.1 Component Variables

v_[(<*PROTOCOL*>|<*component*>)_]<*descriptive name*>

The *protocol* name shall be used for variables to store a *protocol* related data (i.e. received call reference values etc.), the component name shall be used for non-*protocol* related variables utilised in a specific test component.

Example:

```
v_flag                  // variable used to store non-protocol specific data;
v_L3_callReference      // variable used to store a L3 call reference value;
```

9.1.1 Component References

vc_(<*simulated node*>|<*PROTOCOL*>|<*simulated node*>_<*PROTOCOL*>)[_<*descriptive name*>]

Variables defined within a component type and used to store component instance references (returned by a **create** operation). The type of these variables always shall be one of the user defined component types.

Example:

```
vc_UE_with_IntProt      //variable which will be used to refer to an UE
                        //component instance providing integrity protection

vc_DRNC_RNSAP           //variable which will be used to refer to the RNSAP
                        //component instance from the set of components
                        //jointly providing a DRNC emulation

vc_MGW1                 //variable which will be used to refer to a component
                        // instance emulating an MGW
```

9.1.2 Default References

vd_[<*PROTOCOL*>_]<*descriptive name*>[_(MTC|PTC)]

Variables defined within a component type and used to store references to activated defaults (returned by an **activate** statement). The type of these variables always shall be **default**.

Example:

vd_RANAP_PTC // default reference used in RANAP PTCs

9.2 Other Local Variables

[vl_][<*PROTOCOL*>_]<*descriptive name*>

Variables declared in testcases, functions, altsteps, block of statements or in the control part. The prefix may be omitted for non-protocol related variables (like loop counters, for loop control variables, variables used in calculations etc.)

Example:

vl_endFlag // test case variable to control test
// behaviour

count, i, j //variables used to control e.g. for loops

flag //variable used to control e.g. while/do while loops

NOTE: Control part variables used to store test case verdicts are not distinguished in their prefixes but naturally the descriptive part of the name may identify such use of a variable.

9.2.1 Component References

**vlc_(<*simulated node*>|<*PROTOCOL*>|
<*simulated node*>_<*PROTOCOL*>)[_<*descriptive name*>]**

Variables defined in testcases or functions and used to store component instance references (returned by a **create** operation). The type of these variables always shall be one of the user defined component types.

Example:

vlc_UE //variable which will be used to refer to an UE
//component instance providing integrity protection

9.2.2 Default References

vld_[<*PROTOCOL*>_]<*descriptive name*>[_(MTC|PTC)]

Variables defined in testcases or functions and used to store references to activated defaults (returned by an **activate** statement). The type of these variables always shall be **default**.

Example:

vld_RANAP // default reference used for RANAP PTCs

10 Timer Declarations

10.1 Component Timers

T_[(<*PROTOCOL*>)_]<*descriptive name*>

Timers declared within component type definitions. The *protocol* name shall be used for protocol-related timers.

For short descriptive names it is recommended to use ALL UPPERCASE letters. This makes timer operations well visible within the code of the dynamic behaviour.

Example:

```
T_WAIT          // timer to wait the operator response; may be declared
                // in different test components simultaneously,
                // what aids code transportability

T_ GUARD        // an guard timer declared in the MTC component
                // type declaration

T_RRC_301       // an RRC protocol timer
```

10.2 Local Timer Declarations

Tl_[<*PROTOCOL*>_]<*descriptive name*>

Timers declared in test cases, functions, altsteps, block of statements or in the control part.

For short descriptive names it is recommended to use ALL UPPERCASE letters. This makes timer operations well visible within the code of the dynamic behaviour.

11 Template definitions

TTCN-3 does not make a functional distinction between templates used for different purposes (e.g. ASP-, PDU-, CM-, simple/structured templates or signature templates etc.). This functional distinction has to be done by the test suite writer by using an appropriate naming scheme. Such distinction may not be appropriate or possible in some cases (e.g. in some IP testing scenarios) but its use is anticipated in all cases when applicable.

11.1 Data Templates

t[r]_[PDU|ASP|CM]_[<*PROTOCOL*>_]<*type reference*> [_<*explanative part*>]

For fully defined or modified template definitions used in message-based communication (based on a type definition). The PDU, ASP and CM modifiers may be used when appropriate. They can make sending and receiving operations more talkative.

NOTE: No distinction is made between templates used in message-based and procedure based communications in their prefixes as they are not used in ambiguous situations. When they nevertheless wished to be differentiated, the explanative part can be used for that.

The "r" modifier shall be used for receiving templates, which directly contain matching symbol(s) or matching mechanism(s) in any of its fields (like "?", "*", a value range, a list of values etc.) and hence can ONLY be used in receiving operations.

Note: A template type formal parameter alone does not force the use of the "r" modifier.

Example:

```
template RRC States t_RRC_States (template Id pl_Id) := {…}
                                        //template not containing any matching symbol

template GMM_Attach t_PDU_GMM_Attach_type := {…}
                                        // template usable for sending (no matching symbols
                                        inside)

template MAP_ParametersType tr_PDU_MAP_prepareHandover_toGSM
     ( …) :=
  {
  …
      ho_NumberNotRequired := *,
  …
  }                                     //template usable in receiving (receive, trigger)
                                        //operations only
t_PDU_RRC_RBSetup                       //An RRC PDU template  without any
                                        // matching symbols

tr_ASP_RLC_AMDataReq_DCCH               // RLC ASP template usable for receiving
                                        //operations only (containing matching symbol(s)

t_CM_Token                              //a co-ordination message template
```

11.2 Signature Templates

t[r]_[<API>|<Interface>_]<*signature reference*>
[_<*explanative part*>]

For fully defined or modified template declarations used in procedure-based communication (based on a signature definition).

The use of the "r" modifier is given in clause 11.1.

Example:

```
template S_RPC_RestartCP  t__RPC_RestartCP  := { cpName := "CP1"}
                                        //template not containing any matching symbol

template S_RPC_waitForStart  tr_RPC_waitForStart  := { cpName := ?}
                                        //template containing a matching symbol
```

12 Formal parameters

pl_[<*PROTOCOL*>_]<*parameter name*>

Templates, testcases, functions and altsteps may have formal parameters. From the naming perspective (but only from that) formal parameters can be considered as local definitions (e.g. establishing names with the same visibility as testcase, function or altsteps local variable, constant and timer declarations).

For the "*parameter name*" part it is anticipated to use the name of the parameter's type for testcase, function and altstep parameters and to use the name of the type or the name of the template field (in which it is used) for formal parameters of templates. The <Protocol> part of the name shall be used when the testcase, function or alstep handles multiple protocols, therefore the specific protocol can not be identified just from the context.

Example:

```
template  MAP_ParametersType  tr_PDU_MAP_prepareHandover_toGSM
     ( template CellId pl_targetCellId,
       template RNCId pl_targetRNCId
     ) :=
  {
  …
       targetCellId  :=  pl_targetCellId,
  …
  }                                  //pl_targetCellId and pl_targetRNCId are
                                     //formal parameters of the template
                                     //tr_PDU_MAP_prepareHandover_toGSM
function f_sendRANAPMessage( RANAP_Cause pl_Cause) {…}
                                     //pl_Cause is a formal parameter of the function
                                     //f_sendMessage; the protocol is not identified in the
                                     //name of the parameter because the function in which
                                     //it is used relates to the RANAP protocol unanimously.
```

13 Test Port Instance Names

The TTCN-3 language does not make a functional distinction between ports used for different purposes (as PCOs or CPs). When such functional distinction is necessary the test suite writer shall establish it by using naming schemes in clauses 1913.1.1 and 13.1.2.

13.1 General Case

Test port instances do not have any prefix. It is recommended to use ALL UPPERCASE letter names for test port instances. Except distinguishing from other names this convention makes port operations well visible within the code of the dynamic behaviour.

In the general case it is not required to identify the role of the port (PCO, CP etc.).

13.1.1 PCO Declarations

[<*simulated node*>_]<*service provider*>
[_<*SAP type*>][_<*PROTOCOL*>]_PCO

When test port instances wished to be identified as in the "point of control and observations" role (when such distinction of the port's role is applicable). PCOs may be connected to other test components PCO(s) or may be mapped to a TSI port.

Note: *the <PROTOCOL> part of the name shall be used to avoid ambiguity e.g. in test configurations, when more than one protocol uses the services of the same lower service provider layer*

Example:

```
MTP3_PCO          // for an MTP3 port; as port instances are always bound
                  // to test components, usually no need to identify the
                  // simulated node

RNC_RLC_AM_PCO    // for UE testing, when the tester simulates the RNC)
```

13.1.2 Coordination Point Declarations

<*target PTC/MTC*>_CP

When test port instances wished to be identified as connecting two TTCN-3 test components and in the role of an co-ordination point (when such distinction of the port's role is applicable).

Example:

```
BOIP_CP           // for a CP used the MTC and facing the BOIP PTC
                  // only; note, that port instances are always bound to
                  // test components (no need to identify the "home"
                  // component)
```

14 Dynamic part

14.1 Groups

<*GroupName*>

The name of a group should start with a Uppercase letter.

Example:

RNC_RRC_ConnectionManagement_RRCConnectionEstablishment

BothWayCircuitSelection

UCN_RANAP_RABAssignment_RABSetup

14.2 Test Cases

tc_[<IUT>_][<PROTOCOL>_]<group>[_<i>]_
(<descriptiveName>|<sequenceNumber>)
or
tc_<*TC_number*>[_<*IUT*>][_<*PROTOCOL*>]_<*testCaseName*>

Test case names may use two naming schemes. According to the first one the name shall reference the test group directly and have a descriptive name of the behaviour or a number differentiating the test case from other test cases within the same group.

According to the second scheme the test group is referenced in the TC name indirectly, via the test case number as the first element of the name.

It is proposed not to use underscore in the descriptive name part to allow to separate the two parts of the name easily.

Example:

tc_RNC_RRC_CM_CE_ConnRequestWithCauseOrigSpeechCall
// A test case name using a direct group reference and a
// descriptive name

tc_CN_RANAP_RA_EH_01 // A test case name using a direct group reference and
// sequence numbering within the group

tc_1_1_RNC_RRC_ConnectionRequestWithCauseOrigSpeechCall

tc_2_1_1_IAMsentByControllingSP
// Test case names using indirect group reference in the
// test case number part of the name

14.3 Functions

f_[<*PROTOCOL*>_][<*test component*>_]<*descriptive name*>
[_<*number*>]

TTCN-3 does not make a functional distinction between functions used for test steps or other operations. This has to be done by the test suite writer by using an appropriate descriptive name. Test steps (fragments of the component behaviour specified as separate functions or altsteps) may be defined as functions with a runs on clause (normally without a return value).

The <test component> part of the name allows identifying the component type in the “runs on” clause of non-protocol related test steps (e.g. a configuration supervision executed on the MTC). Also permits the distinction of test steps participating in the same phase of the test behaviour but executed on different component types (e.g. connection setup between RBSs via an MGW).

Example:

f_RRC_ CELL_PCH // Test step in the RRC test steps group

f_RRC_RBRelease // RRC test step

```
f_MTC_SetupTestConfig1      //test step called in the MTC (only) and executing
                            // dynamic test configuration.
```

14.4 Altsteps

as_[<*PROTOCOL*>_][<*descriptive name*>]

Altsteps called directly and/or activated as default. TTCN-3 does not make a functional distinction between altsteps called directly in alt statements or activated as defaults. This functional distinction – when necessary - has to be done by the test suite writer by using an appropriate descriptive part of the name. Sometimes the same altsteps may be used directly or activated as defaults depending on the test case or the test configuration.

Note: Though no distinction is made by the prefix, it is anticipated that the descriptive part of names of altsteps known to be used as defaults at the time of their definition, contain a "def" or "Def" marking.

Example:

```
as_RANAP_iuReleaseDifferentCauses ( RANAP_Cause  pl_Cause )
                            // altstep to receive an Iu Release message with diffe-
                            // rent possible specific cause values

as_RANAP_Def_timeouts       //altsteps activated as default in the RANAP
                            //component
```

15 Summary

Table 1: Lookup table of naming convention syntax

Language element	Options	Name convention syntax	Clause
TTCN-3 module		[<TestObject>_][<PROTOCOL>_]<descriptive *name>[<objid>]*	5.1
ASN.1 module		<PROTOCOL>-(PDU|Type|Constant)-Defs<{objid value}>	5.2
module parameter		tsp[(c|x)]_[<PROTOCOL>_]<descriptive *name*>	6
ASN.1 type		name unchanged or naming conflict(s) resolved according to [1]	7.1
TTCN-3 type		<Descriptive *name*> Note: first letter is uppercase if ASN.1 also used	7.2.1
	PDU type	PDU_[<PROTOCOL>_]<descriptive *name*>	7.2.2
	ASP type	ASP_[<service *provider*>_]*<descriptive name>*	7.2.3
	CM type	CM_[<component>[_<component>]]_<descriptive *name*>	7.2.4
	IE, parame-ter or field	[<PROTOCOL>_](IE|PARAM|FIELD)_<Descriptive *name*>	7.2.5
component type		[<simulated object>_] (<PROTOCOL>|<function>|(<PROTOCOL>|MTC)_<function>|MTC) _CT	7.2.6
test system interface component type		(<Test *Object*>|*<Test Object>_<configuration>)_SCT*	7.2.7
port type		<service *provider>[_<SAP type>]asp[_<PROTOCOL>][_SP]_PT* or *<PROTOCOL>msg[_SP]_PT*	7.2.8
signature		S_[<API>|<interface>_]<descriptive *name*>	7.3
constant, global scope		cg_[<PROTOCOL>_]<descriptive *name*>	8.1

constant, component scope		c_[(<PROTOCOL>|<component>)_]<descriptive *name*>	8.2
constant, local scope		cl_[<PROTOCOL>_]<descriptive *name*>	8.3
variable, component scope		v_[(<PROTOCOL>|<component>)_]<descriptive *name*>	9.1
variable, component scope, storing component reference		vc_(<simulated *node*>|<*PROTOCOL*>| <simulated node>_<PROTOCOL>)[_<descriptive *name>]*	9.1.1
variable, component scope, storing default reference		vd_[<PROTOCOL>_]<descriptive *name>[_(MTC|PTC)]*	9.1.2
variable, local scope		[vl_][<PROTOCOL>_]<descriptive *name*>	9.2
variable, local scope, storing component ref.		vlc_(<simulated *node*>|<*PROTOCOL*>| <simulated node>_<PROTOCOL>)[_<descriptive *name>]*	9.2.1
variable, local scope, storing default ref.		vld_[<PROTOCOL>_]<descriptive *name>[_(MTC|PTC)]*	9.1.2
timer, component scope		T_[(<PROTOCOL>)_]<descriptive *name*>	10.1
timer, local scope		Tl_[<PROTOCOL>_]<descriptive *name*>	10.2
template, data		t[r]_[PDU|ASP|CM]_[<PROTOCOL>_]<type reference> [_<explanative part>]	11.1
template, signature		t[r]_[<API>|<Interface>_]<signature reference> [_<explanative part>]	11.2
formal parameters		pl_[<PROTOCOL>_]<parameter name>	12
test port instance		All uppercase letters	13.1
	PCO	[<simulated *node*>_]<*service provider*> [_<SAP type>][_<PROTOCOL>]_PCO	13.1.1
	CP	<target PTC/MTC>_CP	13.1.2

group		<GroupName>	14.1
testcase		tc_[<IUT>_][<PROTOCOL>_]<group>[_<i>]_ (<descriptiveName>|<sequenceNumber>) or tc_<TC_number>[_<IUT>][_<PROTOCOL>]_<testCaseName>	14.2
function		f_[<PROTOCOL>_][<test component>_]<descriptive name> [_<number>]	14.3
altstep		as_[<PROTOCOL>_][<descriptive name>]	14.4

Table 2: Constant, variable, timer and formal parameter prefixes

		Scope	
	global	*component*	*local*
constant	cg_	c_	cl
variable		v_	vl_
variable storing component ref.		vc_	vlc_
variable storing default reference		vd_	vld_
timer		T_	Tl_
formal parameter			pl_