UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.292

(09/98)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE X: RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

Interconnexion des systèmes ouverts – Tests de conformité

Cadre et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Notation combinée arborescente et tabulaire (TTCN)

Recommandation UIT-T X.292

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

RÉSEAUX PUBLICS POUR DONNÉES	
Services et fonctionnalités	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalisation et commutation	X.50–X.89
Aspects réseau	X.90–X.149
Maintenance	X.150–X.179
Dispositions administratives	X.180–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	
Modèle et notation	X.200–X.209
Définitions des services	X.210–X.219
Spécifications des protocoles en mode connexion	X.220–X.229
Spécifications des protocoles en mode sans connexion	X.230–X.239
Formulaires PICS	X.240–X.259
Identification des protocoles	X.260–X.269
Protocoles de sécurité	X.270–X.279
Objets gérés des couches	X.280–X.289
Tests de conformité	**X.290–X.299**
INTERFONCTIONNEMENT DES RÉSEAUX	
Généralités	X.300–X.349
Systèmes de transmission de données par satellite	X.350–X.399
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	
Réseautage	X.600–X.629
Efficacité	X.630–X.639
Qualité de service	X.640–X.649
Dénomination, adressage et enregistrement	X.650–X.679
Notation de syntaxe abstraite numéro un (ASN.1)	X.680–X.699
GESTION OSI	
Cadre général et architecture de la gestion-systèmes	X.700–X.709
Service et protocole de communication de gestion	X.710–X.719
Structure de l'information de gestion	X.720–X.729
Fonctions de gestion et fonctions ODMA	X.730–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850–X.859
Traitement transactionnel	X.860–X.879
Opérations distantes	X.880–X.899
TRAITEMENT RÉPARTI OUVERT	X.900–X.999

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

RECOMMANDATION UIT-T X.292

CADRE ET MÉTHODOLOGIE DES TESTS DE CONFORMITÉ OSI POUR LES RECOMMANDATIONS SUR LES PROTOCOLES POUR LES APPLICATIONS DE L'UIT-T – NOTATION COMBINÉE ARBORESCENTE ET TABULAIRE (TTCN)

Résumé

La présente Recommandation définit une notation informelle de tests, appelée notation combinée arborescente et tabulaire (TTCN, *tree and tabular combined notation*), destinée à être utilisée pour spécifier des suites de tests de conformité abstraites OSI. Cette notation est indépendante des méthodes de test, des couches et des protocoles et reflète la méthodologie de test abstraite définie dans les Recommandations X.290 et X.291.

Source

La Recommandation UIT-T X.292, révisée par la Commission d'études 7 de l'UIT-T (1997-2000), a été approuvée le 25 septembre 1998 selon la procédure définie dans la Résolution n° 1 de la CMNT.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'études à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution n° 1 de la CMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, le terme *exploitation reconnue (ER)* désigne tout particulier, toute entreprise, toute société ou tout organisme public qui exploite un service de correspondance publique. Les termes *Administration*, *ER* et *correspondance publique* sont définis dans la *Constitution de l'UIT (Genève, 1992)*.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Introduction

La présente Recommandation définit une notation informelle de tests, appelée notation combinée arborescente et tabulaire (TTCN, *tree and tabular combined notation*), destinée à être utilisée pour spécifier des suites de tests de conformité abstraites OSI.

Pour élaborer une suite de tests abstraite normalisée, on utilise une notation de test décrivant des tests élémentaires abstraits. Cette notation de test peut être une notation informelle (sans sémantique formellement définie) ou une technique de description formelle (FDT, *formal description technique*). La notation TTCN est une notation informelle à sémantique expressément, mais non formellement, définie.

La notation TTCN est destinée à répondre aux objectifs suivants:

a) fournir une notation dans laquelle il soit possible d'exprimer des tests élémentaires abstraits dans des suites de tests normalisées;

b) fournir une notation indépendante des méthodes de test, des couches et des protocoles;

c) fournir une notation reflétant la méthodologie de test abstraite définie dans les Recommandations de la série X.290;

d) fournir une capacité permettant d'utiliser la concomitance lors de la spécification des tests élémentaires abstraits, le cas échéant, tant pour la configuration de test multiparti que pour la configuration de test biparti.

La méthodologie de test abstraite considère une suite de tests comme une hiérarchie allant de la suite de tests complète en passant par les groupes de tests, les tests élémentaires et les modules de tests jusqu'aux événements de test. La notation TTCN donne une structure de dénomination reflétant la position des tests élémentaires dans cette hiérarchie. Elle permet également de structurer les tests élémentaires en une hiérarchie de modules de tests aboutissant à des événements de test. En notation TTCN, les événements de test de base sont l'émission et la réception de primitives de service abstraites (ASP, *abstract service primitives*) et d'unités de données protocolaires (PDU, *protocol data units*) ainsi que les événements de temporisation.

Cette notation est donnée sous deux formes: une forme tabulaire lisible par l'être humain, appelée notation graphique TTCN.GR, destinée à être utilisée dans les normes relatives aux suites de tests de conformité OSI; une forme exploitable par machine, appelée notation TTCN.MP, destinée à être utilisée pour représenter la notation TTCN sous une forme canonique dans des systèmes informatiques et servant de syntaxe de transfert des tests élémentaires entre différents systèmes informatiques. Ces deux formes sont sémantiquement équivalentes.

Recommandation X.292

CADRE ET MÉTHODOLOGIE DES TESTS DE CONFORMITÉ OSI POUR LES RECOMMANDATIONS SUR LES PROTOCOLES POUR LES APPLICATIONS DE L'UIT-T – NOTATION COMBINÉE ARBORESCENTE ET TABULAIRE (TTCN)[1]

(révisée en 1998)

L'UIT-T,

considérant

a) que la Recommandation X.200 définit le modèle de référence pour l'interconnexion des systèmes ouverts (OSI) pour les applications de l'UIT-T;

b) que l'objectif de l'OSI ne sera complètement réalisé que si les systèmes peuvent être testés pour déterminer leur conformité aux Recommandations pertinentes sur les protocoles OSI;

c) que des suites de tests normalisées devraient être élaborées pour chaque Recommandation sur les protocoles OSI, de telle façon:

– que les résultats des tests de conformité provenant de services de test différents soient largement acceptés et accueillis avec confiance;

– qu'elles assurent la possibilité d'interfonctionnement des équipements qui ont subi avec succès les tests de conformité normalisés;

d) qu'il est nécessaire de normaliser le processus des tests de conformité afin d'atteindre un niveau de comparabilité utile et acceptable des résultats d'évaluation de conformité de produits similaires,

déclare à l'unanimité

que la notation à utiliser pour les tests graphiques et abstraits doit être conforme à la présente Recommandation.

1 Domaine d'application

1.1 La présente Recommandation définit une notation informelle de tests, appelée notation combinée arborescente et tabulaire (TTCN), applicable aux suites de tests de conformité OSI. Cette notation indépendante des méthodes de test, des couches et des protocoles, reflète la méthodologie de test abstraite définie dans les Recommandations X.290 et X.291.

1.2 Elle fournit également des spécifications et des directives concernant l'utilisation de la notation TTCN pour spécifier des suites de tests de conformité indépendantes du système dans une ou plusieurs Recommandations OSI. Cette notation est donnée sous deux formes: la première, lisible par l'être humain, est applicable à l'élaboration de Recommandations traitant des suites de tests de conformité pour les protocoles OSI; la seconde, exploitable par machine, s'applique au traitement et aux échanges informatiques.

1.3 La présente Recommandation s'applique à la spécification des tests élémentaires de conformité qui peuvent s'exprimer de manière abstraite en termes de contrôle et d'observation d'unités de données protocolaires et de primitives de service abstraites. Cependant, certains protocoles peuvent requérir des tests élémentaires qui ne peuvent s'exprimer en ces termes. La spécification de ces tests élémentaires n'entre pas dans le cadre de la présente Recommandation, bien qu'ils puissent avoir à figurer dans une Recommandation sur les suites de tests de conformité.

Certaines spécifications de conformité statique relatives à un service d'application peuvent, par exemple, nécessiter des techniques de test spécifiques à cette application particulière.

La spécification des tests élémentaires dans lesquels plusieurs descriptions de comportement doivent se faire en parallèle est traitée par les caractéristiques de concomitance (en particulier pour la définition des composantes de test et des configurations de composantes de test).

---

[1] La Recommandation X.292 et l'ISO/CEI 9646-3, Technologies de l'information – Interconnexion de systèmes ouverts – Essais de conformité – Méthodologie générale et procédures – Partie 3: notation combinée arborescente et tabulaire (TTCN), sont alignées sur le plan technique.

1.4 La présente Recommandation indique ce qu'une Recommandation sur des suites de tests peut spécifier à propos de la conformité de réalisation d'une suite de tests, notamment à propos de la sémantique opératoire des suites de tests en notation TTCN.

1.5 La présente Recommandation s'applique à la spécification de suites de tests de conformité pour les protocoles des couches 2 à 7 de l'OSI, et notamment aux protocoles exprimés en notation de syntaxe abstraite numéro un (ASN.1, *abstract syntax notation one*). N'entrent pas dans le cadre de la présente Recommandation:

a) la spécification de suites de tests de conformité pour les protocoles de couche Physique;

b) les rapports entre la notation TTCN et les techniques de description formelle;

c) la réalisation de suites de tests exécutables (ETS, *executable test suites*) à partir des suites de tests abstraites.

1.6 La présente Recommandation définit les mécanismes permettant d'utiliser la concomitance pour la spécification des tests élémentaires abstraits. La concomitance en notation TTCN est applicable à la spécification des tests élémentaires:

a) dans un contexte de test multiparti;

b) qui traitent du multiplexage et du démultiplexage, soit dans un contexte de test multiparti, soit dans un contexte de test biparti;

c) qui traitent de l'éclatement ou de la recombinaison, soit dans un contexte de test multiparti, soit dans un contexte de test biparti;

d) dans un contexte de test biparti, quand la complexité du protocole ou de l'ensemble de protocoles traité par l'instance IUT est telle que la concomitance peut simplifier la spécification du test élémentaire.

1.7 Les modules TTCN sont définis de manière à permettre le partage des spécifications TTCN communes entre suites de tests.

2 Références normatives

La présente Recommandation se réfère à certaines dispositions des Recommandations UIT-T et textes suivants qui de ce fait en sont partie intégrante. Les versions indiquées étaient en vigueur au moment de la publication de la présente Recommandation. Toute Recommandation ou tout texte étant sujet à révision, les utilisateurs de la présente Recommandation sont invités à se reporter, si possible, aux versions les plus récentes des références normatives suivantes. La liste des Recommandations de l'UIT-T en vigueur est régulièrement publiée.

– Recommandation UIT-T X.200 (1994) | ISO/CEI 7498-1:1994, *Technologies de l'information – Interconnexion des systèmes ouverts – Modèle de référence de base: le modèle de référence de base.*

– Recommandation UIT-T X.210 (1993) | ISO/CEI 10731:1994, *Technologies de l'information – Interconnexion des systèmes ouverts – Modèle de référence de base: conventions pour la définition des services de l'interconnexion de systèmes ouverts.*

– Recommandation UIT-T X.290 (1995), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Concepts généraux.*

  ISO/CEI 9646-1:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI – Partie 1: Concepts généraux.*

– Recommandation UIT-T X.291 (1995), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Spécification de suite de tests abstraite.*

  ISO/CEI 9646-2:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI – Partie 2: Spécification des suites de tests abstraites.*

– Recommandation UIT-T X.293 (1995), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Réalisation des tests.*

  ISO/CEI 9646-4:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI – Partie 4: Réalisation des tests.*

– Recommandation UIT-T X.294 (1995), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Prescriptions des laboratoires de test et des clients en matière de processus d'évaluation de conformité.*

ISO/CEI 9646-5:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI – Partie 5: Spécifications pour laboratoires d'essais et clients pour le procédé d'évaluation de conformité.*

– Recommandation UIT-T X.295 (1995), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Spécification des tests de profil de protocole.*

ISO/CEI 9646-6:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI – Partie 6: Spécification de test pour les profils de protocoles.*

– Recommandation UIT-T X.296 (1995), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Déclarations de conformité d'instance.*

ISO/CEI 9646-7:1995, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Essais de conformité – Méthodologie générale et procédures – Partie 7: Déclarations de conformité des mises en œuvre.*

– Recommandation UIT-T X.680 (1997) | ISO/CEI 8824-1:1998, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base.*

– Recommandation UIT-T X.681 (1997) | ISO/CEI 8824-2:1998, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des objets informationnels.*

– Recommandation UIT-T X.682 (1997) | ISO/CEI 8824-3:1998, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des contraintes.*

– Recommandation UIT-T X.683 (1997) | ISO/CEI 8824-4:1998, *Technologies de l'information – Notation de syntaxe abstraite numéro un: Paramétrage des spécifications de la notation de syntaxe abstraite numéro un.*

– Recommandation UIT-T X.690 (1997) | ISO/CEI 8825-1:1998, *Technologies de l'information – Règles de codage de la notation de syntaxe abstraite numéro un: Spécification des règles de codage de base, des règles de codage canoniques et des règles de codage distinctives.*

– Recommandation UIT-T X.691 (1997) | ISO/CEI 8825-2:1998, *Technologies de l'information – Règles de codage ASN.1 – Spécification des règles de codage compact.*

– ISO/CEI 646:1991, *Technologies de l'information – Jeu ISO de caractères codés à 7 éléments pour l'échange d'informations. (Publiée actuellement en anglais seulement).*

– ISO/CEI 10646-1:1993, *Technologies de l'information – Jeu universel de caractères codés à plusieurs octets – Partie 1: Architecture et table multilingue. (Publiée actuellement en anglais seulement).*

3 Définitions

La présente Recommandation définit les termes suivants:

3.1 Termes de base tirés de la Recommandation X.290

Les termes suivants, définis dans la Recommandation X.290, sont utilisés:

a) primitive de service abstraite;

b) méthodologie de test abstraite;

c) test élémentaire abstrait;

d) méthode de test (abstraite);

e) suite de tests abstraite;

f) journal de conformité;

g) suite de tests (de conformité);

h) méthode de test coordonnée;

i) méthode de test répartie;

j) test élémentaire exécutable;

k) erreur de test élémentaire exécutable;

l) suite de tests exécutables;

m) échec (verdict);

n) état de test "au repos";

o) implémentation sous test;

p) non concluant (verdict);

q) événement de test non valide;

r) méthode de test locale;

s) testeur inférieur;

t) moyens de test;

u) succès (verdict);

v) formulaire PICS;

w) formulaire PIXIT;

x) déclaration de conformité d'implémentation de protocole;

y) informations supplémentaires sur l'implémentation de protocole destinées au test;

z) point de contrôle et d'observation;

aa) méthode de test à distance;

ab) état de test stable;

ac) suite de tests abstraite normalisée;

ad) condition de conformité statique;

ae) événement de test non syntaxiquement valide;

af) système sous test;

ag) corps de test;

ah) test élémentaire;

ai) erreur de test élémentaire;

aj) procédures de coordination de tests;

ak) événement de test;

al) groupe de tests;

am) objectif de groupe de tests;

an) laboratoire (de test);

ao) protocole de gestion de tests;

ap) résultat d'un test;

aq) épilogue de test;

ar) préambule (de test);

as) objectif d'un test;

at) réalisation d'un test;

au) réalisateur de tests;

av) module de test;

aw) suite de tests (de conformité);

ax) système de test;

ay) testeur supérieur;

az) verdict (d'un test);

ba) état de test.

3.2 Termes tirés de la Recommandation X.200

Les termes suivants, définis dans la Recommandation X.200 (1995), sont utilisés:

a) couche (N) (en particulier pour les couches Application, Session et Transport);

b) unité de données protocolaires (N);

c) point d'accès au service (N);

d) sous-réseau;

e) syntaxe de transfert.

3.3 Termes tirés de la Recommandation X.210

Le terme suivant, défini dans la Recommandation X.210, est utilisé:

– fournisseur de services OSI.

3.4 Termes tirés de la Recommandation X.680

Les termes suivants, définis dans la Recommandation X.680, sont utilisés:

a) type chaîne binaire;

b) type chaîne de caractères;

c) type énuméré;

d) type externe;

e) identificateur d'objet;

f) type chaîne d'octets;

g) type réel;

h) type sélection;

i) type séquence;

j) type séquence de;

k) type ensemble;

l) type ensemble de;

m) sous-type.

NOTE – Quand il y a risque de confusion avec la notation TTCN, ces termes sont précédés du préfixe ASN.1.

3.5 Termes tirés de la Recommandation X.690

Le terme suivant, défini dans la Recommandation X.690, est utilisé:

– codage.

3.6 Termes spécifiques à la notation TTCN

Pour les besoins de la présente Recommandation, les définitions suivantes s'appliquent:

3.6.1 règles de codage applicables: règles de codage réelles qui doivent être utilisées lors de l'envoi ou de la réception d'une unité PDU, après combinaison de toutes les valeurs de codage par défaut et des priorités.

3.6.2 construction attach: déclaration en notation combinée arborescente et tabulaire rattachant un module de test à un arbre d'appel.

3.6.3 contrainte de base: spécifie un ensemble de valeurs par défaut pour chaque champ dans une définition du type de primitive de service abstraite ou d'unité de données protocolaires.

3.6.4 type de base: type duquel est dérivé un type défini dans une suite de tests.

3.6.5 ligne comportementale: entrée dans une table de comportement dynamique, représentant un événement de test ou une autre déclaration en notation combinée arborescente et tabulaire, éventuellement munie d'une étiquette, d'un verdict, d'une référence aux contraintes et d'un commentaire.

3.6.6 arbre comportemental: spécification d'un ensemble de séquences d'événements de test et d'autres déclarations en notation combinée arborescente et tabulaire.

3.6.7 entrée muette: dans une table compacte de contraintes modifiées, une entrée muette (laissée en blanc) dans un paramètre ou dans un champ de contrainte traduit le fait que sa valeur est fixée par héritage.

3.6.8 arbre d'appel: arbre comportemental auquel un module de test est rattaché.

3.6.9 table compacte des contraintes: déclaration sur une seule table d'un ensemble de contraintes pour une primitive de service abstraite, une unité de données protocolaires ou un type structuré.

3.6.10 table compacte des tests élémentaires: déclaration sur une seule table d'un ensemble de tests élémentaires appartenant à un même groupe de tests.

3.6.11 test élémentaire concomitant: test élémentaire spécifié à l'aide de la notation TTCN concomitante.

3.6.12 notation TTCN concomitante: notation TTCN qui utilise des composantes de test et des configurations de composantes de test afin d'exprimer la concomitance dans le comportement dynamique des tests élémentaires.

3.6.13 partie contraintes: partie d'une suite de tests en notation combinée arborescente et tabulaire dans laquelle sont spécifiées les valeurs des paramètres de primitives de service abstraites ou des champs d'unités de données protocolaires envoyées à l'implémentation sous test, ainsi que les conditions imposées aux paramètres de primitives de service abstraites et aux champs d'unités de données protocolaires par l'implémentation sous test.

3.6.14 référence aux contraintes: référence à une contrainte, indiquée dans une ligne comportementale.

3.6.15 message de coordination (CM, *coordination message*): élément d'information structurée qui peut être transféré d'une composante de test à une autre, en un point de coordination.

3.6.16 point de coordination (CP, *coordination point*): point d'un environnement de test affecté à deux composantes de test dans une configuration de composantes de test, où les messages de coordination peuvent être échangés de manière asynchrone entre ces composantes de test.

3.6.17 partie déclarative: partie d'une suite de tests en notation combinée arborescente et tabulaire se rapportant à la définition et à la déclaration de tous les objets non prédéfinis utilisés dans cette suite de tests.

3.6.18 comportement par défaut: événements et autres déclarations en notation combinée arborescente et tabulaire qui peuvent se produire à un niveau quelconque de l'arbre associé et qui sont indiqués dans le formulaire des comportements par défaut.

3.6.19 groupe de comportements par défaut: ensemble nommé de comportements par défaut.

3.6.20 référence de groupe d'un comportement par défaut: chemin d'accès spécifiant l'emplacement logique d'un comportement par défaut dans la bibliothèque des comportements par défaut.

3.6.21 identificateur par défaut: nom unique d'un comportement par défaut.

3.6.22 bibliothèque des comportements par défaut: ensemble des comportements par défaut dans une suite de tests.

3.6.23 référence de comportement par défaut: référence à un comportement par défaut dans la bibliothèque des comportements par défaut, faite à partir d'une table de tests élémentaires ou de modules de test.

3.6.24 chemin de dérivation: identificateur composé d'un identificateur de contrainte de base concaténé avec un ou plusieurs identificateurs de contraintes modifiées séparés par des points et se terminant par un point.

3.6.25 chaînage dynamique: établissement d'un lien par paramétrage, allant des déclarations de contrainte d'un paramètre de primitive de service abstraite ou d'un champ d'unité de données protocolaires vers la déclaration de contrainte d'une autre unité de données protocolaires. La référence aux contraintes d'une ligne comportementale spécifie quelles unités de données protocolaires sont chaînées.

3.6.26 partie dynamique: partie d'une suite de tests en notation combinée arborescente et tabulaire spécifiant les descriptions de comportement dynamique des tests élémentaires, des modules de test et des comportements par défaut.

3.6.27 suite de tests développée: suite de tests dont tous les objets importés sont développés. Il s'agira du résultat de la conversion d'une suite de tests modulaire selon l'algorithme de l'Annexe B.

3.6.28 objet externe explicite: objet nommé d'une table d'objets externes. Un objet qui est explicitement déclaré comme étant externe dans un module doit être explicitement défini ou être exporté en tant qu'objet externe.

3.6.29 objet explicitement défini: objet pour lequel une définition ou une déclaration existe dans le module ou dans la suite de tests.

3.6.30 objet explicitement exporté: objet nommé des tables d'exportation, qui est disponible aux fins d'utilisation. Si l'objet est un objet importé, le nom de l'objet source doit être donné.

3.6.31 objet explicitement importé: objet nommé des tables d'importation, qui est disponible aux fins de référence explicite.

3.6.32 objet exporté: objet explicitement défini ou objet explicitement importé dans un objet source, pouvant être utilisé dans tout autre module ou suite de tests. Un objet exporté est soit un objet explicitement exporté, soit un objet implicitement exporté.

3.6.33 objet externe: objet auquel il est fait référence par son nom dans un module, mais qui n'est ni importé, ni explicitement défini. Un objet externe doit être déclaré dans la table d'objets externes. Un objet externe peut être soit explicitement externe, soit implicitement externe.

3.6.34 variable de résultat global: variable de test élémentaire prédéfinie tenue à jour par une composante de test principale dans le contexte MPyT ou par le test élémentaire dans le contexte SPyT, pour enregistrer l'effet accumulé de tous les résultats préliminaires du test élémentaire, afin de déterminer le verdict du test.

3.6.35 objet externe implicite: objet déclaré comme étant externe dans une table d'exportation, qui est omis de la table d'importation correspondante.

3.6.36 objet implicitement exporté: objet explicitement défini ou explicitement importé, qui n'est pas lui-même explicitement exporté, mais auquel un objet explicitement exporté fait référence.

3.6.37 objet implicitement importé: objet auquel un objet explicitement importé fait référence. L'utilisation d'un objet implicitement importé est réservée aux objets explicitement importés (provenant du même objet source) qui y font référence.

3.6.38 événement d'envoi implicite: mécanisme utilisé dans les méthodes de test à distance pour spécifier qu'il revient à l'implémentation sous test de déclencher une unité PDU ou une primitive ASP particulière.

3.6.39 objet importé: objet copié à partir d'un autre objet source, disponible aux fins d'utilisation. Un objet importé est soit un objet explicitement importé, soit un objet implicitement importé.

3.6.40 niveau d'indentation: indique la structure de l'arbre dans une description comportementale. Le niveau d'indentation du texte reflète la structure arborescente d'une description comportementale.

3.6.41 variable de résultat local: variable prédéfinie tenue à jour par une composante de test pour enregistrer l'effet accumulé de ses résultats préliminaires.

3.6.42 arbre local: arbre comportemental défini dans le même formulaire que son arbre d'appel.

3.6.43 composante de test principale (MTC, *main test component*): il s'agit de la seule composante de test d'une configuration de composantes de test, qui est responsable de la création et du contrôle des composantes de test parallèles, ainsi que du calcul et de l'affectation du verdict de test.

3.6.44 contrainte modifiée: contrainte définie pour une primitive de service abstraite ou une unité de données protocolaires ayant déjà une contrainte de base, et correspondant à une modification de cette contrainte de base.

3.6.45 suite de tests modulaire: suite de tests contenant des tables d'importation.

3.6.46 module: collection autonome d'objets TTCN. Tous les objets référencés sont soit définis explicitement dans le module, soit importés d'autres sources, ou encore sont définis en tant qu'objets externes dans le module.

3.6.47 test élémentaire non concomitant: test élémentaire spécifié en notation TTCN, mais sans utiliser la notation TTCN concomitante.

3.6.48 objet: élément de l'une des catégories d'objets énumérées en A.4.2.2 (pour les objets TTCN ayant un identificateur globalement unique) ou en A.4.2.6 (pour les identificateurs ASN.1 globalement uniques dans la suite de tests).

3.6.49 sémantique opératoire: sémantique expliquant l'exécution d'un arbre comportemental en notation combinée arborescente et tabulaire.

3.6.50 objet source d'origine: module ou suite de tests dans lequel ou laquelle un objet est explicitement défini.

3.6.51 événement sinon: mécanisme en notation combinée arborescente et tabulaire permettant de traiter de manière contrôlée des événements de test imprévus.

3.6.52 partie présentation générale: partie d'une suite de tests en notation combinée arborescente et tabulaire relative à la présentation générale de la structure de la suite de tests, de la structure de la bibliothèque des modules de test (si elle existe), de la structure de la bibliothèque des comportements par défaut (si elle existe) et de l'association des expressions de sélection (s'il y en a) aux tests élémentaires et aux groupes de tests. Cette partie fournit également les index des tests élémentaires, des modules de test et des comportements par défaut.

3.6.53 composante de test parallèle (PTC, *parallel test component*): composante de test créée par la composante de test principale.

3.6.54 résultat préliminaire: résultat enregistré avant la fin d'un test élémentaire, indiquant si le verdict établi pour la partie associée du test élémentaire est un succès, un échec ou un verdict non concluant.

3.6.55 pseudo-événement: expression en notation combinée arborescente et tabulaire ou opération de temporisation apparaissant sur une ligne de déclaration dans une description comportementale, sans être associée à un événement.

3.6.56 événement qualifié: événement auquel est associée une expression booléenne.

3.6.57 événement de réception: réception d'une primitive de service abstraite ou d'une unité de données protocolaires en un point de contrôle et d'observation nommé ou implicite.

3.6.58 variable de résultat: variable de test élémentaire prédéfinie, destinée à emmagasiner les résultats préliminaires. En notation TTCN non concomitante, il existe une seule variable de résultat appelée R. En notation TTCN concomitante, il existe une seule variable de résultat global appelée R, chaque composante PTC ayant une variable de résultat local appelée R et la composante MTC ayant une variable de résultat local appelée MTC_R.

3.6.59 arbre racine: arbre comportemental principal d'un test élémentaire, intervenant au niveau d'entrée de ce test élémentaire.

3.6.60 événement d'envoi: envoi d'une primitive de service abstraite ou d'une unité de données protocolaires vers un point de contrôle et d'observation nommé ou implicite.

3.6.61 ensemble d'options: déclarations en notation combinée arborescente et tabulaire codées au même niveau d'indentation et raccordées au même nœud antécédent. Ces déclarations représentent les événements, pseudo-événements et constructions possibles, devant être examinés au point approprié dans l'exécution du test élémentaire.

3.6.62 table simple de contrainte: déclaration dans une table simple d'une contrainte pour une primitive de service abstraite ou une unité de données protocolaires simple d'un type donné.

3.6.63 sémantique d'image instantanée: modèle sémantique supprimant l'effet temporel sur l'exécution d'un test élémentaire, et défini en termes d'image instantanée de l'environnement du test pendant laquelle cet environnement est gelé pour une période spécifiée.

3.6.64 objet source: module importé ou suite de tests importée auxquels correspond une table d'importation.

3.6.65 valeur spécifique: valeur en notation combinée arborescente et tabulaire ne contenant ni mécanisme de concordance, ni variable non liée.

3.6.66 chaînage statique: établissement d'un lien allant des déclarations de contrainte d'un paramètre de primitive de service abstraite ou d'un champ d'unité de données protocolaires à la déclaration de contrainte d'un autre champ d'unité de données protocolaires en prenant explicitement une contrainte pour valeur.

3.6.67 sémantique statique: règles de sémantique limitant l'utilisation de la syntaxe en notation combinée arborescente et tabulaire.

3.6.68 type structuré: collection d'un ou plusieurs paramètres de primitive de service abstraite ou champs d'unité de données protocolaires pouvant apparaître dans une ou plusieurs définitions de type de primitive de service abstraite ou d'unité de données protocolaires, définie dans une déclaration distincte et pouvant servir à spécifier une partie d'une structure plate ou d'une sous-structure dans cette primitive de service abstraite ou cette unité de données protocolaires.

3.6.69 sous-module: module inclus dans un autre module.

3.6.70 identificateur de test élémentaire: nom unique d'un test élémentaire.

3.6.71 variable de test élémentaire: élément d'un ensemble de variables déclaré globalement pour la suite de tests, mais dont la valeur n'est utilisée que dans un seul test élémentaire.

3.6.72 composante de test: subdivision nommée d'un test élémentaire concomitant pouvant être exécutée en parallèle avec d'autres composantes de test, et déclarée comme ayant un nombre fixe de points PCO et un nombre fixe ou maximal de points CP.

3.6.73 configuration de composantes de test: configuration fixe de composantes de test, de points PCO et de points CP, déclarée aux fins d'utilisation dans les tests élémentaires concomitants.

3.6.74 référence de groupe d'un test: chemin spécifiant l'emplacement logique d'un test élémentaire dans une structure de suite de tests abstraite.

3.6.75 groupe de modules de test: ensemble nommé de modules de test.

3.6.76 référence de groupe d'un module de test: trajet spécifiant l'emplacement logique d'un module de test dans la bibliothèque des modules de test.

3.6.77 identificateur de module de test: nom unique d'un module de test.

3.6.78 bibliothèque des modules de test: ensemble des descriptions des comportements dynamiques des modules de test autres que les modules locaux, dans la suite de tests.

3.6.79 objectif de module de test: déclaration informelle de l'objectif que le module de test doit réaliser.

3.6.80 constante de suite de tests: élément d'un ensemble de constantes, non dérivé de la déclaration de conformité d'une implémentation de protocole ou des informations supplémentaires sur l'implémentation de protocole destinées au test, qui reste inchangé dans toute la suite de tests.

3.6.81 paramètre de suite de tests: élément d'un ensemble de constantes, dérivé de la déclaration de conformité d'une implémentation de protocole ou des informations supplémentaires sur l'implémentation de protocole destinées au test, qui paramètre globalement une suite de tests.

3.6.82 variable de suite de tests: élément d'un ensemble de variables déclaré globalement pour la suite de tests et dont la valeur est transmise entre les différents tests élémentaires.

3.6.83 événement de fin de temporisation: événement utilisé dans un arbre comportemental pour vérifier une fin de temporisation donnée.

3.6.84 rattachement à un arbre: méthode permettant d'indiquer qu'un arbre comportemental spécifié ailleurs (en un autre point du formulaire courant ou comme module de test dans la bibliothèque des modules de test) doit être inclus dans l'arbre comportemental courant.

3.6.85 en-tête d'arbre: identificateur d'arbre local suivi d'une liste facultative des paramètres formels de cet arbre.

3.6.86 identificateur d'arbre: nom identifiant un arbre local.

3.6.87 feuille d'arbre: déclaration en notation combinée arborescente et tabulaire dans un arbre comportemental ou un module de test pour laquelle aucun comportement subséquent n'est spécifié.

3.6.88 nœud en arbre: déclaration simple en notation combinée arborescente et tabulaire.

3.6.89 notation d'arbre: notation en notation combinée arborescente et tabulaire utilisée pour représenter les tests élémentaires sous forme d'arbres.

3.6.90 déclaration en notation combinée arborescente et tabulaire: événement, pseudo-événement ou construction spécifiés dans une description comportementale.

3.6.91 événement de test imprévu: événement de test non identifié comme tel dans les résultats de test prévus de la suite de tests. Il est normalement traité par l'événement SINON.

3.6.92 événement non qualifié: événement auquel n'est associée aucune expression booléenne.

4 Abréviations

La présente Recommandation utilise les abréviations suivantes.

4.1 Abréviations définies dans la Recommandation X.290

Pour les besoins de la présente Recommandation, les abréviations suivantes, définies au paragraphe 4/X.290, sont utilisées:

ASP primitive de service abstraite (*abstract service primitive*)

ATS suite de tests abstraite (*absract test suite*)

ETS suite de tests exécutables (*executable test suite*)

IUT implémentation sous test (*implementation under test*)

LT testeur inférieur (*lower tester*)

LTCF fonction de commande du testeur inférieur (*lower tester control function*)

MOT moyens de test (*means of testing*)

PCO point de contrôle et d'observation (*point of control and observation*)

PICS déclaration de conformité d'une implémentation de protocole (*protocol implementation conformance statement*)

PIXIT informations supplémentaires sur l'implémentation de protocole destinées au test (*protocol implementation extra information for testing*)

SUT système sous test (*system under test*)

TMP protocole de gestion de test (*test management protocol*)

UT testeur supérieur (*upper tester*)

UTCF fonction de commande du testeur supérieur (*upper tester control function*)

4.2 Abréviations définies dans la Recommandation X.291

Pour les besoins de la présente Recommandation, les abréviations suivantes, définies au paragraphe 4/X.291, sont utilisées:

DS méthode de test monocouche répartie [*distributed single-layer (test method)*]

LS méthode de test monocouche locale [*local single-layer (test method)*]

RS méthode de test monocouche à distance [*remote single-layer (test method)*]

TTCN notation combinée arborescente et tabulaire (*tree and tabular combined notation*)

4.3 Autres abréviations

Pour les besoins de la présente Recommandation, les abréviations suivantes sont également utilisées:

ASN.1 notation de syntaxe abstraite numéro un (*abstract syntax notation one*)

BNF forme de Backus-Naur (étendue, utilisée en notation TTCN) (*the extended Backus-Naur form used in TTCN*)

CM message de coordination (*coordination message*)

CP point de coordination (*coordination point*)

FDT technique de description formelle (*formal description technique*)

FIFO premier entré premier sorti (*first in first out*)

MTC composante de test principale (*main test component*)

OSI interconnexion des systèmes ouverts (*open systems interconnection*)

PDU unité de données protocolaires (*protocol data unit*)

PTC composante de test parallèle (*parallel test component*)

SAP point d'accès au service (*service access point*)

TCP procédures de coordination des tests (*test coordination procedures*)

TTCN.GR notation combinée arborescente et tabulaire, forme graphique (*tree and tabular combined notation, graphical form*)

TTCN.MP notation combinée arborescente et tabulaire, forme exploitable par machine (*tree and tabular combined notation, machine processable form*)

5 Formes syntaxiques de la notation TTCN

La notation TTCN existe sous deux formes:

– une forme graphique (TTCN.GR), adaptée à la lecture par l'être humain;

– une forme exploitable par machine (TTCN.MP), adaptée à la transmission de descriptions TTCN entre machines, pouvant également convenir à d'autres formes de traitement automatique.

La forme TTCN.GR est définie à l'aide de formulaires tabulaires, tandis que la forme TTCN.MP est définie à l'aide de productions syntaxiques utilisant comme délimiteurs des mots clés TTCN.MP spéciaux à la place des éléments de délimitation fixes des formulaires tabulaires (cadres et en-têtes par exemple). Dans les tables en notation TTCN.GR, les entrées sont définies par des productions syntaxiques ne faisant intervenir aucun mot clé de la notation TTCN.MP; ces productions sont communes aux notations TTCN.GR et TTCN.MP.

Les productions syntaxiques en notation TTCN.MP sont spécifiées dans l'Annexe A. Pour éclaircir la notation TTCN.GR, de nombreuses productions syntaxiques communes aux notations TTCN.GR et TTCN.MP sont incluses dans le corps de la présente Recommandation, où elles sont indiquées par le titre DÉFINITION SYNTAXIQUE. Pour une meilleure lisibilité, certaines productions apparaissent en plusieurs endroits du texte.

Les productions syntaxiques incluses dans le texte sont censées être des copies exactes des productions correspondantes de l'Annexe A; en cas de différence, l'Annexe A fait foi.

La description textuelle de la notation TTCN.GR est censée correspondre à la syntaxe sous-jacente définie dans les productions syntaxiques en notation TTCN.MP, à l'exception des différences indiquées en A.5 et des restrictions de sémantique statique spécifiées à l'Annexe A (qui sont communes aux notations TTCN.GR et TTCN.MP).

Si une contradiction apparaît entre la syntaxe en notation TTCN.GR, d'une part, et la sémantique statique et opératoire, d'autre part, telles qu'elles sont décrites dans le texte et dans l'Annexe A, les règles suivantes s'appliquent:

a) les productions syntaxiques en notation TTCN.MP ont la priorité sur la présentation textuelle et les productions syntaxiques du corps de la présente Recommandation, à l'exception des différences indiquées en A.5;

b) les restrictions de sémantique statique spécifiées en A.4 et dans les commentaires de sémantique statique (indiqués par la mention SÉMANTIQUE STATIQUE) des productions syntaxiques en A.3 restreignent les formes valides en notation TTCN, en indiquant les formes de productions syntaxiques autorisées;

c) de manière semblable, les restrictions de sémantique opératoire spécifiées dans les commentaires de la sémantique opératoire (indiqués par la mention SÉMANTIQUE OPÉRATOIRE) des productions syntaxiques en A.3 restreignent les formes valides en notation TTCN au moment de l'exécution, en indiquant les formes de productions syntaxiques autorisées;

d) les restrictions de sémantiques statique et opératoire spécifiées dans l'Annexe A ont priorité sur le texte apparaissant dans le corps de la présente Recommandation.

Si une suite ATS est spécifiée en notation TTCN.GR conformément à la présente Recommandation, il n'existe alors qu'une seule représentation correspondante en notation TTCN.MP de cette suite ATS ayant la même syntaxe sous-jacente. Ces deux représentations ont des sémantiques opératoires identiques. Deux représentations différentes d'une suite ATS sont équivalentes si et seulement si elles ont des sémantiques opératoires identiques.

NOTE – Si une suite ATS normalisée spécifiée en notation TTCN.GR a une représentation apparemment équivalente en notation TTCN.MP, mais que les interprétations de leur sémantique opératoire diffèrent, c'est la sémantique opératoire de la notation TTCN.GR qui prime, car c'est elle qui constitue la suite ATS normalisée.

6 Conformité

6.1 Pour être conformes à la présente Recommandation les suites ATS répondront aux spécifications des notations TTCN.GR ou TTCN.MP.

NOTE – Le paragraphe 10/X.290 explique l'utilisation du terme "conformité" dans les Recommandations de la série X.290.

6.2 Pour être conformes aux spécifications de la notation TTCN.GR, les suites ATS répondront aux spécifications syntaxiques de la notation TTCN.GR énoncées aux paragraphes 9 à 16 et en A.4.

6.3 Pour être conformes aux spécifications de la notation TTCN.MP, les suites ATS répondront aux spécifications syntaxiques de la notation TTCN.MP énoncées en A.3.

6.4 Pour être conformes à la présente Recommandation et être sémantiquement valides, les suites ATS répondront aux spécifications de sémantique statique énoncées dans les paragraphes 7 à 16, ainsi qu'à l'Annexe A, et auront une sémantique opératoire conforme à la définition de la sémantique opératoire donnée dans l'Annexe B et aux restrictions de la sémantique opératoire spécifiées en A.3.

6.5 Pour être conforme à la présente Recommandation, une suite ATS normalisée sera telle que toute instance de cette suite de tests se voulant conforme à cette suite ATS normalisée:

a) ait une sémantique opératoire équivalant à celle de la suite de tests comme défini dans l'Annexe B;

b) satisfasse aux exigences supplémentaires de la sémantique opératoire spécifiées en A.3;

c) soit conforme à la Recommandation X.293.

NOTE – Si une erreur de sémantique statique ou opératoire est décelée pendant l'exécution du test élémentaire exécutable conforme à la spécification TTCN du test élémentaire abstrait correspondant, un laboratoire d'essai conforme à la Recommandation X.294 consignera une erreur de test élémentaire abstrait ou exécutable, selon l'endroit où l'erreur se produit.

7 Conventions

7.1 Introduction

Les conventions suivantes ont été utilisées pour définir les formulaires tabulaires TTCN.GR et la grammaire TTCN.MP.

7.2 Métanotation syntaxique

Le Tableau 1 définit la métanotation utilisée pour spécifier la forme étendue de la grammaire de la forme Backus-Naur pour la notation TTCN (dénotée BNF dans ce qui suit).

Tableau 1/X.292 – Métanotation syntaxique de la notation TTCN.MP

::=	Est par définition
abc xyz	abc suivi de xyz
|	ou
[abc]	0 ou 1 instance d'abc
{abc}	0 ou plusieurs instances d'abc
{abc}+	1 ou plusieurs instances d'abc
(...)	Groupement textuel
abc	Symbole non terminal abc
abc	Symbole terminal abc
"abc"	Symbole terminal abc

EXEMPLE 1 – Utilisation de la métanotation BNF:

FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"

On utilise les conventions suivantes pour les textes des formulaires tabulaires:

a) un texte en caractères gras (**comme ceci**) apparaît mot pour mot dans chaque table réelle d'une suite de tests TTCN;

b) un texte en italique (*comme ceci*) n'apparaît pas mot pour mot dans une suite de tests TTCN. Cette police de caractères sert à indiquer que le texte réel doit remplacer le symbole en italique. Les spécifications syntaxiques concernant le texte réel peuvent être trouvées dans la production BNF correspondante en notation TTCN.MP.

EXEMPLE 2 – *SuiteIdentifier* correspond à la production 3 de l'Annexe A.

7.3 Formulaires tabulaires en notation TTCN.GR

7.3.1 Introduction

La notation TTCN.GR est définie à l'aide de deux types de tables:

a) les tables d'objets TTCN simples (voir 7.3.2),

utilisées pour définir, déclarer ou décrire un objet TTCN simple, comme une déclaration d'unité PDU ou un comportement dynamique de test élémentaire;

b) les tables d'objets TTCN multiples (voir 7.3.3),

utilisées pour définir un certain nombre d'objets TTCN du même type dans une seule table, comme des définitions de type simple ou des variables de test élémentaire.

7.3.2 Tables d'objets TTCN simples

La disposition générale d'une table pour un objet TTCN simple est indiquée dans la Figure 1.

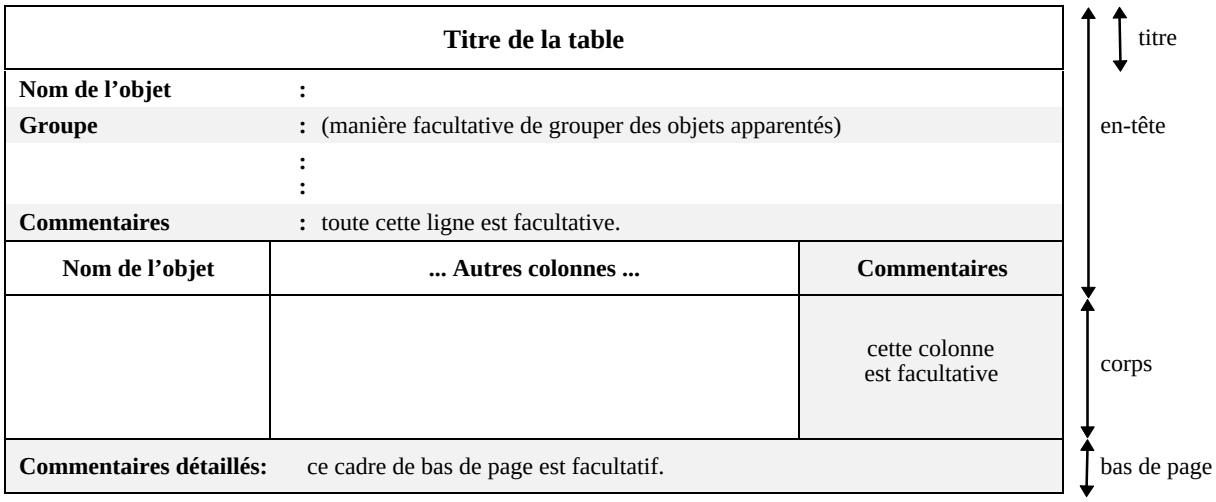


Figure 1/X.292 – Disposition générale d'une table déclarative simple

L'en-tête de la table contient des informations générales relatives à l'objet défini par cette table. La première entrée de l'en-tête, appelée *Nom de l'objet (Object Name)*, contient un identificateur de l'objet. La deuxième, appelée *Groupe (Group)*, peut servir à fournir un identificateur permettant de regrouper des objets apparentés dans la même catégorie. Elle peut être omise. La dernière entrée, appelée *Commentaires (Comments)*, contient une description informelle de l'objet. Elle peut également être omise.

Le corps de la table comprend une ou plusieurs colonnes portant chacune un titre. La colonne la plus à droite, appelée *Commentaires*, contient des descriptions informelles des composantes de l'objet spécifié dans le corps de la table. Elle n'est pas présente dans tous les formulaires et peut être omise si cette colonne est vide.

Le cadre de bas de page contient un article appelé *Commentaires détaillés (Detailed Comments)*. Il peut être utilisé aux mêmes fins que la colonne Commentaires du corps de la table. Le concepteur de la suite de tests peut utiliser ce cadre de bas de page en association avec la colonne Commentaires, à la place de celle-ci, ou ne pas l'utiliser du tout, auquel cas ce cadre peut être omis.

7.3.3 Tables d'objets TTCN multiples

La disposition générale d'une table pour des objets TTCN multiples est indiquée dans la Figure 2.

Le *Commentaire collectif* facultatif peut être utilisé avant un groupe d'objets apparentés, déclarés dans une table d'objets multiples, tant pour indiquer le groupement que pour donner un commentaire s'appliquant à chaque membre du groupe ou au groupe dans son ensemble.

Ce type de table comporte une section d'en-tête facultative minimale, qui peut comprendre un identificateur de *groupe* et un *commentaire collectif*. Le corps de la table comporte une ou plusieurs colonnes dotées chacune d'un titre. La colonne la plus à gauche, appelée *Nom de l'objet*, contient les identificateurs des objets définis ou déclarés dans cette table. La colonne la plus à droite, appelée *Commentaires*, contient des descriptions informelles des objets définis ou déclarés dans la table. Elle n'existe pas dans tous les formulaires. Lorsqu'elle existe, son utilisation est optionnelle pour le concepteur de la suite de tests. Le cadre de bas de page est identique à celui d'une table d'objet simple.

Titre de la table		
Groupe : (manière facultative de grouper des objets apparentés)		
Commentaire collectif: *commentaire valable pour les objets définis/déclarés ci-dessous. Ce commentaire vaut jusqu'au prochain commentaire collectif ou jusqu'à la fin de la présente table.*		
Nom de l'objet	**... Autres colonnes ...**	**Commentaires**
Commentaire collectif: *commentaire valable pour les objets définis/déclarés ci-dessous. Ce commentaire vaut jusqu'au prochain commentaire collectif ou jusqu'à la fin de la présente table.*		
Nom de l'objet	**... Autres colonnes ...**	**Commentaires**
Commentaires détaillés:		

Figure 2/X.292 – Disposition générale d'une table déclarative multiple

7.3.4 Autres tables compactes

Dans certains cas, il est possible de représenter plusieurs tables d'objets simples TTCN sous un format plus compact, c'est-à-dire de présenter plusieurs tables d'objets simples TTCN en une table compacte. Les seules tables pouvant être présentées ainsi sont les suivantes:

– contraintes de primitive ASP (tabulaires et en notation ASN.1);

– contraintes d'unité PDU (tabulaires et en notation ASN.1);

– contraintes du type structuré;

– contraintes du type ASN.1;

– comportements dynamiques de test élémentaire.

Les formats de ces formulaires compacts sont définis dans l'Annexe E.

7.3.5 Spécification des formulaires

La présente Recommandation spécifie de nombreux types de tables en notation TTCN.GR et donne une représentation graphique des formulaires correspondants. Ces formulaires sont conformes à la disposition générale indiquée aux 7.3.2 et 7.3.3. Les colonnes facultatives sont ombrées pour mémoire.

7.4 Texte libre et texte libre borné

Certaines entrées des tables permettent l'utilisation de texte libre, c'est-à-dire de caractères provenant de l'un quelconque des jeux de caractères définis dans l'ISO/CEI 10646-1, sous réserve des restrictions suivantes:

a) la combinaison de caractères "*/", utilisée en notation TTCN.MP pour indiquer la fin d'une chaîne de texte libre, ne doit pas apparaître dans le texte libre à moins d'être précédée par une barre oblique inverse (\). Une double barre oblique inverse (\\) sera utilisée pour représenter une barre oblique inverse dans le texte;

b) les combinaisons des caractères "/*" et "*/" ouvrant et fermant les chaînes de texte libre borné BoundedFreeText en notation TTCN.MP seront omises en notation TTCN.GR, c'est-à-dire que si une chaîne de texte libre borné apparaît dans un champ de la table, par exemple dans le champ de l'identificateur complet, ces combinaisons de caractères ne seront pas imprimées.

8 Concomitance en notation TTCN

8.1 Composantes de test

La notation TTCN permet la spécification de composantes de test qui peuvent être exécutées de manière concomitante. Le présent paragraphe donne un aperçu général des formulaires et mécanismes supplémentaires qui existent en notation TTCN concomitante. Ces formulaires et mécanismes ne doivent pas être utilisés dans les suites ATS qui n'utilisent pas la concomitance (c'est-à-dire que l'utilisation de la concomitance est facultative).

Un testeur comprend une composante de test principale (MTC, *main test component*) et zéro ou une ou plusieurs composantes de test parallèles (PTC, *parallel test component*). En notation TTCN non concomitante, il est inutile de déclarer la composante de test principale, puisqu'il n'y a qu'une seule composante de test et que, par défaut, il s'agit de la composante de test principale.

Les composantes de test sont déclarées dans la table de déclaration des composantes de test. Une composante de test peut communiquer, avec l'implémentation IUT, par l'intermédiaire d'un ou de plusieurs points de contrôle et d'observation (PCO, *point of control and observation*). Les composantes de test peuvent communiquer les unes avec les autres en échangeant des messages de coordination (CM, *coordination message*) par l'intermédiaire de points de coordination (CP, *coordination point*). Les composantes PTC peuvent aussi communiquer implicitement avec la composante MTC, au moyen d'affectations à la variable de résultat global, la composante principale étant capable de vérifier si une ou plusieurs composantes parallèles ont terminé l'exécution ou non. Les tables de déclaration de configuration de composantes de test servent à spécifier les configurations (abstraites) de composantes de test. Ces déclarations (une par configuration) montrent quels sont les points PCO et CP utilisés, le cas échéant, par les composantes de test. Les messages CM sont spécifiés d'une manière très semblable à la méthode servant à spécifier les primitives ASP. La notation ASN.1 peut être utilisée pour spécifier les messages CM. Les contraintes des messages CM sont aussi très semblables à celles des primitives ASP. Des formulaires particuliers sont fournis pour la définition des types de message CM et pour la déclaration des contraintes de message CM. Les messages de coordination sont envoyés et reçus en utilisant les déclarations TTCN normales SEND et RECEIVE.

En résumé, dans le cas de l'utilisation de la notation TTCN concomitante, il convient d'utiliser les formulaires suivants:

a) déclarations de composante de test;

b) déclarations de configuration de composantes de test.

De plus, dans le cas de l'utilisation de la notation TTCN concomitante, on peut utiliser les formulaires suivants:

c) déclarations de point CP;

d) définitions de type de message CM et définitions de type de message CM en notation ASN.1, à condition que les déclarations de point CP soient utilisées;

e) déclarations des contraintes de message CM, à condition que les définitions de type de message CM soient utilisées;

f) déclarations des contraintes de message CM en notation ASN.1, à condition que les définitions de type de message CM en notation ASN.1 soient utilisées.

8.2 Configurations des composantes de test

Les Figures 3 et 4 illustrent quelques configurations possibles de composantes de test. Dans une instance de ces configurations abstraites, les composantes de test peuvent résider dans une seule machine ou être réparties sur plusieurs.

Il est possible d'utiliser différentes configurations de composantes PTC dans différents tests élémentaires d'une suite de tests abstraite. Chaque test élémentaire abstrait qui utilise la concomitance doit utiliser l'une des configurations de composantes de test déclarées.

Il convient de noter les cas inhabituels, mais valides suivants:

a) une composante PTC n'a pas besoin d'avoir de point PCO;

b) une composante PTC n'a pas besoin d'avoir de point CP vers une composante MTC. Dans un tel cas, la seule interaction entre la composante parallèle et la composante principale sera la création de la composante parallèle et les rapports de résultats implicites provenant de la composante PTC, c'est-à-dire que la composante MTC n'a pas de contrôle explicite sur la composante PTC après sa création;

c) deux composantes PTC peuvent être raccordées par plus d'un point CP;

d) un test élémentaire dont la configuration de composantes de test se rapporte à une composante PTC n'a pas besoin de contenir de déclaration CREATE pour déclencher cette composante PTC;

e) un test élémentaire dont la configuration de composantes de test se rapporte à un point CP n'a pas besoin de contenir de déclaration SEND ou RECEIVE utilisant ce point CP.

Les points a), b) et c) sont illustrés aux Figures 3 et 4.

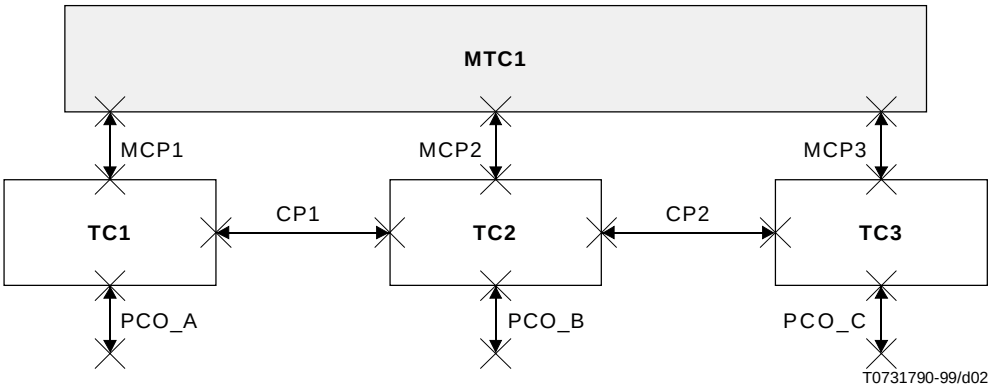


Figure 3/X.292 – Exemple de configuration de composantes de test CONFIG1

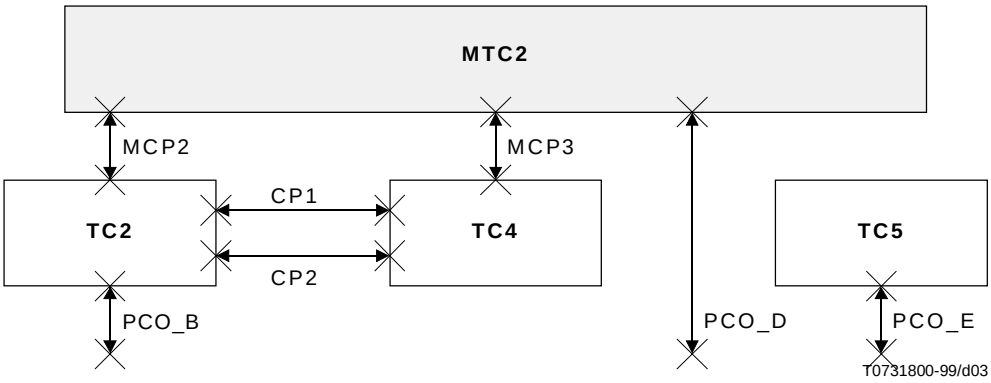


Figure 4/X.292 – Exemple de configuration de composantes de test CONFIG2

9 Structure des suites de tests TTCN

9.1 Introduction

La notation TTCN permet de conférer une structure hiérarchique à une suite de tests conformément au 8.1/X.290. Cette structure comprend les composantes suivantes:

a) groupes de tests;

b) tests élémentaires;

c) modules de test.

Une suite de tests en notation TTCN peut être complètement plate (c'est-à-dire dépourvue de structure); dans ce cas, il n'existe pas de groupes de tests.

La notation TTCN permet d'utiliser des groupes de modules de test et des groupes de comportements par défaut, conceptuellement similaires aux groupes de tests, pour structurer hiérarchiquement les modules de test et les comportements par défaut. Cette structure hiérarchique est facultative.

9.2 Références de groupe de tests

La notation TTCN prend en charge une structure de dénomination reflétant le regroupement conceptuel des tests élémentaires. Les groupes de tests peuvent être imbriqués. Les tests élémentaires peuvent également être autonomes (voir la Figure 9/X.290). Les références de groupe de tests définissent la structure de la suite de tests; elles obéissent à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

EXEMPLE 3 – Référence de groupe de transport: TRANSPORT/CLASS0/CONN_ESTAB/

9.3 Références de groupe de modules de test

9.3.1 Les modules de test peuvent être explicitement identifiés en notation TTCN et utilisés pour structurer des tests élémentaires ou d'autres modules de test. Les modules de test peuvent aussi être implicites dans la description de comportement d'un test élémentaire. Les modules de test explicites peuvent être spécifiés:

– soit localement dans un test élémentaire ou dans une description de comportement d'un module de test;

– soit globalement dans une bibliothèque des modules de test; celle-ci peut être structurée hiérarchiquement en groupes de modules de test.

NOTE – Un préambule peut, par exemple, comprendre quelques lignes d'une description de comportement d'un test élémentaire; il est alors implicite. Un préambule peut aussi être spécifié explicitement par sa propre description de comportement. Si un tel préambule explicite est utilisé uniquement dans un test élémentaire, il peut être spécifié localement dans ce même test. Par contre, s'il est utilisé dans plusieurs tests élémentaires, il doit être spécifié dans la bibliothèque des modules de test.

9.3.2 Les modules de test locaux sont identifiés simplement par un identificateur d'arbre. Les modules de test globaux sont identifiés par un identificateur de module de test. Les modules de test globaux ont aussi une référence de groupe de modules de test qui indique l'emplacement d'un module de test dans la bibliothèque des modules de test. La structure de la bibliothèque des modules de test est indépendante de la structure de la suite de tests. Les références de groupe de modules de test obéissent à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

641 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}

EXEMPLE 4 – Référence de groupe de modules de test de transport:
TRANSPORT/STEP_LIBRARY/CLASS0/CONN_ESTAB/

9.4 Références de groupe de comportements par défaut

Les comportements par défaut (lorsqu'ils existent) sont regroupés dans une bibliothèque de comportements par défaut.

Une référence de groupe d'un comportement par défaut spécifie l'emplacement du comportement par défaut dans la bibliothèque correspondante, qui peut être structurée hiérarchiquement. La structure de cette bibliothèque n'a aucune influence sur celle de la suite de tests elle-même. Les références de groupe de comportements par défaut obéissent à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

651 DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}

EXEMPLE 5 – Référence de groupe de comportements par défaut:
TRANSPORT/DEFAULT_LIBRAR/CLASS0/

9.5 Parties composantes d'une suite de tests TTCN

Une suite ATS écrite en notation TTCN comporte dans l'ordre les quatre sections suivantes:

a) présentation générale de la suite (voir le paragraphe 10),

contenant les informations nécessaires à la présentation générale et à la bonne compréhension de la suite de tests, par exemple les références de test et une description de son objectif global;

b) partie importation (voir 10.7),

qui contient les déclarations des objets utilisés dans la suite de tests ou dans le module de test, qui sont importés d'un objet source;

c) partie déclarative (voir le paragraphe 11),

contenant les définitions ou déclarations de toutes les composantes de la suite de tests (par exemple, des points PCO, des temporisations, des primitives ASP, des unités PDU, et de leurs paramètres ou champs);

d) partie contraintes (voir les paragraphes 12, 13 et 14),

contenant les déclarations des valeurs des primitives ASP, des unités PDU, ainsi que de leurs paramètres utilisés dans la partie dynamique. Ces contraintes sont spécifiées à l'aide:

1) des tables TTCN;

2) de la notation des valeurs en syntaxe ASN.1;

3) des deux éléments ci-dessus à la fois;

e) partie dynamique (voir le paragraphe 15),

comprenant trois sections contenant les tables spécifiant le comportement du test exprimé principalement en termes d'instances de primitives ASP ou d'unités PDU aux points PCO. Ces sections sont les suivantes:

1) descriptions de comportements dynamiques de tests élémentaires;

2) bibliothèque contenant les descriptions de comportements dynamiques de modules de test (s'il y en a);

3) bibliothèque contenant les descriptions de comportements dynamiques par défaut (s'il y en a).

10 Aperçu général de la suite de tests

10.1 Introduction

La partie aperçu général d'une suite de tests a pour objet de fournir les informations nécessaires à la présentation générale et à la bonne compréhension de cette suite de tests. Elle comprend les éléments suivants:

a) structure de la suite de tests (voir 10.2);

b) index des tests élémentaires (voir 10.3);

c) index des modules de test (voir 10.4);

d) index des comportements par défaut (voir 10.5);

e) table d'exportation de suite de tests (voir 10.6).

10.2 Structure de suite de tests

La structure de la suite de tests contient l'identification des documents de référence correspondants, la spécification de la structure de la suite de tests, une brève description de sa finalité générale et des références aux critères de sélection des groupes de tests.

Cette section comporte au moins les informations suivantes:

a) le nom de la suite de tests;

b) les références aux normes de base pertinentes;

c) une référence au formulaire de déclaration PICS;

d) une référence au formulaire partiel d'informations PIXIT (voir 14.1/X.291, ainsi que l'Appendice V/X.296);

e) une indication de la ou des méthodes de test auxquelles cette suite de tests s'applique, plus, dans le cas des méthodes de test coordonnées, une référence à l'endroit où le protocole TMP est spécifié;

f) toute autre information pouvant aider à la compréhension de la suite de tests, par exemple son numéro de version ou son mode de dérivation; cette information doit apparaître sous forme de commentaires;

g) une liste des groupes de tests de la suite de tests (s'il y en a),

   comportant, pour chacun de ces groupes, les informations suivantes:

   1) la référence de groupe des tests,

      dans laquelle le premier identificateur peut être le nom de la suite, et chacun des identificateurs suivants représente l'ordonnancement conceptuel de la suite de tests. Les groupes de tests seront énumérés dans l'ordre d'apparition des tests élémentaires correspondants dans la suite ATS. Ils doivent de plus être classés de telle sorte que tous les groupes d'un groupe simple suivent immédiatement celui-ci. Tous les groupes de tests d'une suite de tests doivent être énumérés;

      les tests élémentaires importés peuvent être inclus dans n'importe quel groupe, indépendamment du groupe dans lequel ils sont définis dans l'objet source d'origine. Un nouveau groupe peut être inclus dans la liste, même s'il n'apparaît pas dans la partie dynamique. Un tel groupe ne doit comprendre que des tests élémentaires importés;

      les groupes de la partie dynamique doivent apparaître dans le même ordre où ils apparaissent dans cette partie, mais la liste peut être précédée, interrompue ou suivie par de nouveaux groupes de tests élémentaires importés. Pour ces nouveaux groupes, le numéro de page ne doit pas être fourni;

      la colonne Référence de sélection peut contenir l'identificateur d'une expression de sélection applicable aux nouveaux groupes de tests. Cette nouvelle expression de sélection doit avoir priorité sur l'expression de sélection spécifiée dans le groupe de tests d'origine (s'il y en a). L'absence d'identificateur d'expression de sélection dans cette colonne signifie que l'expression de sélection spécifiée dans le groupe de tests d'origine (s'il y en a) est omise;

      la colonne Objectif du groupe de tests peut comprendre une nouvelle déclaration informelle de l'objectif du nouveau groupe de tests. Ce nouvel objectif doit avoir priorité sur l'objectif du groupe de tests importé (s'il y en a). L'absence d'objectif du groupe de tests dans cette colonne indique que l'objectif du groupe de tests spécifié est omis;

   2) un identificateur facultatif de l'expression de sélection,

      qui donne comme référence une entrée de la table des définitions des expressions de sélection du test élémentaire, servant à déterminer si les tests élémentaires du groupe s'appliquent à des implémentations sous test particulières. Cette colonne peut contenir l'identificateur d'une expression de sélection applicable au groupe de tests. Si un identificateur d'expression de sélection est indiqué pour un groupe et si l'expression de sélection mentionnée prend la valeur FALSE (faux), aucun test élémentaire de ce groupe n'est sélectionné pour exécution. Si l'expression de sélection prend la valeur TRUE (vrai), la sélection pour exécution de tests élémentaires appartenant à ce groupe dépendra alors de la valeur prise par les expressions de sélection correspondant aux sous-groupes de ce groupe et aux tests élémentaires individuels. L'omission d'un identificateur d'expression de sélection équivaut à la valeur booléenne TRUE;

   3) l'objectif du groupe de tests,

      qui est une déclaration informelle de la finalité du groupe de tests;

   4) un numéro de page,

      indiquant l'emplacement du premier test élémentaire du groupe dans la suite ATS. Le numéro de page accompagnant chaque référence de groupe de tests dans la table de la structure de la suite de tests est celui de la première description comportementale de test élémentaire de ce groupe.

Ces informations doivent être fournies dans le format de Formulaire 1, ci-dessous.

Structure de la suite de tests			
Nom de la suite : *SuiteIdentifier*			
Réf. aux normes : *Free Text*			
Réf. de déclaration PICS : *Free Text*			
Réf. d'informations PIXIT : *Free Text*			
Méthode(s) de test : *FreeText*			
Commentaires : *[FreeText]*			
Référence de groupe de tests	**Référence de sélection**	**Objectif du groupe de tests**	**n° de page**
⋮ *TestGroupReference* ⋮	⋮ *[SelectExprIdentifier]* ⋮	⋮ *FreeText* ⋮	⋮ *Number* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 1 – Structure de la suite de tests

DÉFINITION SYNTAXIQUE:

41 SuiteIdentifier ::= Identifier

626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

202 SelectExprIdentifier ::= Identifier

741 Number ::= (NonZeroNum {Num}) | 0

10.3 Index des tests élémentaires

L'index des tests élémentaires contient la liste complète des tests élémentaires de la suite ATS. Les informations suivantes seront données pour chacun d'eux:

a) une référence facultative de groupe de tests (si la suite ATS est structurée en groupes de tests),

qui définit l'emplacement du test élémentaire dans la structure du groupe de la suite de tests. Si cette référence est omise, le test élémentaire est supposé être situé dans le même groupe de tests que le test élémentaire le précédant dans l'index. Les groupes de tests sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Une référence de groupe de tests explicite sera indiquée pour le premier test élémentaire de chaque groupe. Une telle référence de groupe de tests explicite sera également fournie pour chaque test élémentaire suivant immédiatement le dernier test élémentaire du groupe; ceci est nécessaire si le groupe de tests contient à la fois des groupes de tests et des tests élémentaires;

b) le nom du test élémentaire,

c'est-à-dire l'identificateur indiqué dans la table des comportements dynamiques des tests élémentaires. Les tests élémentaires seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) un identificateur facultatif d'expression de sélection,

qui donne comme référence une entrée de la table des définitions des expressions de sélection du test élémentaire, afin de déterminer si le test élémentaire doit être sélectionné pour exécution. Cette colonne peut contenir l'identificateur d'une expression de sélection applicable au test élémentaire. Si un identificateur d'expression de sélection est indiqué et si l'expression de sélection prend la valeur FALSE (faux), ce test élémentaire n'est pas sélectionné pour exécution. Si l'expression de sélection prend la valeur TRUE (vrai), la sélection pour exécution du test élémentaire dépendra alors de la valeur prise par les expressions de sélection des groupes de tests contenant le test élémentaire. Un test élémentaire est sélectionné si son expression de sélection, ainsi que celle de tous les groupes le contenant, prennent la valeur TRUE (vrai). L'omission d'un identificateur d'expression de sélection équivaut à la valeur booléenne TRUE;

d) une description du test élémentaire,

pouvant être un résumé de l'objectif du test;

e) un numéro de page,

indiquant l'emplacement du test élémentaire dans la suite ATS. Le numéro de page figurant vis-à-vis de chaque identificateur de test élémentaire dans la table d'index des tests élémentaires est celui de la description comportementale qui lui correspond.

Ces informations seront fournies dans le format illustré dans le Formulaire 2, ci-dessous.

Index des tests élémentaires				
Référence du groupe de tests	**Identificateur du test élémentaire**	**Réf. de sélection**	**Description**	**n⁰ de page**
⋮ *TestGroupReference* ⋮	⋮ *TestCaseIdentifier* ⋮	⋮ *[SelectExprIdentifier]* ⋮	⋮ *FreeText* ⋮	⋮ *Number* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 2 – Index des tests élémentaires

Des commentaires collectifs peuvent être utilisés dans cette table, conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

624 TestCaseIdentifier ::= Identifier

202 SelectExprIdentifier ::= Identifier

La liste complète des tests élémentaires doit comprendre les tests élémentaires importés. Les tests élémentaires explicitement définis sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Les numéros de page ne doivent pas être fournis pour les tests élémentaires importés.

La colonne Référence de sélection a une sémantique semblable à celle qui est donnée au sous-paragraphe précédent (voir 10.2).

La colonne Description peut contenir une nouvelle forme abrégée de l'objectif du test. La nouvelle description doit avoir priorité sur celle du test élémentaire importé (s'il y en a). L'absence de description dans cette colonne indique que la description spécifiée est omise.

10.4 Index des modules de test

L'index des modules de test contient la liste complète de tous les modules de test de la suite ATS. Les informations suivantes seront fournies pour chaque module de test:

a) une référence facultative de groupe de modules de test (si la suite ATS est structurée en groupes de modules de test),

qui définit l'emplacement du module de test dans la structure de la bibliothèque des modules de test. Si cette référence est omise, le module de test est supposé classé dans le même groupe que le module de test le précédant dans l'index. Les groupes de modules de test sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Une référence explicite de groupe de modules de test sera fournie pour le premier module de test de chaque groupe. Une telle référence explicite de groupe de modules de test sera également fournie pour chaque module de test suivant immédiatement le dernier module de test du groupe; ceci est nécessaire si un groupe de modules de test contient à la fois des groupes de modules de test et des modules de test;

b) le nom du module de test,

c'est-à-dire l'identificateur fourni dans la table des comportements dynamiques des modules de test. Les modules de test seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) une description du module de test,

pouvant être un résumé de l'objectif du module de test;

d) un numéro de page,

indiquant l'emplacement du module de test dans la suite ATS. Le numéro de page indiqué vis-à-vis de chaque identificateur de module de test dans la table d'index des modules de test est celui de la description comportementale qui lui correspond.

Ces informations seront fournies dans le format illustré dans le Formulaire 3, ci-dessous.

Index des modules de test			
Référence du groupe de modules de test	**Identificateur du module de test**	**Description**	**n° de page**
⋮ *TestStepGroupReference* ⋮	⋮ *TestStepIdentifier* ⋮	⋮ *FreeText* ⋮	⋮ *Number* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 3 – Index des modules de test

Des commentaires collectifs peuvent être utilisés dans cette table, conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

641 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
639 TestStepIdentifier ::= Identifier

La liste complète des tests élémentaires doit comprendre les tests élémentaires importés. Les tests élémentaires explicitement définis sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Les numéros de page ne doivent pas être fournis pour les tests élémentaires importés.

La colonne Description peut contenir une nouvelle forme abrégée de l'objectif du test. La nouvelle description doit avoir priorité sur celle du test élémentaire importé (s'il y en a). L'absence de description dans cette colonne indique que la description spécifiée est omise.

10.5 Index des comportements par défaut

L'index des comportements par défaut contient la liste complète de tous les comportements par défaut de la suite ATS. Les informations suivantes seront fournies pour chaque comportement par défaut:

a) une référence facultative de groupe de comportements par défaut (si la suite ATS est structurée en groupes de comportements par défaut),

qui définit l'emplacement du comportement par défaut dans la structure de la bibliothèque des comportements par défaut. Si cette référence est omise, le comportement par défaut est supposé être situé dans le même groupe que le comportement par défaut le précédant dans l'index. Les comportements par défaut seront énumérés dans l'ordre de leur apparition dans la suite ATS. Une référence explicite de groupe de comportements par défaut sera fournie pour le premier comportement par défaut de chaque groupe et pour chaque comportement par défaut suivant immédiatement le dernier comportement par défaut du groupe;

b) le nom du comportement par défaut,

c'est-à-dire l'identificateur indiqué dans la table des comportements dynamiques par défaut. Les comportements par défaut seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) une description du comportement par défaut,

pouvant être un résumé de l'objectif du comportement par défaut;

d) un numéro de page,

indiquant l'emplacement du comportement par défaut dans la suite ATS. Le numéro de page indiqué vis-à-vis de chaque identificateur de comportement par défaut dans la table d'index des comportements par défaut est celui de la description comportementale qui lui correspond.

Ces informations seront fournies dans le format illustré dans le Formulaire 4, ci-dessous.

Index des comportements par défaut			
Référence de groupe de comportements par défaut	**Identificateur du comportement par défaut**	**Description**	**n° de page**
⋮ *DefaultGroupReference* ⋮	⋮ *DefaultIdentifier* ⋮	⋮ *FreeText* ⋮	⋮ *Number* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 4 – Index des comportements par défaut

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

651 DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}

650 DefaultIdentifier ::= Identifier

La liste complète des comportements par défaut doit comprendre les comportements par défaut importés. Les comportements par défaut explicitement définis sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Les numéros de page ne doivent pas être fournis pour les comportements par défaut importés.

La colonne Description peut contenir une nouvelle forme abrégée de l'objectif du comportement par défaut. La nouvelle description doit avoir priorité sur celle du comportement par défaut importé (s'il y en a). L'absence de description dans cette colonne indique que la description spécifiée est omise.

10.6 Table d'exportation des suites de tests

La table d'exportation des suites de tests peut servir à spécifier explicitement quels objets de la suite de tests sont conçus comme étant réutilisables et peuvent, par conséquent, être importés dans d'autres suites de tests ou modules TTCN.

Le formulaire d'exportation de suite de tests sert à identifier les objets qui peuvent être exportés.

Si un type de point PCO est donné comme objet exporté dans la table d'exportation, il doit être défini dans la table facultative de type de point PCO.

Il convient de donner le nom de l'objet source d'origine si l'objet est lui-même importé.

Si l'objet est déclaré comme étant un objet externe (objet externe explicite) ou est un objet omis dans l'objet source importé (objet externe implicite), le mot clé EXTERNAL (externe) est donné à la place du nom de l'objet source.

L'exportation d'un objet de type Enumeration (énumération) ou Named Number (nombre nommé) exige que l'on donne le type correspondant. Les autres objets qui sont définis dans le type correspondant ne sont pas exportés de la même manière. Néanmoins, ils sont implicitement exportés et on peut y faire référence dans d'autres objets exportés. Le nom du type est donné comme suffixe du nom de l'objet, entre crochets.

Les informations suivantes seront fournies dans la table d'exportation de suites de tests, pour chacun des objets exportés:

a) le nom de l'objet – Si l'objet est du type NamedNumber ou Enumeration, le type correspondant doit être donné en suffixe du nom de l'objet, entre crochets;

b) le type d'objet;

c) le nom de l'objet source d'origine si l'objet est importé, ou la directive d'objet EXTERNAL;

d) un numéro de page,

indiquant l'emplacement de l'objet dans la suite de tests (ne pas donner de numéro de page pour les objets importés);

e) un commentaire facultatif.

Ces informations seront fournies dans le format illustré dans le Formulaire 5, ci-dessous.

Table d'exportation de suite de tests				
Nom de l'objet	**Type d'objet**	**Nom de la source**	**n° de page**	**Commentaires**
⋮ *ObjectIdentifier* ⋮	⋮ *TTCN_ObjectType* ⋮	⋮ *[SourceIdentifier | ObjectDirective]* ⋮	⋮ *Number* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 5 – Table d'exportation de suite de tests

DÉFINITION SYNTAXIQUE:

12 ObjectIdentifier ::= Identifier | ObjectTypeReference

15 TTCN_ObjectType ::= **SimpleType_Object** | **StructType_Object** | **ASN1_Type_Object** | **TS_Op_Object** | **TS_Proc_Object** | **TS_Par_Object** | **SelectExpr_Object** | **TS_Const_Object** | **TS_Var_Object** | **TC_Var_Object** | **PCO_Type_Object** | **PCO_Object** | **CP_Object** | **Timer_Object** | **TComp_Object** | **TCompConfig_Object** | **TTCN_ASP_Type_Object** | **ASN1_ASP_Type_Object** | **TTCN_PDU_Type_Object** | **ASN1_PDU_Type_Object** | **TTCN_CM_Type_Object** | **ASN1_CM_Type_Object** | **EncodingRule_Object** | **EncodingVariation_Object** | **InvalidFieldEncoding_Object** | **Alias_Object** | **StructTypeConstraint_Object** | **ASN1_TypeConstraint_Object** | **TTCN_ASP_Constraint_Object** | **ASN1_ASP_Constraint_Object** | **TTCN_PDU_constraint_Object** | **ASN1_PDU_Constraint_Object** | **TTCN_CM_Constraint_Object** | **ASN1_CM_Constraint_Object** | **TestCase_Object** | **TestStep_Object** | **Default_Object** | **NamedNumber_Object** | **Enumeration_Object**

17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier

18 ObjectDirective ::= Omit | EXTERNAL

EXEMPLE 6 – Table d'exportation de suite de tests:

Table d'exportation de suite de tests				
Nom de l'objet	**Type d'objet**	**Nom de la source**	**n° de page**	**Commentaires**
String5	SimpleTypeDef		3	
wait	TimerDcl	Module_B		
INTC	TTCN_PDU_Type		13	
DEF1	Default	TestSuite_1		
TC_2	TestCase	TestSuite_2		
TC_3	TestCase		33	
Preamble	TestStep	EXTERNAL		
Commentaires détaillés:				

10.7 Partie importation

10.7.1 Introduction

L'objectif de la partie importation est de déclarer les objets utilisés dans la suite de tests qui sont importés d'un objet source. L'effet des importations équivaut à avoir une copie des objets importés dans la suite de tests.

Un objet ne peut être importé que s'il est exporté par un objet source. Une suite de tests qui n'a pas de table d'exportation exporte tous les objets qui ont un nom global. Un module et une suite de tests qui ont au moins une table d'exportation exportent les objets contenus dans les tables d'exportation. Un objet qui n'est pas lui-même explicitement importé est implicitement importé si un objet importé y fait référence.

10.7.2 Table d'importation

La table d'importation identifie l'objet source et fournit des informations sur l'objectif général de l'objet source. Les informations suivantes seront fournies dans la table d'importation:

a) le nom de l'objet source;

b) une description de l'objectif de l'objet source;

c) une référence complète à l'objet source, qui devrait comprendre un identificateur de document et d'autres informations, comme la version et la date;

d) toute autre information qui pourrait aider à comprendre l'objet source; de telles informations devraient être incluses en commentaire;

e) une liste des objets provenant de l'objet source importé; pour chaque objet, il convient de donner les informations suivantes:

   1) le nom de l'objet, tel qu'il est utilisé dans l'objet source;

   2) le type de l'objet, qui doit être le même que le type donné dans l'objet source;

   3) le nom de l'objet source d'origine si l'objet est importé d'un autre objet source, ou la directive d'objet OMIT ou "-" si l'objet doit être omis de l'ensemble d'objets importés de l'objet source, ou encore la directive d'objet EXTERNAL si l'objet est déclaré comme étant externe dans l'objet source.

Ces informations seront fournies dans le format illustré dans le Formulaire 6, ci-dessous.

Table d'importation			
Nom de la source	**:** *SourceIdentifier*		
Groupe	**:** *[ImportsGroupReference]*		
Réf. aux normes	**:** *[FreeText]*		
Commentaires	**:** *[FreeText]*		
Nom de l'objet	**Type d'objet**	**Nom de la source**	**Commentaires**
⋮ *ObjectIdentifier* ⋮	⋮ *TTCN_ObjectType* ⋮	⋮ *[SourceIdentifier | ObjectDirective]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés:	*[FreeText]*		

Formulaire 6 – Table d'importation

DÉFINITION SYNTAXIQUE:

17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier

34 ImportsGroupReference ::=[(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ImportsGroupIdentifier "/"}

12 ObjectIdentifier ::= Identifier | ObjectTypeReference

15 TTCN_ObjectType ::= **SimpleType_Object** | **StructType_Object** | **ASN1_Type_Object** | **TS_Op_Object** | **TS_Proc_Object** | **TS_Par_Object** | **SelectExpr_Object** | **TS_Const_Object** | **TS_Var_Object** | **TC_Var_Object** | **PCO_Type_Object** | **PCO_Object** | **CP_Object** | **Timer_Object** | **TComp_Object** | **TCompConfig_Object** | **TTCN_ASP_Type_Object** | **ASN1_ASP_Type_Object** | **TTCN_PDU_Type_Object** | **ASN1_PDU_Type_Object** | **TTCN_CM_Type_Object** | **ASN1_CM_Type_Object** | **EncodingRule_Object** | **EncodingVariation_Object** | **InvalidFieldEncoding_Object** | **Alias_Object** | **StructTypeConstraint_Object** | **ASN1_TypeConstraint_Object** | **TTCN_ASP_Constraint_Object** | **ASN1_ASP_Constraint_Object** | **TTCN_PDU_constraint_Object** | **ASN1_PDU_Constraint_Object** | **TTCN_CM_Constraint_Object** | **ASN1_CM_Constraint_Object** | **TestCase_Object** | **TestStep_Object** | **Default_Object** | **NamedNumber_Object**| **Enumeration_Object**

18 **ObjectDirective ::= Omit | EXTERNAL**

EXEMPLE 7 – Une table d'importation:

Table d'importation

Nom du test élémentaire : Module A
Référence de la source : {ISO standard 1234}
Référence aux normes : ISO 300 313
Commentaires : Suite de tests pour la Couche 2

Nom de l'objet	Type d'objet	Nom de la source	Commentaires
String5	SimpleTypeDef		
Wait	TimeDcl	Module B	1)
R1_POSTAMBLE	TestStep	EXTERNAL	2)
TSAP	PCO_TypeDcl		3)
blue[ColorEnum]	Enumeration	OMIT	
a[NN_type1]	NamedNumber		4)

Commentaires détaillés:

1) la source d'origine est le Module B;
2) ce module de test est déclaré comme étant externe dans le Module A et doit être explicitement défini ou être importé à l'endroit où ce module est utilisé;
3) le point TSAP doit être défini dans la table de déclaration de type de point PCO;
4) ce nombre nommé est omis de la table d'importation et, par conséquent, devrait être explicitement redéfini dans la suite de tests.

11 Partie déclarative

11.1 Introduction

La partie déclarative d'une suite ATS a pour objet de définir et de déclarer tous les objets utilisés dans cette suite de tests. Les objets suivants d'une suite ATS cités en référence dans la partie présentation générale, la partie contraintes et la partie dynamique, devront avoir été déclarés dans la partie déclarative:

a) *définitions:*
   1) types de suites de tests (voir 11.2.3);
   2) opérations des suites de tests (voir 11.3.4);

b) *paramétrage et sélection des tests élémentaires:*
   1) paramètres de suites de tests (voir 11.4);
   2) expressions de sélection des tests élémentaires (voir 11.5);

c) *déclarations/définitions:*
   1) constantes de suites de tests (voir 11.6 et 11.7);
   2) variables de suites de tests (voir 11.8.1);
   3) variables de test élémentaire (voir 11.8.3);
   4) types de points PCO (voir 11.9);
   5) points de contrôle et d'observation (voir 11.10);
   6) points de coordination (CP) (voir 11.11);
   7) temporisateurs (voir 11.12);
   8) composantes de test (voir 11.13.1);
   9) configurations de composantes de test (voir 11.13.2);
   10) types de primitives ASP (voir 11.14);
   11) types d'unités PDU (voir 11.15);

12) règles de codage (voir 11.16.1);

13) variations de codage (voir 11.16.2);

14) codage de champ non valide (voir 11.16.3);

15) types de messages de coordination (CM) (voir 11.17);

16) alias (pseudonymes) (voir 11.21).

11.2 Types TTCN

11.2.1 Introduction

La notation TTCN prend en charge un certain nombre de types et de mécanismes prédéfinis qui permettent de définir des types spécifiques de suites de tests. Ces types peuvent être utilisés dans toute la suite de tests et référencés lors de la déclaration des paramètres, des constantes et des variables de suites de tests, ainsi que des paramètres de primitives ASP, des champs d'unités PDU, etc.

La notation TTCN est un langage à typage faible en ce sens que les valeurs de deux types quelconques ayant le même type de base sont considérées comme étant de type compatible (par exemple aux fins d'affectation ou de transfert de paramètres).

11.2.2 Types TTCN prédéfinis

Un certain nombre de types couramment employés sont prédéfinis pour être utilisés en notation TTCN. Tous les types définis en notation ASN.1 et dans le présent paragraphe peuvent être référencés, même s'ils n'apparaissent pas dans une définition de type d'une suite de tests. Tous les autres types utilisés dans une suite de tests devront être déclarés dans les définitions des types, des primitives ASP et des unités PDU de la suite de tests. Ils seront référencés par leur nom.

Les types TTCN prédéfinis indiqués ci-dessous sont considérés comme identiques à leurs homologues en notation ASN.1:

a) **type prédéfini INTEGER** (entier): type dont les valeurs distinctives sont les entiers relatifs (positifs, négatifs ou nuls).

   Les valeurs du type INTEGER sont représentées par un ou plusieurs chiffres, le premier étant différent de zéro, sauf si la valeur est nulle; la valeur nulle est représentée par un seul zéro;

b) **type prédéfini BOOLEAN** (booléen): type à deux valeurs distinctives.

   Les valeurs de type booléen (BOOLEAN) sont TRUE (vrai) et FALSE (faux);

c) **type prédéfini BITSTRING** (chaîne binaire): type dont les valeurs distinctives sont des séquences ordonnées de zéro, un ou plusieurs bits.

   Les valeurs du type BITSTRING sont représentées par un nombre arbitraire (éventuellement zéro) de chiffres binaires (0 et 1), précédés d'une apostrophe ' et suivis des deux caractères 'B:

   EXEMPLE 8 – '01101'B

d) **type prédéfini HEXSTRING** (chaîne hexadécimale): type dont les valeurs distinctives sont des séquences ordonnées de zéro, un ou plusieurs chiffres hexadécimaux, chaque chiffre correspondant à une séquence ordonnée de quatre bits.

   Les valeurs du type HEXSTRING sont représentées par un nombre arbitraire (éventuellement zéro) de chiffres hexadécimaux:

   0 1 2 3 4 5 6 7 8 9 A B C D E F

   précédés d'une apostrophe ' et suivis des deux caractères 'H; chaque chiffre hexadécimal (type HEX) est utilisé pour dénoter la valeur d'un demi-octet en utilisant une représentaion hexadécimale:

   EXEMPLE 9 – 'AB01D'H

e) **type prédéfini OCTETSTRING** (chaîne d'octets): type dont les valeurs distinctives sont des séquences ordonnées comprenant zéro ou un nombre pair positif de chiffres hexadécimaux (chaque paire de chiffres correspondant à une séquence ordonnée de huit bits).

Les valeurs du type OCTETSTRING sont représentées par un nombre pair arbitraire (éventuellement nul) de chiffres hexadécimaux:

0 1 2 3 4 5 6 7 8 9 A B C D E F

précédés d'une apostrophe ' et suivis des deux caractères 'O, chaque chiffre hexadécimal (type HEX) représentant un demi-octet:

EXEMPLE 10 – 'FF96'O

f) **type prédéfini OBJECTIDENTIFIER** (identificateur d'objet): type dont les valeurs distinctives sont l'ensemble de tous les identificateurs d'objet affectés conformément aux règles mentionnées dans la Recommandation X.680;

g) **type prédéfini R_TYPE** (type R): type dont les valeurs distinctives sont les suivantes:

pass (succès), fail (échec), inconc (non concluant) et none (aucun).

Ces valeurs sont des identificateurs prédéfinis et, comme tels, sont sensibles aux majuscules et aux minuscules. Ce type prédéfini s'utilise avec les verdicts, voir 15.17;

h) **type prédéfini CharacterString** (chaîne de caractères): type dont les valeurs distinctives sont des séquences de zéro, un ou plusieurs caractères tirés d'un jeu de caractères donné; les types CharacterString énumérés dans le Tableau 2 peuvent être utilisés; ils sont définis au paragraphe 31/X.680.

Tableau 2/X.292 – Type prédéfini CharacterString

NumericString
PrintableString
TeletexString
T61String
VideotexString
VisibleString
ISO646String
IA5String
GraphicString
GeneralString
BMPString
UniversalString

Les valeurs des types CharacterString sont représentées par un nombre arbitraire (éventuellement nul) de caractères tirés du jeu de caractères indiqué par le type CharacterString, précédés et suivis de guillemets ("); un guillemet faisant partie de la chaîne de caractères sera dénoté par un double guillemet ("").

DÉFINITION SYNTAXIQUE:

735 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | **OBJECTIDENTIFIER** | **R_Type** | CharacterString

736 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString** | **T61String** | **ISO646String**

741 **Number ::= (NonZeroNum {Num}) | 0**

742 **NonZeroNum ::= 1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

743 **Num ::= 0 | NonZeroNum**

744 **BooleanValue ::= TRUE** | **FALSE**

745 **Bstring ::= "_" {Bin | Wildcard} "_" B**

746 **Bin ::= 0** | **1**

747 **Hstring ::= "_" {Hex | Wildcard} "_" H**

748 **Hex ::= Num | A** | **B** | **C** | **D** | **E** | **F**

749 **Ostring ::= "_" {Oct | Wildcard} "_" O**

750 **Oct ::= Hex Hex**

751 **Cstring ::= """ {Char | Wildcard | "\"} """**

752 Char ::= /* REFERENCE – caractère défini par le type chaîne de caractères approprié. */

753 Wildcard ::= AnyOne | AnyOrNone

754 AnyOne ::= "?"

755 AnyOrNone ::= ""*

11.2.3 Définitions des types de suites de tests

11.2.3.1 Introduction

Les définitions des types devant servir de types d'objets de données et de sous-types de primitives ASP et d'unités PDU structurées, etc., peuvent être présentées en format tabulaire ou en notation ASN.1. La récursivité directe ou indirecte n'est pas admise dans les définitions de types de suites de tests lorsque des types y sont cités en référence.

11.2.3.2 Définition d'un type simple à l'aide de tables

Pour définir un nouveau type simple, les informations suivantes seront fournies:

a) un nom pour le type considéré;

b) le type de base,

qui doit être un type prédéfini ou un type simple. Ce type de base est suivi par la restriction de type, qui prend l'une des formes suivantes:

1) une liste des valeurs distinctives du type de base; ces valeurs comprennent le nouveau type;

2) la spécification d'un intervalle de valeurs de type INTEGER; le nouveau type englobe les valeurs de cet ensemble, limites comprises. Pour spécifier un intervalle infini, on peut utiliser le mot clé INFINITY (infini) à la place d'une valeur, pour indiquer qu'il n'y a pas de limite supérieure ou inférieure;

3) la spécification d'une longueur particulière ou d'un intervalle de longueurs pour un type de chaîne prédéfini ou de chaîne de suite de tests; cette ou ces valeurs de longueur seront interprétées selon le Tableau 5, en 11.18.2; pour la limite supérieure, seuls seront utilisés des entiers INTEGER non négatifs ou le mot clé INFINITY;

c) facultativement, un identificateur de codage spécifique, suivi par toute liste de paramètres effectifs nécessaires, afin de spécifier un codage explicite pour le type simple, qui a priorité sur les règles de codage et les variations de codage applicables à une unité PDU quelconque dans laquelle le type simple est utilisé; l'identificateur de codage, s'il y en a, doit identifier soit l'une des variations de codage, soit une définition de codage de champ non valide précisée dans la suite de tests [par exemple LD(10)]; voir 11.16.4.

Ces informations seront fournies dans le format illustré dans le Formulaire 7, ci-dessous.

Définitions de type simple			
Groupe : *[SimpleTypeGroupReference]*			
Nom du type	**Définition du type**	**Codage du type**	**Commentaires**
⋮ *SimpleTypeIdentifier* ⋮	⋮ *Type&Restriction* ⋮	⋮ *[PDU_FieldEncodingCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 7 – Définitions de type simple

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

```
75  SimpleTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {SimpleTypeGroupIdentifier "/"}
79  SimpleTypeIdentifier ::= Identifier
80  Type&Restriction ::= Type[Restriction]
515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall
```

Lorsqu'un type est défini sous la forme d'un intervalle de valeurs ou de longueurs (pour des chaînes), la limite inférieure sera celle de gauche. Un intervalle de valeurs entières ne sera utilisé que pour le type de base INTEGER ou pour un type dérivé de INTEGER. Dans ce dernier cas, l'intervalle de valeurs entières sera un sous-intervalle de l'ensemble des valeurs définies par le type de base.

Lorsqu'un type est défini par une liste de valeurs, celles-ci doivent appartenir au type de base et former un sous-ensemble strict des valeurs du type de base. Lorsqu'un type est défini par restriction de longueur, l'ensemble de ce type doit former un sous-ensemble strict des valeurs du type de base.

Les valeurs de deux types simples quelconques ayant le même type de base sont considérées comme étant de type compatible (par exemple aux fins d'affectation ou de transfert de paramètres).

EXEMPLE 11 – Définition du type simple de suite de tests:

Définition du type simple		
Nom du type	**Définition du type**	**Commentaires**
Transport_classes	INTEGER(0, 1, 2, 3, 4)	Classes pouvant servir à la liaison de couche Transport
String5	IA5string[5]	Chaîne de longueur 5
SeqNumbers	INTEGER(0,127)	Tous les nombres de 0 à 127
PositiveNumbers	INTEGER(1...INFINITY)	Tous les ENTIERS positifs
String 10 to 20	IA5String[10..20]	Chaîne de longueur minimale (10 caractères) et de longueur maximale (20 caractères)

11.2.3.3 Définition des types structurés à l'aide de tables

Les types structurés peuvent être définis sous forme de tables utilisables pour la déclaration d'objets structurés comme les sous-types dans des définitions de primitives ASP et d'unités PDU ou dans d'autres types structurés, etc.

Pour chaque type structuré, les informations suivantes seront fournies:

a) son nom,

   si nécessaire, il s'agira du nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; lorsqu'une abréviation est utilisée, le nom complet devra suivre entre parenthèses;

b) les variations de codage à utiliser pour les structures de ce type dans une unité PDU,

   afin de spécifier des variations de codage explicites pour l'ensemble des types structurés, qui ont priorité sur les variations de codage à n'importe quelle unité PDU dans laquelle ce type structuré est utilisé, cette entrée facultative doit faire référence à une entrée dans la table des variations de codage pertinente [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage applicables sont celles qui s'appliquent à chaque unité PDU dans laquelle ce type structuré est utilisé. Voir 11.16.4;

c) la liste des éléments associés à ce type structuré,

   comportant, pour chacun de ces éléments, les informations suivantes:

   1) son nom,

      c'est-à-dire le nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; lorsqu'on utilise une abréviation, le nom complet devra suivre entre parenthèses;

   2) son type et un attribut facultatif,

      ces éléments pouvant appartenir à un type structuré arbitrairement complexe; les références récursives (directes ou indirectes) ne sont pas admises;

      il est possible d'utiliser l'élément facultatif de restriction de longueur pour indiquer les longueurs minimale et maximale d'un élément de type chaîne (voir 11.18);

   3) facultativement, un identificateur de codage spécifique, suivi par toute liste de paramètres effectifs nécessaires, afin de spécifier un codage explicite pour le type structuré, qui a priorité sur les règles de codage et les variations de codage applicables à une unité PDU quelconque dans laquelle le type structuré est utilisé; l'identificateur de codage, s'il y en a, doit identifier soit l'une des variations de codage, soit une définition de codage de champ non valide précisée dans la suite de tests [par exemple LD(10)]; voir 11.16.4.

Les éléments des définitions des types structurés sont considérés comme facultatifs, c'est-à-dire que dans certaines instances de ces types, ils peuvent ne pas être tous présents.

Ces informations seront fournies dans le format illustré dans le Formulaire 8, ci-dessous.

Définition d'un type structuré			
Nom du type : *StructId&FullId*			
Groupe : *[StructTypeGroupReference]*			
Variation de codage : *[EncVariationCall]*			
Commentaires : *[FreeText]*			
Nom de l'élément	**Définition du type**	**Codage du champ**	**Commentaires**
⋮ *ElemId&FullId* ⋮	⋮ *Type&Attributes* ⋮	⋮ *[PDU_FieldEncodingCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 8 – Définition d'un type structuré

DÉFINITION SYNTAXIQUE:

97 StructId&FullId ::= StructIdentifier [fullIdentifier]
101 StructTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {StructTypeGroupIdentifier "/"}
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]
106 ElemId&FullId ::= ElemIdentifier [FullIdentifier]
396 Type&Attributes ::= (Type [LengthAttribute]) | PDU
515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall

11.2.3.4 Définition de types de suites de tests à l'aide de la notation ASN.1

Il est possible de spécifier des types de suites de tests à l'aide de la notation ASN.1, en les définissant selon la syntaxe ASN.1 indiquée dans la Recommandation X.680 (1994). Les informations suivantes seront fournies pour chaque type ASN.1:

a) son nom,

   c'est-à-dire le nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, le nom complet suivra entre parenthèses;

b) les variations de codage à utiliser pour les structures de ce type dans une unité PDU,

   afin de spécifier des variations de codage explicites pour l'ensemble des types ASN1_Types, qui ont priorité sur les variations de codage applicables à n'importe quelle unité PDU dans laquelle ce type ASN1_Type est utilisé, cette entrée facultative doit faire référence à une entrée de la table des variations de codage pertinente [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage applicables sont celles qui s'appliquent à chaque unité PDU dans laquelle ce type ASN1_Type est utilisé. Voir 11.16.4;

c) la définition du type ASN.1,

   qui respectera la syntaxe définie dans la Recommandation X.680, sauf qu'il existe une option supplémentaire permettant de spécifier une variation de codage ou un codage de champ non valide en association soit avec la totalité du type ASN.1 (ASN1_Type) ou avec tout type ASN.1 à l'intérieur du type général ASN.1 (ASN1_Type). A cet effet, on donne un identificateur de codage spécifique suivi de toute liste de paramètres effectifs nécessaires, afin de spécifier un codage explicite pour les champs individuels ou d'autres sous-types d'une unité PDU, qui a priorité sur les règles de codage et les variations de codage applicables à l'unité PDU dans son ensemble; l'identificateur de codage, s'il y en a, doit identifier soit l'une des variations de codage, soit une définition de codage de champ non valide définie dans la suite de tests [par exemple LD(10)]; voir 11.16.4.

Le trait d'union (-) ne devra pas être utilisé dans les identificateurs de cette définition; il est possible d'utiliser à la place le caractère de soulignement (_). L'identificateur de type indiqué dans l'en-tête de la table est le nom du premier type défini dans le corps de la table.

Les types cités en référence dans la définition de type doivent être définis dans d'autres tables de définition de types ASN.1, par référence dans la table de références de types ASN.1 ou être définis localement dans la même table, après la première définition de type. Les types définis localement ne seront pas utilisés dans d'autres parties de la suite de tests.

Les définitions de types ASN.1 utilisées en notation TTCN ne feront pas référence à des types externes définis dans la Recommandation X.680. Des commentaires en notation ASN.1 peuvent figurer dans le corps de la table. La colonne "commentaires" n'existe pas dans cette table.

Les commentaires en ASN.1 commencent par "--" et finissent soit par la prochaine occurrence de "--" ou par l'indication "fin de ligne", suivant ce qui se produit en premier. Cela évite qu'un simple commentaire ASN.1 s'étale sur plusieurs lignes. L'indication "fin de ligne" n'est toutefois pas un symbole défini dans la notation TTCN.MP. Il est par conséquent recommandé aux concepteurs de suites ATS de faciliter l'échange de suites ATS en notation TTCN.MP en terminant toujours les commentaires en notation ASN.1 par "--".

Ces informations seront fournies dans le format de Formulaire 9, ci-dessous.

Définition de type en notation ASN.1	
Nom du type	**:** *ASN1_TypeId&FullId*
Groupe	**:** *[ASN1_TypeGroupReference]*
Variation de codage	**:** *[EncVariationCall]*
Commentaires	**:** *[FreeText]*
Définition du type	
⋮ *ASN1_Type&LocalTypes* ⋮	
Commentaires détaillés:	*[FreeText]*

Formulaire 9 – Définition de type en notation ASN.1

DÉFINITION SYNTAXIQUE:

115 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]

118 ASN1_TypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_TypeGroupIdentifier "/"}

511 Enc VariationCall ::= EncVariationIdentifier [ActualParList]

121 ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}

EXEMPLE 12 – Définition de type de suite de tests en notation ASN.1:

Définition de type en notation ASN.1	
Nom du type	**:** DATE_type
Commentaires	**:** afin d'illustrer la structure des définitions de types en notation ASN.1
Définition du type	

SEQUENCE {
jour DAY_type,
mois MONTH_type,
année YEAR_type
}
-- *local DAY_type*
DAY_type ::= INTEGER {first(), last(31)}
-- *Les types MONTH_type et YEAR_type sont définis dans d'autres tables de définition de type ASN.1.*	

11.2.3.5 Définition de types ASN.1 par référence

Il est possible de spécifier des types par une référence précise à un type ASN.1 défini dans une Recommandation OSI ou par référence à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les informations suivantes seront fournies pour chaque type:

a) son nom,

   qui pourra être utilisé dans toute la suite de tests; il sera spécifié sans identificateur complet FullIdentifier;

b) la référence de type,

   qui respectera les règles énoncées dans la Recommandation X.680 pour les identificateurs;

c) l'identificateur du module,

   composé d'une référence à un module respectant les règles énoncées dans la Recommandation X.680 pour les identificateurs et d'un identificateur d'objet ObjectIdentifier facultatif; ce module sera unique dans le domaine considéré;

d) les variations de codage à utiliser pour ces types ASN1_Types dans une unité PDU,

   afin de spécifier des variations de codage explicites pour l'ensemble des types ASN.1 (ASN1_Types), qui ont priorité sur les variations de codage applicables à toute unité PDU dans laquelle ce type ASN.1 (ASN1_Type) est utilisé, cette entrée facultative doit faire référence à une entrée de la table de variations de codage appropriée [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage applicables sont celles qui s'appliquent à chaque unité PDU dans laquelle ce type ASN1_Type est utilisé. Voir 11.16.4.

Ces informations seront fournies dans le Formulaire 10, ci-dessous.

Définitions de types ASN.1 par référence				
Groupe : *[ASN1_TypeGroupReference]*				
Nom du type	**Référence du type**	**Identificateur du module**	**Variation de codage**	**Commentaires**
⋮ *ASN1_TypeId&FullId* ⋮	⋮ *TypeReference* ⋮	⋮ *ASN1_ModuleIdentifier* ⋮	⋮ *[EncVariationCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 10 – Définitions de types ASN.1 par référence

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

118 ASN1_TypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_TypeGroupIdentifier "/"}
115 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
131 TypeReference ::= typereference
133 ASN1_ModuleIdentifier ::= ModuleIdentifier
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]

Les types importés de modules ASN.1 pouvant contenir des identificateurs, des références de types et des références de valeur respectant les règles énoncées dans la Recommandation X.680 pour les identificateurs, peuvent donc comprendre des traits d'union. Pour pouvoir utiliser en notation TTCN les définitions importées, il est donc nécessaire de transformer les traits d'union contenus dans les identificateurs importés en caractères de soulignement. Cette modification s'effectuera dans le processus d'importation.

EXEMPLE 13 – La définition suivante de type contenue dans un module ASN.1:

```
module-1 DEFINITIONS BEGIN
    Type-1 ::= SEQUENCE        {   field1    Sub-Type-1,
                                   field2    BIT STRING {first-bit(0), second-bit(1) } }
END
```

peut être importée en TTCN avec:

Définition des types ASN.1 par référence			
Nom du type	**Référence du type**	**Identificateur du module**	**Commentaires**
Type_1 Sub_Type_1	Type-1 Sub-Type-1	module-1 module-1	

La définition par référence du Type-1, indiquée ci-dessus, équivaut à la définition suivante:

Définition de type en notation ASN.1		
Nom du type	**:**	Type_1
Commentaires	**:**	
Définition du type		
SEQUENCE {	field	Sub_Type_1,
	Field	BIT STRING {first_bit(0), second_bit(1) } }

11.3 Opérateurs et opérations en notation TTCN

11.3.1 Introduction

La notation TTCN prend en charge un certain nombre d'opérateurs, d'opérations et de mécanismes prédéfinis permettant de définir des opérations de suite de tests. Ces opérateurs et opérations peuvent être utilisés dans toutes les descriptions et contraintes de comportement dynamique.

11.3.2 Opérateurs en notation TTCN

11.3.2.1 Introduction

Les opérateurs prédéfinis sont classés en trois catégories:

a) arithmétiques;

b) relationnels;

c) booléens.

Les règles de préséance entre ces opérateurs sont indiquées au Tableau 3. Les parenthèses servent à regrouper des opérandes en expressions, une expression entre parenthèses étant évaluée prioritairement.

Les opérateurs indiqués sur une même ligne du Tableau 3 ont la même préséance. Lorsque plusieurs opérateurs de même préséance apparaissent, ils sont évalués de gauche à droite.

Tableau 3/X.292 – Préséance entre les opérateurs

maximale ↓ minimale	Unaires	( ) + – NOT
	Binaires	* / MOD AND + – OR = < > <> >= <=

DÉFINITION SYNTAXIQUE:

```
723  AddOp ::= "+"|"–"| OR
724  MultiplyOp ::= "*"|"/"| MOD | AND
725  UnaryOp  ::= "+"|"–"| NOT
726  RelOp ::= "="|"<"|">"|"<>"|">="|"<="
```

11.3.2.2 Opérateurs arithmétiques prédéfinis

Les opérateurs arithmétiques prédéfinis sont:

"+", "–", "*", "/", **MOD**

Ils représentent les opérations d'addition, de soustraction, de multiplication, de division et de modulo. Leurs opérandes sont du type INTEGER (entiers) (types TTCN et ASN.1 prédéfinis) ou d'un type dérivé d'INTEGER (un sous-ensemble des entiers). Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans des expressions arithmétiques.

Le résultat des opérations arithmétiques est du type INTEGER.

Les mêmes règles s'appliquent également aux opérandes des opérateurs unaires (monadiques) + et –. L'opérateur "–" inverse le signe de l'opérande.

La division (/) d'un entier INTEGER par un autre donne pour résultat un entier, la partie fractionnaire étant ignorée.

L'opération MOD entre deux entiers a pour résultat le reste de la division du premier entier par le second.

11.3.2.3 Opérateurs relationnels prédéfinis

Les opérateurs relationnels prédéfinis sont:

"=" | "<" | ">" | "<>" | ">=" | "<="

Ils représentent les relations d'égalité, d'infériorité, de supériorité, d'inégalité, de supériorité ou égalité et d'infériorité ou égalité. Les opérandes d'égalité (=) et d'inégalité (<>) peuvent être d'un type arbitraire. Les deux opérandes doivent être compatibles. Tous les autres opérateurs relationnels auront des opérandes de type entier (INTEGER) ou dérivé. Ces opérations ont un résultat de type booléen (BOOLEAN).

Dans les comparaisons de chaînes, les chaînes binaires BITSTRING, hexadécimales HEXSTRING, d'octets OCTETSTRING et toutes les chaînes de caractères CharacterString peuvent comporter les caractères génériques (*) (AnyOrNone = zéro ou un caractère quelconque) et (?) (AnyOne = un caractère quelconque). Dans ce cas, la comparaison s'effectue conformément aux règles de concordance de modèle définies en 12.6.2.

11.3.2.4 Opérateurs booléens prédéfinis

Les opérateurs booléens prédéfinis sont:

NOT AND OR (NON ET OU)

Ils représentent les opérations de négation, d'intersection logique et de réunion logique. Leurs opérandes sont de type booléen (TTCN ou ASN.1 ou prédéfinis). Leur résultat est de type booléen. L'opérateur logique ET prend la valeur TRUE (vrai) si ses deux opérandes ont la valeur "vrai", la valeur FALSE (faux) dans les autres cas. L'opérateur logique OU prend la valeur "vrai" si l'un au moins de ses opérandes a la valeur "vrai", la valeur "faux" si et seulement si ses deux opérandes ont la valeur "faux". Le NON logique est l'opérateur unaire qui prend la valeur "vrai" si son opérande a la valeur "faux", et la valeur "faux" si son opérande a la valeur "vrai".

11.3.3 Opérations prédéfinies

11.3.3.1 Introduction

11.3.3.1.1 Les opérations prédéfinies se classent en deux catégories:

a) opérations de conversion;

b) autres opérations.

11.3.3.1.2 Les opérations prédéfinies peuvent être utilisées dans toute suite de tests. Elles n'ont pas besoin d'être explicitement définies par une table de définition d'opérations de suite de tests. Lorsqu'une opération prédéfinie est appelée:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur correspondant au type du paramètre formel;

c) toutes les variables apparaissant dans la liste des paramètres seront liées.

Chacune des opérations prédéfinies se présente sous la forme suivante:

OPERATION_NAME (FORMAL_PARAMETER_LIST) ➔ RESULT_TYPE

11.3.3.2 Opérations prédéfinies de conversion

11.3.3.2.1 La notation TTCN prend en charge les opérations prédéfinies de conversion de types suivantes:

a) HEX_TO_INT convertit une chaîne hexadécimale en un entier (HEXSTRING en INTEGER);

b) BIT_TO_INT convertit une chaîne binaire en un entier (BITSTRING en INTEGER);

c) INT_TO_HEX convertit un entier en une chaîne hexadécimale (INTEGER en HEXSTRING);

d) INT_TO_BIT convertit un entier en une chaîne binaire (INTEGER en BITSTRING).

Les règles de codage indiquées ici ne s'appliquent que dans le contexte de ces opérations. Il serait erroné de supposer que ces règles s'appliquent en dehors du domaine de ces opérations en notation TTCN.

11.3.3.2.2 HEX_TO_INT(hexvalue:HEXSTRING) ➔ INTEGER

Cette opération convertit une valeur hexadécimale en un entier.

A cette fin, la chaîne hexadécimale est interprétée comme un entier positif en base 16, le poids des chiffres allant décroissant de gauche à droite. Les chiffres hexadécimaux de 0 à F représentent respectivement les valeurs décimales de 0 à 15.

11.3.3.2.3 BIT_TO_INT(bitvalue:BITSTRING) ➔ INTEGER

Cette opération convertit une chaîne binaire en un entier.

A cette fin, la chaîne binaire est interprétée comme un entier positif en base 2, le poids des chiffres allant décroissant de gauche à droite. Les bits 0 et 1 représentent respectivement les valeurs décimales 0 et 1.

11.3.3.2.4 INT_TO_HEX(intvalue, slength:INTEGER) ➔ HEXSTRING

Cette opération convertit un entier simple en une chaîne hexadécimale. La longueur de la chaîne résultante est de *slength* chiffres hexadécimaux.

A cette fin, une chaîne HEXSTRING est interprétée comme un entier positif en base 16, le poids des chiffres décroissant de gauche à droite. Les chiffres hexadécimaux de 0 à F représentent respectivement les valeurs décimales de 0 à 15.

Si la valeur résultante comporte moins de chiffres hexadécimaux que le nombre spécifié par le second paramètre, la chaîne HEXSTRING est complétée à gauche par des zéros.

Une erreur de test élémentaire intervient si l'entier *intvalue* est négatif ou si le résultat hexadécimal comporte plus de chiffres que ne le permet le second paramètre.

11.3.3.2.5 INT_TO_BIT(intvalue, slength:INTEGER) ➔ BITSTRING

Cette opération convertit un entier en une chaîne binaire. La longueur de la chaîne résultante est de *slength* chiffres binaires.

A cette fin, une chaîne BITSTRING est interprétée comme un entier positif en base 2, le poids des chiffres décroissant de gauche à droite. Les bits 0 et 1 représentent respectivement les valeurs décimales 0 et 1.

Si la valeur résultante comporte moins de bits que le nombre spécifié par le second paramètre, la chaîne BITSTRING est complétée à gauche par des zéros.

Une erreur de test élémentaire intervient si l'entier *intvalue* est négatif ou si le résultat binaire comporte plus de bits que ne le permet le second paramètre.

11.3.3.3 Autres opérations prédéfinies

La notation TTCN comprend également les opérations prédéfinies suivantes:

a) IS_PRESENT (est présent);

b) IS_CHOSEN (est choisi);

c) NUMBER_OF_ELEMENTS (nombre d'éléments);

d) LENGTH_OF (longueur de);

e) SIZE_OF (taille de).

11.3.3.3.1 IS_PRESENT(DataObjectReference) ➔ BOOLEAN

Cette opération ne prend comme argument une référence à un champ dans un objet de données que si ce champ est défini comme facultatif (OPTIONAL) ou s'il a une valeur par défaut (DEFAULT). Le champ peut être d'un type quelconque. Cette opération a pour résultat la valeur booléenne vrai (TRUE) si et seulement si la valeur du champ est présente dans l'instance effective de l'objet de données. Dans les autres cas, le résultat a la valeur "faux" (FALSE).

L'argument de cette opération aura le format défini en 15.10.2.

EXEMPLE 14 – Utilisation de IS_PRESENT:

```
Si l'unité received_PDU est de type ASN.1
    SEQUENCE    {    field_1 INTEGER OPTIONAL,
                     field_2 SEQUENCE OF INTEGER        }
Alors l'opération
    IS_PRESENT(received_PDU.field_1)
```
prend la valeur "vrai" (TRUE) si le champ field_1 de l'instance effective de l'unité received_PDU est présent.

11.3.3.3.2 IS_CHOSEN(DataObjectReference) ➔ BOOLEAN

Cette opération renvoie la valeur booléenne "vrai" (TRUE) si et seulement si la référence de l'objet de données spécifie la variante du type CHOICE effectivement sélectionnée pour un objet de données particulier. Dans les autres cas, elle a pour résultat la valeur "faux" (FALSE). Cette opération ne s'applique pas à des objets de données ou à des champs d'objets de données d'un type autre que le type ASN.1 CHOICE. L'argument de cette opération aura le format défini en 15.10.2.

EXEMPLE 15 – Utilisation de IS_CHOSEN:

```
Si l'unité received_PDU est de type ASN.1
    CHOICE  {        p1 PDU_type1,
                     p2 PDU_type2,
                     p3 PDU_type }
Alors l'opération
    IS_CHOSEN(received_PDU.p2)
```
renvoie la valeur "vrai" (TRUE) si l'instance effective de l'unité received_PDU achemine une unité du type PDU_type2.

11.3.3.3.3 NUMBER_OF_ELEMENTS(Value) ➔ INTEGER

Cette opération renvoie le nombre effectif d'éléments d'une valeur de type ASN.1 SEQUENCE OF ou SET OF. Son résultat est pleinement compatible avec celui de la contrainte équivalente ASN.1 SIZE (taille ASN.1) appliquée aux objets de ces types. Elle ne s'applique pas aux valeurs d'un type autre que les types ASN.1 SEQUENCE OF ou SET OF. L'argument de cette opération aura le format défini en 15.10.2.

EXEMPLE 16 – Utilisation de NUMBER_OF_ELEMENTS:

```
Si l'unité received_PDU est de type ASN.1
    SEQUENCE    {    field_1 INTEGER OPTIONAL,
                     field_2 SEQUENCE OF INTEGER }
Alors l'opération
    NUMBER_OF_ELEMENTS(received_PDU.field_2)
```
renvoie le nombre d'éléments de la séquence SEQUENCE OF INTEGER contenue dans l'unité received_PDU d'objet effectif de données.
De plus, NUMBER_OF_ELEMENTS ({3, 0, 5}) renvoie 3.

11.3.3.3.4 LENGTH_OF(Value) ➔ INTEGER

Cette opération renvoie la longueur effective d'une valeur de type BITSTRING, HEXSTRING, OCTETSTRING ou CharacterString ou d'un type ASN.1 BIT STRING ou OCTET STRING. Les unités de longueur sont définies pour chaque type de chaîne au Tableau 5 en 11.18.2.

NOTE – Ces unités de longueur sont compatibles avec celles qui sont utilisées dans les contraintes de taille ASN.1 (ASN.1 SIZE) pour les objets de type ASN.1, mais non pour les valeurs littérales qui, dans ce contexte en notation TTCN, sont considérées comme étant du type TTCN correspondant. Ainsi, une chaîne hexadécimale comme 'F3'H, qui en ASN.1 pourrait être du type BIT STRING ou OCTET STRING, sera interprétée comme étant du type HEXSTRING en notation TTCN.

L'argument de cette opération aura le format défini en 15.10.2.

Cette opération ne s'applique pas aux valeurs d'un type autre que les types BITSTRING, HEXSTRING, OCTETSTRING et CharacterString ou les types ASN.1 BIT STRING ou OCTET STRING.

EXEMPLE 17 – Utilisation de LENGTH_OF:

Si S est du type BITSTRING ou du type ASN.1 BIT STRING et =’010’B, alors LENGTH_OF(S) renvoie 3.
Si S est du type HEXSTRING et =’F3’H, alors LENGTH_OF(S) renvoie 2.
Si S est du type OCTETSTRING et =’F2’O, alors LENGTH_OF(S) renvoie 1.
Si S est du type CharacterString et ="EXAMPLE", alors LENGTH_OF(S) renvoie 7.
Si S est du type ASN.1 BIT STRING et =’F3’H, alors LENGTH_OF(S) renvoie 8.
Si S est du type ASN.1 OCTET STRING et =’F3’H, alors LENGTH_OF(S) renvoie 1.
Si S est du type ASN.1 OCTET STRING et =’01010011’B, alors LENGTH_OF(S) renvoie 1.
De plus, LENGTH_OF (INT_TO_HEX (26, 4)) renvoie 4.
LENGTH_OF (’F3’H) renvoie 2
et, LENGTH_OF ("Length_of Example") renvoie 17.

11.3.4 Définitions des opérations des suites de tests et descriptions

11.3.4.1 Introduction

Le concepteur d'une suite ATS peut introduire des opérations spécifiques à cette suite de tests. Pour définir une nouvelle opération, il faut fournir les informations suivantes:

a) le nom de l'opération;

b) la liste des paramètres d'entrée avec leurs types,

qui est la liste des noms et des types des paramètres formels. Chaque nom de paramètre sera suivi de deux points (":"), puis du nom du type du paramètre.

Lorsque plusieurs paramètres sont du même type, ils pourront être regroupés en sous-listes. Dans une sous-liste, les noms des paramètres sont séparés par une virgule. Le dernier paramètre de la liste est suivi de deux points (":"), puis du nom du type commun.

Lorsqu'une liste comporte plus d'un couple paramètre/type (ou plus d'un couple liste de paramètres/type), les couples seront séparés par des points-virgules (";").

Seuls les types prédéfinis et les types de données précisés dans les définitions de type de suite de tests, de type de primitive ASP et de type d'unité PDU pourront servir de types de paramètres formels. Les types de points PCO ne seront pas utilisés comme types de paramètre formel. Tous les paramètres doivent être transférés par valeur, ce qui signifie que lors de l'évaluation d'un appel d'opération de suite de tests, les paramètres effectifs sont affectés aux paramètres formels correspondants, comme s'il s'agissait d'une déclaration d'affectation.

EXEMPLE 18 – Listes des paramètres

Les deux manières suivantes de spécifier une liste de paramètres comportant deux paramètres INTEGER et un paramètre BOOLEAN sont équivalentes:

(A:INTEGER; B:INTEGER; C:BOOLEAN)

(A, B:INTEGER; C:BOOLEAN)

c) le type du résultat,

qui obéira aux règles concernant les types de paramètres énoncées au point b);

d) la définition de l'opération,

qui consistera en l'une des deux options suivantes:

1) définition de procédure qui, lorsqu'elle est évaluée, résulte dans l'évaluation d'une déclaration RETURNVALUE pour donner le résultat de l'opération, y compris les commentaires explicatifs incorporés dans la définition de procédure aux endroits appropriés, en tant que texte délimité par "/*" et "*/";

2) description de l'opération, sous forme textuelle, comprenant si possible une référence à une spécification disponible au public de l'algorithme à appliquer lors de l'invocation de l'opération, plus au moins un exemple illustrant un appel de l'opération et le résultat correspondant; l'explication commencera par l'énoncé du nom de l'opération, suivi entre parenthèses d'une liste des noms des paramètres de l'opération; tout cela constitue un "canevas" d'invocation de l'opération;

e) facultativement, d'autres commentaires décrivant l'opération, figurent soit dans la partie commentaires de l'en-tête de la table ou dans la zone commentaires détaillés de la table.

Il est recommandé d'utiliser les définitions de procédure, afin de définir avec précision les opérations, mais une explication textuelle est admise comme solution de rechange, aux fins de compatibilité amont.

Dans le cas d'une définition de procédure, ces informations seront fournies dans le format illustré dans le Formulaire 11, ci-dessous.

Définition de la procédure d'une opération de suite de tests	
Nom de l'opération	**:** *TS_ProcId&ParList*
Groupe	**:** *[TS_ProcGroupReference]*
Type de résultat	**:** *Type*
Commentaires	**:** *[FreeText]*
Définition	
TS_OpProcDef	
Commentaires détaillés:	*[FreeText]*

Formulaire 11 – Définition de la procédure d'une opération de suite de tests

DÉFINITION SYNTAXIQUE:

155 TS_Protocol&ParList ::= TS_ProcIdentifier [FormalParList]
158 TS_ProcGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ProcGroupIdentifier "/"}
734 Type ::= PredefinedType | Reference Type
162 TS_OpProcDef ::= [VarBlock] ProcStatement

Dans le cas d'une description textuelle, ces informations seront fournies dans le format illustré dans le Formulaire 12, ci-dessous.

Description d'une opération de suite de tests	
Nom de l'opération	**:** *TS_OpId&ParList*
Groupe	**:** *[TS_OpGroupReference]*
Type de résultat	**:** *Type*
Commentaires	**:** *[FreeText]*
Description	
FreeText	
Commentaires détaillés:	*[FreeText]*

Formulaire 12 – Description d'une opération de suite de tests

DÉFINITION SYNTAXIQUE:

141 TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]
144 TS_OpGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_OpGroupIdentifier "/"}
734 Type ::= PredefinedType | ReferenceType

11.3.4.2 Paramètres

Une opération de suite de tests peut être comparée à une fonction d'un langage de programmation classique. Les valeurs ne doivent être transférées dans l'opération que par des paramètres formels. Chaque paramètre formel doit être déclaré comme étant un type prédéfini, un identificateur de type de suite de tests, un identificateur de type de primitive ASP, un

identificateur de type d'unité PDU, un identificateur de type de message CM ou comme étant le métatype **PDU**. Les variables de suite de tests, les variables de test élémentaire, les constantes de suite de tests, les paramètres de suite de tests et les contraintes ne doivent pas être directement utilisés dans la définition de procédure d'une opération de suite de tests, mais doivent être transférés comme paramètres effectifs, s'ils sont nécessaires dans l'opération de suite de tests.

Il ne doit y avoir aucun effet secondaire, c'est-à-dire que les paramètres de l'opération ne doivent pas être changés par suite d'un appel de l'opération. Des opérations prédéfinies et d'autres opérations de suite de tests peuvent être utilisées dans la définition de procédure d'une opération de suite de tests sans devoir être transférées en tant que paramètres effectifs.

Lorsqu'on invoque une opération de suite de tests:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra la valeur d'un élément du même type que le paramètre formel correspondant;

c) toutes les variables apparaissant dans la liste de paramètres effectifs seront liées;

d) les paramètres effectifs seront transférés par valeur.

11.3.4.3 Variables et identificateurs

Dans le cas de l'utilisation d'une définition de procédure, cette définition peut comprendre la déclaration de variables locales, placées au début de la définition de procédure, entre les mots clés **VAR** (variable) et **ENDVAR** (fin de la variable). Ces variables peuvent être de tout type admis en notation TTCN. Le domaine d'application de ces variables locales est la définition de procédure elle-même. Ces déclarations donnent des listes d'identificateurs de variable, chacun étant d'un type donné et chaque liste pouvant être déclarée comme étant **STATIC** (statique) ou non. Il est possible de donner une valeur initiale facultative aux variables, tant déclarées comme étant **STATIC** que ne l'étant pas.

NOTE – Il est recommandé de toujours donner aux variables **STATIC** une valeur initiale.

Les variables qui ne sont pas déclarées comme étant **STATIC** sont initialisées chaque fois que l'opération est invoquée, avec la valeur initiale spécifiée, s'il y en a, et ne doivent donc pas transmettre une valeur d'une évaluation de l'opération de suite de tests à une autre. Les variables qui sont déclarées comme étant **STATIC** sont initialisées avec la valeur initiale spécifiée, s'il y en a, la première fois que l'opération est invoquée dans une composante de test donnée, ou dans un test élémentaire donné, si les composantes de test ne sont pas utilisées, et elles conservent leur valeur d'une invocation d'opération à l'autre dans cette composante de test ou dans ce test élémentaire.

Les variables auxquelles aucune valeur initiale n'est affectée sont considérées comme étant non liées et doivent être explicitement liées à une valeur par une affectation dans le corps de l'opération, avant d'être utilisées dans une expression. Si une variable non liée est utilisée dans une expression, il s'agit alors d'une erreur de test élémentaire.

Chaque identificateur utilisé dans la définition de procédure d'une opération de suite de tests doit être l'un des éléments suivants:

a) un nom de variable déclarée localement;

b) un nom de type utilisé dans une déclaration de variable;

c) un nom de paramètre formel déclaré dans une liste de paramètres formels de l'opération;

d) un nom d'opération de suite de tests.

Le domaine d'application des noms de paramètre formel et des noms de variable déclarée localement est la définition de procédure de l'opération de suite de tests. Par conséquent, la valeur de tous les autres types d'identificateur n'est pas accessible directement dans la définition de procédure d'une opération de suite de tests. Pour accéder à une telle valeur, celle-ci doit être transférée en tant que paramètre effectif à l'opération de suite de tests.

11.3.4.4 Déclarations de procédure

Dans une définition de procédure, après la déclaration des variables locales, s'il y en a, il doit y avoir une déclaration de procédure de l'une des sortes suivantes:

a) une déclaration Return (renvoi);

b) une déclaration Assignment (affectation);

c) une déclaration If (si);

d) une boucle While (tandis);

e) une déclaration Case (cas);

f) un bloc contenant une séquence de déclarations de procédure séparées par des points-virgules et toutes délimitées par les mots clés **BEGIN** (début) et **END** (fin).

Les commentaires peuvent être incorporés dans les déclarations de procédure, en tant que texte délimité par "/*" et "*/". Les commentaires ne doivent pas être incorporés dans d'autres commentaires.

11.3.4.5 Déclarations ReturnValue (renvoi de la valeur)

Chaque évaluation d'une opération de suite de tests doit se terminer par l'évaluation d'une déclaration ReturnValue, comprenant le mot clé **RETURNVALUE** suivi d'une expression. Cette déclaration doit renvoyer la valeur de l'expression donnée comme étant le résultat de l'opération de suite de tests. Le type de ce résultat doit correspondre au type de résultat spécifié dans l'en-tête de la table de définition de l'opération de suite de tests.

11.3.4.6 Déclarations Assignment (affectation)

La forme de la déclaration d'affectation Assignment est la même que dans les descriptions de comportement en notation TTCN (voir 15.10.4), sauf que la déclaration n'est pas entre parenthèses. L'élément DataObjectReference à gauche doit commencer par une variable locale. Si la variable locale est de type structuré, l'élément DataObjectReference peut accéder à une composante de cette structure (en utilisant une référence à un enregistrement, une référence à un ensemble ou une référence à un bit, selon le cas; voir 15.10.2 et 15.10.3).

11.3.4.7 Déclarations If (si)

La déclaration If existe sous les deux formes suivantes:

– **IF** expression **THEN** déclaration de procédure **ELSE** déclaration de procédure **ENDIF**;

– **IF** expression **THEN** déclaration de procédure **ENDIF**.

L'expression suivant le mot clé **IF** doit être évaluée en premier et doit prendre une valeur booléenne. Si celle-ci prend la valeur **TRUE**, la déclaration de procédure suivant le mot clé **THEN** doit être évaluée. Si l'expression prend la valeur **FALSE,** la déclaration de procédure suivant le mot clé **ELSE**, s'il y en a, est évaluée. L'utilisation du mot clé **ENDIF** pour terminer la déclaration If permet aux déclarations de procédure suivant **THEN** et **ELSE** d'être des déclarations If, sans qu'elles soient enfermées dans un bloc.

11.3.4.8 Boucle While (tandis)

Une boucle While prend la forme suivante:

– **WHILE** expression **DO** déclaration de procédure **ENDWHILE**.

L'expression suivant le mot clé **WHILE** doit être évaluée en premier et doit prendre une valeur booléenne. Si celle-ci prend la valeur **TRUE**, la déclaration de procédure suivant le mot clé **DO** doit être évaluée, puis, si aucune déclaration ReturnValue n'a été évaluée, le processus se répète en commençant par l'évaluation de l'expression à nouveau. Dès que l'expression prend la valeur **FALSE**, l'évaluation de la boucle While est terminée.

11.3.4.9 Déclaration Case (cas)

Une déclaration Case prend l'une des deux formes suivantes:

– **CASE** expression **OF**

integer-label_1: procedure-statement_1;
integer-label_2: procedure-statement_2;

...

integer-label_n: procedure-statement_n;

ELSE

déclaration de procédure

ENDCASE

– **CASE** expression **OF**

integer-label_1: procedure-statement_1;
integer-label_2: procedure-statement_2;
...
integer-label_n: procedure-statement_n;

ENDCASE

L'expression suivant le mot clé **CASE** doit être évaluée en premier et doit prendre une valeur entière positive qui doit correspondre à au plus une étiquette d'entier dans le corps de la déclaration Case. La déclaration de procédure suivant l'étiquette d'entier concordante, s'il y en a, doit être évaluée, ce qui termine l'évaluation de la déclaration Case. Si,

toutefois, le résultat de l'évaluation de l'expression ne concorde avec aucune étiquette d'entier, la déclaration de procédure qui suit le mot clé **ELSE**, s'il y en a, doit être évaluée, ce qui termine l'évaluation de la déclaration Case. Si, toutefois, le résultat de l'évaluation de l'expression ne correspond à aucune étiquette d'entier ni à un mot clé **ELSE**, le résultat de la déclaration Case est une erreur de test élémentaire. Par conséquent, la déclaration Case équivaut à une séquence emboîtée de déclarations If, chacune faisant le test de l'expression "(expression) = integer-label_i", pouvant être suivie d'un mot clé **ELSE** au niveau le plus profond d'emboîtage.

11.3.4.10 Utilisation des opérations de suite de tests

Une opération de suite de tests, accompagnée de ses paramètres effectifs, peut être utilisée à chaque fois qu'une expression est admise.

Chaque opération de suite de tests devrait comprendre une détection d'erreur appropriée. Si une erreur (par exemple division par zéro, paramètre non valide, manque de concordance des types ou évaluation d'une variable non liée) est détectée pendant l'évaluation d'une opération de suite de tests, elle doit résulter en une erreur de test élémentaire.

EXEMPLE 19 – Définition de l'opération SUBSTR (sous-chaîne):

Description de l'opération de suite de tests	
Nom de l'opération	**:** SUBSTR (source:IA5String; start_index, length:INTEGER)
Type du résultat	**:** IA5String
Description	
SUBSTR(source, start_index, length) est la chaîne de longueur *len* commençant à l'indice *start_index* de la chaîne d'origine *source*. Par exemple: SUBSTR("abcde",3,2) = "cd" SUBSTR("abcde",1,3) = "abc" *SUBSTR(source, start_index, len)* ne doit être définie que si: *start_index* ≥ 1, *len* ≥ 0, et *start_index + len* ≤ *(length of source)* + 1. Toute tentative d'évaluer SUBSTR appliquée aux arguments sur lesquels cet élément n'est pas défini résultera en une erreur de test élémentaire.	

EXEMPLE 20 – Définition de l'opération NUMBER_OF_INVOCATIONS:

Définition de la procédure d'une opération de suite de tests	
Nom de l'opération	**:** NUMBER_OF_INVOCATIONS
Type du résultat	**:** INTEGER
Définition	
VAR STATIC COUNT : INTEGER : 0 ENDVAR BEGIN COUNT := COUNT + 1; **RETURNVALUE** COUNT END	
Commentaires détaillés: NUMBER_OF_INVOCATIONS() donne une valeur entière qui est égale au nombre de fois où cette opération a été invoquée dans la composante de test courante, ou dans le test élémentaire si l'on n'utilise pas de composantes de test.	

11.4 Déclarations des paramètres de suites de tests

Cette section a pour objectif de déclarer les constantes dérivées de la déclaration PICS et des informations PIXIT, qui servent au paramétrage global de la suite de tests. Ces constantes sont appelées paramètres de la suite de tests et servent de base à la sélection et au paramétrage des tests élémentaires.

La section déclaration fournira pour chaque paramètre de suite de tests les informations suivantes:

a) son nom;

b) son type,

il s'agira d'un type prédéfini, d'un type ASN.1, d'un type suite de tests ou d'un type unité PDU;

c) sa valeur par défaut, s'il y en a,

qui peut servir à suggérer des valeurs appropriées pour certains paramètres de suite de tests, comme les durées de temporisation;

d) la référence d'entrée PICS ou PIXIT,

il s'agit d'une référence à une entrée individuelle du formulaire PICS ou PIXIT identifiant clairement l'emplacement où se trouve la valeur à utiliser pour ce paramètre de suite de tests.

Ces informations seront fournies dans le format illustré dans le Formulaire 13, ci dessous.

Déclaration de paramètres d'une suite de tests				
Groupe : *[TS_ParGroupReference]*				
Nom de paramètre	**Type**	**Valeur par défaut**	**Réf. PICS/PIXIT**	**Commentaires**
⋮ *TS_ParIdentifier* ⋮	⋮ *Type* ⋮	⋮ *[DefaultValue]* ⋮	⋮ *FreeText* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 13 – Déclaration de paramètres d'une suite de tests

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

183 TS_ParGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ParGroupIdentifier "/"}
187 TS_ParIdentifier ::= Identifier
734 Type ::= PredefinedType | ReferenceType

EXEMPLE 21 – Déclaration de paramètres d'une suite de tests:

Déclaration de paramètres d'une suite de tests			
Nom du paramètre	**Type**	**Réf. PICS/PIXIT**	**Commentaires**
PAR1	INTEGER	PICS question xx	
PAR2	INTEGER	PICS question yy	
PAR3	INTEGER	PICS question zz	

11.5 Définition des expressions de sélection de tests élémentaires

Cette section de la suite ATS a pour objet de définir les expressions de sélection à utiliser dans le processus de sélection des tests élémentaires. Elle répondra aux spécifications de la Recommandation X.291.

Une expression de sélection est associée à un ou plusieurs tests élémentaires ou groupes de tests en plaçant son identificateur dans la colonne référence de sélection des tests élémentaires de la structure de la suite de tests ou de l'index des tests élémentaires. Une même expression peut être indiquée en référence par plus d'un test élémentaire ou groupe de tests.

Il convient de considérer que l'utilisation d'une expression de sélection signifie que le test élémentaire ne sera exécuté que si l'expression de sélection prend la valeur TRUE.

Les informations suivantes seront fournies pour chaque expression de sélection d'un test élémentaire:

a) son nom;

b) une expression de sélection,

dont le résultat sera de type BOOLÉEN et dont les termes seront exclusivement des valeurs littérales, des paramètres de suite de tests, des constantes de suite de tests et d'autres identificateurs d'expression de sélection.

Ces informations seront fournies dans le format illustré dans le Formulaire 14, ci dessous.

Définition des expressions de sélection de tests élémentaires		
Groupe : *[SelectionGroupReference]*		
Nom de l'expression	**Expression de sélection**	**Commentaires**
⋮ *SelectExprIdentifier* ⋮	⋮ *SelectionExpression* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 14 – Définition des expressions de sélection de tests élémentaires

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

198 SelectExprGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {SelectExprGroupIdentifier "/"}

202 SelectExprIdentifier ::= Identifier

204 SelectionExpression ::= Expression

11.6 Déclaration des constantes de suites de tests

Cette section de la suite ATS a pour objet de déclarer un ensemble de noms pour des valeurs *non* dérivées de la déclaration PICS ou des informations PIXIT, et qui resteront constantes dans toute la suite de tests.

Pour chaque constante d'une suite de tests, les informations suivantes seront fournies:

a) son nom;

b) son type,

il s'agira d'un type prédéfini, d'un type simple ou d'un type ASN.1 (y compris les unités PDU, les primitives ASP et les messages CM exprimés en notation ASN.1);

c) sa valeur,

les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de tests élémentaires; la valeur prise correspondra au type indiqué dans la colonne type.

Ces informations seront fournies dans le format illustré dans le Formulaire 15, ci-dessous.

Déclaration des constantes de suites de tests			
Groupe : *[TS_ConstGroupReference]*			
Nom de la constante	**Type**	**Valeur**	**Commentaires**
⋮	⋮	⋮	⋮
TS_ConstIdentifier	*Type*	*DeclarationValue*	*[FreeText]*
⋮	⋮	⋮	⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 15 – Déclaration des constantes de suites de tests

Des commentaires collectifs peuvent être utilisés dans ce formulaire conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

212 TS_ConstGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ConstGroupIdentifier "/"}
228 ValueReference ::= valuereference
734 Type ::= PredefinedType | ReferenceType
219 DeclarationValue ::= Expression

EXEMPLE 22 – Déclaration des constantes de suites de tests:

Déclaration des constantes de suites de tests			
Nom de la constante	**Type**	**Valeur**	**Commentaires**
TS_CONST1	BOOLEAN	TRUE	
TS_CONST2	IA5String	"Chaîne quelconque"	

11.7 Déclarations des constantes de suites de tests par référence

Cette section a pour objet de déclarer un ensemble de noms pour des valeurs *non* dérivées de la déclaration PICS ou des informations PIXIT, et qui resteront constantes dans toute la suite de tests.

Pour chaque constante d'une suite de tests, les informations suivantes seront fournies:

a) son nom;

b) son type,

c'est-à-dire un type prédéfini, un type simple ou un type ASN.1 (y compris les types d'unité PDU, de primitive ASP et de message CM exprimés en notation ASN.1) importé par une définition de type ASN.1 par référence, à partir du module ASN.1 identifié par l'identificateur de module spécifié;

c) sa référence à une valeur,

la valeur correspondant à un élément du type indiqué dans la colonne type;

d) l'identificateur du module,

qui comprend une référence au module qui suit les règles applicables à l'identificateur énoncées dans la Recommandation X.680, ainsi qu'un identificateur d'objet (ObjectIdentifier) facultatif; le module doit être unique dans le domaine d'intérêt.

Ces informations seront fournies dans le format illustré dans le Formulaire 16, ci-dessous.

Déclaration des constantes de suites de tests par référence				
Groupe : *[TS_ConstGroupReference]*				
Nom de la constante	**Type**	**Référence à une valeur**	**Identificateur du module**	**Commentaires**
⋮ *TS_ConstIdentifier* ⋮	⋮ *Type* ⋮	⋮ *ValueReference* ⋮	⋮ *ASN1_ModuleIdentifier* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 16 – Déclaration des constantes de suites de tests par référence

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

212 TS_ConstGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ConstGroupIdentifier "/"}
228 ValueReference ::= valuereference
734 Type ::= PredefinedType | ReferenceType
228 ValueReference ::= valuereference
133 ASN1_ModuleIdentifier ::= ModuleIdentifier

11.8 Variables TTCN

11.8.1 Déclaration des variables de suites de tests

Une suite de tests peut utiliser un ensemble de variables définies globalement pour cette suite de tests, dont la valeur se transmet dans toute la suite de tests. Ces variables sont appelées variables de la suite de tests.

On utilise une variable de suite de tests chaque fois qu'il est nécessaire de transmettre des informations entre deux tests élémentaires. Dans la notation TTCN concomitante, les variables de suites de tests ne doivent être utilisées que par la composante de test principale (MTC).

Les informations suivantes seront fournies pour chacune des variables déclarées:

a) son nom;

b) son type,

il s'agira d'un type prédéfini, d'un type ASN.1, d'un type de suite de tests ou d'un type d'unité PDU;

c) sa valeur initiale (s'il y a lieu),

la colonne valeur initiale servira à affecter une valeur initiale à une variable de suite de tests en son point de déclaration; les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de test élémentaire; la valeur prise par l'expression sera du type indiqué dans la colonne type. La spécification d'une valeur initiale est facultative.

Ces informations seront fournies dans le format illustré dans le Formulaire 17, ci-dessous.

Déclaration des variables de suites de tests			
Groupe : *[TS_VarGroupReference]*			
Nom de la variable	**Type**	**Valeur**	**Commentaires**
⋮ *TS_VarIdentifier* ⋮	⋮ *Type* ⋮	⋮ *[DeclarationValue]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 17 – Déclaration des variables de suites de tests

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

235 TS_VarGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_VarGroupIdentifier "/"}

239 TS_VarIdentifier ::= Identifier

734 Type ::= PredefinedType | ReferenceType

219 DeclarationValue ::= Expression

L'utilisation des variables d'une suite de tests ne devra pas supposer un ordre particulier d'exécution des tests élémentaires, car ces tests peuvent être exécutés indépendamment les uns des autres dans la suite de tests.

EXEMPLE 23 – Déclaration des variables de suites de tests:

Déclaration des variables de suites de tests			
Nom de la variable	**Type**	**Valeur**	**Commentaires**
Etat	IA5String	"idle" (repos)	Indique l'état stable final du test élémentaire précédent, s'il y en a, pour faciliter la détermination du préambule à utiliser.

11.8.2 Liaison des variables de suites de tests

A l'origine, les variables de suites de tests ne sont pas liées. Elles peuvent être liées ou liées à nouveau dans les contextes suivants:

a) au point de déclaration si une valeur initiale leur est affectée;

b) lorsque la variable de suite de tests apparaît dans le membre gauche d'une déclaration d'affectation (voir 15.10.4).

Une fois la variable de suite de tests liée, elle conserve sa valeur jusqu'à ce qu'il lui soit affecté une autre valeur, sinon jusqu'à la fin de l'exécution de la suite de tests.

Si une variable non liée d'une suite de tests apparaît dans le membre droit d'une affectation, il s'agit d'une erreur de test élémentaire.

11.8.3 Déclarations des variables de test élémentaire

Une suite de tests peut utiliser un ensemble de variables déclarées globalement pour cette suite de tests, mais dont le domaine d'application est défini comme local pour le test élémentaire.

En notation TTCN concomitante, chaque composante de test, y compris la composante MTC, reçoit une nouvelle copie de toutes les variables de test élémentaire lors de sa création. Ces variables sont appelées variables de test élémentaire.

Les informations suivantes sont fournies pour chaque variable déclarée:

a) son nom;

b) son type,

il s'agira d'un type prédéfini, d'un type ASN.1, d'un type de suite de tests ou d'un type d'unité PDU;

c) sa valeur initiale (le cas échéant),

la colonne valeur initiale servira à affecter une valeur initiale à une variable de test élémentaire en son point de déclaration; les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de test élémentaire; la valeur prise par l'expression sera du type indiqué dans la colonne type. La spécification d'une valeur initiale est facultative.

Ces informations seront fournies dans le format illustré dans le Formulaire 18, ci-dessous.

Déclaration des variables de test élémentaire			
Groupe : *[TC_VarGroupReference]*			
Nom de la variable	**Type**	**Valeur**	**Commentaires**
⋮ *TC_VarIdentifier* ⋮	⋮ *Type* ⋮	⋮ *[DeclarationValue]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 18 – Déclaration des variables de test élémentaire

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

248 TC_VarGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TC_VarGroupIdentifier "/"}
252 TC_VarIdentifier ::= Identifier
734 Type ::= PredefinedType | ReferenceType
219 DeclarationValue ::= Expression

NOTE – Lorsqu'on utilise des variables de test élémentaire comme variables locales dans un module de test, il faut prêter attention aux possibles conflits d'appellation avec les variables des autres modules de test et tests élémentaires. Pour éviter ces problèmes, un concepteur de suite de tests peut adopter une convention de dénomination garantissant l'unicité d'appellation de ces variables dans la suite de tests.

11.8.4 Liaison des variables de test élémentaire

A l'origine, les variables de suite de tests ne sont pas liées. Elles peuvent être liées ou liées à nouveau dans les contextes suivants:

a) au point de déclaration si une valeur initiale leur est affectée;

b) lorsque la variable de test élémentaire apparaît dans le membre gauche d'une déclaration d'affectation (voir 15.10.4).

Une fois la variable de test élémentaire liée, elle conserve sa valeur jusqu'à ce qu'il lui soit affecté une autre valeur en cours d'exécution, sinon jusqu'à la fin de l'exécution du test élémentaire. A la fin du test élémentaire, la variable de test élémentaire reprend sa valeur initiale, si une telle valeur est spécifiée; sinon, elle devient non liée.

Si une variable de test élémentaire non liée apparaît dans le membre droit d'une affectation, il s'agit d'une erreur de test élémentaire.

11.9 Déclaration des types de point PCO

Cette partie de la suite de tests abstraite (ATS) énumère l'ensemble des limites de service où se trouvent les points de contrôle et d'observation (PCO).

Les informations suivantes seront fournies pour chaque point PCO utilisé dans la suite de tests:

a) son nom,

   qui est le même que celui donné dans la table des points PCO;

b) son rôle,

   qui doit être déclaré comme étant soit UT ou LT dans la colonne rôle ou par un texte descriptif dans la colonne commentaires; l'identificateur prédéfini **UT** (*upper tester*) indique que le point PCO est un point PCO de testeur supérieur, l'identificateur **LT** (*lower tester*) spécifiant qu'il s'agit d'un point PCO de testeur inférieur; si la colonne rôle est utilisée, son contenu doit correspondre au rôle, s'il y en a, qui est donné dans la table de déclaration des points PCO.

   NOTE – Dans une suite de tests utilisant la concomitance, le rôle d'un type de point PCO peut devoir être décrit en fonction de la nature de la composante de test et du fournisseur de services sous-jacent qui doivent être couplés par les points PCO de ce type.

Ces informations seront fournies dans le format illustré dans le Formulaire 19, ci-dessous.

Déclaration du type des points PCO		
Groupe : *[PCO_GroupReference]*		
Type du point PCO	**Rôle**	**Commentaires**
⋮ *PCO_TypeIdentifier* ⋮	⋮ *[PCO_Role]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 19 – Déclaration du type des points PCO

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

271 PCO_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PCO_GroupIdentifier "/"}

263 PCO_TypeIdentifier ::= Identifier

279 PCO_Role ::= **UT** | **LT**

Les types de points PCO sont définis dans la table des points PCO et, par conséquent, la table des types de points PCO est facultative. Si un type de point PCO est donné en tant qu'objet exporté dans une table d'exportation, il doit être défini dans la table des types de points PCO.

11.10 Déclaration de points PCO

Cette partie de la suite de tests abstraite (ATS) énumère l'ensemble des points de contrôle et d'observation (PCO) à utiliser dans la suite de tests et explique où ils existent dans l'environnement de test.

NOTE 1 – Le nombre des points PCO sera, le cas échéant, celui qui est spécifié dans les Recommandations X.290 et X.291, pour la ou les méthodes identifiées dans la table de structure de la suite de tests. Dans la notation TTCN, les points PCO peuvent aussi être utilisés selon une manière non décrite dans la Recommandation X.291, par exemple pour communiquer avec des parties du système de test ou de l'environnement de test qui ne sont pas définis dans la suite de tests (par exemple pour manipuler des fréquences ou simuler des transferts pour les tests de protocole en radiocommunications).

NOTE 2 – Les déclarations comportementales TTCN spécifiées pour être exécutées au point PCO du testeur supérieur n'auront pas de spécifications allant au-delà de celles de la Recommandation X.291.

En notation TTCN, le modèle de point PCO repose sur deux files d'attente (FIFO, *first in first out*) (premier entré, premier sorti):

– une file de sortie pour l'envoi de primitives ASP et/ou d'unités PDU;

– une file d'entrée pour la réception de primitives ASP et d'unités PDU.

La file de sortie est supposée se trouver au niveau du fournisseur de services sous-jacent ou, dans le cas du testeur supérieur, au niveau de l'implémentation sous test (IUT).

Un événement SEND (envoi) à un point PCO est réussi lorsqu'il est transmis du testeur inférieur au fournisseur de services, ou du testeur supérieur à l'implémentation IUT.

Le testeur est doté d'une file d'attente d'entrée pour la réception des événements. Tous les événements entrants sont placés à la suite dans cette file et traités dans leur ordre d'arrivée, sans qu'aucun ne soit perdu.

NOTE 3 – Ce modèle de file d'attente est purement abstrait et ne suppose aucune mise en œuvre particulière.

Les informations suivantes seront fournies pour chaque point PCO utilisé dans la suite de tests:

a) son nom,

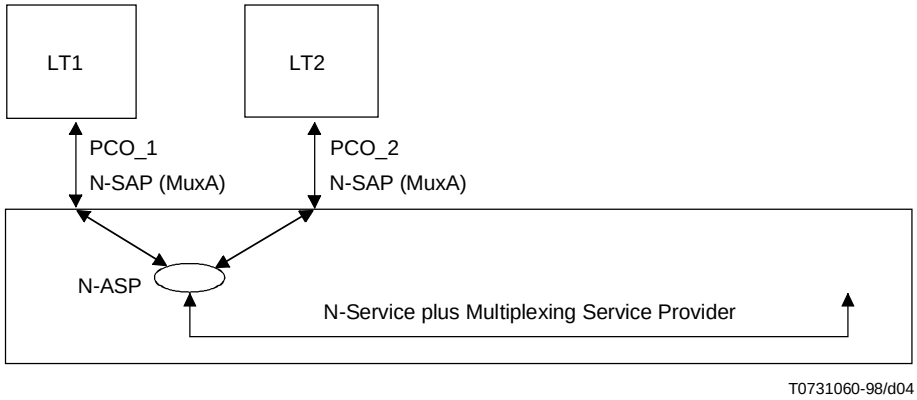
utilisé dans les descriptions comportementales pour spécifier l'endroit où des événements particuliers interviennent;

b) son type,

utilisé pour identifier la limite du service où se trouve le point PCO, et qui peut, au besoin, être suivi des informations relatives aux exigences de multiplexage à satisfaire immédiatement en dessous de ce point PCO, mais au-dessus de la limite de service; si les activités se produisant à deux points PCO ou plus doivent être multiplexées par le fournisseur de services (par exemple en un point final d'une seule connexion), le type de point PCO doit alors, dans les déclarations de point PCO pour ces points PCO, être suivi par la même valeur de multiplexage MuxValue (paramètre de suite de tests), donnée entre parenthèses; la signification précise de ce paramètre de suite de tests doit être spécifiée dans le formulaire PIXIT approprié;

NOTE 4 – Voir également I.11 pour de plus amples explications de la valeur MuxValue.

EXEMPLE 24 – Utilisation de MuxValue:



c) son rôle,

qui peut être omis s'il est précisé dans la table de déclaration de type de point PCO pour chaque type de point PCO utilisé; si le rôle n'est pas spécifié dans une table de déclaration de type de point PCO, il doit alors être déclaré soit comme testeur supérieur (UT) ou comme testeur inférieur (LT) dans la colonne rôle, ou par un texte descriptif dans la colonne commentaires; l'identificateur prédéfini **UT** indique que le point PCO est un point PCO de testeur supérieur, tandis que l'identificateur **LT** indique un point PCO de testeur inférieur; si la colonne rôle est utilisée, son contenu doit correspondre au rôle, le cas échéant, donné dans la table de déclaration de type de point PCO.

NOTE 5 – Dans une suite de tests utilisant la concomitance, le rôle d'un point PCO peut devoir être décrit en fonction de la nature de la composante de test et du fournisseur du service sous-jacent qui doivent être couplés par ce point PCO.

Ces informations seront fournies dans le format illustré dans le Formulaire 20, ci-dessous.

Déclaration des points PCO			
Groupe : *[PCO_GroupReference]*			
Nom du point PCO	**Type**	**Rôle**	**Commentaires**
⋮ *PCO_Identifier* ⋮	⋮ *PCO_TypeIdentifier [(MuxValue)]* ⋮	⋮ *[PCO_Role]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 20 – Déclaration des points PCO

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

271 PCO_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PCO_GroupIdentifier "/"}
275 PCO_Identifier ::= Identifier
263 PCO_TypeIdentifier ::= Identifier
277 MuxValue ::= TS_ParIdentifier
279 PCO_Role ::= **UT** | **LT**

EXEMPLE 25 – Déclaration des points PCO:

Déclaration des points PCO			
Nom du point PCO	**Type du point PCO**	**Rôle**	**Commentaires**
L	TSAP	LT	Point d'accès au service transport, niveau testeur inférieur.
U	SSAP	UT	Point d'accès au service session, niveau testeur supérieur.

Les points de contrôle et d'observation PCO sont d'habitude des points d'accès au service (SAP), mais plus généralement, il peut s'agir de n'importe quel point d'où il est possible d'observer et de contrôler les événements de test. Il est toutefois possible de définir un point PCO correspondant à un *ensemble* de points SAP, à condition que tous ces points SAP comprenant ce point PCO:

– soient situés au même endroit (c'est-à-dire dans le testeur inférieur ou dans le testeur supérieur);

– donnent accès au même service.

Lorsqu'un point PCO correspond à plusieurs points SAP, on identifie un point SAP particulier par son adresse. Les points PCO sont normalement associés à un point d'accès au service (SAP) de l'implémentation IUT ou du fournisseur de service de la couche (N – 1).

NOTE 6 – Un point PCO ne peut être associé à aucun point SAP, notamment dans le cas d'une couche composée de sous-couches (par exemple, dans la couche Application, ou dans les couches inférieures, où un point de rattachement de sous-réseau n'est pas un point SAP).

11.11 Déclarations des points de coordination (CP)

Les points CP sont utilisés pour faciliter l'échange des messages de coordination (CM) entre les composantes de test. Les points CP sont modélisés selon deux files d'attente, une pour chaque sens de communication. Ils sont, sous ce rapport, semblables aux points de contrôle et d'observation (PCO) (voir la Figure 3). Une différence entre les points CP et les points PCO est que les points CP raccordent deux composantes de test, tandis que les points PCO raccordent une composante de test à l'environnement externe constitué habituellement soit par l'implémentation IUT, soit par un fournisseur de service (voir la Figure 5).

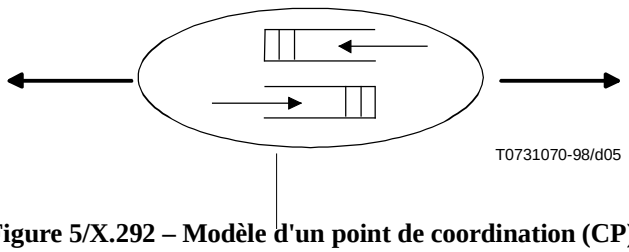


Figure 5/X.292 – Modèle d'un point de coordination (CP)

Les points CP peuvent être réalisés soit par une communication locale, soit par une communication atteignant les limites physiques.

La communication se faisant par l'intermédiaire des points CP est asynchrone, c'est-à-dire qu'elle est effectuée par une composante de test envoyant un message CM à sa partenaire, cette partenaire recevant le message CM quand elle est prête. Toutefois, la composante de test qui a déclenché le message CM passe à l'exécution immédiatement après avoir envoyé le message CM. S'il est nécessaire que la composante de test d'envoi suspende son activité jusqu'à ce que le message CM ait été reçu, le spécificateur de suites de tests devrait utiliser un mécanisme de prise de contact. La Figure 6 illustre un exemple de spécification de la prise de contact.

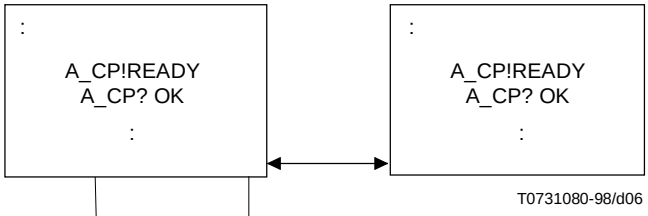


Figure 6/X.292 – Exemple d'une prise de contact simple

Tous les points CP doivent être déclarés. Le nom de chaque point CP doit être unique à l'intérieur de la suite de tests.

Ces informations seront fournies dans le format illustré dans le Formulaire 21, ci-dessous.

Déclarations des points de coordination (CP)	
Groupe : *[CP_GroupReference]*	
Nom du point CP	**Commentaires**
⋮ *CP_Identifier* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*	

Formulaire 21 – Déclarations des points de coordination (CP)

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

286 CP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {CP_GroupIdentifier"/"}
290 CP_Identifier ::= Identifier

11.12 Déclarations des temporisateurs

Une suite de tests peut utiliser des temporisateurs. Les informations suivantes seront fournies pour chaque temporisateur:

a) son nom;

b) sa durée facultative,

où la durée par défaut de la temporisation est une expression qui pourra être omise si sa valeur ne peut être déterminée avant l'exécution de la suite de tests; les termes de l'expression ne contiendront ni variables de suite de tests, ni variables de test élémentaire. L'expression de la durée de temporisation prendra une valeur ENTIÈRE non signée (donc positive);

c) l'unité de temps,

c'est-à-dire:

1) **ps** (picoseconde);
2) **ns** (nanoseconde);
3) **µs** (microseconde);

4) **ms** (milliseconde);

5) **s** (seconde);

6) **min** (minute).

Les unités de temps sont déterminées par le concepteur de la suite de tests et fixées au moment de la spécification. Différents temporisateurs pourront utiliser des unités de temps différentes dans la même suite de tests. S'il existe une entrée de déclaration PICS ou d'information PIXIT, la déclaration de temporisateur doit spécifier les mêmes unités que dans l'entrée PICS/PIXIT.

Ces informations seront fournies dans le format illustré dans le Formulaire 22, ci-dessous.

Déclaration des temporisateurs			
Groupe : *[TimerGroupReference]*			
Nom du temporisateur	**Durée**	**Unité**	**Commentaires**
⋮ *TimerIdentifier* ⋮	⋮ *[DeclarationValue]* ⋮	⋮ *TimeUnit* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 22 – Déclaration des temporisateurs

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

297 TimerGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TimerGroupIdentifier "/"}

301 TimerIdentifier ::= Identifier

219 DeclarationValue ::= Expression

304 TimerUnit ::= **ps | ns | ms | s | min**

Chaque composante de test obtient une nouvelle copie de tous les temporisateurs quand elle commence à exécuter son comportement.

EXEMPLE 26 – Déclaration des temporisateurs:

Déclaration des temporisateurs			
Nom du temporisateur	**Durée**	**Unité**	**Commentaires**
wait (attente)	15	s	Temporisation à usage général
no_response (pas de réponse)	A	min	Sert à attendre que l'implémentation sous test se connecte ou réagisse à l'établissement d'une connexion; sa durée est supérieure à celle d'une temporisation à usage général. Elle tire sa valeur des informations PIXIT.
Delay_time (délai)		ms	Durée à déterminer pendant l'exécution de la suite de tests

11.13 Déclaration des composantes de test et des configurations

11.13.1 Composantes de test

11.13.1.1 Composante de test principale

La composante de test principale a pour but de remplir le rôle de la fonction de commande de testeur inférieur (LTCF, *lower tester control function*), tel qu'il est défini au 12.5.2/X.291. Son comportement est décrit dans le premier arbre de la table de description des comportements des tests élémentaires et dans tous les arbres qui lui sont rattachés. Elle est chargée de:

a) créer toutes les composantes de test parallèles (PTC) nécessaires dans la configuration courante et de surveiller leur fin;

b) gérer les points CP qui existent entre elle et les composantes PTC;

c) calculer et affecter le verdict du test, en utilisant sa connaissance de l'effet combiné des résultats préliminaires provenant des composantes PTC.

De plus, la composante de test principale peut gérer le ou les points PCO.

Seule la composante de test principale utilisera des variables de suite de tests. Ces variables de suite de tests ne doivent pas être transférées aux points PTC dans la construction CREATE.

11.13.1.2 Composantes de test parallèles

Les composantes de test parallèles ont pour but de remplir le rôle de testeurs inférieurs ou de testeurs supérieurs. Leur comportement est décrit dans l'arbre référencé dans une déclaration CREATE dans la composante MTC, et dans tous les arbres qui lui sont rattachés. Une composante PTC affecte les résultats préliminaires, mais n'affecte pas les verdicts de test.

Une composante PTC ne doit pas:

a) utiliser de variables de suite de tests;

b) créer d'autres composantes de test.

11.13.1.3 Déclaration des composantes de test

Dans le cas de l'utilisation de la notation TTCN concomitante, cette section de la suite ATS doit déclarer toutes les composantes de test individuelles qui sont utilisées. Ces composantes de test sont par la suite référencées à partir des déclarations de configurations de composantes de test qui définissent les configurations spécifiques.

Les informations suivantes sont fournies pour chaque composante de test:

a) son nom,

   qui doit être unique dans la suite de tests;

b) son rôle,

   qui doit indiquer s'il s'agit d'une composante de test principale ou d'une composante de test parallèle, et l'endroit où au moins une composante de test doit être une composante de test principale et où au moins une composante de test doit être une composante de test parallèle;

c) le nombre de points PCO utilisés,

   dans le cas où zéro, un ou plusieurs points PCO peuvent être associés à la composante de test;

d) le nombre de points CP utilisés,

   dans le cas où zéro, un ou plusieurs points CP peuvent être associés à la composante de test.

Ces informations seront fournies dans le format illustré dans le Formulaire 23, ci-dessous.

Déclaration des composantes de test				
Groupe : *[TcompGroupReference]*				
Nom de la composante	**Rôle de la composante**	**Nombre de points PCO**	**Nombre de points CP**	**Commentaires**
⋮ *TcompIdentifier* ⋮	⋮ *TCompRole* ⋮	⋮ *Num_PCOs* ⋮	⋮ *Num_CPs* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 23 – Déclaration des composantes de test

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

311 TcompGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TcompGroupIdentifier "/"}
315 TcompIdentifier ::= Identifier
317 TcompRole ::= MTC | PTC
318 Num_PCOs ::= Number
321 Num_Cos ::= Number

EXEMPLE 27 – Déclaration des composantes de test:

Cette table de déclaration des composantes de test peut être utilisée de concert avec les configurations de composantes de test CONFIG1 et CONFIG2, illustrées aux Figures 3 et 4, et déclarées dans les Exemples 28 et 29.

Déclaration des composantes de test				
Nom de la composante	**Rôle de la composante**	**Nombre de points PCO**	**Nombre de points CP**	**Commentaires**
MTC1	MTC	0	3	Utilisée dans la Config 1
MTC2	MTC	1	2	Utilisée dans la Config 2, avec un point PCO
TC1	PTC	1	2	Utilisée dans la Config 1
TC2	PTC	1	3	Utilisée dans la Config 1 et la Config 2
TC3	PTC	1	2	Utilisée dans la Config 1
TC4	PTC	0	3	Utilisée dans la Config 2
TC5	PTC	1	0	Utilisée dans la Config 2, sans point CP

11.13.2 Déclarations des configurations de composantes de test

Les composantes de test servent à construire une architecture logique, ou configuration, qui facilite l'exécution concomitante des arbres de comportement dynamique en notation TTCN. Toute configuration de composantes de test utilisée dans un test élémentaire abstrait utilisant la concomitance doit être déclarée.

Les informations suivantes seront fournies pour chaque configuration de composantes de test:

a) son nom,

qui doit être unique dans la suite de tests et doit être référencé à partir d'un en-tête de table de comportement dynamique de test élémentaire;

b) une liste des composantes de test appartenant à la configuration de test,

les informations suivantes étant fournies pour chaque composante de test:

1) son nom,

qui doit avoir été déclaré comme nom de composante de test. Exactement une des composantes de test de la configuration doit être déclarée comme étant la composante principale MTC;

2) les points PCO utilisés,

une liste de zéro, un ou plusieurs points PCO déclarés étant associée à chaque composante de test. Le nombre de points PCO de la liste doit être égal au nombre de points PCO déclarés dans la déclaration de composantes de test appropriée. Aucun point PCO ne doit être utilisé plus d'une fois dans une même configuration (c'est-à-dire que les composantes de test d'une même configuration ne doivent pas partager de points PCO);

3) les points CP utilisés,

une liste de zéro, un ou plusieurs points CP déclarés étant associée à chaque composante de test. Le nombre de points CP de la liste pour une composante PTC doit être égal au nombre de points CP déclarés dans la déclaration de composantes de test appropriée. Le nombre de points CP de la liste d'une composante MTC ne doit pas dépasser le nombre de points CP déclarés. Aucun nom de point CP ne doit apparaître plus d'une fois dans chaque liste de points CP. Chaque nom de point CP de la liste correspondant à une composante de test doit apparaître dans la liste pour exactement une des autres composantes de test de la configuration. Autrement dit, chaque nom de point CP utilisé dans la configuration apparaîtra exactement deux fois dans la table de configuration. Ces paires de points CP servent à spécifier la connectivité des composantes de test de la configuration.

Ces informations seront fournies dans le format illustré dans le Formulaire 24, ci-dessous.

Déclaration des configurations de composantes de test			
Nom de la configuration : *TCompConfigIdentifier*			
Groupe : *[TCompConfigGroupReference]*			
Commentaires : *[FreeText]*			
Composantes utilisées	**Points PCO utilisés**	**Points CP utilisés**	**Commentaires**
⋮ *TcompIdentifier* ⋮	⋮ *[PCO_List]* ⋮	⋮ *[CP_List]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 24 – Déclaration des configurations de composantes de test

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

329 TcompConfigIdentifier ::= Identifier
330 TcompConfigGroupreference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TcompConfigGroupIdentifier "/"}
315 TcompIdentifier ::= Identifier
337 PCO_List ::= PCO_Identifier {Comma PCO_Identifier}
339 CP_List ::= CP_Identifier {Comma CP_Identifier}

EXEMPLE 28 – Déclaration des configurations de composantes de test correspondant à la Figure 3:

Déclaration des configurations de composantes de test		
Nom de la configuration : CONFIG_1		
Composantes utilisées	**Points PCO utilisés**	**Points CP utilisés**
MTC1		MCP1, MCP2, MCP3
TC1	PCO_A	MCP1, CP1
TC2	PCO_B	MCP2, CP1, CP2
TC3	PCO_C	MCP3, CP2

EXEMPLE 29 – Déclaration des configurations de composantes de test correspondant à la Figure 4:

Déclaration des configurations de composantes de test		
Nom de la configuration : CONFIG_2		
Composantes utilisées	**Points PCO utilisés**	**Points CP utilisés**
MTC2	PCO_D	MCP2, MCP3
TC2	PCO_B	MCP2, CP1, CP2
TC4		MCP3, CP1, CP2
TC5	PCO_E	

11.14 Définition des types de primitives ASP

11.14.1 Introduction

Cette partie d'une suite de tests TTCN a pour objet de déclarer les types de primitives ASP pouvant être envoyées ou reçues aux points PCO déclarés. Les définitions des types de primitives ASP peuvent inclure, si nécessaire, des définitions de types en notation ASN.1.

11.14.2 Définition des types de primitives ASP à l'aide de tables

Les informations suivantes seront fournies pour chaque primitive ASP:

a) son nom,

c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondante; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à la primitive ASP,

ce type de point PCO étant l'un de ceux qui sont employés dans le formulaire de déclaration des points PCO. Si un seul point PCO est défini dans la suite de tests, la spécification du type de point PCO dans la définition du type de primitive ASP est facultative;

c) la liste des paramètres associés à la primitive ASP,

donnant pour chaque paramètre les informations suivantes:

1) son nom,

c'est-à-dire:

– soit le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondante; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

– soit le macrosymbole (<-) indiquant que l'entrée dans la colonne type identifie un ensemble de paramètres qui doit être inséré directement dans la liste des paramètres de la primitive ASP; ce macrosymbole ne sera utilisé qu'avec les types structurés mentionnés dans les définitions des types structurés;

2) son type et un attribut facultatif,

les paramètres pouvant être d'un type à structure arbitrairement complexe, notamment d'un type de suite de tests (prédéfini, simple, structuré ou ASN.1); si un paramètre est structuré comme une unité PDU, son type est alors déclaré:

– soit comme un identificateur d'unité PDU pour indiquer que, dans la contrainte de primitive ASP, ce paramètre peut être chaîné à une contrainte d'unité PDU d'un type PDU spécifique;

– soit sous la forme du symbole **PDU** pour indiquer que, dans la contrainte de primitive ASP, ce paramètre peut être chaîné à une contrainte d'unité PDU d'un type PDU quelconque, l'attribut facultatif étant Length (longueur);

la spécification de cet attribut peut restreindre ce paramètre à une longueur ou à un intervalle de longueurs particulier, conformément au 11.18. Les valeurs de longueur doivent être interprétées selon le Tableau 5 en 11.18. Les limites seront spécifiées sous forme de valeurs littérales d'ENTIERS non négatifs, de paramètres de suite de tests, de constantes de suite de tests ou du mot clé INFINITY (infini).

Les spécifications de longueur définies pour le type de paramètre de la primitive ASP dans les définitions de type de suite de tests ne doivent pas être contraires aux spécifications de longueur figurant dans la définition du type de la primitive ASP, c'est-à-dire que l'ensemble de chaînes défini par restriction de longueur dans la définition d'une primitive ASP doit former un sous-ensemble strict de l'ensemble de chaînes précisé dans la définition du type de la suite de tests.

Le mot clé INFINITY peut servir de limite supérieure pour indiquer une longueur maximale illimitée.

NOTE – Il n'est généralement pas nécessaire de limiter la longueur des paramètres des primitives ASP; cependant, dans certains cas, cela peut s'avérer nécessaire pour limiter effectivement la longueur d'un champ de l'unité PDU correspondante dans un protocole sous-jacent.

Les paramètres des définitions de types de primitives ASP sont considérés comme facultatifs; c'est-à-dire que, dans des instances de ces types, des paramètres entiers peuvent être omis.

Ces informations seront fournies dans le format illustré dans le Formulaire 25, ci-dessous.

Définition du type de primitive ASP		
Nom de la primitive ASP : *ASP_Id&FullId*		
Groupe : *[ASP_GroupReference]*		
Type de point PCO : *[PCO_TypeIdentifier]*		
Commentaires : *[FreeText]*		
Nom du paramètre	**Type du paramètre**	**Commentaires**
⋮ *ASP_ParIdOrMacro* ⋮	⋮ *Type&Attributes* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 25 – Définition du type de primitive ASP

Les colonnes nom du paramètre et type du paramètre doivent être soit toutes deux présentes, soit toutes deux absentes.

DÉFINITION SYNTAXIQUE:

348 ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]

351 ASP_Groupreference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASP_GroupIdentifier "/"}

263 PCO_TypeIdentifier ::= Identifier

357 ASP_ParIdOrMacro ::= ASP_ParId&FullId | MacroSymbol

396 Type&Attributes ::= (Type[LengthAttribute]) |**PDU**

EXEMPLE 30 – Primitive de demande de service abstraite T_CONNECT:

La figure ci-dessous représente un exemple tiré du service transport (Recommandation X.214). Il pourrait s'agir d'une des primitives ASP utilisées pour décrire le comportement d'un testeur supérieur abstrait dans une suite de tests DS (méthode de test monocouche répartie) pour le transport de classe 0. CDA, CGA et QS sont des types de suite de tests (Recommandation X.224).

Définition du type de primitive ASP		
Nom de la primitive ASP : CONreq (demande T_CONNECT) **Type du point PCO** : *TSAP* **Commentaires** :		
Nom du paramètre	**Type du paramètre**	**Commentaires**
Cda (called address)	CDA	adresse appelée du testeur supérieur
Cga (calling address)	CGA	adresse appelante du testeur inférieur
QS	QS	qualité de service – doit vérifier que la classe 0 est utilisée
Commentaires détaillés: primitive ASP à transmettre au point d'accès au service transport		

11.14.3 Utilisation de types structurés dans les définitions de types des primitives ASP

Les définitions de primitives ASP peuvent se référer de deux manières possibles à un type structuré:

a) si la définition contient un nom de paramètre, le type structuré mentionné forme une sous-structure. Ceci permet de définir des primitives ASP à sous-structure paramétrique multiniveau;

b) si on utilise le macrosymbole (<-) à la place d'un nom de paramètre, ceci équivaut alors à un développement de macro; l'entrée dans la définition du type de primitive ASP se développe directement en une liste de paramètres sans introduire de niveau supplémentaire de sous-structure.

Ce macrosymbole ne doit pas être utilisé sur la même ligne que des références à des types ASN.1 ou à des types simples; en d'autres termes, seuls des types structurés définis sous forme tabulaire peuvent être développés dans d'autres types structurés en tant que développements de macro.

11.14.4 Définitions de types de primitives ASP à l'aide de la notation ASN.1

Les primitives ASP peuvent, si nécessaire, être spécifiées en notation ASN.1. On a alors recours à une définition ASN.1 en utilisant la syntaxe ASN.1 telle qu'elle est définie dans la Recommandation X.680. Les informations suivantes seront fournies pour chaque primitive ASP en notation ASN.1:

a) son nom,

   c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à la primitive ASP,

   ce type de point PCO étant l'un de ceux employés dans le formulaire de déclaration des points PCO. Si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type de la primitive ASP devient facultative;

c) la définition ASN.1 du type de primitive ASP,

   qui doit respecter la syntaxe définie dans la Recommandation X.680. Le symbole trait d'union (-) ne doit pas figurer dans les identificateurs de cette définition; il peut être remplacé par le symbole de soulignement (_). L'identificateur de primitive ASP porté dans l'en-tête de la table est le nom du premier type défini dans le corps de la table.

   Les types référencés à partir de la définition de la primitive ASP seront définis soit dans d'autres tables de définition de types en notation ASN.1, soit par référence dans la table de référence des types en notation ASN.1, soit enfin, localement dans la même table, à la suite de la définition du premier type. Les types définis localement ne seront pas utilisés ailleurs dans la suite de tests.

   Des commentaires ASN.1 peuvent être portés dans le corps de la table. La colonne commentaires n'existe pas.

Les commentaires en notation ASN.1 commencent par "--" et finissent soit par la prochaine occurrence de "--" ou par l'indication "fin de ligne", suivant ce qui se produit en premier. Cela évite qu'un simple commentaire ASN.1 s'étale sur plusieurs lignes. L'indication "fin de ligne" n'est toutefois pas un symbole défini dans la notation TTCN.MP. Il est par conséquent recommandé aux spécificateurs de suites ATS de faciliter l'échange de suites ATS en notation TTCN.MP en terminant toujours les commentaires en notation ASN.1 par "--".

Ces informations seront fournies dans le format illustré dans le Formulaire 26, ci-dessous.

Définition du type de primitive ASP en notation ASN.1	
Nom de la primitive ASP	**:** *ASP_Id&FullId*
Groupe	**:** *[ASN1ASP_GroupReference]*
Type de point PCO	**:** *[PCO_TypeIdentifier]*
Commentaires	**:** *[FreeText]*
Définition du type	
ASN1_Type&LocalTypes	
Commentaires détaillés: *[FreeText]*	

Formulaire 26 – Définition du type de primitive ASP en notation ASN.1

DÉFINITION SYNTAXIQUE:

348 ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]
367 ASN1_ASP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_ASP_GroupIdentifier "/"}
263 PCO_TypeIdentifier ::= Identifier
121 ASN1_Type&Localtypes ::= ASN1_Type {ASN1_LocalType}

11.14.5 Définitions par référence de types de primitives ASP en notation ASN.1

Les primitives ASP peuvent être spécifiées par une référence précise à une primitive ASP en notation ASN.1 définie dans une Recommandation sur l'OSI ou par référence à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les informations suivantes seront fournies pour chaque primitive ASP:

a) son nom,

qui peut être utilisé dans toute la suite de tests;

b) le type de point PCO associé à cette primitive ASP,

ce type devant être l'un de ceux mentionnés dans le formulaire de déclaration des points PCO. Si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO devient facultative dans la définition de type de primitive ASP;

c) la référence du type,

qui respectera les règles relatives aux identificateurs énoncées dans la Recommandation X.680;

d) l'identificateur du module,

composé d'une référence de module conforme aux règles relatives aux identificateurs énoncées dans la Recommandation X.680 et d'un identificateur facultatif d'objet ObjectIdentifier.

Ces informations seront fournies dans le Formulaire 27, ci-dessous.

Définitions par référence de types de primitives ASP en notation ASN.1				
Groupe **:** *[ASN1ASP_GroupReference]*				
Nom de la primitive ASP	**Type de point PCO**	**Référence du type**	**Identificateur de module**	**Commentaires**
⋮	⋮	⋮	⋮	⋮
ASP_Id&FullId	*[PCO_TypeIdentifier]*	*TypeReference*	*ModuleIdentifier*	*[FreeText]*
⋮	⋮	⋮	⋮	⋮
Commentaires détaillés: *[FreeText]*				

Formulaire 27 – Définitions par référence de types de primitives ASP en notation ASN.1

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

367 ASN1_ASP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_ASP_GroupIdentifier "/"}

348 ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]

263 PCO_TypeIdentifier ::= Identifier

131 TypeReference ::= typereference

133 ASN1_ModuleIdentifier ::= ModuleIdentifier

Les identificateurs, les références de types et les références de valeur en notation ASN.1 peuvent comporter des traits d'union. Pour pouvoir utiliser des définitions importées en notation TTCN, il est nécessaire de remplacer ces traits d'union par des caractères de soulignement (voir A.4.2.1).

11.15 Définition des types d'unité PDU

11.15.1 Introduction

Cette partie d'une suite de tests abstraite en notation TTCN a pour objet de déclarer les types d'unités PDU qui peuvent être envoyées ou reçues, directement ou imbriquées dans des primitives ASP, aux points PCO déclarés. Les définitions des types d'unité PDU peuvent comporter si nécessaire des définitions de type en notation ASN.1. Les définitions d'unités PDU définissent l'ensemble des unités PDU échangées avec l'implémentation sous test et qui sont syntaxiquement valides par rapport à la suite ATS, mais pas nécessairement valides par rapport à la spécification de protocole.

Tous les champs des unités PDU définis dans la Recommandation sur les protocoles pertinente doivent être déclarés soit explicitement, soit implicitement par référence à des règles de codage (les règles de codage ASN.1, si elles sont applicables).

Le codage des champs d'unités PDU sera conforme à la spécification de protocole correspondante, à moins que les informations de codage ne soient incluses dans la suite de tests.

11.15.2 Définition des types d'unités PDU à l'aide de tables

La définition des unités PDU est semblable à celle des primitives ASP. Les informations suivantes seront fournies pour chaque unité PDU:

a) son nom,

   c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type de point PCO associé à l'unité PDU,

   ce type de point PCO étant l'un de ceux mentionnés dans les déclarations de points PCO; si, dans toute la suite de tests, une unité PDU n'est envoyée ou reçue qu'imbriquée dans des primitives ASP, la spécification du type de point PCO devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type d'unité PDU devient facultative;

c) les règles de codage à utiliser pour les unités PDU de ce type,

   afin de spécifier un codage explicite pour des unités PDU complètes, qui a priorité sur les règles de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des définitions de codage appropriée (par exemple pour passer des règles BER aux règles DER). Si cette entrée n'est pas utilisée, les règles de codage globales par défaut s'appliquent. Voir 11.16.4;

d) les variations de codage à utiliser pour les unités PDU de ce type,

   afin de spécifier des variations de codage explicites pour des unités PDU complètes, qui ont priorité sur les variations de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des variations de codage appropriée [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage globales par défaut s'appliquent. Voir 11.16.4;

e) la liste des champs associés à l'unité PDU,

   donnant, pour chaque champ, les informations suivantes:

   1) son nom,

c'est-à-dire:

– soit le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

– soit le macrosymbole (<-) indiquant que l'entrée dans la colonne type identifie un ensemble de champs à insérer directement dans la liste des champs de l'unité PDU; ce macrosymbole ne sera utilisé qu'avec les types structurés mentionnés dans les définitions correspondantes des types structurés;

2) son type et un attribut facultatif,

les champs pouvant être d'un type à structure arbitrairement complexe, notamment d'un type de suite de tests (prédéfini, simple, structuré ou ASN.1); si un champ doit être structuré comme une unité PDU, son type peut alors être déclaré:

– soit comme identificateur d'unité PDU pour indiquer que, dans la contrainte relative à l'unité PDU, ce champ peut être chaîné à une contrainte d'unité PDU d'un type PDU spécifique;

– soit sous la forme du symbole **PDU** pour indiquer que, dans la contrainte relative à l'unité PDU, ce champ peut être chaîné à une contrainte d'unité PDU d'un type PDU quelconque;

et l'attribut facultatif étant Length (longueur),

la spécification de cet attribut pouvant restreindre ce champ à une longueur ou à un intervalle de longueurs particulier, conformément au 11.18. Les valeurs de longueur s'interprètent conformément au Tableau 5 en 11.18. Les limites seront spécifiées en termes de valeurs littérales "entières" non négatives, de paramètres de suite de tests, de constantes de suite de tests ou du mot clé INFINITY (infini).

Les spécifications de longueur définies pour le type de champ d'unité PDU dans les définitions de suite de tests ne doivent pas être contraires aux spécifications de longueur figurant dans la définition du type d'unité PDU, c'est-à-dire que l'ensemble de chaînes défini par restriction de longueur figurant dans la définition d'une unité PDU doit former un sous-ensemble strict de l'ensemble de chaînes précisé dans la définition du type de la suite de tests.

Le mot clé INFINITY peut servir de limite supérieure pour indiquer une longueur maximale illimitée;

3) de manière facultative, un identificateur de codage spécifique suivi par toute liste de paramètres effectifs nécessaires, afin de spécifier un codage explicite pour les champs individuels d'une unité PDU, qui a priorité sur les règles de codage et les variations de codage applicables à l'unité PDU dans son ensemble; l'identificateur de codage, s'il y en a, doit identifier soit l'une des variations de codage, soit une définition de codage de champ non valide précisée dans la suite de tests [par exemple LD(10)]; voir 11.16.4.

Les champs des définitions des types d'unité PDU sont considérés comme facultatifs, c'est-à-dire que dans des instances de ces types, des champs entiers peuvent être omis.

Ces informations seront fournies dans le format illustré dans le Formulaire 28, ci-dessous.

Définition de type d'unité PDU			
Nom de l'unité PDU : *PDU_Id&FullId*			
Groupe : *[PDU_GroupReference]*			
Type de point PCO : *[PCO_TypeIdentifier]*			
Nom de la règle de codage : *[EncodingRuleIdentifier]*			
Variation de codage : *[EncVariationCall]*			
Commentaires : *[FreeText]*			
Nom du champ	**Type du champ**	**Codage du champ**	**Commentaires**
⋮	⋮	⋮	⋮
PDU_FieldIdOrMacro	*Type&Attributes*	*[PDU_FieldEncodingCall]*	*[FreeText]*
⋮	⋮	⋮	⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 28 – Définition de type d'unité PDU

Les colonnes nom du champ et type du champ doivent être soit toutes deux présentes, soit toutes deux absentes.

DÉFINITION SYNTAXIQUE:

382 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
385 PDU_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PDU_GroupIdentifier "/"}
263 PCO_TypeIdentifier ::= Identifier
452 EncodingRuleIdentifier ::= Identifier
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]
391 PDU_FieldIdORMacro ::= PDU_FieldId&FullId | MacroSymbol
396 Type&Attributes ::= (Type [LengthAttribute]) | PDU
515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall

EXEMPLE 31 – Définition d'un type courant d'unité PDU:

Définition de type d'unité PDU		
Nom de l'unité PDU : INTC (interrupt Confirm)		
Type du point PCO : NSAP		
Nom du champ	**Type du champ**	**Commentaires**
GFI	BITSTRING	Identificateur général de format
LCGN	BITSTRING	Numéro du groupe de canaux logiques
LCN	BITSTRING	Identificateur de canal logique
PTI	OCTETSTRING	Identificateur de type de paquet
EXTRA	OCTETSTRING	Pour créer des paquets INTC (confirmation d'interruption) longs

11.15.3 Utilisation de types structurés dans les définitions d'unités PDU

Les définitions d'unités PDU peuvent se référer de deux manières possibles à un type structuré:

a) si la définition contient un nom de champ, le type structuré mentionné forme une sous-structure. Ceci permet de définir des unités PDU à structure de champs multiniveau;

b) si on utilise le macrosymbole (<-) à la place d'un nom de champ, cela équivaut alors à un développement de macro; l'entrée dans la définition du type d'unité PDU se développe directement en une liste de champs sans introduire de niveau supplémentaire de sous-structure.

Ce macrosymbole ne doit pas être utilisé sur la même ligne que des références à des types ASN.1 ou à des types simples; en d'autres termes, seuls des types structurés définis sous forme tabulaire peuvent être développés dans d'autres types structurés en tant que développements de macro.

11.15.4 Définition de types d'unités PDU à l'aide de la notation ASN.1

Les unités PDU peuvent, si nécessaire, être spécifiées en notation ASN.1. On a alors recours à une définition ASN.1 en utilisant la syntaxe ASN.1 telle qu'elle est définie dans la Recommandation X.680. Les informations suivantes seront fournies pour chaque unité PDU en notation ASN.1:

a) son nom,

   c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondante; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à l'unité PDU,

   ce type de point PCO étant l'un de ceux qui sont employés dans les déclarations des points PCO; si une unité PDU est toujours envoyée ou reçue imbriquée dans des primitives ASP, la spécification du type de point PCO dans la définition du type d'unité PDU devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type de l'unité PDU devient facultative;

c) les règles de codage à utiliser pour les unités PDU de ce type,

   afin de spécifier un codage explicite pour des unités PDU complètes, qui a priorité sur les règles de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des définitions de codage appropriée (par exemple pour passer des règles BER aux règles DER). Si cette entrée n'est pas utilisée, les règles de codage globales par défaut s'appliquent. Voir 11.16.4;

d) les variations de codage à utiliser pour les unités PDU de ce type,

afin de spécifier des variations de codage explicites pour des unités PDU complètes, qui ont priorité sur les variations de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des variations de codage appropriée [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage globales par défaut s'appliquent. Voir 11.16.4;

e) la définition ASN.1 du type d'unité PDU,

qui respectera la syntaxe définie dans la Recommandation X.680, sauf qu'il existe une option supplémentaire permettant de spécifier une variation de codage ou un codage de champ non valide en association soit avec la totalité du type ASN.1 (ASN1_Type) ou avec tout type ASN.1 à l'intérieur du type général ASN.1 (ASN1_Type). A cet effet, on donne un identificateur de codage spécifique suivi de toute liste de paramètres effectifs nécessaires, afin de spécifier un codage explicite pour les champs individuels ou d'autres sous-types d'une unité PDU, qui a priorité sur les règles de codage et les variations de codage applicables à l'unité PDU dans son ensemble; l'identificateur de codage, s'il y en a, doit identifier soit l'une des variations de codage, soit une définition de codage de champ non valide définie dans la suite de tests [par exemple LD(10)]; voir 11.16.4.

Le symbole trait d'union (-) ne doit pas figurer dans les identificateurs de cette définition; il peut être remplacé par le symbole de soulignement (_). L'identificateur d'unité PDU porté dans l'en-tête de la table est le nom du premier type défini dans le corps de la table.

Les types référencés à partir de la définition de l'unité PDU seront définis soit dans d'autres tables de définition de type en notation ASN.1, soit par référence dans la table de référence des types en notation ASN.1, soit enfin localement dans la même table, à la suite de la définition du premier type. Les types définis localement ne seront pas utilisés ailleurs dans la suite de tests.

Des commentaires ASN.1 peuvent être utilisés dans le corps de la table. La colonne commentaires n'existe pas.

Les commentaires en notation ASN.1 commencent par "--" et finissent soit par la prochaine occurrence de "--" ou par l'indication "fin de ligne", suivant ce qui se produit en premier. Cela évite qu'un simple commentaire ASN.1 s'étale sur plusieurs lignes. L'indication "fin de ligne" n'est toutefois pas un symbole défini dans la notation TTCN.MP. Il est par conséquent recommandé aux spécificateurs de suites ATS de faciliter l'échange de suites ATS en notation TTCN.MP en terminant toujours les commentaires en notation ASN.1 par "--".

Ces informations seront fournies dans le Formulaire 29, ci-dessous.

Définition de type d'unité PDU en notation ASN.1	
Nom de l'unité PDU	**:** *PDU_Id&FullId*
Groupe	**:** *[ASN1_PDU_GroupReference]*
Type de point PCO	**:** *[PCO_TypeIdentifier]*
Nom de la règle de codage	**:** *[EncodingRuleIdentifier]*
Variation de codage	**:** *[EncVariationCall]*
Commentaires	**:** *[FreeText]*
Définition du type	
ASN1_Type&LocalTypes	
Commentaires détaillés:	*[FreeText]*

Formulaire 29 – Définition de type d'unité PDU en notation ASN.1

DÉFINITION SYNTAXIQUE:

382 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
409 ASN1_PDU_Grouppreference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_PDU_GroupIdentifier"/"}
263 PCO_TypeIdentifier ::= Identifier
452 EncodingRuleIdentifier ::= Identifier
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]
121 ASN1_Type&LocalTypes ::= ASN1_Type {ANS1_LocalType}

EXEMPLE 32 – Définition ASN.1 d'une unité FTAM (transfert, accès et gestion de fichiers):

Définition du type d'unité PDU à l'aide de la notation ASN.1	
Nom de l'unité PDU	**:** *F_INIT (F_INITIALIZE_response)*
Type du point PCO	**:**
Commentaires	**:**
Définition du type	

SEQUENCE {
state_result State_result DEFAULT success,
action_result Action_Result multiple success,
protocol_id Protocol_Version,
}	

11.15.5 Définitions par référence de types d'unités PDU en notation ASN.1

Les unités PDU peuvent être spécifiées par une référence précise à une unité PDU en notation ASN.1 définie dans une Recommandation sur l'OSI ou par référence à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les identificateurs, les références de type et les références de valeur en notation ASN.1 peuvent contenir des traits d'union. Pour pouvoir utiliser des définitions importées en notation TTCN, il est nécessaire de remplacer les traits d'union par des soulignés (voir A.4.2.1).

Les informations suivantes seront fournies pour chaque unité PDU:

a) son nom,

   qui peut être utilisé dans toute la suite de tests;

b) le type de point PCO associé à cette unité PDU,

   ce type devant être l'un de ceux qui sont mentionnés dans les déclarations de points PCO; si une unité PDU n'est envoyée ou reçue qu'imbriquée dans des primitives ASP dans toute la suite de tests, la spécification du type de point PCO devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO devient facultative dans la définition du type d'unité PDU;

c) la référence du type,

   qui respectera les règles relatives aux identificateurs énoncées dans la Recommandation X.680;

d) l'identificateur du module,

   composé d'une référence de module conforme aux règles relatives aux identificateurs énoncées dans la Recommandation X.680 et d'un identificateur facultatif d'objet ObjectIdentifier;

e) les règles de codage à utiliser pour les unités PDU de ce type,

   afin de spécifier un codage explicite pour des unités PDU complètes, qui a priorité sur les règles de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des définitions de codage appropriée (par exemple pour passer des règles BER aux règles DER). Si cette entrée n'est pas utilisée, les règles de codage globales par défaut s'appliquent. Voir 11.16.4;

f) les variations de codage à utiliser pour les unités PDU de ce type,

   afin de spécifier des variations de codage explicites pour des unités PDU complètes, qui ont priorité sur les variations de codage globales par défaut pour la suite de tests dans son ensemble, cette entrée facultative doit faire référence à une entrée dans la table des variations de codage appropriée [par exemple pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les variations de codage globales par défaut s'appliquent. Voir 11.16.4.

Ces informations seront fournies dans le Formulaire 30, ci-dessous.

Définition par référence de types d'unités PDU en notation ASN.1						
Groupe : *[ASN1PDU_GroupReference]*						
Nom de l'unité PDU	**Type de point PCO**	**Référence du type**	**Identificateur de module**	**Règles de codage**	**Variation de codage**	**Commentaires**
⋮ *PDU_Id&FullId* ⋮	⋮ *[PCO_TypeIdentifier]* ⋮	⋮ *TypeReference* ⋮	⋮ *ModuleIdentifier* ⋮	⋮ *[EncodingRuleIdentifier]* ⋮	⋮ *[EncVariationCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*						

Formulaire 30 – Définition par référence de types d'unités PDU en notation ASN.1

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

409 ASN1_PDU_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_GroupIdentifier "/"}
382 PDU_Id&FullId ::= PDU_Identifier {FullIdentifier}
263 PCO_TypeIdentifier ::= Identifier
131 TypeReference ::= typereference
133 ASN1_ModuleIdentifier ::= ModuleIdentifier
452 EncodingruleIdentifier ::= ModuleIdentifier
511 EncVariationCall ::= EncVariationIdentifier {ActualParList}

11.16 Informations de codage de suite de tests

11.16.1 Définitions de codage

Afin de faciliter la spécification et le test des règles de codage d'un protocole OSI, quand une certaine souplesse est admise dans les règles de codage applicables au protocole, il convient de fournir une définition de codage. Si une telle définition de codage est fournie, une référence doit être donnée, dans la suite ATS, à la spécification dans laquelle les règles de codage sont précisées. La référence peut être faite à la spécification de protocole elle-même, ou à une spécification distincte de règles de codage. Si une telle référence ne peut être donnée, c'est-à-dire si les règles de codage du protocole ne sont pas normalisées, les règles de codage ne doivent pas être testées.

Les informations suivantes seront fournies pour chaque ensemble de règles de codage relatives au protocole:

a) le nom de la règle de codage, qui est un identificateur unique à utiliser dans l'ensemble de la suite de tests pour faire référence à une définition de codage;

b) la référence à la Recommandation appropriée qui définit les règles de codage;

c) une expression par défaut, identifiant les règles de codage à utiliser par défaut; cette expression prendra une valeur booléenne et utilisera seulement des valeurs littérales, des paramètres de suite de tests et des constantes de suite de tests dans ses termes;

d) facultativement, d'autres commentaires, dans la colonne commentaires, ou dans la zone des commentaires détaillés de la table.

Dans le cas où plusieurs ensembles de règles de codage pourraient être utilisés pour un protocole, le nom des règles de codage doit être énuméré dans la colonne nom de la règle de codage de la table des définitions de codage. Le nom de la règle de codage associée à l'expression par défaut qui prend la valeur TRUE doit être choisi comme l'ensemble par défaut pour la suite de tests. Si, dans la table des définitions de codage, plus d'une expression par défaut prend la valeur TRUE, ou si aucune expression par défaut ne prend la valeur TRUE, il s'agit d'une erreur de test élémentaire. Si aucune expression par défaut n'est spécifiée, c'est l'équivalent de la spécification de la valeur FALSE.

Ces informations seront fournies dans le Formulaire 31, ci-dessous.

Définitions de codage			
Groupe : *[EncodingGroupReference]*			
Nom de la règle de codage	**Référence**	**Valeur par défaut**	**Commentaires**
⋮ *EncodingRuleIdentifier* ⋮	⋮ *EncodingReference* ⋮	⋮ *[DefaultExpression]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 31 – Définitions de codage

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

448 EncodingGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {EncodingGroupIdentifier "/"}
452 EncodingRuleIdentifier ::= Identifier
454 EncodingReference ::= BoundedFreeText
456 DefaultExpression ::= Expression

Les règles de codage spécifiées dans ce formulaire ne s'appliquent qu'aux unités PDU.

EXEMPLE 33 – Définitions de codage:

Définitions de codage			
Nom de la règle de codage	**Référence**	**Expression par défaut**	**Commentaires**
BER	Recommandations X.690	TRUE	Règles de codage de base
PER	Recommandations X.690		Règles de codage compact
DER	Recommandations X.690		Règles de codage distinctives
Commentaires détaillés: *[FreeText]*			

11.16.2 Variations de codage

Les variations admissibles de chaque définition de codage qui peut être utilisée dans la suite de tests peuvent être fournies.

Pour définir de telles variations de codage, les informations suivantes seront fournies:

a) le nom de la règle de codage, c'est-à-dire le nom des règles de codage identifiées dans la table des définitions de codage à laquelle cette variation s'applique;

b) une liste facultative des types, énumérant les types auxquels cette variation de codage pourrait s'appliquer; une liste vide signifie que les variations de codage peuvent être appliquées à n'importe quel champ de l'unité PDU. Les types peuvent être n'importe quel type d'unité PDU ou tout type pouvant apparaître dans une unité PDU;

c) une liste des variations de codage,

les informations suivantes étant fournies pour chaque variation de codage:

1) le nom de la variation de codage, qui est un identificateur unique se rapportant à une définition admise de codage pour un type spécifique, tel qu'il est compris dans la spécification de règles de codage appropriée;

2) une référence, servant à identifier la section de la spécification des règles de codage qui décrit cet ensemble de variations de codage;

3) une expression par défaut, indiquant la variation de codage à utiliser par défaut; cette expression prendra une valeur booléenne et utilisera seulement des valeurs littérales, des paramètres de suite de tests et des constantes de suite de tests dans ses termes;

d) facultativement, d'autres commentaires, dans la partie commentaires de l'en-tête de la table, dans la colonne commentaires ou dans la zone des commentaires détaillés de la table.

La variation de codage associée à l'expression qui prend la valeur TRUE doit être choisie comme la variation de codage par défaut pour la liste donnée des types, le cas échéant, ou sinon pour tous les types figurant dans la suite de tests. Si plus d'une expression par défaut de la table des variations de codage prend la valeur TRUE, il s'agit d'une erreur de test élémentaire. Si aucune expression par défaut n'est spécifiée pour une variation de codage, c'est l'équivalent de la spécification de la valeur FALSE. Si aucune expression par défaut n'est spécifiée, ou si elles prennent toutes la valeur FALSE, c'est la première variation de codage qui doit être considérée comme la variation par défaut.

Les variations de codage seront fournies dans le format illustré dans le Formulaire 32, ci-dessous.

Variations de codage			
Groupe : *[EncVariationGroupReference]*			
Nom de la règle de codage : *EncodingRuleIdentifier*			
Liste des types : *[TypeList]*			
Commentaires : *[FreeText]*			
Variation de codage	**Référence**	**Expression par défaut**	**Commentaires**
⋮ *EncVariationId&ParList* ⋮	⋮ *VariationReference* ⋮	⋮ *[DefaultExpression]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 32 – Variations de codage

DÉFINITION SYNTAXIQUE:

463 EncVariationGroupreference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {EncVariationGroupIdentifier "/"}
452 EncodingRuleIdentifier ::= Identifier
467 TypeList ::= Type {Comma Type}
470 EncVariationId&ParList ::= EncVariationIdentifier [FormalParList]
473 Variationreference ::= BoundedFreeText
456 DefaultExpression ::= Expression

EXEMPLE 34 – Variations de codage:

Variations de codage			
Nom de la règle de codage : BER			
Liste des types : Length			
Commentaires : Length est défini comme étant du type INTEGER (entier).			
Variation de codage	**Référence**	**Expression par défaut**	**Commentaires**
SD LD(len: INTEGER)	6.3.3.1 6.3.3.2	TRUE	
Commentaires détaillés:			

11.16.3 Définitions de codage de champ non valides

Afin de tester à fond les règles de codage, il peut être nécessaire de définir des variations interdites des définitions de codage utilisées par le protocole. Des définitions de codage de champ non valides peuvent être fournies pour n'importe lequel des types utilisés dans les champs d'unité PDU de la suite de tests. Une fois définie, une définition de codage de champ non valide peut être utilisée pour avoir priorité sur le codage normal d'une valeur de champ de contrainte d'unité PDU spécifique du même type (voir 13.4).

Les informations suivantes relatives à une définition de codage de champ non valide seront fournies:

a) un nom de codage de champ non valide, qui est un identificateur particulier à utiliser dans l'ensemble de la suite de tests pour faire référence à cette définition de codage de champ non valide, suivi d'une liste facultative de paramètres formels;

b) une liste de types facultative, énumérant les types auxquels ce codage pourrait s'appliquer; une liste vide signifie que la définition de codage peut être appliquée à n'importe quel champ d'une unité PDU;

c) une définition d'opération de codage qui définit la manière dont les valeurs doivent être codées,

   cette définition étant une définition de procédure ayant la même forme que la définition de procédure d'une opération de suite de tests (voir 11.3.4) et qui, lorsqu'elle est évaluée, résulte en l'évaluation d'une déclaration ReturnValue pour donner le résultat de l'opération, y compris les commentaires explicatifs incorporés à la définition de procédure aux endroits appropriés, en tant que texte délimité par "/*" et "*/"; les commentaires explicatifs doivent comprendre un exemple montrant un appel; le résultat de l'opération de codage sera une chaîne binaire, avec un ordre défini de transmission, qui représentera la valeur correspondante;

d) facultativement, d'autres commentaires décrivant l'opération, soit dans la partie commentaires de l'en-tête de la table ou dans la zone commentaires détaillés de la table.

Il est recommandé d'utiliser des définitions de procédure afin de définir avec précision les opérations.

Si une liste de paramètres formels est spécifiée, les valeurs transférées à l'opération de codage servent à effectuer le codage du champ d'unité PDU. Chaque paramètre formel doit être déclaré comme étant un type prédéfini, un identificateur de type de suite de tests ou un identificateur de type d'unité PDU. Par exemple, une valeur entière peut être transférée à une opération de codage qui calcule la longueur d'un champ d'unité PDU. La manière dont on utilise les paramètres transférés à l'opération doit être expliquée dans la définition de l'opération de codage.

Il convient d'utiliser un formulaire pour chaque définition de codage de champ non valide.

Les définitions d'opération de codage de champ non valides seront fournies dans le Formulaire 33, ci-dessous.

Définition d'opération de codage de champ non valide	
Groupe	**:** *[InvalidFieldEncodingGroupReference]*
Nom de l'opération	**:** *InvalidFieldEncodingId&ParList*
Type de résultat	**:** *[TypeList]*
Commentaires	**:** *[FreeText]*
Définition	
TS_OpProcDef	
Commentaires détaillés:	*[FreeText]*

Formulaire 33 – Définition d'opération de codage de champ non valide

DÉFINITION SYNTAXIQUE:

484 InvalidFieldEncodingGroupReference:: =
[(SuiteIdentifier | TTCN_ModuleIdentifier)"/"](InvalidFieldEncodingGroupIdentifier"/")

481 InvalidFieldEncodingId&ParList ::= InvalidFieldEncodingIdentifier [FormalParList]

467 TypeList ::= Type {Comma Type}

162 TS_OpProcDef ::= [VarBlock]ProcStatement

11.16.4 Application des règles de codage

Les règles de codage spécifiées dans la suite de tests s'appliquent à toutes les unités PDU envoyées ou reçues dans la partie comportement. Des règles de codage peuvent être spécifiées pour la totalité de la suite de tests ou pour des déclarations de type ou encore pour des déclarations de contraintes, comme l'indique le Tableau 4. Les endroits du Tableau 4 marqués ✪ indiquent le domaine d'application permis de chaque sorte d'information de codage.

Tableau 4/X.292 – Applicabilité des définitions de codage

		Définitions de codage				
		Règles de codage		**Variations de codage**		**Codage de champ non valide**
Priorité	**Domaine d'application**	**Expression par défaut**	**Autre**	**Expression par défaut**	**Autre**	
Minimale	Suite de tests	✪		✪		
	Déclarations de type					
	Unités PDU		✪	✪	✪	
	Types structurés ou types ASN.1			✪	✪	
	Types simples ou champs/ éléments d'unité PDU			✪	✪	✪
	Déclarations de contraintes					
	Unités PDU		✪	✪	✪	
Maximale	Types structurés ou types ASN.1			✪	✪	
	Champs/ éléments d'unité PDU			✪	✪	✪
Priorité dans un rang		Minimale				Maximale

Les règles de codage doivent être appliquées selon les valeurs prioritaires des rangées indiquées dans la première colonne du Tableau 4, "(4)" ayant la plus grande priorité et "(1)" la plus faible. Dans une rangée, la priorité va de gauche à droite, l'entrée la plus à droite ayant la plus grande priorité. Par conséquent, les règles de codage de champ de contrainte ont priorité sur toutes les autres, tandis que les règles de codage par défaut appliquées au niveau de la suite de tests peuvent être supplantées par n'importe quelle autre méthode de spécification. Les règles de codage réelles à utiliser pour une unité PDU, une fois que toutes les priorités ont été appliquées, sont appelées les règles de codage applicables.

Si aucune information de codage n'est spécifiée pour une contrainte de type structuré ou de type ASN.1, elle "hérite" des règles de codage appliquées au niveau de l'unité PDU. Ainsi, les règles de codage appliquées à une contrainte de type structuré ou de type ASN.1 varieront, selon l'unité PDU dans laquelle elles sont utilisées. Par contre, si des informations de codage sont spécifiées pour une contrainte du type structuré ou du type ASN.1, ces informations auront priorité sur les informations de codage de chaque unité PDU dans laquelle elles seront utilisées. Si une telle contrainte de type structuré ou de type ASN.1 est utilisée dans une primitive ASP, il n'est pas tenu compte des informations de codage.

Pour les événements de RÉCEPTION, si aucune règle de codage spécifique ne s'applique à l'unité PDU entrante, on peut appliquer un codage selon toute variation permise par la définition de codage applicable (par exemple toute forme de codage de longueur permise par les règles de codage de base BER).

11.17 Définitions de types de messages de coordination (CM)

11.17.1 Introduction

Les paramètres de message CM peuvent être de n'importe quel type pouvant être spécifié en notation TTCN. Les messages CM simples peuvent ne contenir aucun paramètre associé ou peuvent comprendre un seul paramètre, par exemple un nombre naturel, un résultat préliminaire ou une chaîne de caractères comme "suspend" ou "continue". Les messages CM plus complexes peuvent véhiculer des informations supplémentaires, par exemple une unité PDU complète, un champ d'unité PDU ou la valeur lue à partir d'un temporisateur. Il n'existe pas de message CM prédéfini.

11.17.2 Définitions de types de messages CM à l'aide des tables

Les types de messages CM peuvent être déclarés à l'aide des tables TTCN. Les informations suivantes seront fournies pour chaque type de message CM:

a) son nom,

   chaque nom étant unique dans la suite de tests;

b) une liste des paramètres associés au message CM,

   dans laquelle les informations suivantes seront fournies pour chaque paramètre:

   1) son nom,

      qui est unique dans le message CM;

   2) son type et un attribut facultatif,

      comme pour les champs d'unité PDU,

      auquel cas la spécification peut restreindre le champ à une longueur ou à un intervalle particulier, selon 11.18. Les valeurs de longueur doivent être interprétées d'après le Tableau 5 en 11.18. Les limites doivent être spécifiées en fonction de valeurs littérales "entières" non négatives, de paramètres de suite de tests, de constantes de suite de tests ou du mot clé INFINITY (infini).

      Les spécifications de longueur définies dans le type de champ d'unité PDU dans les définitions de type de suite de tests ne doivent pas contredire les spécificationns de longueur données dans la définition de type d'unité PDU, c'est-à-dire que l'ensemble de chaînes défini par une restriction de longueur dans une définition d'unité PDU doit être un sous-ensemble strict de l'ensemble de chaînes précisé par la définition de type de suite de tests.

      Le mot clé INFINITY peut être utilisé comme valeur de la limite supérieure pour indiquer qu'il n'y a pas de limite supérieure de longueur.

      Tous les paramètres des messages CM sont facultatifs, c'est-à-dire qu'ils peuvent être omis lorsque le message CM est utilisé.

Ces informations seront fournies dans le format illustré dans le Formulaire 34, ci-dessous.

Définition de types de messages CM		
Nom du message CM : *CM_Identifier*		
Groupe : *[CM_GroupReference]*		
Commentaires : *[FreeText]*		
Nom du paramètre	**Type de paramètre**	**Commentaires**
⋮	⋮	⋮
CM_ParIdOrMacro	*Type&Attributes*	*[FreeText]*
⋮	⋮	⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 34 – Définition de types de messages CM

Les colonnes nom du paramètre et type de paramètre doivent soit être toutes deux présentes, soit toutes deux absentes.

DÉFINITION SYNTAXIQUE:

424 CM_Identifier ::= Identifier

426 CM_GroupReference ::= [(SuiteIdentifier|TTCN_ModuleIdentifier) "/"] {CM_GroupIdentifier "/"}

431 CM_ParIdOrMacro ::= CM_ParIdentifier|MacroSymbol

396 Type&Attributes ::= (Type[LengthAttribute]) |PDU

11.17.3 Définitions de type de message CM à l'aide de la notation ASN.1

Les types de message CM peuvent être déclarés à l'aide de la notation ASN.1. Les informations suivantes seront fournies pour chaque type de message CM en notation ASN.1:

a) son nom,

chaque nom étant unique dans la suite de tests;

b) la définition de type de message CM en notation ASN.1,

qui doit suivre la syntaxe définie dans la Recommandation X.680. Pour les identificateurs de cette définition, le symbole trait d'union (-) ne doit pas être utilisé. Le symbole soulignement (_) peut être utilisé à sa place. L'identificateur d'unité PDU de l'en-tête de la table est le nom du premier type défini dans le corps de la table.

Les types auxquels il est fait référence à partir de la définition d'unité PDU doivent être définis dans d'autres tables de définition de type en notation ASN.1, par référence dans la table de référence de types en notation ASN.1 ou être définis localement dans la même table, à la suite de la définition du premier type. Les types définis localement ne doivent pas être utilisés dans d'autres parties de la suite de tests.

Des commentaires en notation ASN.1 peuvent être utilisés dans le corps de la table. La colonne commentaires ne doit pas apparaître dans cette table.

Les commentaires en notation ASN.1 commencent par "--" et finissent soit par la prochaine occurrence de "--" ou par l'indication "fin de ligne", suivant ce qui se produit en premier. Cela évite qu'un simple commentaire ASN.1 s'étale sur plusieurs lignes. L'indication "fin de ligne" n'est toutefois pas un symbole défini dans la notation TTCN.MP. Il est par conséquent recommandé aux spécificateurs de suites ATS de faciliter l'échange de suites ATS en notation TTCN.MP en terminant toujours les commentaires en notation ASN.1 par "--".

Ces informations seront fournies dans le format illustré dans le Formulaire 35, ci-dessous.

Définition de types de messages CM en notation ASN.1	
Nom du message CM	**:** *CM_Identifier*
Groupe	**:** *[ASN1CM_GroupReference]*
Commentaires	**:** *[FreeText]*
Définition du type	
ASN1_Type&LocalTypes	
Commentaires détaillés:	*[FreeText]*

Formulaire 35 – Définition de types de messages CM en notationASN.1

DÉFINITION SYNTAXIQUE:

424 CM_Identifier ::= Identifier

440 ASN1_CM_GroupReference ::= [(SuiteIdentifier|TTCN_ModuleIdentifier) "/"]{ASN1_CM_GroupIdentifier "/"}

121 ASN1_Type&LocalTypes ::= ASN1_Type{ASN1_LocalType}

11.18 Spécifications de longueur des chaînes

11.18.1 La notation TTCN prévoit la spécification de restrictions de longueur pour les types de chaînes (BITSTRING, HEXSTRING, OCTETSTRING et pour tous les types de chaînes de caractères CharacterString), plus les types BIT STRING et OCTET STRING en notation ASN.1, dans les cas suivants:

a) lors de la déclaration de types de suite de tests en tant que restrictions de type;

b) lors de la déclaration de paramètres simples de primitives ASP, de champs simples d'unités PDU ou d'éléments simples de types structurés en tant qu'attributs de paramètre, de champ ou de type d'élément;

c) lors de la définition de contraintes de primitives ASP, d'unités PDU ou de type structuré en tant qu'attribut d'une valeur de contrainte.

11.18.2 Les spécifications de longueur peuvent être indiquées dans les formats suivants:

a) [Length]

restreignant la longueur des valeurs des chaînes d'un type donné à la seule valeur *Length*;

b) [MinLength **TO** MaxLength] ou [MinLength .. MaxLength]

spécifiant une longueur minimale et une longueur maximale pour les valeurs d'un type de chaîne donné.

Les limites de longueur: *Length*, *MinLength* et *MaxLength* auront divers degrés de complexité selon l'endroit où elles sont utilisées. Dans tous les cas, elles prendront des valeurs entières non négatives. On peut également utiliser le mot clé INFINITY pour indiquer que la longueur maximale est illimitée. Lorsqu'un intervalle de longueurs est spécifié, la limite inférieure sera celle de gauche.

Dans le contexte des contraintes, des restrictions de longueur peuvent également être spécifiées pour les valeurs des types SEQUENCE OF et SET OF, limitant ainsi le nombre de leurs éléments.

Les unités de longueur des différents types de chaînes sont indiquées dans le Tableau 5.

Tableau 5/X.292 – Unités de longueur utilisées pour les spécifications de longueur de champ

Type	Unités de longueur
BITSTRING ou BIT STRING	Bits
HEXSTRING	Chiffres hexadécimaux
OCTETSTRING ou OCTET STRING	Octets
CharacterString	Caractères
SEQUENCE OF	Nombre d'éléments du type de base
SET OF	Nombre d'éléments du type de base

Les spécifications de longueur ne doivent pas être contradictoires, c'est-à-dire qu'une restriction portant sur un type (un ensemble de valeurs) lui-même déjà restreint doit spécifier un sous-ensemble des valeurs de ce type.

EXEMPLE 35 – Spécification de longueur:

Si l'on considère les définitions suivantes de types ASN.1:

type1 ::= OCTETSTRING [0 .. 25]

type2 ::= type1 [15 .. 24]

la restriction de longueur portant sur le type2 est correcte puisque le type2 comprend toutes les chaînes OCTETSTRING d'une longueur minimale de 15 et d'une longueur maximale de 24, qui constituent un sous-ensemble strict de toutes les chaînes OCTETSTRING d'une longueur maximale de 25. Par contre, la définition:

type2 ::= type1[15 .. 30]

n'est pas valide puisqu'elle contient des valeurs non incluses dans le type1.

11.19 Définition de primitives ASP, d'unités PDU et de messages CM pour les événements SEND (envoi)

Dans les primitives ASP et les unités PDU envoyées depuis le testeur, les valeurs des paramètres ASP et des champs PDU définies dans la partie contraintes (voir les paragraphes 12, 13 et 14) correspondront à la définition de ces paramètres ou champs. En d'autres termes:

a) cette valeur sera du type spécifié pour ce paramètre de primitive ASP ou pour ce champ d'unité PDU;

b) chacune des valeurs respectera toutes les restrictions de longueur associées à ce type;

c) les valeurs de champ d'unité PDU doivent être codées conformément aux règles de codage applicables.

Les opérations de codage définies dans la suite de tests sont exécutées implicitement comme parties de l'événement SEND. Les valeurs par défaut et les éléments prioritaires sont appliqués, selon les besoins. Par conséquent, à la sortie de l'événement SEND se trouvent les données codées à transférer au fournisseur de service approprié.

11.20 Définition de primitives ASP, d'unités PDU et de messages CM pour les événements RECEIVE (réception)

Le type de primitives ASP et d'unités PDU pouvant être reçues par le testeur définit la classe de primitives ASP et d'unités PDU entrantes pouvant concorder avec une spécification d'événement de ce type. Une primitive ASP ou une unité PDU entrante est considérée comme appartenant à cette classe si et seulement si:

a) les valeurs des paramètres de la primitive ASP et des champs de l'unité PDU sont du type spécifié dans la définition de la primitive ASP et de l'unité PDU;

b) cette valeur répond à toutes les restrictions de longueur associées à ce type;

c) les valeurs de champ d'unité PDU peuvent être décodées conformément aux règles de codage applicables.

Dans tous les autres cas, une primitive ASP ou une unité PDU est considérée comme ne concordant pas avec une spécification d'événement de ce type.

Dans le cas de primitives ASP ou d'unités PDU possédant une sous-structure, qu'il s'agisse de types structurés ou de types ASN.1, les règles ci-dessus s'appliquent récursivement aux champs des sous-structures.

11.21 Définitions des alias (pseudonymes)

11.21.1 Introduction

Pour améliorer la lisibilité des descriptions comportementales en notation TTCN, il est possible d'utiliser un alias (pseudonyme) pour faciliter la redésignation des identificateurs de primitives ASP ou d'unités PDU dans les descriptions comportementales. Cette redésignation peut servir à souligner l'échange d'unités PDU imbriquées dans des primitives ASP.

Les informations suivantes seront fournies pour chaque pseudonyme:

a) un identificateur d'alias (pseudonyme);

b) son développement,

qui est lui-même un identificateur.

Ces informations seront fournies dans le format illustré dans le Formulaire 36, ci-dessous.

Définition des alias (pseudonymes)		
Groupe : *[AliasGroupReference]*		
Nom des alias (pseudonymes)	**Développement**	**Commentaires**
⋮ *AliasIdentifier* ⋮	⋮ *Expansion* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 36 – Définition des alias (pseudonymes)

Des commentaires collectifs peuvent être utilisés dans cette table conformément à la Figure 2.

DÉFINITION SYNTAXIQUE:

493 AliasGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {AliasGroupIdentifier "/"}

497 Alias Identifier ::= Identifier

499 Expansion ::=ASP_Identifier | PDU_Identifier

11.21.2 Développement des alias

On appliquera les règles suivantes:

a) un alias est un identificateur respectant les règles syntaxiques relatives aux identificateurs définies dans la notation TTCN.MP, c'est-à-dire qu'un alias est délimité par un caractère (symbole) quelconque non autorisé dans un identificateur en TTCN;

b) les alias ne sont pas transitifs: si un alias apparaît en tant que développement d'un autre alias, il ne sera pas développé (c'est-à-dire qu'il s'agit d'un développement en une passe);

c) un alias ne sera utilisé que pour remplacer un identificateur de primitive ASP ou d'unité PDU dans une seule déclaration TTCN dans un arbre comportemental. Il ne sera utilisé que dans une colonne de description comportementale;

d) le développement d'un alias respectera les règles syntaxiques relatives aux identificateurs, définies dans la notation TTCN.MP.

EXEMPLE 36 – Définition d'alias à partir d'une suite de tests de transport:

Définition des alias (pseudonymes)		
Nom des alias (pseudonymes)	**Développement**	**Commentaires**
CR	demande N_DATA	Alias de la primitive ASP de demande N_DATA, utilisé pour acheminer une unité CR_TPDU
DR	demande N_DATA	Alias de la primitive ASP de demande N_DATA, utilisé pour acheminer une unité DR_TPDU
CC	indication N_DATA	Alias de la primitive d'indication ASP N_DATA, utilisé pour acheminer une unité CC_TPDU

NOTE – Les alias étant traités comme des développements de macro-instructions, le terme AliasIdentifier (identificateur d'alias) n'apparaît pas en format BNF dans les lignes d'événement TTCN.

12 Partie contraintes

12.1 Introduction

Une suite ATS spécifiera les valeurs des paramètres des primitives ASP et des champs des unités PDU envoyées et reçues par le système de test. Dans la notation TTCN, la partie contraintes remplit ce rôle.

Les descriptions de comportement dynamique (voir le paragraphe 15) font référence aux contraintes pour élaborer les primitives ASP et les unités PDU sortantes des événements SEND (envoi), et pour spécifier le contenu prévu des primitives ASP et des unités PDU entrantes des événements RECEIVE (réception).

Ces contraintes peuvent être spécifiées sous l'une des deux formes suivantes:

a) contraintes sous forme tabulaire (voir le paragraphe 13);

b) contraintes en notation ASN.1 (voir le paragraphe 14).

Les valeurs réelles ou les contraintes des valeurs d'un message CM seront déclarées de la même manière que les contraintes des unités PDU.

12.2 Principes généraux

Le présent sous-paragraphe décrit les principes généraux régissant l'élaboration des contraintes pour les événements SEND (envoi) et l'étude de la concordance des événements RECEIVE (réception), et en définit les mécanismes. Ces principes sont communs aux deux formes de contraintes, tabulaire et ASN.1.

Les contraintes sont des spécifications détaillées des primitives ASP et des unités PDU. Normalement, chaque contrainte est définie spécifiquement aussi bien pour des événements SEND (envoi) que des événements RECEIVE (réception). Il n'est pas nécessaire de spécifier une contrainte lorsqu'une primitive ASP ou un message CM ne contient aucun paramètre ou lorsqu'une unité PDU ne contient aucun champ. Une même contrainte peut s'utiliser dans l'un ou l'autre contexte, sous réserve des restrictions de sémantique opératoire définies dans l'Annexe B.

La spécification de contrainte d'une primitive ASP ou d'une unité PDU aura la même structure que celle de la définition de type de cette primitive ASP ou de cette unité PDU.

Si une primitive ASP ou une unité PDU possède une sous-structure, les contraintes applicables aux primitives ASP et aux unités PDU de ce type auront la même structure tabulaire ou une structure ASN.1 compatible (c'est-à-dire pouvant comporter des groupages).

Dans une définition de primitive ASP ou d'unité PDU, les types structurés construits à l'aide du macrosymbole (<-) ne sont pas considérés comme des sous-structures. Les contraintes relatives à ces primitives ASP ou à ces unités PDU soit auront une structure complètement plate (les éléments de toute structure développée étant explicitement énumérés dans la contrainte de primitive ASP ou d'unité PDU), soit feront référence à une contrainte de structure correspondante pour un développement de macro.

Les contraintes spécifient les valeurs des paramètres de primitives ASP et des champs d'unités PDU à l'aide de diverses combinaisons de valeurs littérales, de références à des objets de données, d'expressions, de valeurs construites en notation ASN.1, de mécanismes spéciaux de concordance et de références à d'autres contraintes. Les contraintes s'appliquant à l'ensemble d'une unité PDU ou à un seul de ses champs peuvent aussi spécifier des règles de codage qui ont priorité sur les règles de codage générales appliquées à la suite de tests. Ces règles de codage peuvent être spécifiées pour l'ensemble de la contrainte ou pour un seul de ses champs.

Les contraintes peuvent utiliser les valeurs de tous les types TTCN et ASN.1. Les expressions utilisées dans les contraintes prendront une valeur spécifique lors de l'utilisation de ces contraintes pendant les événements d'envoi et de réception.

De quelque manière qu'on les obtienne, ces valeurs correspondront aux entrées de paramètre ou de champ dans les définitions des types de primitives ASP ou d'unités PDU. En d'autres termes:

a) la valeur sera du type spécifié pour ce paramètre ou champ;

b) la longueur obéira à toutes les restrictions associées à ce type.

Dans une contrainte, une expression ne comportera que des valeurs (y compris, par exemple, la valeur et les attributs de la contrainte), des paramètres de suite de tests, des constantes de suite de tests, des paramètres formels, des références aux composantes et des opérations de suite de tests.

Il est également possible d'utiliser une référence à une contrainte (éventuellement paramétrée) comme valeur de paramètre ou de champ (chaînage statique).

Les contraintes n'utiliseront ni des variables de suite de tests, ni des variables de test élémentaire, sauf si ces variables sont transmises comme paramètres effectifs. Dans ce cas, elles seront liées à une valeur et ne seront pas modifiées par l'occurrence d'un événement d'envoi ou de réception.

Les mécanismes de concordance sont définis au 12.6.2.

12.3 Paramétrage des contraintes

Les contraintes peuvent être paramétrées. Dans ce cas, le nom de la contrainte est suivi de la liste des paramètres formels entre parenthèses. Les paramètres formels sont utilisés pour spécifier les valeurs des paramètres de primitives ASP et des champs d'unités PDU dans cette contrainte.

Chaque nom de paramètre formel sera suivi de deux points ":" et du nom du type du paramètre. Si plusieurs paramètres sont du même type, ils peuvent être regroupés en sous-liste. Dans ce cas, les noms des paramètres du même type seront séparés par une virgule, le dernier étant suivi de deux points ":" et du nom du type commun à la sous-liste. Lorsqu'on utilise plus d'un couple paramètre/type (ou couple sous-liste/type), les couples seront séparés par un point-virgule ";".

Les valeurs littérales, les paramètres de suite de tests, les constantes de suite de tests, les variables de suite de tests, les variables de test élémentaire et les contraintes de type PDU ou suite de tests peuvent être transmis comme paramètres effectifs à une contrainte par une référence à une contrainte effectuée depuis une description comportementale. Ces paramètres ne seront pas du type point PCO ou primitive ASP.

12.4 Chaînage des contraintes

Les contraintes peuvent être chaînées en citant en référence une contrainte comme valeur de paramètre ou de champ dans une autre contrainte. Par exemple, la valeur du paramètre Data (données) d'une primitive ASP de demande de données de réseau N-DATA pourrait être une référence à une contrainte d'unité PDU de demande de connexion transport T-CRPDU, c'est-à-dire que la contrainte T-CRPDU est chaînée à la primitive de demande N-DATA.

Le chaînage des contraintes peut s'effectuer de deux manières:

a) par chaînage statique, dans lequel la valeur dans une contrainte d'un paramètre de primitive ASP ou d'un champ d'unité PDU, est une référence explicite à une autre contrainte;

b) par chaînage dynamique, dans lequel la valeur dans une contrainte d'un paramètre de primitive ASP ou d'un champ d'unité PDU, est un paramètre formel de cette contrainte. Lorsqu'une telle référence est appelée dans un comportement dynamique, le paramètre effectif correspondant à cette contrainte est une référence à une autre contrainte (voir les exemples de chaînages statique et dynamique donnés dans l'Appendice I).

Une contrainte ne peut faire référence à elle-même de manière récursive (directe ou indirecte) dans les déclarations de contraintes.

Le chaînage des contraintes n'est possible que lorsque les déclarations correspondantes ont été définies pour permettre le chaînage. Par exemple, lors du chaînage d'un paramètre d'une primitive ASP à une contrainte d'une unité PDU, le type déclaré pour le paramètre de la primitive ASP sera un type d'unité PDU approprié ou sera de métatype **PDU**. Dans les déclarations d'unité PDU en notation ASN.1, le type d'unité PDU sera défini comme un choix parmi tous les types d'unité PDU valides, alors que dans les déclarations sous forme tabulaire, le métatype **PDU** sera utilisé pour obtenir le même résultat. De même, lors du chaînage d'un champ d'une unité PDU à une contrainte de structure, le type déclaré pour le champ de l'unité PDU sera un type structuré approprié.

12.5 Contraintes relatives aux événements SEND (envoi)

Les contraintes données en référence d'un événement SEND (envoi) ne doivent pas contenir de caractères génériques [c'est-à-dire AnyValue (?) (valeur quelconque) ou AnyOrOmit (*) (quelconque ou omission)], sauf si ceux-ci sont des valeurs spécifiques explicitement affectées dans la ligne d'événement SEND (envoi) de la description du comportement.

Dans une contrainte tabulaire, tous les paramètres de primitive ASP et les champs d'unité PDU sont facultatifs et peuvent donc être omis en utilisant le symbole d'omission pour indiquer que ces paramètres ou ces champs sont absents de l'événement d'envoi.

Dans une contrainte ASN.1, seuls les paramètres de primitive ASP et les champs d'unité PDU déclarés facultatifs peuvent être omis. On peut les omettre soit en utilisant le symbole d'omission, soit simplement en ne mentionnant pas le paramètre de primitive ASP ou le champ d'unité PDU pertinent.

Aucun des mécanismes de concordance définis au 12.6.2, à l'exception de la valeur spécifique SpecificValue, ne fournit de valeur de paramètre de primitive ASP ou de champ d'unité PDU pour un événement d'envoi.

Lorsqu'on utilise des valeurs ASN.1 du type SET ou SET OF dans une contrainte, les valeurs des éléments de cet ensemble sont envoyées dans l'ordre spécifié par la contrainte correspondante.

12.6 Contraintes relatives aux événements RECEIVE (réception)

12.6.1 Valeurs de concordance

Si une contrainte est utilisée pour déterminer les valeurs des paramètres de primitive ASP ou des champs d'unité PDU avec lesquelles une primitive ASP ou une unité PDU reçue doit concorder, cette contrainte ne comportera que des valeurs spécifiques évaluées conformément au 12.6.3, ou des mécanismes spéciaux de concordance lorsqu'il n'est pas souhaitable ou possible de spécifier des valeurs particulières. Les mécanismes de concordance spécifient des concordances autres que la méthode d'égalité à une valeur spécifique.

Une primitive ASP ou une unité PDU entrante concorde avec une contrainte utilisée dans un événement RECEIVE si, et seulement si, toutes les conditions suivantes sont remplies:

a) tous les paramètres de la primitive ASP ou tous les champs de l'unité PDU sont du type spécifié dans les définitions de la primitive ASP ou de l'unité PDU;

b) la valeur, le jeu de caractères et la longueur obéissent aux restrictions associées à ce type;

c) les valeurs des paramètres de la primitive ASP ou des champs de l'unité PDU concordent correctement avec celles de la contrainte;

d) pour les unités PDU, les unités PDU ont été correctement décodées en tenant compte des règles de codage par défaut et des éléments prioritaires qui s'appliquent; si les règles de codage de l'unité PDU reçue ne sont pas les mêmes que celles qui ont été spécifiées pour la contrainte, alors l'unité PDU ne concordera pas.

Dans le cas de primitives ASP ou d'unités PDU dotées de sous-structures, qu'elles soient définies par des types structurés ou en ASN.1, les règles ci-dessus s'appliquent récursivement aux champs de la ou des sous-structures.

NOTE – Si un événement RECEIVE est qualifié par une expression booléenne, il y aura concordance si la primitive ASP ou l'unité PDU entrante concorde avec la contrainte et si le qualificateur prend la valeur TRUE.

12.6.2 Mécanismes de concordance

Le Tableau 6 donne un aperçu général des mécanismes de concordance pris en charge, ainsi que des symboles spéciaux et de leur domaine d'application. La colonne de gauche donne la liste des types ASN.1 et des types TTCN équivalents auxquels ces mécanismes de concordance s'appliquent. Les différents types de mécanismes de concordance, correspondant aux lignes du tableau, sont subdivisés verticalement en quatre groupes:

a) spécifications de valeurs;

b) spécificateurs de *substitution* (à des valeurs);

c) spécificateurs d'*appartenance* (à une valeur);

d) spécificateurs d'*attributs* (de valeurs).

Certains de ces spécificateurs peuvent être combinés comme indiqué dans les sous-paragraphes suivants.

La partie ombrée du Tableau 6 indique les mécanismes qui s'appliquent à la fois aux types prédéfinis TTCN et ASN.1.

Dans la spécification d'une contrainte, les mécanismes de concordance peuvent remplacer les valeurs des paramètres individuels de primitive ASP ou de champs individuels d'unité PDU et même l'ensemble d'une primitive ASP ou d'une unité PDU.

NOTE – Quand ces mécanismes de concordance sont utilisés, soit seuls, soit en combinaison, il est possible de spécifier dans les contraintes de nombreuses restrictions de protocole, ce qui évite les calculs détaillés inutiles dans la partie comportement.

12.6.3 Spécificateurs de valeurs

Il s'agit du mécanisme de concordance de base. Les valeurs spécifiques indiquées dans les contraintes sont des expressions. Sauf spécification contraire, un paramètre de primitive ASP ou un champ d'unité PDU d'une contrainte concorde avec le paramètre correspondant de la primitive ASP entrante ou le champ correspondant de l'unité PDU entrante si, et seulement si, la valeur de ce paramètre de primitive ASP entrante ou de champ d'unité PDU entrante est identique à la valeur prise par l'expression de la contrainte.

Deux valeurs d'un type tabulaire de primitive ASP, d'unité PDU ou de structure, ou deux valeurs ASN.1 du type SEQUENCE ou SEQUENCE OF sont considérées comme identiques si tous leurs paramètres, champs ou éléments concordent et apparaissent dans le même ordre. Deux valeurs ASN.1 de type SET ou SET OF sont identiques si elles ont le même nombre d'éléments et si chacun des éléments d'une valeur concorde exactement avec un élément de l'autre et réciproquement. Les éléments de deux valeurs de type SET ou SET OF ne doivent pas obligatoirement apparaître dans le même ordre pour concorder.

Tableau 6/X.292 – Mécanismes de concordance en notation TTCN

	VALEUR	SPÉCIFICATEURS DE SUBSTITUTION								SPÉCIFICATEURS D'APPARTENANCE			SPÉCIFICATEURS D'APPARTENANCE	
TYPE	**Specific Value** (valeur spécifique)	**Complement**	**Omit(-)**	**AnyValue (?)**	**AnyOrOmit(*)**	**ValueList**	**Range**	**SuperSet**	**SubSet**	**AnyOne (?)**	**AnyOrNone(*)**	**Permutation**	**Length**	**IfPresent**
BOOLEAN	•	•	•	•	•	•								•
INTEGER	•	•	•	•	•	•	•							•
ENUMERATED	•	•	•	•	•	•								•
BITSTRING	•	•	•	•	•	•				•	•		•	•
OCTETSTRING	•	•	•	•	•	•				•	•		•	•
HEXSTRING	•	•	•	•	•	•				•	•		•	•
CHARSTRNGS	•	•	•	•	•	•				•	•		•	•
SEQUENCE		•	•	•	•	•								•
SEQUENCE OF	•	•	•	•	•	•				•	•	•	•	•
SET	•	•	•	•	•	•								•
SET OF	•	•	•	•	•	•		•		•	•		•	•
ANY	•	•	•	•	•	•								•
CHOICE	•	•	•	•	•	•								•
OBJECT ID	•	•	•	•	•	•								•

12.6.4 Spécificateurs de substitution

12.6.4.1 Complement

Complement est une spécification de concordance qui peut s'appliquer à des valeurs de tous types. Cette opération est indiquée par le mot clé COMPLEMENT (complément) suivi de la liste des valeurs des contraintes. Chaque valeur de contrainte de cette liste doit être du type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel le mécanisme Complement est utilisé.

DÉFINITION SYNTAXIQUE:

566 Complement ::= **COMPLEMENT** ValueList

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par le mécanisme Complement concorde avec le paramètre de primitive ASP ou le champ d'unité PDU correspondant si, et seulement si, le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante ne concorde avec aucune des valeurs de la liste ValueList.

EXEMPLE 37 – Contraintes utilisant le mécanisme Complement à la place d'une valeur avec une liste des valeurs:

Type	*Contrainte*
INTEGER	COMPLEMENT(5)
INTEGER	COMPLEMENT(1, 3, 5)

12.6.4.2 Omit (omission)

Omit est un spécificateur spécial de concordance qui peut s'appliquer à des valeurs de tous types, à condition que le paramètre de primitive ASP ou le champ d'unité PDU soit facultatif.

Dans les contraintes ASN.1, il est également possible d'omettre un paramètre de primitive ASP ou un champ d'unité PDU de type OPTIONAL (facultatif) au lieu d'utiliser explicitement le symbole OMIT.

NOTE – Dans les contraintes tabulaires, tous les paramètres, les champs et les éléments sont considérés comme implicitement facultatifs; ils peuvent être omis en utilisant le spécificateur Omit. Dans les contraintes en notation ASN.1, les paramètres, les champs et les éléments qui ne sont pas explicitement identifiés comme OPTIONAL (facultatif) dans la définition du type sont obligatoires; ils ne peuvent donc pas être omis, car la définition du type ne sera pas respectée. Si un de ces paramètres, champs ou éléments doit être omis dans une contrainte en particulier, il faut soit définir un autre type dans lequel ce paramètre, ce champ ou cet élément est explicitement identifié comme OPTIONAL (par exemple, en les marquant tous comme facultatifs), soit appliquer un codage de champ non valide à ce paramètre, à ce champ ou à cet élément, ce qui l'omettra du codage.

Dans les contraintes tabulaires, le spécificateur Omit est représenté par un trait d'union (-). Dans les contraintes en notation ASN.1, il est indiqué par **OMIT**.

DÉFINITION SYNTAXIQUE:

567 Omit ::= Dash | **OMIT**

Dans une contrainte, un spécificateur Omit indique qu'un paramètre de primitive ASP ou un champ d'unité PDU facultatif sera absent.

EXEMPLE 38 – Contrainte utilisant Omit en lieu et place d'une valeur, au niveau le plus élevé:

Type	*Contrainte*
INTEGER OPTIONAL	OMIT

12.6.4.3 AnyValue (valeur quelconque)

AnyValue est un spécificateur spécial de concordance qui peut s'appliquer à des valeurs de tous types. Dans les contraintes tant ASN.1 que tabulaires, AnyValue est indiqué par le signe "?".

DÉFINITION SYNTAXIQUE:

568 AnyValue ::= "?"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant ce symbole concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant prend la valeur d'un élément simple du type spécifié.

EXEMPLE 39 – Contrainte utilisant une combinaison de valeurs spécifiques et le symbole AnyValue:

Type	*Contrainte*
SEQUENCE OF SET OF INTEGER	{ {1, 2}, ?, {1, 2, ?} }

12.6.4.4 AnyOrOmit (élément quelconque ou omission)

AnyOrOmit est un spécificateur spécial de concordance applicable à des valeurs de tous types, à condition que le paramètre de primitive ASP ou le champ d'unité PDU soit déclaré comme facultatif. Dans les contraintes tant ASN.1 que tabulaires, AnyOrOmit est indiqué par le signe "*".

NOTE – Le symbole "*" est utilisé pour les deux symboles AnyOrOmit et AnyOrNone (quelconque ou aucun). Cette ambiguïté d'interprétation est levée par les spécifications du présent sous-paragraphe et 12.6.5.2.

DÉFINITION SYNTAXIQUE:

569 AnyOrOmit ::= "*"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le symbole AnyOrOmit concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant est absent ou prend la valeur d'un élément quelconque du type spécifié.

EXEMPLE 40 – Contrainte utilisant une combinaison de valeurs spécifiques et du symbole AnyOrOmit:

Type	*Contrainte*
SEQUENCE OF { id1 SET OF INTEGER id2 SET OF INTEGER	{ id1 {2, 5}, id2 * }

12.6.4.5 ValueList (liste de valeurs)

La spécification par liste ValueList peut s'utiliser avec des valeurs de tous types. Dans les contraintes tant ASN.1 que tabulaires, ces listes apparaissent sous forme d'une série de valeurs séparées par des virgules et placées entre parenthèses.

DÉFINITION SYNTAXIQUE:

570 ValueList ::= "("ConstraintValue&Attributes {Comma ConstraintValue&Attributes} ")"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant une liste de valeurs concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant concorde avec l'une quelconque des valeurs de la liste des valeurs ValueList. Chaque valeur de la liste ValueList doit appartenir au type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel le mécanisme ValueList est utilisé.

EXEMPLE 41 – Contrainte utilisant le mécanisme ValueList en lieu et place d'une valeur spécifique pour un type INTEGER:

Type	*Contrainte*
INTEGER	(2, 4, 6)

EXEMPLE 42 – Contrainte utilisant le mécanisme ValueList en lieu et place d'une valeur spécifique pour un type CHOICE:

Type	*Contrainte*
CHOICE { a INTEGER, b BOOLEAN }	(a 2, b TRUE)

12.6.4.6 Range (intervalle)

La spécification par intervalles "Range" ne s'applique qu'à des valeurs de type entier. Un intervalle est indiqué par deux limites, séparées par le signe ".." ou le mot TO (à), le tout entre parenthèses. Une limite sera:

a) soit INFINITY ou -INFINITY;

b) soit une expression de contrainte de valeur entière.

La limite inférieure sera inscrite à la gauche du symbole ".." ou TO, la limite supérieure à droite. La limite inférieure doit être inférieure à la limite supérieure.

DÉFINITION SYNTAXIQUE:

571 ValueRange ::= "(" ValRange ")"
572 ValRange ::= (LowerRangeBound To UpperRangeBound)
573 LowerRangeBound ::= ConstraintExpression | Minus **INFINITY**
574 UpperRangeBound ::= ConstraintExpression | **INFINITY**

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par un intervalle correspond au paramètre de la primitive ASP entrante ou au champ de l'unité PDU entrante correspondant si, et seulement si, la valeur de ce paramètre ou de ce champ entrant est égale à l'une de celles de l'intervalle.

EXEMPLE 43 – Contrainte utilisant le mécanisme d'intervalle "Range" en lieu et place d'une valeur:

Type	*Contrainte*
INTEGER	(1 .. 6) (-INFINITY .. 8) (12 .. INFINITY)

12.6.4.7 SuperSet (surensemble)

La spécification par surensemble est un mécanisme de concordance qui ne s'applique qu'à des valeurs du type SET OF (ensemble de). Ce mécanisme ne sera utilisé que pour des contraintes ASN.1. Il est indiqué par la mention **SUPERSET**.

DÉFINITION SYNTAXIQUE:

575 SuperSet ::= SUPERSET "(" ConstraintValue&Attributes ")"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par un surensemble concorde avec le paramètre de la primitive ASP entrante ou le champ de l'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant contient tous les éléments du surensemble, plus éventuellement d'autres. L'argument du spécificateur SuperSet doit être du type déclaré pour le paramètre de la primitive ASP ou le champ de l'unité PDU pour lequel ce mécanisme est utilisé.

EXEMPLE 44 – Contrainte spécifiée par un surensemble en lieu et place d'une valeur spécifique:

Type	*Contrainte*
SET OF INTEGER	SUPERSET({1, 2, 3})

12.6.4.8 SubSet (sous-ensemble)

La spécification par sous-ensemble est un mécanisme de concordance qui ne s'applique qu'aux valeurs du type SET OF. Ce mécanisme ne sera utilisé que pour des contraintes ASN.1. Il est indiqué par la mention **SUBSET**.

DÉFINITION SYNTAXIQUE:

576 SubSet ::= SUBSET "(" ConstraintValue&Attributes")"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par un sous-ensemble concorde avec le paramètre de la primitive ASP entrante ou le champ de l'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant ne contient que des éléments du sous-ensemble, et pas nécessairement tous. L'argument du spécificateur Subset doit être du type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel ce mécanisme est utilisé.

EXEMPLE 45 – Contrainte spécifiée par un sous-ensemble en lieu et place d'une valeur spécifique:

Type	*Contrainte*
SET OF INTEGER	SUBSET({2, 4, 6, 8, 10})

12.6.5 Spécificateurs d'appartenance

12.6.5.1 AnyOne (un quelconque)

AnyOne est un spécificateur spécial de concordance qui peut s'utiliser avec des valeurs des types chaînes, SEQUENCE OF et SET OF. Dans les contraintes tant ASN.1 que tabulaires, ce spécificateur est représenté par le signe "?".

DÉFINITION SYNTAXIQUE:

754 AnyOne ::= "?"

Dans une valeur de type chaîne, SEQUENCE OF ou SET OF, le signe "?" mis à la place d'un élément simple indique que tout élément est acceptable à cette position. Dans une chaîne CharacterString, le caractère "?" (point d'interrogation) sera représenté par le signe "\?"; quant au caractère "\" (barre oblique inverse), il sera représenté par le signe "\\".

EXEMPLE 46 – Contrainte utilisant AnyOne:

Type	*Contrainte*
IA5String	"a?cd"
SEQUENCE OF INTEGER	{1, 2, ? }

NOTE – Dans le second exemple, le signe "?" peut être interprété comme le spécificateur "valeur quelconque" (AnyValue) représentant une valeur entière quelconque au niveau de la contrainte ou comme le spécificateur "un quelconque" (AnyOne) représentant une valeur quelconque parmi les valeurs de la suite d'entiers SEQUENCE OF INTEGER. Ces interprétations aboutissant toutes deux au même ensemble d'événements mis en concordance, l'ambiguïté ne pose pas de problème.

12.6.5.2 AnyOrNone (quelconque ou aucun)

"AnyOrNone" est un spécificateur spécial de concordance qui peut s'utiliser avec des valeurs des types chaînes, SEQUENCE OF et SET OF. Dans les contraintes tant ASN.1 que tabulaires, ce spécificateur est représenté par le signe "*".

Si le symbole "*" apparaît au niveau supérieur d'une valeur de type chaîne, SEQUENCE OF ou SET OF, il est interprété comme le spécificateur AnyOrNone.

NOTE – Cette règle évite l'interprétation possible du signe "*" en tant que symbole de AnyOrOmit, qui remplace un élément d'une valeur de type chaîne, SEQUENCE OF ou SET OF.

DÉFINITION SYNTAXIQUE:

755 AnyOrNone ::= "*"

Dans une valeur de type chaîne, SEQUENCE OF ou SET OF, un signe "*" mis à la place d'un élément simple remplace zéro ou plusieurs éléments consécutifs. Le mécanisme de concordance fait correspondre au symbole "*" la plus longue séquence possible d'éléments, conformément au modèle spécifié par les symboles entourant le signe "*". Dans une chaîne CharacterString, le caractère "*" (astérisque) est représenté par le signe "*"; quant au caractère "\" (barre oblique inverse), il est représenté par le signe "\\".

EXEMPLE 47 – Contraintes utilisant le spécificateur AnyOne:

Type	*Contrainte*
IA5String	"ab*z"
SEQUENCE OF INTEGER	{1, 2, *, 10 }
SEQUENCE OF IA5String	{ "ab*z", *, "abc" }

12.6.5.3 Permutation

La permutation est une opération de concordance qui ne s'applique qu'aux éléments d'une valeur de type SEQUENCE OF. Elle ne s'utilise qu'avec des contraintes ASN.1. Elle est indiquée par le spécificateur **PERMUTATION**.

DÉFINITION SYNTAXIQUE:

577 Permutation ::= **PERMUTATION** ValueList

La mention Permutation, en lieu et place d'un élément simple, signifie que cet élément peut prendre pour valeur n'importe quelle permutation des éléments de la liste de valeurs ValueList. Si les spécificateurs Permutation et AnyOrNone apparaissent tous deux dans une valeur, le spécificateur AnyOrNone sera évalué en premier. Chacun des éléments cités en argument de la Permutation doit être du type déclaré dans le type SEQUENCE OF du paramètre de la primitive ASP ou du champ de l'unité PDU.

EXEMPLE 48 – Contrainte spécifiée par Permutation:

Type	*Contrainte*
SEQUENCE OF INTEGER	{PERMUTATION (1, 2, 3), 5}

EXEMPLE 49 – Contraintes spécifiées par une combinaison de Permutation et AnyOrNone:

Type	*Contrainte*
SEQUENCE OF INTEGER	{PERMUTATION (1,2,3), *}
	{PERMUTATION (1,2,3,*)}

A noter que la première contrainte concorde avec toute primitive ASP ou unité PDU entrante composée d'une séquence de valeurs entières, commençant par une des six permutations: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; ou 3,2,1 et suivie d'un nombre quelconque d'entiers. La seconde contrainte concorde avec toute primitive ASP ou unité PDU entrante de type suite d'entiers contenant les éléments 1, 2 et 3 en position quelconque. Elle concorde, par exemple, avec les suites {5,2,7,1,3} et {9,3,7,2,12,1,17}.

12.6.6 Attributs de valeurs

12.6.6.1 Length (longueur)

La spécification de longueur est une opération de concordance qui ne s'utilise comme attribut que pour les mécanismes suivants: Complement, AnyValue, AnyOrOmit, AnyOne, AnyOrNone, Permutation, SuperSet and SubSet. Elle peut être utilisée en combinaison avec l'attribut IfPresent.

Dans les contraintes tant ASN.1 que tabulaires, la longueur peut être spécifiée sous forme d'une valeur exacte ou d'un intervalle de valeurs de types chaîne, SEQUENCE OF et SET OF, conformément au 11.18. Les unités de longueur seront interprétées selon le Tableau 5. Les limites seront des valeurs entières non négatives. Le mot clé INFINITY peut également servir de limite supérieure pour indiquer que la longueur maximale est illimitée.

Une spécification de longueur définie pour un type de paramètre de primitive ASP ou de champ d'unité PDU dans une définition de type de suite de tests respectera les spécifications de longueur de la contrainte de la primitive ASP ou de l'unité PDU, c'est-à-dire que l'ensemble de chaînes défini dans une contrainte de primitive ASP ou d'unité PDU par restriction de longueur doit former un sous-ensemble strict de l'ensemble de chaînes correspondant à la définition de la primitive ASP ou de l'unité PDU.

DÉFINITION SYNTAXIQUE:

580 ValueLength ::= SingleValueLength | RangeValueLength
581 SingleValueLength ::= "[" ValueBound "]"
582 ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
583 RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
584 LowerValueBound ::= ValueBound
89 To ::=TO |".."
585 UpperValueBound ::= ValueBound | **INFINITY**

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le mécanisme de spécification de l'attribut Length concorde avec le paramètre de la primitive ASP entrante ou avec le champ de l'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant concorde avec la valeur de la contrainte tout en respectant la spécification de l'attribut associé. La valeur de l'attribut de longueur est considérée comme acceptable si la longueur du paramètre ou du champ entrant appartient à l'intervalle spécifié (limites comprises) lorsque la spécification d'attribut est donnée sous forme d'intervalle, ou est égale à la valeur spécifiée lorsque la spécification d'attribut est donnée sous forme de valeur unique.

Lorsqu'un paramètre, un champ ou un élément est omis, l'attribut Length concorde toujours. Par conséquent, lorsque le spécificateur Omit est utilisé, la longueur est redondante; lorsque le symbole AnyOrOmit et le spécificateur IfPresent sont utilisés, elle impose une restriction sur la valeur entrante, s'il y en a une.

EXEMPLE 50 – Contraintes utilisant une combinaison de spécification d'appartenance et d'attribut de longueur:

Type	*Contrainte*
IA5String	"ab*ab" [13]

12.6.6.2 IfPresent (si présent)

IfPresent est un spécificateur spécial de concordance pouvant être utilisé comme attribut de tous les mécanismes de concordance, à condition que le type soit déclaré comme facultatif. Dans les contraintes tant ASN.1 que tabulaires, cette spécification est indiquée par le spécificateur **IF_PRESENT**.

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le spécificateur IfPresent comme attribut d'un autre spécificateur concorde avec le paramètre de la primitive ASP entrante ou le champ de l'unité PDU entrante correspondant si, et seulement si, ce paramètre ou ce champ entrant concorde avec la contrainte ou est absent.

NOTE – Le spécificateur AnyOrOmit (*) a exactement la même signification que le spécificateur ? doté de l'attribut IF_PRESENT.

EXEMPLE 51 – Contrainte utilisant une valeur combinée avec le spécificateur IfPresent:

Type	*Contrainte*
IA5String OPTIONAL	"abcdef" IF_PRESENT

13 Spécification des contraintes à l'aide de tables

13.1 Introduction

Le présent paragraphe a pour objet de décrire la spécification sous forme tabulaire de contraintes imposées à des primitives ASP et à des unités PDU. Il indique comment utiliser des tables de contraintes simples pour imposer des contraintes à des primitives ASP ou à des unités PDU plates (non structurées) et comment spécifier des contraintes structurées en déclarant des contraintes relatives à des types structurés définis dans les types des suites de tests.

L'Annexe C définit des tables complémentaires permettant de regrouper plusieurs définitions de contraintes dans une seule table.

13.2 Déclarations de contraintes de type structuré

Si une primitive ASP ou une unité PDU est définie à l'aide de types structurés, soit par développement de macros soit par des sous-structures, les contraintes imposées seront sous-structurées de la même façon. Les informations suivantes doivent être fournies pour chacune des contraintes de type structuré:

a) le nom de la contrainte,

   éventuellement suivi d'une liste facultative de paramètres formels;

b) le nom du type structuré;

c) le chemin de dérivation (voir 13.6);

d) les variations de codage à utiliser pour la contrainte,

   pour spécifier des variations de codage explicites pour l'ensemble des contraintes de type structuré, qui ont priorité sur les règles et les variations de codage qui s'appliquent à la contrainte de l'unité PDU dans laquelle cette contrainte de type structuré est utilisée, cette entrée facultative fera référence à une entrée de la table des variations de codage appropriée [par exemple, pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les règles et les variations de codage applicables à la contrainte de l'unité PDU s'appliquent aussi à la contrainte de type structuré. Voir 11.16.4;

e) une valeur pour la contrainte de chaque élément,

   donnant les informations suivantes pour chaque élément:

   1) son nom,

      chaque entrée de la colonne nom de l'élément devra avoir été déclarée dans la définition du type structuré correspondant. Si le nom abrégé et l'identificateur complet d'un des éléments d'origine sont définis, la contrainte ne répétera pas l'identificateur complet.

Si, dans le développement d'une macro, la définition du type structuré se réfère à un autre type structuré (c'est-à-dire que "<-" remplace le nom de l'élément), la contrainte correspondante prendra l'une des formes suivantes:

– ou bien chacun des éléments du type structuré sera directement inclus dans les contraintes;

– ou bien le macrosymbole (<-) sera placé dans la position correspondante dans la colonne nom de l'élément de la contrainte et sa valeur fera référence à une contrainte appliquée au type structuré mentionné dans la définition du type structuré.

On n'utilisera pas de contraintes structurées par développement de macro, sauf si la définition du type structuré correspondant fait elle aussi référence au type structuré interne obtenu par développement de macro;

2) sa valeur et un attribut facultatif;

3) facultativement, un identificateur de codage spécifique suivi de toute liste de paramètres effectifs nécessaires, pour spécifier explicitement le codage de chaque élément dans une contrainte de type structuré, qui a priorité sur les règles et les variations de codage qui s'appliquent à l'ensemble de la contrainte de type structuré et qui a également priorité sur le codage spécifié pour cet élément dans la déclaration du type structuré; l'identificateur de codage, s'il y en a, doit indiquer soit l'une des variations de codage, ou une définition de codage de champ non valide qui figure dans une suite de tests [par exemple, LD(10)]; voir 11.16.4.

Les valeurs des éléments des contraintes de type structuré seront fournies dans le format illustré dans le Formulaire 37, ci-dessous.

Déclaration de contrainte de type structuré			
Nom de la contrainte : *ConsId&ParList*			
Groupe : *[StructTypeConstraintGroupReference]*			
Type structuré : *StructIdentifier*			
Chemin de dérivation : *[DerivationPath]*			
Variation de codage : *[EncVariationCall]*			
Commentaires : *[FreeText]*			
Nom de l'élément	**Valeur de l'élément**	**Codage de l'élément**	**Commentaires**
⋮ *ElemIdentifier* ⋮	⋮ *ConstraintValue&Attributes* ⋮	⋮ *[PDU_FieldEncodingCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 37 – Déclaration de contrainte de type structuré

Ce formulaire est utilisé pour les structures et leurs éléments de la même manière que le formulaire de déclaration de contrainte d'unité PDU l'est pour les unités PDU et leurs champs (voir 13.4).

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier [FormalParList]
508 StructTypeConstraintGroupReference
::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{StructTypeConstraintGroupIdentifier"/"}
99 StructIdentifier ::=Identifier
558 DerivationPath ::= {ConstraintIdentifier Dot }+
511 EncVariationCall ::=EncVariationIdentifier [ActualParList]
107 ElemIdentifier ::=Identifier
562 ConstraintValue&Attributes ::=ConstraintValue ValueAttributes
515 PDU_FieldEncodingCall ::= EncVariationCall| InvalidFieldEncodingCall

Si une définition de primitive ASP ou d'unité PDU se réfère à un type structuré comme sous-structure de paramètre ou de champ (c'est-à-dire avec le nom du paramètre ou du champ spécifié pour cette sous-structure), la contrainte correspondante indiquera alors le même nom de paramètre ou de champ dans la position correspondante de la colonne nom du paramètre ou nom du champ, et sa valeur sera une référence à une contrainte portant sur ce paramètre ou ce

champ (c'est-à-dire sur cette sous-structure, conformément à la définition du type structuré). Si la définition de la primitive ASP ou de l'unité PDU se réfère à un paramètre ou à un champ spécifié comme étant du métatype PDU, la valeur de ce paramètre ou champ sera alors spécifiée dans une contrainte correspondante comme le nom d'une contrainte d'unité PDU ou comme paramètre formel.

13.3 Déclaration des contraintes de primitive ASP

Les valeurs des paramètres des contraintes de primitive ASP seront fournies dans le format illustré dans le Formulaire 38, ci-dessous.

Déclaration de contrainte de primitive ASP		
Nom de la contrainte : *ConsId&ParList*		
Groupe : *[ASP_ConstraintGroupReference]*		
Type de la primitive : *ASP_Identifier*		
Chemin de dérivation : *[DerivationPath]*		
Commentaires : *[FreeText]*		
Nom du paramètre	**Valeur du paramètre**	**Commentaires**
⋮ *ASP_ParIdOrMacro* ⋮	⋮ *ConstraintValue&Attributes* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 38 – Déclaration de contrainte de primitive ASP

Les colonnes nom du paramètre et valeur du paramètre doivent être toutes les deux présentes, ou toutes les deux omises.

Ce formulaire est utilisé pour spécifier les contraintes portant sur les primitives ASP et leurs paramètres de la même manière qu'un formulaire de déclaration de contrainte d'unité PDU (voir 13.4), sauf les renseignements sur le codage qui ne sont pas pertinents et qu'il ne faut pas spécifier.

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier [FormalParList]
532 ASP_ConstraintGroupReference ::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{ASP_ConstraintGroupIdentifier"/"}
349 ASP_Identifier ::=Identifier
558 DerivationPath ::= {ConstraintIdentifier Dot}+
357 ASP_ParIdOrMacro ::=ASP_ParId&FullId| MacroSymbol
562 ConstraintValue&Attributes ::=ConstraintValue ValueAttributes

13.4 Déclarations des contraintes d'unité PDU

En format tabulaire, une contrainte est définie par la spécification d'une valeur et d'attributs facultatifs pour chaque champ d'unité PDU. Les informations suivantes seront fournies pour chaque contrainte d'unité PDU:

a) le nom de la contrainte,

   éventuellement suivi d'une liste facultative de paramètres formels;

b) le nom du type de l'unité PDU;

c) son chemin de dérivation (voir 13.6);

d) les règles de codage à utiliser pour la contrainte,

   pour spécifier le codage explicite pour l'ensemble des contraintes d'unité PDU, qui a priorité sur les règles de codage qui s'appliquent au type d'unité PDU spécifié, cette entrée facultative fera référence à une entrée qui figure dans la table de définitions de codage (par exemple, pour passer de BER à DER). Si cette entrée n'est pas utilisée, les règles de codage utilisées pour le type d'unité PDU s'appliquent. Voir 11.16.4;

e) les variations de codage à utiliser pour la contrainte,

pour spécifier les variations de codage explicites pour l'ensemble des contraintes d'unité PDU, qui ont priorité sur les variations de codage qui s'appliquent au type d'unité PDU spécifié, cette entrée facultative fera référence à une entrée qui figure dans la table de variations de codage [par exemple, pour passer de SD à LD(3)]. Si cette entrée n'est pas utilisée, les règles de codage utilisées pour le type d'unité PDU s'appliquent. Voir 11.16.4;

f) une valeur de contrainte pour chaque champ,

donnant les informations suivantes pour chaque champ:

1) son nom,

chaque entrée de champ dans la colonne nom du champ devant avoir été déclarée dans la définition correspondante du type d'unité PDU. Si l'un des champs de l'unité PDU a été défini comme ayant un nom abrégé et un identificateur complet, la contrainte ne répétera pas l'identificateur complet.

Si la définition de l'unité PDU se réfère à un type structuré résultant d'un développement de macro (c'est-à-dire dans laquelle le signe "<-" remplace le nom d'un champ), la contrainte correspondante prendra l'une des formes suivantes:

– ou bien chacun des éléments correspondant au type structuré sera directement inclus dans la contrainte;

– ou alors le macrosymbole (<-) sera placé dans la position correspondante de la colonne nom du champ d'unité PDU de la contrainte et sa valeur fera référence à une contrainte appliquée au type structuré mentionné dans la définition de l'unité PDU.

On n'utilisera pas de contraintes structurées par développement de macro, sauf si la définition de l'unité PDU correspondante fait elle aussi référence au même type structuré par développement de macro;

2) sa valeur et un attribut facultatif;

3) facultativement, un identificateur de codage spécifique suivi de toute liste de paramètres effectifs nécessaires, pour spécifier explicitement le codage de chaque champ dans une contrainte d'unité PDU, qui a priorité sur les règles et les variations de codage qui s'appliquent à l'ensemble de la contrainte d'unité PDU et qui a également priorité sur le codage spécifique applicable à ce champ pour les unités PDU de ce type; l'identificateur de codage, s'il y a lieu, doit identifier soit l'une des variations de codage, ou une définition de codage de champ non valide précisée dans la suite de tests [par exemple, LD(10)]; voir 11.16.4.

Ce mécanisme de codage ne doit pas être utilisé dans les contraintes de primitives ASP.

Ces informations seront fournies dans le format illustré dans le Formulaire 39, ci-dessous.

Déclaration de contrainte d'unité PDU			
Nom de la contrainte : *ConsId&ParList*			
Groupe : *[PDU_ConstraintGroupReference]*			
Type de l'unité : *PDU_Identifier*			
Chemin de dérivation : *[DerivationPath]*			
Nom de la règle de codage : *[EncodingRuleIdentifier]*			
Variation de codage : *[EncVariationCall]*			
Commentaires : *[FreeText]*			
Nom du champ	**Valeur du champ**	**Codage du champ**	**Commentaires**
⋮ *PDU_FieldIdOrMacro* ⋮	⋮ *ConstraintValue&Attributes* ⋮	⋮ *[PDU_FieldEncodingCall]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire 39 – Déclaration de contrainte d'unité PDU

Les colonnes nom du champ et valeur du champ doivent toutes les deux être présentes, ou toutes les deux omises. La colonne codage du champ ne doit pas apparaître seule.

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]
551 PDU_ConstraintGroupReference ::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{PDU_ConstraintGroupIdentifier "/"}
383 PDU_Identifier ::= Identifier
558 DerivationPath ::={ConstraintIdentifier Dot}+
452 EncodingRuleIdentifier ::= Identifier
511 EncVariationCall ::= EncVariationIdentifier[ActualParList]
391 PDU_FieldOrMacro ::= PDU_ Field&Id| MacroSymbol
562 ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
515 PDU_FieldEncodingCall ::= EncVariationCal| InvalidFieldEncodingCall

EXEMPLE 52 – Contrainte C1 agissant sur l'unité PDU_A:

Déclaration de contrainte d'unité PDU		
Nom de la contrainte : C1 **Type de l'unité** : PDU_A **Chemin de dérivation** : **Commentaires** :		
Nom du champ	**Valeur du champ**	**Commentaires**
FIELD1 FIELD2 FIELD3	(4 .. INFINITY) TRUE "A STRING"	

13.5 Paramétrage des contraintes

Les contraintes peuvent être paramétrées à l'aide d'une liste de paramètres formels. Les paramètres effectifs sont transmis à une contrainte à partir d'une référence à une contrainte dans une description comportementale.

EXEMPLE 53 – Contrainte paramétrée:

Déclaration de contrainte d'unité PDU		
Nom de la contrainte : C2(P1:INTEGER;P2:BOOLEAN) **Type d'unité** : PDU_B **Chemin de dérivation** : **Commentaires** :		
Nom du champ	**Valeur du champ**	**Commentaires**
FIELD1 FIELD2 FIELD3	P1 P2 "A STRING"	
Commentaires détaillés: une référence possible à C2 à partir d'un test élémentaire ou d'un module de test serait: C2(0,TRUE)		

13.6 Contraintes de base et contraintes modifiées

Au moins une contrainte de base sera spécifiée pour chaque type défini de primitive ASP, d'unité PDU ou de message CM. Lorsqu'une primitive ASP ou un message CM ne contient pas de paramètres, ou lorsqu'une unité PDU ne contient pas de champs, les contraintes ne sont pas pertinentes et, par conséquent, les contraintes de base ne sont pas obligatoires. Une contrainte de base spécifie un ensemble de valeurs de base (par défaut) ou un spécificateur de concordance pour chacun des champs définis. Une unité PDU peut comporter un nombre quelconque de contraintes de base (voir les exemples donnés dans l'Appendice I).

Lorsqu'une contrainte est spécifiée comme étant une modification d'une contrainte de base, tous les champs non spécifiés à nouveau prennent par défaut la valeur des champs correspondants de la contrainte de base. Le nom de la contrainte modifiée sera un identificateur unique. Le nom de la contrainte de base dont elle dérive sera indiqué à la rubrique chemin de dérivation de l'en-tête de la contrainte. Cette rubrique est laissée en blanc pour une contrainte de base. Une

contrainte modifiée peut être modifiée à son tour. Dans ce cas, le chemin de dérivation indiquera les noms concaténés et séparés par des points (.) des contraintes de base et des contraintes successives déjà modifiées. Le nom de la dernière contrainte modifiée sera suivi d'un point. Les règles de formation d'une contrainte modifiée à partir d'une contrainte de base sont les suivantes:

a) si un paramètre ou un champ et sa valeur ou son spécificateur de concordance ne sont pas spécifiés dans la contrainte modifiée, ce paramètre ou ce champ, ou son spécificateur correspondant, prend alors la valeur de la contrainte mère (c'est-à-dire que la valeur est héritée);

b) si un paramètre ou un champ et sa valeur ou son spécificateur de concordance sont spécifiés dans la contrainte modifiée, la valeur spécifiée remplace alors la valeur ou le spécificateur de concordance du paramètre ou du champ dans la contrainte mère.

13.7 Listes de paramètres formels dans les contraintes modifiées

Si une contrainte de base est définie comme ayant une liste de paramètres formels, les règles suivantes s'appliqueront à toutes les contraintes modifiées dérivées de cette contrainte de base, que leur modification résulte d'une ou plusieurs dérivations successives:

a) la contrainte modifiée aura la même liste de paramètres formels que la contrainte de base. Aucun paramètre ne sera notamment omis ou ajouté;

b) la liste des paramètres formels suivra le nom de chaque contrainte modifiée;

c) les paramètres de primitive ASP et les unités PDU apparaissant dans les champs d'une contrainte de base ne doivent pas être modifiés ou explicitement omis dans une contrainte modifiée.

13.8 Déclaration des contraintes des messages CM

Ces informations seront fournies dans le format illustré dans le Formulaire 40, ci-dessous.

Déclaration de contrainte de message CM		
Nom de la contrainte : *ConsId&ParList*		
Groupe : *[CM_ConstraintGroupReference]*		
Type de message CM : *CM_Identifier*		
Chemin de dérivation : *[DerivationPath]*		
Commentaires : *[FreeText]*		
Nom du paramètre	**Valeur du paramètre**	**Commentaires**
⋮ *CM_ParIdOrMacro* ⋮	⋮ *ConstraintValue&Attributes* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire 40 – Déclaration de contrainte de message CM

Les colonnes nom du paramètre et valeur du paramètre doivent être toutes les deux présentes, ou toutes les deux omises.

Ce formulaire est utilisé pour les messages CM de la même manière que le formulaire de déclaration de contrainte d'unité PDU l'est pour les unités PDU (voir 13.4).

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]
605 CM_ConstraintGroupReference ::= [(SuiteIdentifier| TTCN_ModuleIdentifier) "/"]{CM_ConstraintGroupIdentifier "/"}
424 CM_Identifier ::= Identifier
558 DerivationPath ::={ConstraintIdentifier Dot}+
431 CM_ParIdOrMacro ::= CM_ParIdentifier| MacroSymbol
562 ConstraintValue&Attributes ::= ConstraintValue ValueAttributes

Ce formulaire est utilisé pour le type CM de la même manière que le formulaire de déclaration de contrainte d'unité PDU l'est pour les unités PDU.

14 Spécification des contraintes en notation ASN.1

14.1 Introduction

Le présent paragraphe décrit une méthode pour spécifier les contraintes relatives au type, aux primitives ASP et aux unités PDU en notation ASN.1, d'une manière similaire à la définition des contraintes tabulaires. La déclaration normale des valeurs ASN.1 est étendue pour permettre l'utilisation des mécanismes de concordance. Des mécanismes assurant la substitution ou l'omission de parties de contraintes en notation ASN.1 sont définis pour les besoins des contraintes modifiées.

Par ailleurs, la notation ASN.1 est utilisée dans les contraintes de la même manière que pour les types. En particulier:

a) le trait d'union ("-") ne doit pas être utilisé pour les identificateurs dans une contrainte en notation ASN.1. Le symbole de souligné peut être utilisé à sa place ("_");

b) les contraintes en notation ASN.1 ne doivent pas utiliser de références à des valeurs externes, telles que définies dans la Recommandation X.680;

c) les commentaires en notation ASN.1 peuvent être utilisés dans le corps de la table. La colonne de commentaires ne doit pas être présente dans cette table. Les commentaires en notation ASN.1 commencent par "--" et se terminent par l'occurrence suivante de "--" ou par une "fin de ligne", soit la première occurrence d'un de ces deux éléments. Cela empêche qu'un seul commentaire ASN.1 puisse s'étendre sur plusieurs lignes. "Fin de ligne" ne représente toutefois pas un symbole déterminé en notation TTCN.MP. Il est recommandé aux spécificateurs ATS de faciliter l'échange des suites ATS en notation TTCN.MP en terminant toujours les commentaires ASN.1 au moyen de "--".

14.2 Déclarations des contraintes des types en notation ASN.1

Les contraintes des primitives ASP et des unités PDU en notation ASN.1 peuvent être structurées par des références à des contraintes de type suites de tests en notation ASN.1 pour les valeurs de champs complexes. Les types suites de tests en notation ASN.1 sont définis dans la partie déclarative de la suite ATS.

Les informations suivantes seront fournies pour chaque déclaration de contrainte de primitive ASP en notation ASN.1:

a) le nom de la contrainte,

   qui peut être suivi d'une liste facultative de paramètres formels;

b) le nom du type ASN.1;

c) le chemin de dérivation (voir 13.6 et 14.6),

   afin de préciser des variations de codage explicites pour l'ensemble des contraintes relatives aux types ASN.1, qui ont priorité sur les variations de codage de la contrainte d'unité PDU qui renvoie à la contrainte de type ASN.1 et sur les variations de codage globales par défaut pour la suite de tests, cette rubrique facultative doit indiquer en référence une rubrique dans la table de variations de codage pertinente [par exemple pour passer de SD à LD(3)]; si cette rubrique n'est pas utilisée, les variations de codage par défaut s'appliquent à toutes les contraintes de type ASN.1 de ce type, à moins d'être supplantées spécifiquement dans une contrainte particulière;

d) les variations de codage devant être utilisées comme contrainte,

   si une déclaration de contrainte en notation ASN.1 est une modification d'une contrainte en notation ASN.1 existante, le nom de la contrainte en notation ASN.1 qui est à la base de la modification sera indiqué dans la table, à la rubrique de chemin de dérivation;

e) la valeur de la contrainte,

   où le corps de la table de contrainte de type ASN.1 contient la déclaration de la contrainte en notation ASN.1 et les attributs facultatifs. Tous les attributs et toutes les valeurs de contrainte définis en 12.6 sont utilisables dans des contraintes en notation ASN.1.

   Afin de préciser un codage explicite pour des valeurs individuelles dans une contrainte de type ASN.1, qui a priorité sur toutes les autres variations de codage s'appliquant au codage de contrainte de type ASN.1 spécifique [voir c) ci-dessus], le mot clé **ENC** est utilisé après la valeur pertinente, suivi d'un identificateur de codage

spécifique et de toute liste de paramètres effectifs nécessaires. L'identificateur de codage doit identifier soit une des variations de codage, soit une définition de codage de champ non valide présentée dans la suite de tests.

Les déclarations de contraintes de type ASN.1 seront données dans le format du Formulaire 41, ci-dessous.

Déclaration de contrainte d'un type en notation ASN.1	
Nom de la contrainte	**:** *ConsId&ParList*
Groupe	**:** *[ASN1_TypeConstraintGroupReference]*
Type structuré	**:** *ASN1_TypeIdentifier*
Chemin de dérivation	**:** *[DerivationPath]*
Variation de codage	**:** *[EncVariationCall]*
Commentaires	**:** *[FreeText]*
Valeur de la contrainte	
ConstraintValue&AttributesOrReplace	
Commentaires détaillés:	*[FreeText]*

Formulaire 41 – Déclaration de contrainte d'un type en notation ASN.1

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]

523 ASN1_TypeConstraintGroupReference
::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{ASN1_TypeConstraintGroupIdentifier"/"}

116 ASN1_Identifier ::=Identifier

558 DerivationPath ::={ConstraintIdentifier Dot}+

511 EncVariationCall ::=EncVariationIdentifier[ActualParList]

595 ConstraintValue&AttributesOrReplace ::=ConstraintValue&Attribute| Replacement{Comma Replacement}

Ce formulaire est utilisé pour les types ASN.1, de la même manière que le formulaire de déclaration de contraintes d'unités PDU en notation ASN.1 (voir 14.4).

14.3 Déclarations de contraintes de primitive ASP en notation ASN.1

Les informations suivantes seront fournies pour chaque déclaration de contrainte de primitive ASP en notation ASN.1:

a) le nom de la contrainte,

éventuellement suivi d'une liste facultative de paramètres formels;

b) le nom du type de la primitive ASP;

c) le chemin de dérivation (voir 13.6 et 14.6),

si une déclaration de contrainte en notation ASN.1 est une modification d'une contrainte en notation ASN.1 existante, le nom de la contrainte en notation ASN.1 prise comme base de cette modification sera indiqué en référence dans la table, à la rubrique chemin de dérivation;

d) la valeur de la contrainte,

le corps de la table de contrainte de la primitive ASP contenant la déclaration de la contrainte en notation ASN.1 et les attributs facultatifs. Tous les attributs et toutes les valeurs de contrainte définis en 12.6 sont utilisables dans des contraintes en notation ASN.1.

Les déclarations de contraintes de primitives ASP en notation ASN.1 seront données dans le format du Formulaire 42, ci-dessous.

Déclaration de contrainte de primitive ASP en notation ASN.1	
Nom de la contrainte	**:** *ConsId&ParList*
Groupe	**:** *[ASN1ASP_ConstraintGroupReference]*
Type de la primitive	**:** *ASP_Identifier*
Chemin de dérivation	**:** *[DerivationPath]*
Commentaires	**:** *[FreeText]*
Valeur de la contrainte	
ConstraintValue&AttributesOrReplace	
Commentaires détaillés:	*[FreeText]*

Formulaire 42 – Déclaration de contrainte de primitive ASP en notation ASN.1

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]

542 ASN1_ASP_ConstraintGroupReference ::=[(SuiteIdentifier|
TTCN_ModuleIdentifier)"/"]{ASN1_ASP_ConstraintGroupIdentifier"/"}

349 ASP_Identifier ::=Identifier

558 DerivationPath ::={ConstraintIdentifier Dot}+

595 ConstraintValue&AttributesOrReplace ::=ConstraintValue&Attribute| Replacement{Comma Replacement}

Ce formulaire est utilisé pour les primitives ASN.1, de la même manière que le formulaire de déclaration de contraintes d'unités PDU en notation ASN.1 (voir 14.4).

14.4 Déclarations de contraintes d'unités PDU en notation ASN.1

Les informations suivantes seront fournies pour chaque déclaration de contrainte d'unité PDU en notation ASN.1:

a) le nom de la contrainte,

qui peut être suivi d'une liste facultative de paramètres formels;

b) le nom du type d'unité PDU;

c) le chemin de dérivation (voir 13.6 et 14.6);

d) les règles de codage devant être utilisées comme contrainte,

afin de préciser un codage explicite pour l'ensemble des contraintes d'unité PDU en notation ASN.1, qui a priorité sur les règles de codage globales par défaut de la suite de tests, cette rubrique facultative doit indiquer en référence une rubrique dans la table de définitions de codage pertinente (par exemple pour passer de BER à DER); si cette rubrique n'est pas utilisée, les règles de codage par défaut s'appliquent à toutes les contraintes de type d'unité PDU de ce type, à moins d'être supplantées spécifiquement dans une contrainte particulière;

e) les variations de codage devant être utilisées pour la contrainte,

afin de préciser des variations de codage explicites pour l'ensemble des contraintes d'unité PDU en notation ASN.1, qui ont priorité sur les variations de codage globales par défaut pour la suite de tests, cette rubrique facultative doit indiquer en référence une rubrique dans la table de variations de codage pertinente [par exemple pour passer de SD à LD(3)]; si cette rubrique n'est pas utilisée, les variations de codage par défaut s'appliquent à toutes les contraintes de type de l'unité PDU ASN.1 de ce type, à moins d'être supplantées spécifiquement dans une contrainte particulière.

Si une déclaration de contrainte en notation ASN.1 est une modification d'une contrainte en notation ASN.1 existante, le nom de la contrainte en notation ASN.1 qui est à la base de la modification sera indiqué dans la table, dans la rubrique de chemin de dérivation;

f) la valeur de la contrainte,

où le corps de la table de contrainte d'unité PDU contient la déclaration de la contrainte en notation ASN.1 et les attributs facultatifs. Tous les attributs et toutes les valeurs de contrainte définis en 12.6 sont utilisables dans des contraintes en notation ASN.1.

Afin de préciser un codage explicite pour des valeurs individuelles dans une contrainte d'unité PDU en notation ASN.1, qui a priorité sur les règles de codage globales par défaut s'appliquant au codage de contrainte d'unité PDU en notation ASN.1 spécifique [voir c) et d) ci-dessus], le mot clé **ENC** est utilisé après la valeur pertinente, suivi d'un identificateur de codage spécifique et de toute liste de paramètres effectifs nécessaires. L'identificateur de codage doit identifier soit une des variations de codage, soit une définition de codage de champ non valide définie dans la suite de tests.

Les déclarations de contraintes d'unités PDU seront données dans le format du Formulaire 43, ci-dessous.

Déclaration de contrainte d'unité PDU en notation ASN.1	
Nom de la contrainte	**:** *ConsId&ParList*
Groupe	**:** *[ASN1PDU_ConstraintGroupReference]*
Type de l'unité PDU	**:** *PDU_Identifier*
Chemin de dérivation	**:** *[DerivationPath]*
Nom de la règle de codage	**:** *[EncodingRuleIdentifier]*
Variation de codage	**:** *[EncVariationCall]*
Commentaires	**:** *[FreeText]*
Valeur de la contrainte	
ConstraintValue&AttributesOrReplace	
Commentaires détaillés:	*[FreeText]*

Formulaire 43 – Déclaration de contrainte d'unité PDU en notation ASN.1

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]

592 ASN1_PDU_ConstraintGroupReference ::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{ASN1_PDU_ConstraintGroupIdentifier"/"}

383 PDU_Identifier ::=Identifier

558 DerivationPath ::={ConstraintIdentifier Dot}+

452 EncodingRuleIdentifier ::= Identifier

511 EncVariationCall ::= EncVariationIdentifier[ActualParList]

595 ConstraintValue&AttributesOrReplace ::=ConstraintValue&Attribute| Replacement{Comma Replacement}

14.5 Paramétrage des contraintes ASN.1

Les contraintes ASN.1 peuvent être paramétrées (voir 13.5).

14.6 Contraintes ASN.1 modifiées

Des contraintes en notation ASN.1 peuvent être spécifiées par modification d'une contrainte en notation ASN.1 existante. Des parties de contrainte peuvent être spécifiées à nouveau pour former une nouvelle contrainte à l'aide des mécanismes REPLACE et OMIT (remplacer et omettre).

Certains paramètres ou champs d'une contrainte de base ou modifiée peuvent être identifiés à l'aide d'une liste de sélecteurs de champs afin de remplacer leur valeur définie par une nouvelle, ou d'omettre la valeur définie. Une liste de références ReferenceList se compose des identificateurs des sélecteurs de champs (définis dans la définition de type correspondante) séparés par des points, identifiant de manière unique un champ particulier (éventuellement structuré) dans une unité PDU (ou une primitive ASP). Les champs du premier niveau sont identifiés par un seul sélecteur, les champs imbriqués nécessitant la connaissance du chemin de dérivation complet.

Les valeurs de remplacement ne seront utilisées que lorsqu'un chemin de dérivation est spécifié. Un ensemble complet de valeurs ASN.1 ne sera utilisé que lorsque aucun chemin de dérivation n'aura été spécifié. Les valeurs remplacées ou omises peuvent être structurées.

DÉFINITION SYNTAXIQUE:

596 Replacement ::= **REPLACE** ReferenceList **BY** ConstraintValue&Attributes| **OMIT** ReferenceList

597 ReferenceList ::= (ArrayRef| ComponentIdentifier| ComponentPosition) {ComponentReference}

Si un champ appartient à une structure SEQUENCE, SET ou CHOICE, la position du champ entre les parenthèses peut être utilisée en guise d'identificateur de sélecteur de champ. Cette technique sera utilisée lorsque aucun identificateur n'aura été indiqué dans la déclaration du champ.

14.7 Listes des paramètres formels dans des contraintes en notation ASN.1 modifiées

Les spécifications du 13.7 s'appliquent également aux contraintes en notation ASN.1 modifiées.

14.8 Noms des paramètres de primitive ASP et des champs d'unité PDU dans les contraintes ASN.1

Lors de la spécification d'une contrainte de primitive ASP ou d'unité PDU en notation ASN.1, il est possible d'utiliser les identificateurs de paramètres ou de champs spécifiés dans la définition ASN.1 des types SEQUENCE, SET ou CHOICE pour identifier le paramètre de primitive ASP ou le champ d'unité PDU pour lequel la contrainte impose une valeur. Dans le cas des types CHOICE, on utilisera les identificateurs de variantes. Dans le cas des types SEQUENCE, les identificateurs de paramètre ou de champ seront utilisés chaque fois que l'omission de paramètres ou champs facultatifs risque de rendre ambiguë l'attribution de valeur. Pour ce qui est des types SET, les identificateurs de paramètre ou de champ seront toujours utilisés.

EXEMPLE 54 – Valeurs des champs dans une contrainte en notation ASN.1 d'unité PDU:

Soit la définition de type suivante:

Définition de type d'unité PDU en notation ASN.1
Nom de l'unité PDU : XY_PDU **Type de point PCO** : **Commentaires** :
Définition du type
SET { field_1 INTEGER OPTIONAL, field_2 BOOLEAN, field_3 INTEGER OPTIONAL, field_4 INTEGER OPTIONAL }

La contrainte suivante est alors possible:

Déclaration de contrainte en notation ASN.1 d'unité PDU
Nom de la contrainte : CONS1 **Type de l'unité PDU** : XY_PDU **Chemin de dérivation** : **Commentaires** :
Valeur de la contrainte
{ field_1 5, field_2 TRUE, field_3 3 } *-- le champ 4 n'est pas spécifié=>omis au moment de l'envoi* *-- si l'identificateur field_3 n'avait pas été utilisé, l'attribution de la valeur 3 au champ 3 ou au* *-- champ 4 aurait été ambiguë, ces champs étant tous deux facultatifs.*

14.9 Déclarations de contrainte de messages CM en notation ASN.1

Les valeurs de paramètres pour les contraintes de messages CM seront données dans le format du Formulaire 44, ci-dessous.

Déclaration de contrainte de messages CM en notation ASN.1	
Nom de la contrainte	**:** *ConsId&ParList*
Groupe	**:** *[ASN1CM_ConstraintGroupReference]*
Type de message CM	**:** *CM_Identifier*
Chemin de dérivation	**:** *[DerivationPath]*
Commentaires	**:** *[FreeText]*
Valeur de la contrainte	
ConstraintValue&AttributesOrReplace	
Commentaires détaillés:	*[FreeText]*

Formulaire 44 – Déclaration de contrainte de messages CM en notation ASN.1

DÉFINITION SYNTAXIQUE:

555 ConsId&ParList ::= ConstraintIdentifier[FormalParList]

615 ASN1_CM_ConstraintGroupReference ::=[(SuiteIdentifier| TTCN_ModuleIdentifier)"/"]{ASN1_CM_ConstraintGroupIdentifier"/"}

424 CM_Identifier ::=Identifier

558 DerivationPath ::={ConstraintIdentifier Dot}+

595 ConstraintValue&AttributesOrReplace ::=ConstraintValue&Attribute| Replacement{Comma Replacement}

Ce formulaire est utilisé pour les contraintes de messages CM de la même manière que la déclaration de contraintes d'unité PDU est utilisée pour les unités PDU.

15 Partie dynamique

15.1 Introduction

La partie dynamique constitue le corps de la suite de tests. Elle regroupe les descriptions comportementales des tests élémentaires, des modules de test et des comportements par défaut.

15.2 Comportement dynamique de test élémentaire

15.2.1 Spécification de la table de comportement dynamique d'un test élémentaire

15.2.1.1 La table a pour titre "comportement dynamique de test élémentaire".

15.2.1.2 L'en-tête contient les informations suivantes:

a) le nom du test élémentaire,

attribuant un identificateur unique au test élémentaire décrit dans la table;

b) la référence de groupe du test,

indiquant le nom complet (au niveau le plus bas du chemin d'accès) du groupe qui contient le test élémentaire; ce nom complet respectera les spécifications du 9.2 et se terminera par une barre oblique (/);

c) l'objectif du test,

déclaration informelle de l'objectif du test élémentaire, tel qu'il figure dans la Recommandation (si elle existe) relative à la structure de la suite de tests et aux objectifs du test, ou dans une section équivalente de la Recommandation sur les suites de tests (si elle existe);

d) la référence du comportement par défaut,

identificateur (comprenant, si nécessaire, une liste des paramètres effectifs) d'une description de comportement par défaut, s'il y en a, s'appliquant à la description comportementale du test élémentaire (voir 15.4).

15.2.1.3 Le corps de la table comportera les colonnes suivantes avec les informations correspondantes:

a) une colonne numéro de ligne (facultative) (voir 15.2.5),

placée, lorsqu'elle existe, à l'extrême gauche de la table;

b) une colonne étiquette,

dans laquelle seront inscrites les étiquettes facultatives identifiant les déclarations TTCN et utilisées par les constructions de saut GOTO (voir 15.14);

c) une description comportementale,

décrivant le comportement des testeurs LT et UT en termes de déclarations TTCN et de leurs paramètres, à l'aide de la notation arborescente (voir 15.6);

d) une colonne référence aux contraintes,

dans laquelle sont inscrites les références aux contraintes permettant d'associer les déclarations TTCN de l'arbre comportemental à une référence aux valeurs, définies dans la partie contraintes d'une primitive ASP ou d'une unité PDU spécifique (voir le paragraphe 12);

e) une colonne verdict,

dans laquelle est inscrit le verdict ou l'information de résultat associés à des déclarations TTCN de l'arbre comportemental (voir 15.17);

f) une colonne commentaires (facultative),

utilisée pour faciliter la compréhension des déclarations TTCN en les commentant par de courtes remarques ou une référence à un texte complémentaire figurant dans le champ facultatif de bas de page commentaires détaillés.

Les colonnes c), d), e) et f) seront présentées de gauche à droite dans cet ordre. Il est recommandé de placer la colonne obligatoire étiquette immédiatement à gauche de la description comportementale. Sinon, elle peut être placée immédiatement à droite de cette description.

15.2.1.4 Un cadre de bas de page (facultatif) peut contenir les commentaires détaillés.

15.2.2 Formulaire de comportement dynamique de tests élémentaires

Le comportement dynamique d'un test élémentaire sera décrit sous le format du Formulaire 45, ci-dessous.

Comportement dynamique de test élémentaire					
Nom du test élémentaire : *TestCaseIdentifier*					
Groupe : *TestGroupReference*					
Objectif : *FreeText*					
Configuration : *TCompConfigIdentifier*					
Valeurs par défaut : *[DefaultRefList]*					
Commentaires : *[FreeText]*					
n°	**Etiquette**	**Description comportementale**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1 *2* . . . . *n*	. . *[Label]*	. . *StatementLine* *TreeHeader* *StatementLine*	. . *[ConstraintReference]*	. . *[Verdict]*	. . *[FreeText]*
Commentaires détaillés: *[FreeText]*					

Formulaire 45 – Comportement dynamique de test élémentaire

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent être abrégés en **E**, **RefC**, **V** et **C**, ce qui permet d'élargir le plus possible la colonne description comportementale si la dimension du support physique est limitée.

DÉFINITION SYNTAXIQUE:

624 TestCaseIdentifier ::= Identifier

626 TestGroupReference ::= [SuiteIdentifier "/"]{TestGroupIdentifier "/"}

329 TcompConfigIdentifier ::= Identifier

630 DefaultRefList ::=DefaultReference{Comma DefaultReference}

667 Label ::=Identifier

679 StatementLine ::= (Event[Qualifier][AssignmentList][TimerOps])| (Qualifier[AssignmentList][TimerOps])| (AssignmentList[TimerOps])| TimerOps| Construct| ImplicitSend

657 TreeHeader ::= TreeIdentifier [FormalParList]

669 ConstraintReference ::=ConsRef| FormalParIdentifier| AnyValue

674 Verdict ::=Pass| Fail| Inconclusive| Result

15.2.3 Structure du comportement d'un test élémentaire

Chaque test élémentaire contient une description précise des séquences d'événements (anticipés) et des verdicts correspondants. Cette description est structurée en un arbre dont les nœuds sont les déclarations TTCN, et les feuilles les verdicts correspondants. Il est souvent plus efficace de se servir des modules de test pour sous-structurer cet arbre (voir la Figure 7).

En notation TTCN, cette modularisation explicite est exprimée à l'aide de modules de test et de la structure ATTACH.

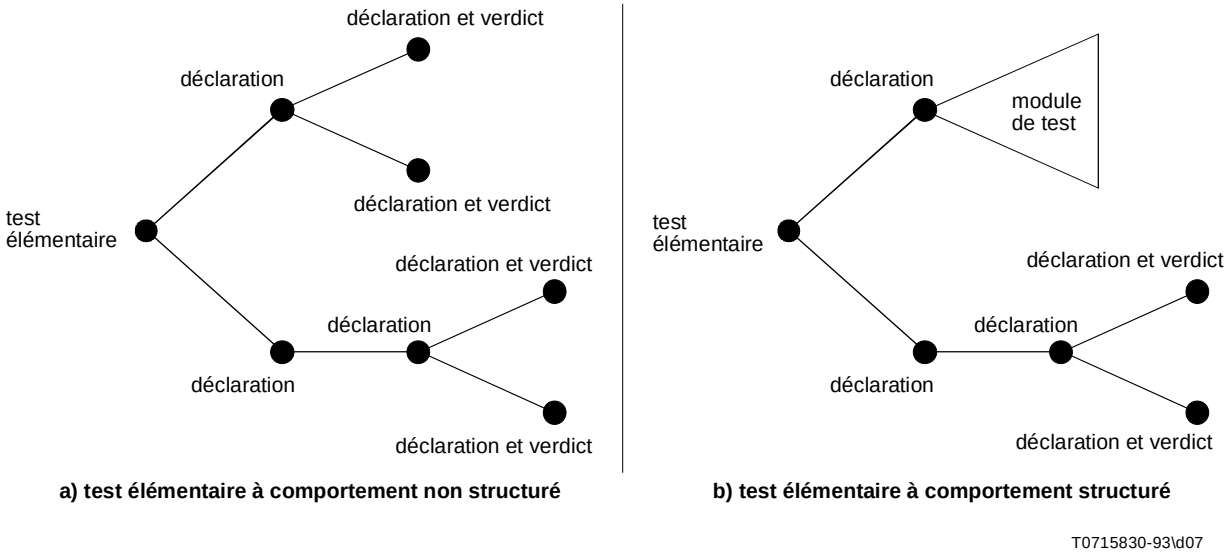


Figure 7/X.292 – Structure comportementale d'un test élémentaire

15.2.4 Description du comportement d'un test élémentaire concomitant

Si des composantes de test parallèles (PTC) sont utilisées dans un test élémentaire, l'en-tête contiendra la rubrique additionnelle configuration, qui identifiera une configuration de composantes de test (TCC, *test component configuration*) déclarée dans la partie déclarative.

Le comportement de la composante de test principale (MTC) est décrit par le premier arbre dans la table de comportement d'un test élémentaire ainsi que par tous les arbres qui lui sont rattachés. L'arbre comportemental de la MTC crée des PTC au besoin et associe chaque PTC à son propre arbre comportemental.

Si le comportement d'une PTC est spécifié en tant qu'arbre local dans le comportement d'un test élémentaire, la rubrique "Defaults Reference" sera vide. Cette restriction empêche la PTC d'hériter du comportement par défaut de la MTC.

Un test élémentaire doit seulement utiliser les composantes de test présentes dans la configuration de composantes de test à laquelle il est fait référence. Le choix de la configuration déterminera l'ensemble de points de contrôle et d'observation (PCO) et de points de coordination (CP) qui peuvent être utilisés dans le test élémentaire. Lorsqu'elle est utilisée, l'entrée configuration dans l'en-tête de comportement dynamique de test élémentaire doit être fournie dans le format indiqué dans le Formulaire 45.

15.2.5 Numérotation et suite des lignes

Une ligne (de description) comportementale pouvant être trop longue pour tenir sur une ligne du support physique, il est nécessaire de recourir à des symboles supplémentaires indiquant l'étalement des lignes comportementales. Deux techniques sont employées à cette fin:

a) indiquer le début d'une nouvelle ligne comportementale; une colonne supplémentaire de numérotation de lignes est ajoutée à gauche du corps de la table; cette colonne ne comporte de numérotation qu'au début de chaque nouvelle ligne comportementale; les numéros utilisés sont les entiers naturels 1, 2, 3, ... sans réinitialisation pour les arbres locaux; en d'autres termes, chaque ligne comportementale de la table aura un numéro de ligne unique;

   NOTE 1 – Les numéros de ligne peuvent être utilisés à des fins de consignation, pour désigner sans ambiguïté la ligne comportementale exécutée.

   NOTE 2 – Les numéros de ligne peuvent servir de références dans la section de commentaires détaillés.

b) indiquer la suite des lignes; si une ligne comportementale doit se poursuivre sur la ligne physique suivante, un signe (#) sera inscrit au début de cette ligne de suite de texte dans la colonne comportement; il est recommandé de donner aux lignes de suite la même indentation que la première ligne.

Si une ligne se poursuit dans une colonne autre que celle de la description comportementale, le signe (#) n'est pas nécessaire.

EXEMPLE 55 – Impression de longues lignes comportementales:

55.1 Style recommandé:

n°	Etiquette	Description comportementale	Réf. aux contraintes	Verdict	Commentaires
1		Ceci est une déclaration TTCN trop longue pour être imprimée sur une seule ligne car la colonne est trop étroite.	Ref1		
2		Ceci est la ligne de déclaration suivante	Ceci est une référence à une contrainte trop longue pour être imprimée sur une seule ligne		
3		Ligne d'une déclaration optionnelle	Ref2		

55.2 Autre style possible:

Etiquette	Description comportementale	Réf. aux contraintes	Verdict	Commentaires
	Ceci est une déclaration TTCN trop longue pour être imprimée sur une seule ligne car la colonne est trop étroite.	Ref1		
	Ceci est la ligne de déclaration suivante	Ceci est une référence à une contrainte trop longue pour être imprimée sur une seule ligne		
	Ligne d'une déclaration optionnelle	Ref2		

15.3 Comportement dynamique des modules de test

15.3.1 Spécification des tables de comportement dynamique des modules de test

Le comportement dynamique des modules de test est défini de la même manière que celui des tests élémentaires, à ceci près que les modules de test peuvent être paramétrés (voir 15.7). Les tables de comportement dynamique des modules de test sont identiques à celles des tests élémentaires, exception faite des différences suivantes:

a) la table a pour titre comportement dynamique de module de test;

b) la première rubrique de l'en-tête est nom du module de test,

qui est un identificateur unique du module de test suivi de la liste facultative des paramètres formels et des types associés. Ces paramètres peuvent être utilisés pour transférer des points PCO, des contraintes et d'autres objets de données vers l'arbre racine du module de test;

c) la deuxième rubrique de l'en-tête est la référence de groupe du module de test (groupe),

qui indique le nom complet (au niveau le plus bas du chemin d'accès) du groupe bibliothèque de modules de test contenant ce module de test; ce nom complet respectera les spécifications du 9.3 et se terminera par une barre oblique (/);

d) la troisième rubrique de l'en-tête est objectif (du module de test),

déclaration informelle de l'objectif du module de test.

15.3.2 Formulaire de comportement dynamique d'un module de test

Le comportement dynamique d'un module de test est décrit sous le format du Formulaire 46, ci-dessous.

Comportement dynamique de module de test					
Nom du module de test : *TestStepId&ParList*					
Groupe : *TestStepGroupReference*					
Objectif : *FreeText*					
Comportement par défaut : *[DefaultRefList]*					
Commentaires : *[FreeText]*					
n°	**Etiquette**	**Description comportementale**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1 *2* .	. . *[Label]*	. . *StatementLine*	. . *[ConstraintReference]*	. . *[Verdict]*	. . *[FreeText]*
. . . . . . *n*	. . . .	. . . *TreeHeader* *StatementLine* . .	. . . .	. . . .	. . . .
Commentaires détaillés: *[FreeText]*					

Formulaire 46 – Comportement dynamique de module de test

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent s'abréger en **E**, **RéfC**, **V** et **C**.

DÉFINITION SYNTAXIQUE:

638 TestStepId ::= TestStepIdentifier[FormalParList]
641 TestStepReference ::= [SuiteIdentifier "/"]{TestStepIdentifier "/"}
630 DefaultRefList ::=DefaultReference{Comma DefaultReference}
667 Label ::=Identifier
679 StatementLine ::= (Event[Qualifier][AssignmentList][TimerOps])| (Qualifier[AssignmentList][TimerOps])| (AssignmentList[TimerOps])| TimerOps| Construct| ImplicitSend
657 TreeHeader ::=TreeIdentifier [FormalParList]
669 ConstraintReference ::=ConsRef| FormalParIdentifier| AnyValue
674 Verdict ::=Pass| Fail| Inconclusive| Result

15.4 Comportement dynamique par défaut

15.4.1 Comportement par défaut

Un test élémentaire TTCN spécifiera un autre comportement pour *tous* les événements possibles (y compris les événements non valides). Dans un arbre comportemental, les séquences d'options (proposition de remplacement) aboutissent souvent toutes au même comportement. Ce comportement peut être mis en facteur comme comportement par défaut dans l'arbre. Ces descriptions de comportements par défaut sont regroupées dans la bibliothèque générale des comportements par défaut.

La dynamique des comportements par défaut est définie à l'aide des mêmes mécanismes que pour les modules de test, aux restrictions suivantes près:

a) un comportement par défaut n'a pas lui-même de comportement par défaut;

b) une description de comportement par défaut peut rattacher des arbres locaux (voir 15.7.1) mais elle ne doit pas rattacher de modules de test;

c) si des arbres locaux sont utilisés dans une description de comportement par défaut, ils ne doivent pas rattacher des modules de test;

d) le ou les arbres de la description comportementale n'utiliseront pas l'opération ACTIVATE (voir 15.18.4).

Les points PCO et les autres paramètres effectifs peuvent être transférés à des descriptions de comportement par défaut de la même manière qu'ils peuvent l'être à des modules de test. On appliquera les mêmes règles de visibilité et de substitution textuelle des paramètres qu'au rattachement d'arbres (voir 15.13).

15.4.2 Spécification des tables de comportement dynamique par défaut

Les tables de comportement dynamique par défaut sont identiques à celles des modules de test, aux différences suivantes près:

a) la table a pour titre "Comportement dynamique par défaut";

b) la première rubrique de l'en-tête est nom du comportement par défaut,

identificateur unique du comportement par défaut suivi de la liste facultative des paramètres formels et des types associés. Ces paramètres peuvent être utilisés pour transférer des points PCO, des contraintes ou d'autres objets de données vers l'arbre racine du comportement par défaut;

c) la deuxième rubrique de l'en-tête est groupe (du comportement par défaut),

indiquant le nom complet (au niveau le plus bas du chemin d'accès) du groupe de comportements par défaut contenant ce comportement par défaut; ce nom complet respectera les spécifications du 9.4 et se terminera par une barre oblique (/);

d) la troisième rubrique de l'en-tête est objectif (du comportement par défaut),

déclaration informelle de l'objectif du comportement par défaut.

15.4.3 Formulaire de comportement dynamique par défaut

Le comportement dynamique par défaut sera décrit dans le format illustré dans le Formulaire 47, ci-dessous.

Comportement dynamique par défaut					
Nom du comportement par défaut : *DefaultId&ParList*					
Groupe : *DefaultGroupReference*					
Objectif : *FreeText*					
Commentaires : *[FreeText]*					
n°	**Etiquette**	**Description comportementale**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1	.	.	.	.	.
2	.	.	.	.	.
.	*[Label]*	*StatementLine*	*[ConstraintReference]*	*[Verdict]*	*[FreeText]*
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	*TreeHeader*	.	.	.
.	.	*StatementLine*	.	.	.
.	.	.	.	.	.
n	.	.	.	.	.
Commentaires détaillés: *[FreeText]*					

Formulaire 47 – Comportement dynamique par défaut

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent s'abréger en **E**, **RéfC**, **V** et **C**.

DÉFINITION SYNTAXIQUE:

649 DefaultId&ParList ::=DefaultIdentifier{FormalParList}

651 DefaultGroupReference ::= [SuiteIdentifier "/"]{DefaultGroupIdentifier "/"}

667 Label ::=Identifier

679 StatementLine ::= (Event[Qualifier][AssignmentList][TimerOps])| (Qualifier[AssignmentList][TimerOps])| (AssignmentList[TimerOps])| TimerOps| Construct| ImplicitSend

657 TreeHeader ::= TreeIdentifier [FormalParList]

669 ConstraintReference ::=ConsRef| FormalParIdentifier| AnyValue

674 Verdict ::=Pass| Fail| Inconclusive| Result

15.5 Description comportementale

La colonne description comportementale d'une table de comportement dynamique regroupe les combinaisons de déclarations TTCN jugées possibles par le spécificateur de la suite de tests. L'ensemble de ces combinaisons est appelé arbre comportemental. Chaque déclaration TTCN constitue un nœud de l'arbre comportemental.

15.6 Notation arborescente

Chaque déclaration TTCN apparaîtra sur une ligne distincte. Les déclarations peuvent être reliées les unes aux autres de deux manières:

– en tant que séquences de déclarations TTCN;

– en tant que déclarations TTCN optionnelles.

Les séquences de déclarations TTCN sont représentées sur des lignes de déclarations successives, chaque déclaration étant décalée d'un niveau d'indentation vers la droite par rapport à la précédente.

EXEMPLE 56 – Séquence de déclarations TTCN:

```
EVENT_A
    CONSTRUCT_B
        EVENT_C
```

Les déclarations ayant le même niveau d'indentation et rattachées au même nœud antécédent représentent des déclarations optionnelles pouvant intervenir à un moment donné. Dès lors, cet ensemble de déclarations en notation TTCN est appelé *ensemble des déclarations optionnelles*, ou simplement *options*.

EXEMPLE 57 – Déclarations TTCN optionnelles:

```
CONSTRUCT_A1
STATEMENT_A2
EVENT_A3
```

EXEMPLE 58 – Combinaison de séquences de déclarations et d'options dans la construction d'un arbre:

```
EVENT_A
    CONSTRUCT_B
        EVENT_C
    STATEMENT_D1
    EVENT_D2
```

Le résultat de l'évaluation d'une déclaration TTCN dépend des diverses conditions qui lui sont associées. Les conditions des différentes déclarations ne sont pas nécessairement mutuellement exclusives, c'est-à-dire qu'il est possible qu'à un moment donné, plusieurs déclarations prennent la valeur TRUE. Les lignes de déclaration optionnelle étant évaluées dans leur ordre d'apparition dans l'ensemble d'options, c'est la première pour laquelle les conditions seront remplies qui sera prise en considération. Ceci pourra engendrer des comportements non atteignables, lorsque des déclarations sont codées comme déclarations optionnelles à la suite d'autres déclarations toujours vraies.

Les opérations GOTO et REPEAT sont assimilées à des déclarations toujours vraies. De même, les opérations SEND (envoi), IMPLICIT SEND (envoi implicite), d'affectation et de temporisation sont également assimilées à des déclarations vraies sous réserve que le qualificateur d'accompagnement, s'il existe, ait la valeur TRUE (vrai).

En notation TTCN.GR, l'indentation graphique des lignes de déclaration est mappée avec les valeurs d'indentation en notation TTCN.MP. Les déclarations du premier niveau d'indentation des déclarations optionnelles n'ayant pas d'antécédent dans l'arbre racine ou dans l'arbre local auquel elles appartiennent auront zéro pour valeur d'indentation. Les déclarations ayant un antécédent prendront comme valeur d'indentation celle de cet antécédent, plus un.

DÉFINITION SYNTAXIQUE:

664 Line ::= **$Line** Indentation StatementLine

EXEMPLE 59 – $Line [6] +R1_POSTAMBLE

15.7 Noms des arbres et listes de paramètres

15.7.1 Introduction

Chaque description comportementale contiendra au moins un arbre comportemental. Chacun de ces arbres recevra un nom pour pouvoir être cité en référence (dans une construction ATTACH, par exemple) de manière non ambiguë.

Le premier arbre apparaissant dans une description comportementale est appelé arbre racine. Le nom d'un arbre racine est l'identificateur apparaissant dans l'en-tête de sa table de comportement dynamique. Par exemple, le nom de l'arbre racine d'un module de test est l'identificateur de ce module de test, et il en est de même pour les arbres racines des comportements dynamiques des tests élémentaires et des comportements dynamiques par défaut.

Les arbres autres que l'arbre racine qui apparaissent dans les tables de comportement dynamique sont appelés arbres locaux. Ils sont précédés d'un en-tête d'arbre contenant le nom de l'arbre.

DÉFINITION SYNTAXIQUE:

654 RootTree ::={BehaviourLine}+
655 LocalTree ::=Header {BehaviourLine}+

15.7.2 Arbres paramétrés

Tous les arbres, à l'exception des arbres racines des tests élémentaires, peuvent être paramétrés. Les paramètres peuvent fournir des points PCO, des contraintes, des variables et d'autres éléments de la sorte à utiliser dans cet arbre. Les arbres racines des tests élémentaires ne sont pas paramétrés.

Si un arbre est paramétré, son nom sera directement suivi par une liste entre parenthèses des paramètres formels avec leurs types. Par exemple, la liste des paramètres formels de l'arbre racine d'un module de test sera indiquée entre parenthèses immédiatement après l'identificateur du module de test dans l'en-tête de la table de comportement dynamique du module de test. De même, la liste des paramètres formels d'un arbre local suivra immédiatement le nom de l'arbre dans son en-tête.

Dans la liste, chaque paramètre formel sera suivi de deux points et de son type. S'il existe plusieurs paramètres formels du même type, ils pourront être regroupés dans une sous-liste. Dans ces sous-listes, les paramètres formels sont séparés les uns des autres par une virgule, le dernier étant suivi de deux points (:) et du nom de son type.

Lorsqu'il y a plus d'un couple paramètre formel/type (ou plus d'un couple sous-liste/type), ces couples sont séparés les uns des autres par un point-virgule (;).

Les paramètres formels peuvent appartenir aux types suivants: point PCO, primitive ASP, unité PDU, structuré, ou appartenir à l'un des autres types prédéfinis ou types de suite de tests.

Si le symbole **PDU** est donné comme type de paramètre formel d'un arbre, il ne faudra pas faire référence dans l'arbre à des champs particuliers de l'unité PDU. Si le paramètre formel est l'identificateur d'une unité PDU particulière, il est alors possible de faire référence à des champs particuliers de cette unité PDU dans cet arbre.

EXEMPLE 60 – Module de test utilisant des paramètres formels: EXEMPLE_TREE (L:TSAP; X:INTEGER; Y:INTEGER)

EXEMPLE 61 – Module de test utilisant un paramètre formel avec sous-liste: EXEMPLE_TREE (L:TSAP; X, Y:INTEGER)

15.8 Déclarations TTCN

La notation arborescente permet de spécifier des événements de test déclenchés par le ou les testeurs inférieurs ou supérieurs [événements SEND (envoi) et IMPLICIT SEND (envoi implicite)], des événements de test reçus par le ou les testeurs inférieurs ou supérieurs [RECEIVE (réception), OTHERWISE (sinon), TIMEOUT (fin de temporisation) et DONE (terminé)], des constructions (GOTO, ATTACH, REPEAT, CREATE, RETURN et ACTIVATE) et des pseudo-événements comprenant des combinaisons de qualificateurs et d'opérations d'affectation et de temporisation. Tous ces événements et toutes ces opérations sont collectivement appelés déclarations TTCN.

Les événements de test peuvent être accompagnés de qualificateurs (expressions booléennes), d'opérations d'affectation et de temporisation. Ces opérations et ces qualificateurs peuvent constituer à eux seuls des déclarations TTCN et sont alors appelés pseudo-événements.

15.9 Evénements de test TTCN

15.9.1 Evénements d'envoi et de réception

La notation TTCN prend en charge le lancement (l'envoi) de primitives ASP et d'unités PDU vers des points PCO nommés, ainsi que l'acceptation (la réception) de primitives ASP et d'unités PDU en des points PCO nommés. Le modèle de point PCO est défini aux 11.10 et 15.9.5.3. La notation TTCN concomitante prend en charge l'envoi de messages de coordination CM à des points de coordination (CP) nommés, ainsi que la réception à ces points CP. Le modèle de CP est défini en 11.11.

DÉFINITION SYNTAXIQUE:

682 Send ::=[PCO_Identifier| CP_Identifier| FormalParIdentifier] "!"(ASP_Identifier| PDU_Identifier| CM_Identifier)

684 Receive ::= [PCO_Identifier| CP_Identifier| FormalParIdentifier] "?" (ASP_Identifier| PDU_Identifier| CM_Identifier)

Dans la forme la plus simple, un identificateur de primitive ASP ou d'unité PDU suit le symbole d'envoi (!) pour les événements devant être déclenchés par les testeurs LT ou UT, ou le symbole de réception (?) pour les événements que ces testeurs peuvent accepter. Le nom facultatif d'un point PCO facultatif n'est pas indiqué. Cette forme n'est valide que lorsque la suite de tests ne comporte qu'un seul point PCO.

EXEMPLE 62 – !CONreq ou ?CONind

Si la suite de tests comporte plus d'un point PCO, le symbole d'envoi (!) ou de réception (?) sera précédé d'un nom de point PCO apparaissant dans la partie déclarative ou dans la liste des paramètres formels de l'arbre. Ce nom de point PCO désigne le point PCO auquel l'événement de test peut se produire.

EXEMPLE 63 – L! CONreq ou L? CONind

Dans le cas des points CP, l'identificateur de CP précédera le symbole d'envoi dans le cas de l'envoi d'un message CM et le symbole de réception dans le cas de la réception d'un message CM.

EXEMPLE 64 – A_CP!A_CM ou A_CP?A_CM

15.9.2 Evénements de réception

Le résultat d'une ligne d'événement RECEIVE est "vrai" si une primitive ASP ou une unité PDU entrante au point PCO spécifié concorde avec la ligne d'événement. Pour que cette concordance ait lieu, les conditions suivantes devront être réunies:

a) l'unité PDU entrante peut être décodée conformément aux règles de codage applicables;

b) la primitive ASP ou l'unité PDU entrante est valide par rapport à la définition du type de primitive ASP ou d'unité PDU donné en référence par le nom d'événement sur cette ligne. En particulier, toutes les valeurs de paramètres et de champs seront du type défini et respecteront toutes les restrictions de longueur spécifiées;

c) la primitive ASP ou l'unité PDU concorde avec la référence à une contrainte mentionnée sur cette ligne d'événement;

d) si un qualificateur est spécifié sur la ligne d'événement, son résultat doit être "vrai" (TRUE); ce qualificateur peut faire référence à des paramètres de primitive ASP et à des champs d'unité PDU.

Un événement entrant n'est retiré de la file d'entrée d'un point PCO que s'il concorde avec une ligne d'événement de réception.

En notation TTCN concomitante, la réception et la mise en correspondance d'un message CM à un point CP sont traitées de la même manière que ci-dessus.

15.9.3 Evénements d'envoi

Le résultat d'une ligne d'événement SEND comportant un qualificateur est "vrai" si l'expression de ce qualificateur prend la valeur "vrai" (TRUE). Les événements d'envoi non qualifiés sont assimilés à des événements toujours "vrais". La primitive ASP ou l'unité PDU sortante résultant d'un événement d'envoi sera formée de la manière suivante:

a) toutes les valeurs des paramètres de la primitive ASP ou des champs de l'unité PDU seront du type spécifié dans les définitions correspondantes et respecteront toutes les restrictions de longueur contenues dans ces définitions;

b) les paramètres de la primitive ASP ou les champs de l'unité PDU prendront les valeurs spécifiées dans la contrainte indiquée en référence sur la ligne d'événement (voir les paragraphes 12, 13 et 14 pour l'explication des constructions de primitives ASP et d'unités PDU avec contraintes);

c) toute valeur directement affectée aux paramètres de primitive ASP ou aux champs d'unité PDU sur la ligne d'événement prendra le pas sur la valeur correspondante éventuellement spécifiée dans la contrainte;

d) tous les paramètres ou les champs de la primitive ASP ou de l'unité PDU sortante comporteront des valeurs spécifiques ou seront explicitement omis avant la réalisation de l'événement d'envoi SEND;

e) l'unité PDU totalement construite doit être codée conformément aux règles de codage applicables.

La génération, par contrainte ou affectation d'une valeur de paramètre de primitive ASP ou de champ d'unité PDU enfreignant la déclaration de type ou la restriction longueur entraînera une erreur de test élémentaire.

En notation TTCN concomitante, l'envoi de messages CM à des points CP est traité de la même manière que ci-dessus.

15.9.4 Durée de vie des événements

Les identificateurs de paramètres de primitive ASP et de champs d'unité PDU associés aux événements SEND et RECEIVE ne pourront faire référence aux valeurs des paramètres de primitive ASP et des champs d'unité PDU que sur la ligne de déclaration elle-même.

Dans le cas des événements SEND, les paramètres des primitives ASP et les champs des unités PDU correspondants peuvent comporter des valeurs, si nécessaire, dans des affectations réalisées sur la ligne d'envoi.

EXEMPLE 65 – !A_PDU (A_PDU.FIELD:=3)

Une telle affectation n'a pas d'effet en dehors de la ligne d'événement dans laquelle elle intervient; elle ne survit pas à l'événement d'envoi.

Dans le cas des événements RECEIVE, s'il est nécessaire de faire référence par la suite à des valeurs de paramètres de primitive ASP ou de champs d'unité PDU, on affectera tout ou partie de la primitive ASP ou de l'unité PDU à des variables sur la ligne de réception elle-même. Il est alors possible de faire référence à ces variables dans les lignes suivantes.

EXEMPLE 66 – ?A_PDU (VAR:=A_PDU.FIELD)

VAR pouvant être utilisé sur des lignes d'événement après la réception de A_PDU.

La durée de vie des messages CM est aussi restreinte à la déclaration RECEIVE pertinente. L'accès aux identificateurs des champs CM peut se faire d'une manière semblable à celle des identificateurs des champs d'unité PDU.

EXEMPLE 67 – A_CP!A_CM ou A_CP?A_CM

15.9.5 Exécution de l'arbre comportemental

15.9.5.1 Introduction

Le spécificateur de la suite de tests organisera l'arbre comportemental représentant un test élémentaire ou un module de test conformément aux règles suivantes relatives à l'exécution des tests:

a) démarrant depuis la racine de l'arbre, le testeur LT ou UT reste au premier niveau d'indentation jusqu'à ce qu'un événement concorde. Si un événement doit être déclenché, le testeur LT ou UT le déclenche; si un événement doit être reçu, il n'est considéré comme concordant que si un événement réel est reçu et qu'il concorde avec la ligne d'événement;

b) dès qu'un événement a concordé, le testeur LT ou UT passe au niveau suivant d'indentation. Il n'est possible de revenir à un niveau d'indentation précédent qu'en utilisant la construction de saut GOTO;

c) les lignes d'événements de même niveau d'indentation venant après une même ligne d'événement antécédente représentent les déclarations optionnelles pouvant être mises en concordance à un moment donné. Les déclarations optionnelles seront indiquées dans l'ordre dans lequel le concepteur de la suite de tests demande au testeur LT ou UT de tenter de les déclencher ou de les recevoir, si nécessaire de manière récurrente, jusqu'à ce que l'une de ces déclarations optionnelles concorde.

EXEMPLE 68 – Exemple d'arbre comportemental TTCN:

Supposons que la séquence d'événements suivante puisse intervenir pendant un test dont l'objectif est d'établir une liaison, d'échanger des données et de libérer cette liaison. Ces événements se produisent au point PCO du testeur inférieur appelé L:

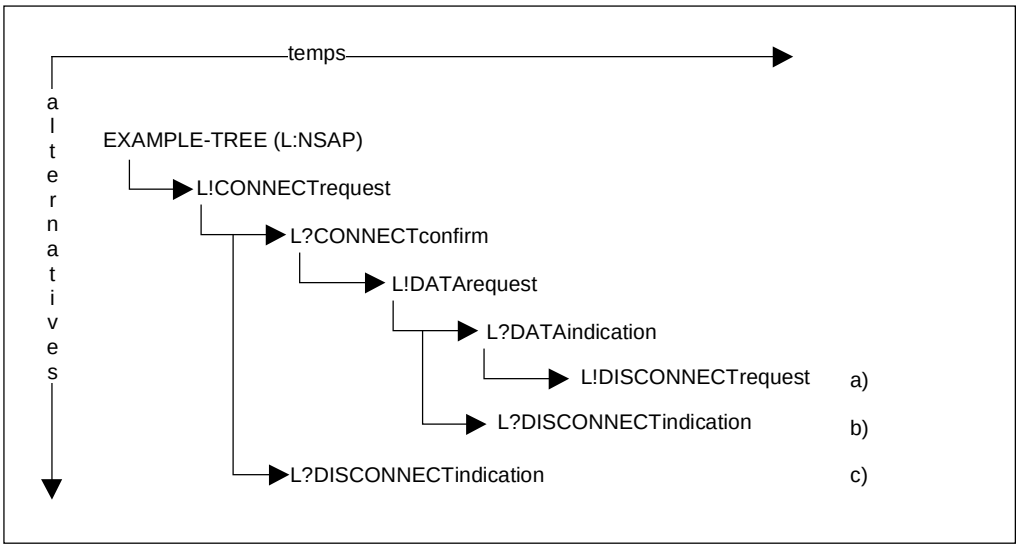
a) CONNECTrequest (demande de connexion), CONNECTconfirm (confirmation de connexion), DATArequest (demande de données), DATAindication (indication de données), DISCONNECTrequest (demande de déconnexion).

   Leur déroulement peut être bloqué à tout moment par l'implémentation sous test ou par le fournisseur de service, ce qui génère deux autres séquences:

b) CONNECTrequest, CONNECTconfirm, DATArequest, DISCONNECTindication;

c) CONNECTrequest, DISCONNECTindication.

Ces trois séquences d'événements peuvent s'exprimer sous forme d'un arbre comportemental TTCN. Il existe cinq niveaux d'options, et trois feuilles seulement [de a) à c)], les événements SEND L! étant toujours "vrais". L'exécution doit se dérouler de gauche à droite (déroulement séquentiel) et de haut en bas (choix conditionnels). La figure suivante illustre ce déroulement, ainsi que le principe d'arbre comportemental TTCN:



T0731090-98\d08

En fait, il n'existe ni traits, ni flèches, ni noms de feuilles en notation TTCN, et l'arbre comportemental de l'exemple précédent devrait être représenté comme suit:

EXEMPLE 69 – Arbre comportemental TTCN:

Comportement dynamique de module de test

Nom du module de test : TREE_EX_1(L:NSAP)
Groupe : TTCN_EXAMPLES/TREE_EXAMPLE_1/
Objectif : illustrer l'utilisation des arbres
Comportement par défaut : NOTE – Cet exemple peut être simplifié en utilisant des comportements par défaut.

n°	Etiquette	Description comportementale	Réf. des contraintes	Verdict	Commentaires
1		L!CONNECTrequest	CR1		Demande de connexion
2		L?CONNECTconfirm	CC1		Confirmation de connexion
3		L!DATArequest	DTR1		Demande d'envoi de données
4		L?DATAindication	DTI1		Réception de données
5		L!DISCONNECTrequest	DSCR1	PASS	Acceptation
6		L?DISCONNECTindication	DSCI1	INCONC	Déconnexion prématurée
7		L?DISCONNECTindication	DSCR1	INCONC	Déconnexion prématurée

15.9.5.2 Sémantique d'image instantanée

Les déclarations optionnelles du niveau d'indentation courant sont traitées dans leur ordre d'apparition. La sémantique opératoire TTCN (voir l'Annexe B) suppose que l'état de tous les événements reste inchangé durant tout le processus de recherche de concordance avec une des déclarations optionnelles appartenant au même ensemble d'options. Cela implique que la sémantique d'image instantanée est utilisée pour les événements reçus et les fins de temporisation; en d'autres termes, chaque fois qu'on aborde un ensemble d'options, on mémorise une image instantanée de l'ensemble des événements reçus et des fins de temporisation apparues à cet instant. Seuls les événements et fins de temporisation identifiés dans cette image instantanée pourront intervenir dans le processus de concordance au cours du cycle suivant de parcours des déclarations optionnelles.

15.9.5.3 Limites d'utilisation des événements

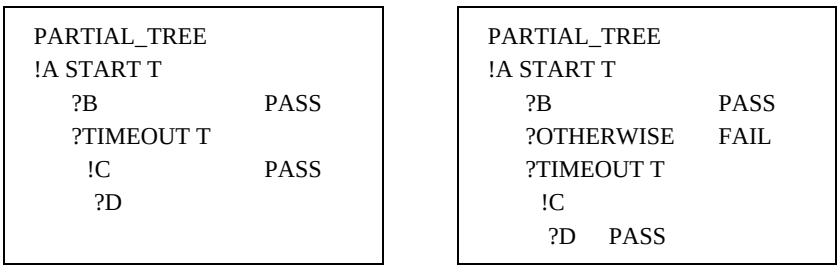
Afin d'éviter toute erreur de test élémentaire, les règles suivantes s'appliquent:

a) un test élémentaire ou un module de test ne doit pas contenir de comportements pour lesquels la vitesse de traitement relative des moyens de tester risque d'influer sur les résultats. Pour éviter de tels problèmes, une ligne d'événement RECEIVE (réception), OTHERWISE (sinon) ou TIMEOUT (fin de temporisation) sera exclusivement suivie, dans un ensemble d'options, par d'autres lignes d'événement du même type (événements réception, sinon ou fin de temporisation). Par conséquent, les arbres de comportement par défaut ne contiennent que des lignes événements RECEIVE, OTHERWISE et TIMEOUT au premier niveau d'indentation des déclarations optionnelles;

b) lorsqu'un événement se trouve sur une file d'attente de points PCO ou CP, ou lorsqu'une fin de temporisation se trouve sur la liste des fins de temporisation, ils ne peuvent en être retirés qu'après une recherche de concordance réussie avec la déclaration TTCN correspondante. Dans le cas d'un ensemble d'options incluant des déclarations RECEIVE, l'ensemble des événements de réception attendus doit être spécifié dans sa totalité. Cela signifie qu'il devra se produire une erreur de test élémentaire si, pendant l'exécution, aucune concordance de déclaration de réception ne se produit et que cependant l'exécution passe au niveau de possibilités conditionnelles suivant du fait de l'apparition d'une fin de temporisation après la réception, sur l'une des files de l'un quelconque des points PCO pertinents, d'une primitive ASP ou d'une unité PDU qui n'était pas spécifiée dans l'ensemble des déclarations de réception. L'envoi implicite ne doit pas être utilisé avec les messages CM;

c) des précautions particulières doivent être prises dans l'emploi de la notation TTCN concomitante afin d'éviter les résultats non fiables obtenus lorsque l'ordre de réception des événements à divers points PCO ou CP est utilisé pour déterminer l'affectation du verdict. Le moment réel où l'unité PDU ou le message CM est reçu, par rapport au moment de réception d'autres unités PDU ou messages CM, peut ne pas être représenté correctement lors de l'exécution de modules de test parallèles;

EXEMPLE 70 – Ensemble incomplet d'événements RECEIVE (réception):



a) **b)**

Dans le cas a), si D est reçu en réponse à !A, le test élémentaire affectera un verdict "succès" erroné dès la fin de la temporisation. Cette procédure peut être évitée par une déclaration OTHERWISE (sinon).

d) en notation TTCN concomitante, le séquencement relatif des événements à divers points PCO ou points CP ne doit pas avoir d'effet sur le verdict affecté, car cela pourrait rendre impossible la répétabilité des résultats en raison des différences sur le plan des vitesses de traitement et de transmission.

15.9.5.4 Précautions lors de l'utilisation de la notation TTCN concomitante

Des précautions particulières doivent être prises dans l'emploi de la notation TTCN concomitante afin d'éviter les résultats non répétables obtenus lorsque l'ordre de réception des événements à divers points PCO ou CP est utilisé pour déterminer l'affectation du verdict. Le moment réel où l'unité PDU ou le message CM est reçu, par rapport au moment de réception d'autres unités PDU ou messages CM, peut ne pas être représenté correctement lors de l'exécution de modules de test parallèles.

15.9.6 Evénement IMPLICIT SEND (envoi implicite)

Dans les méthodes de test distant, bien qu'il n'y ait pas de point PCO explicite au-dessus de l'implémentation sous test, il est nécessaire de pouvoir spécifier, en un point donné de la description comportementale du testeur LT, qu'il faut faire en sorte que cette implémentation IUT lance une primitive ASP ou une unité PDU particulière (mais non un message CM). A cette fin, on définit l'événement envoi implicite répondant à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

683 ImplicitSend ::= "<" IUT "!" (ASP_Identifier | PDU_Identifier)">"

Dans cette syntaxe, l'identificateur du point PCO utilisé avec un événement normal SEND ou RECEIVE est remplacé par le symbole **IUT**, pour indiquer que l'implémentation sous test (IUT) doit envoyer la primitive ASP ou l'unité PDU spécifiée. Les parenthèses angulaires (chevrons couchés) signifient qu'il s'agit d'un événement implicite, c'est-à-dire qu'on ne spécifie pas les actions à exécuter sur l'implémentation IUT pour déclencher cette réaction, mais seulement la réaction voulue elle-même.

Un événement IMPLICIT SEND est toujours considéré comme réussi, en ce sens que toute option codée à la suite de cet événement au même niveau d'indentation sera non atteignable.

On n'utilisera un IMPLICIT SEND que lorsque les Recommandations OSI correspondantes autorisent l'implémentation IUT à envoyer la primitive ASP ou l'unité PDU spécifiée depuis ce point lors d'une communication avec le testeur LT.

Pour chaque événement IMPLICIT SEND dans une suite de tests, le concepteur de cette suite de tests créera et référencera une question dans le formulaire PIXIT partiel, indiquant s'il est possible d'invoquer à la demande cet événement IMPLICIT SEND.

On n'utilisera un événement IMPLICIT SEND que si la méthode de test utilisée est une méthode de test distant et qu'il soit possible d'obtenir le même effet avec la méthode de test monocouche répartie.

NOTE – Par exemple, lorsqu'on teste une implémentation de protocole de transport orientée connexion, si cette restriction n'existait pas, il serait possible d'utiliser un événement IMPLICIT SEND pour demander à l'implémentation IUT de lancer une unité de données protocolaires de transport "demande de connexion" CR TPDU, car la méthode de test monocouche répartie permet de parvenir au même effet en demandant au testeur UT d'envoyer une primitive ASP de demande de connexion-transport T-CONreq. Par ailleurs, il ne serait pas possible d'utiliser un IMPLICIT SEND pour demander à l'implémentation IUT de lancer une primitive ASP de demande de rétablissement-réseau N-RstReq, cet effet ne pouvant être commandé à travers la limite du service transport. Cette restriction a pour objet d'empêcher des tests élémentaires d'exiger d'avoir un contrôle externe sur une implémentation IUT plus important que celui qui est prévu dans les Recommandations correspondantes sur les protocoles.

Lorsqu'un événement IMPLICIT SEND est spécifié, on exécute également les événements internes associés dans l'implémentation IUT (par exemple établir une temporisation, initialiser des variables d'état) qui sont nécessaires au respect des spécifications de la Recommandation relative au protocole testé.

La sémantique de l'événement IMPLICIT SEND fait que le système sous test reçoit les commandes nécessaires au lancement des primitives ASP ou unités PDU spécifiées. Le formulaire PIXIT (ou les documents cités en référence par celui-ci) spécifiera le mode de commande du système sous test.

Ni un verdict final ni un résultat préliminaire ne sera codé sur la ligne événement d'un événement IMPLICIT SEND.

Un événement de réception doit se trouver en un point approprié à la suite d'un événement IMPLICIT SEND afin de vérifier la concordance de la primitive ASP ou de l'unité PDU envoyée par l'implémentation IUT.

EXEMPLE 71 – Exemple d'utilisation d'un événement IMPLICIT SEND:

Comportement dynamique de test élémentaire					
Nom du test élémentaire : IMPI **Groupe** : TTCN_EXAMPLES/IMPLICIT_SEND1/ **Objectif** : arbre partiel illustrant l'utilisation d'un envoi implicite **Comportement par défaut** : **Commentaires** :					
n°	**Etiquette**	**Description comportementale**	**Réf. des contraintes**	**Verdict**	**Commentaires**
: 5 6 7 : 12 :		: <IUT!CR> L?CR L!CC : L?OTHERWISE :	 CR1 CR1 CC1 		

15.9.7 Evénement OTHERWISE (sinon)

L'événement prédéfini OTHERWISE est le mécanisme TTCN permettant de traiter de manière contrôlée les événements de test imprévus. Il a la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

685 Otherwise ::= [PCO_Identifier| CP_Identifier| FormalParIdentifier] "?" **OTHERWISE**

L'événement OTHERWISE est utilisé pour indiquer que le testeur inférieur ou supérieur accepte *tout* événement entrant n'ayant pas précédemment concordé avec l'une des déclarations optionnelles concurrentes de l'événement OTHERWISE. Le testeur doit accepter toute donnée entrante qu'il n'a pas été possible de décoder ou qui n'a pas correspondu à un autre événement que cet événement OTHERWISE.

En notation TTCN non concomitante, lorsqu'une suite de tests comprend plus d'un point PCO, l'événement OTHERWISE sera préfixé par un nom de point PCO apparaissant dans la partie déclarative ou dans la liste des paramètres formels de l'arbre où ce paramètre formel sert à indiquer un nom de point PCO. Le nom de point PCO indique le point PCO où l'événement de test peut intervenir. Les événements entrants, y compris l'événement OTHERWISE, ne sont considérés que du point de vue du point PCO donné.

EXEMPLE 72 – Utilisation de l'événement OTHERWISE avec des identificateurs de point PCO:

PARTIAL_TREE	
PCO1?A	
PCO2?B	PASS
PCO1?C	INCONC
PCO2?OTHERWISE	FAIL

Si aucun événement n'est reçu au point PCO1, la réception de l'événement B au point PCO2 conduit à un verdict de succès. La réception d'un autre événement en ce point PCO2 conduit à un verdict d'échec.

Etant donné l'importance de l'ordre des déclarations optionnelles, les événements entrants constituant des déclarations optionnelles à la suite d'un événement OTHERWISE inconditionnel au même point PCO ne concorderont jamais.

EXEMPLE 73 – Evénements entrants à la suite d'un événement OTHERWISE:

PARTIAL_TREE	
PCO1?A	PASS
PCO1?OTHERWISE	FAIL
PCO1?C	INCONC

L'événement OTHERWISE concorde avec tout événement entrant différent de A et de B. La dernière option (?C) ne pourra jamais concorder.

15.9.8 Evénement OTHERWISE et notation TTCN concomitante

En notation TTCN concomitante, l'événement OTHERWISE peut être utilisé avec des points CP ainsi qu'avec des points PCO. L'événement OTHERWISE à des points CP est permis afin d'offrir un moyen efficace de traitement de "tous les autres messages CM à ce point CP".

15.9.9 Evénement TIMEOUT (fin de temporisation)

L'événement TIMEOUT permet de vérifier dans un test élémentaire l'expiration d'une ou de toutes les temporisations. Lorsqu'un temporisateur s'arrête (conceptuellement immédiatement avant le traitement par image instantanée d'un ensemble d'options concurrentes), un événement TIMEOUT est placé dans une liste de fins de temporisation. Le temporisateur se désactive immédiatement. Pour un temporisateur donné, une seule entrée peut apparaître dans cette liste à un moment donné. Les fins de temporisation n'étant pas associées à des points PCO, les fins de temporisation sont regroupées en une seule liste.

Si un nom de temporisateur est indiqué dans le traitement d'un événement TIMEOUT, un tel événement correspondant à ce nom est recherché dans la liste des fins de temporisation et, s'il est trouvé, il est retiré de la liste et il est considéré comme ayant réussi.

Si aucun nom de temporisateur n'est indiqué, tout événement TIMEOUT de la liste concorde. L'événement TIMEOUT est réussi si la liste n'est pas vide. Dès qu'il y a une concordance, toute la liste de fins de temporisation est vidée.

L'événement TIMEOUT obéit à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

686 Timeout ::= "?"TIMEOUT [TimerIdentifier| FormalParIdentifier]

EXEMPLE 74 – Utilisation de TIMEOUT:

	?TIMEOUT T			

Les événements de TIMEOUT n'étant pas des événements de RECEIVE, les propositions OTHERWISE antérieurement codées ne les rendent pas non atteignables.

15.9.10 Evénements et constructions en notation TTCN concomitante

La construction CREATE et l'événement DONE sont utilisés en notation TTCN concomitante.

15.9.10.1 Construction CREATE

La composante de test principale est déclenchée au début de l'exécution d'un test élémentaire. Au besoin, la composante de test principale déclenche des composantes de test parallèles au moyen de la construction CREATE, qui respecte la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

692 Create ::= CREATE "(" CreateList ")"

La construction invoque un ensemble de composantes de test parallèles (PTC, *parallel test components*). Il y a deux arguments pour chaque cmposante PTC. Le premier est l'identificateur de la PTC créée, qui doit correspondre à l'identificateur d'une PTC dans la configuration de composantes de test mentionnée dans l'en-tête du test élémentaire. Le deuxième est un renvoi à un arbre comportemental (c'est-à-dire module de test ou arbre local), qui peut être accompagné d'une liste de paramètres contenant des valeurs réelles (par exemple des points PCO et CP). L'effet de la construction CREATE est que chaque PTC indiquée commence l'exécution de sa description comportementale en parallèle avec l'exécution de la composante d'essai principale.

NOTE – Le transfert d'identificateurs de points PCO et CP à un arbre comportemental comme paramètres effectifs permet qu'un même arbre comportemental soit utilisé par plus d'une composante de test.

Les points PCO et CP utilisés lors de l'exécution de la description comportementale associée à une composante PTC par la construction CREATE doivent être seulement les points déterminés par la configuration de composantes de test pour le test élémentaire en question.

L'exécution d'une construction CREATE dans une composante PTC qui a déjà été créée doit produire une erreur de test élémentaire. L'exécution d'une construction CREATE par toute composante de test autre que la composante MTC doit produire une erreur de test élémentaire.

Dans la construction CREATE, les identificateurs de points PCO et CP sont transférés à une composante PTC par substitution textuelle, ce qui est habituel lors du rattachement de tests élémentaires. Tous les autres paramètres sont transférés sous forme de valeurs. Cela permet de prévenir les effets secondaires sur les variables qui pourraient avoir un effet sur le traitement d'autres composantes PTC, dont pourraient découler des résultats non répétables.

15.9.10.2 Evénement DONE

Lorsque la composante MTC se termine, le verdict final est affecté par la composante MTC tel qu'il est calculé jusqu'à ce moment (voir 15.17.5). L'événement DONE peut être utilisé dans les composantes MTC et PTC afin de savoir si les composantes PTC sont terminées. Les composantes de test peuvent utiliser cette information afin de déterminer leurs propres résultats préliminaires ainsi que les actions à suivre; en particulier la composante MTC peut éviter de se terminer avant que toutes les autres composantes PTC se soient terminées (voir 15.17.5).

DÉFINITION SYNTAXIQUE:

687 Done ::= "?" DONE "("[TcompIdList] ")"

Une liste d'arguments manquante est interprétée comme une liste de toutes les composantes PTC énoncées dans une construction CREATE exécutée avant l'exécution de l'événement DONE. Un événement DONE sans liste d'arguments ne doit être utilisé que par la composante MTC.

EXEMPLE 75 – Utilisation de l'événement DONE:

```
PARTIAL_MTC_TREE

CREATE(PTC1:TREEA)
  CREATE(PTC2:TREEB)
      START T1
          ?DONE(PTC1,PTC2)
          ?TIMEOUT T1                FAIL
```

NOTE 1 – L'utilisation de ?TIMEOUT est recommandée comme déclaration de remplacement de ?DONE.

NOTE 2 – Si DONE est la seule déclaration optionnelle, elle correspond à un ordre signifiant d'attendre la fin des composantes PTC spécifiées.

NOTE 3 – DONE ne représente pas un moyen pour que la composante MTC coordonne la fin des composantes PTC. La fin ne peut être atteinte qu'au moyen d'un échange approprié de messages CM. La notation TTCN ne fournit pas de messages CM prédéfinis à cet usage.

15.10 Expressions

15.10.1 Introduction

La notation TTCN comporte deux sortes d'expressions: les affectations et les expressions booléennes, qui peuvent toutes deux contenir des valeurs explicites et les formes suivantes de référence à des objets de données:

a) paramètres de suite de tests;

b) constantes de suite de tests;

c) variables de suite de tests et de test élémentaire;

d) paramètres formels d'un arbre de module de test, de comportement par défaut ou d'un arbre local;

e) primitives ASP et unités PDU (sur les lignes d'événement).

Toute variable apparaissant dans une expression booléenne ou dans le membre droit d'une affectation sera valuée. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

DÉFINITION SYNTAXIQUE:

```
703  Expression ::= SimpleExpression {RelOpSimpleExpression}
704  SimpleExpression ::= Term {AddOp Term}
705  Term ::= Factor {MultiplyOp Factor}
706  Factor ::=[UnaryOp]Primary
707  Primary ::=Value| DataObjectReference| OpCall| SelectExprIdentifier| "("Expression ")"
739  Value ::=LiteralValue| ASN1_Value[ASN1_Encoding]
740  LiteralValue ::=Number| BooleanValue| Bstring| Hstring| Ostring| Cstring| R_Value
741  Number ::=(NonZeroNum {Num})| 0
742  NonZeroNum ::= 1| 2| 3| 4| 5| 6| 7| 8| 9
743  Num::=0|  NonZeroNum
744  BooleanValue ::= TRUE| FALSE
745  Bstring ::= "'"{Bin| Wildcard} "'"B
746  Bin ::= 0| 1
747  Hstring ::= "'"{Hex| Wildcard} "'"H
748  Hex ::= Num| A| B| C| D| E| F
749  Ostring ::= "'"{Oct| Wildcard} "'"O
750  Oct ::=Hex Hex
751  Cstring ::= "'" {Char| Wildcard| "\"} """
752  Char ::= /*REFERENCE-A character defined by the relevant CharacterString type. */
753  Wildcard ::=AnyOne| AnyOrNone
754  AnyOne ::= "?"
755  AnyOrNone ::= "*"
708  DataObjectReference {ComponentReference}
709  DataObjectIdentifier ::=TS_ParIdentifier| TS_ConstIdentifier| TS_VarIdentifier| TC_VarIdentifier|
       FormalParIdentifier| ASP_Identifier| PDU_Identifier| CM_Identifier| VarIdentifier
710  ComponentReference ::= RecordRef| ArrayRef| BitRef
711  RecordRef ::= Dot(ComponentIdentifier| ComponentPosition)
712  ComponentIdentifier ::= ASP_ParIdentifier| PDU_FieldIdentifier| CM_ParIdentifier| ElemIdentifier| ASN1_Identifier
714  ComponentPosition ::= "("Number ")"
715  ArrayRef ::=Dot "[" ComponentNumber "]"
716  ComponentNumber ::= Expression
```

717 BitRef ::= Dot (BitIdentifier| "["BitNumber "]")
718 BitIdentifier ::= Identifier
719 BitNumber ::= Expression
720 OpCall ::= OpIdentifier (ActualParList | "("")")
721 OpIdentifier ::=TS_ OpIdentifier| TS_ProcIdentifier| PredefinedOpIdentifier
722 PredefinedOpIdentifier::=BIT_TO_INT| HEX_TO_INT| INT_TO_HEX| IS_CHOSEN| IS_ PRESENT| LENGTH_OF| NUMBER_OF_ELEMENTS
723 AddOp ::= "+"| "/"| OR
724 MultiplyOp ::= "*"| "/"| MOD| AND
725 UnaryOp ::= "+"| "-"| NOT
726 RelOp ::= "="|"<"| ">"|"<>"| ">="|"<="

15.10.2 Références à des objets de données définis en notation ASN.1

15.10.2.1 Introduction

Afin de permettre les références à des composantes d'objets définis au moyen de la notation ASN.1, la notation TTCN fournit trois mécanismes d'accès: les références à des enregistrements, les références à des ensembles et les références au niveau du bit.

DÉFINITION SYNTAXIQUE:

708 DataObjectReference ::= DataObjectIdentifier {ComponentReference}
710 ComponentReference ::= RecordRef | ArrayRef | BitRef
711 RecordRef ::= Dot(ComponentIdentifier| ComponentPosition)
715 ArrayRef ::=Dot "[" ComponentNumber "]"
717 BitRef ::= Dot (BitIdentifier| "["BitNumber "]")

15.10.2.2 Références à des enregistrements

Une référence à un enregistrement peut servir à référer à une composante d'un des types suivants: SEQUENCE, SET et CHOICE. Elle est formée à l'aide d'une notation ponctuée, c'est-à-dire en ajoutant un point et le nom (l'identificateur) ou le numéro (position) de la composante voulue à l'identificateur de l'objet de données. L'identificateur de la composante sera utilisé, s'il est défini, de préférence à sa position. Une référence à une composante sans nom est formée en donnant entre parenthèses le numéro correspondant à la position de cette composante dans la définition de type. Par définition, la numérotation implicite des composantes commence à zéro, la troisième composante a donc la position numéro 2.

La Recommandation X.680 définit les types SET comme ayant des composantes non ordonnées. Cela n'est pertinent que si les valeurs de ce type sont codées et envoyées par le fournisseur de service sous-jacent. La notation TTCN traite donc les objets de données de type SET de la même manière que ceux de type SEQUENCE, c'est-à-dire que la référence à des composantes dont le numéro est *i* correspond toujours à une référence au *i*ème champ déclaré dans le type.

Lorsqu'une primitive ASP, une unité PDU ou un message CM ont été reçus, la référence à une composante ayant l'indice *i* renverra toujours la même valeur. Il n'y a aucune opération en notation TTCN qui entraîne la modification de l'ordre des éléments d'un ensemble SET.

DÉFINITION SYNTAXIQUE:

711 RecordRef ::= Dot (ComponentIdentifier | ComponentPosition)
712 ComponentIdentifier ::= ASP_ParIdentifier | PDU_FieldIdentifier | CM_ParIdentifier | ElemIdentifier | ASN1_Identifier
714 ComponentPosition ::= "(" Number ")"

EXEMPLE 76 – Références à des enregistrements de composantes:

```
Example_type ::= SEQUENCE {
                 field_1                    INTEGER
                 field_2     BOOLEAN,
                             OCTET STRING }
```

Si var1 est du type ASN.1 Example_type, il est possible d'écrire:

var1.field_1 pour désigner le premier champ (INTEGER)

var1.(3) pour désigner le troisième champ (sans nom)

EXEMPLE 77 – Références à des champs d'unité PDU:

```
Example_type ::= SEQUENCE {
                                :
                                user-data    OCTET STRING,
                                :   }
```

Sur une ligne de déclaration contenant le type XY_PDUtype, il est possible d'écrire:

L? XY_PDU (buffer := XY_PDUtype.user_data)

afin de charger le tampon de variable avec le contenu du champ user_data de l'unité PDU entrante.

Lorsqu'un paramètre, un champ ou un élément d'une unité PDU ou de type ASN.1 est chaîné à une primitive ASP, à une autre unité PDU ou à un message CM, une référence à un enregistrement peut être utilisée afin d'identifier une composante de cette unité ou de ce type ASN.1. La référence à l'enregistrement doit identifier la séquence pertinente complète de noms de paramètre, de champ ou d'éléments séparés par des points, en commençant par un identificateur d'objet de données qui désigne l'identificateur de primitive ASP, l'identificateur de message CM ou (si les primitives ASP ne sont pas utilisées dans la suite de tests) l'identificateur d'unité PDU. Après cet identificateur d'objet de données initial, la séquence ne doit contenir aucun identificateur d'unité PDU ou identificateur de type ASN.1. Elle contiendra seulement les identificateurs des paramètres, champs et éléments pertinents. Ce mécanisme ne doit pas être utilisé s'il y a une quelconque ambiguïté au sujet de l'identité d'une contrainte d'unité PDU ou d'une contrainte de type ASN.1 dans la séquence. L'exemple suivant illustre l'utilisation de références à des enregistrements lorsque le chaînage de contraintes est utilisé (voir 12.4).

EXEMPLE 78 – Références à des enregistrements avec chaînage:

```
Définition de type de primitive ASP en ASN.1
ASP1_type ::= SEQUENCE  {
                          par1 OCTET STRING,
                          par2 OCTET STRING,
                          Pdu1 PDU1_type
                        }
PDU1_type ::= SEQUENCE   {
                          Field1 OCTET STRING
                          Field2 OCTET STRING
                          F       F_type
                          }
                    Définition de type de structure ASN.1
F_type ::= SEQUENCE  {
                       Data1 IA5String
                       Data2 IA5String
                     }
```

Lorsque les contraintes de type ASP1_type, PDU1_type et F_type sont utilisées, les valeurs de data1 et de data2 peuvent être référencées comme suit:

ASP1_Type.pdu1.F.data1

ASP1_Type.pdu1.F.data2

De manière semblable, l'ensemble du champ F de l'unité PDU peut être référencé comme suit:

ASP1_Type.pdu1.F

ou l'ensemble de l'unité PDU peut être référencé comme suit:

ASP1_Type.pdu1

Il est à noter que les déclarations utilisées dans cet exemple pourraient s'appliquer au chaînage statique et au chaînage dynamique, car les différences entre les deux types de chaînage ne sont visibles que dans les contraintes. La référence à un enregistrement est donc indépendante de la variété de chaînage utilisée.

15.10.2.3 Références à des ensembles

Une référence à un ensemble peut être utilisée pour faire référence à une composante d'un objet de données du type SEQUENCE OF ou SET OF. Une référence à un ensemble doit être construite au moyen d'une notation ponctuée, en ajoutant un point et l'indice de la composante voulue à l'identificateur d'objet de données. L'indice, qui donne la position de la composante dans l'objet de données (lorsque l'objet est vu comme un ensemble linéaire), est inscrit entre crochets. Par définition en notation ASN.1, l'indexage des composantes commence à zéro. L'indice peut être une expression. Dans ce cas, la résolution de cette expression aura pour résultat un entier non négatif.

La Recommandation X.680 définit des types SET OF comme des ensembles non ordonnés de composantes. Cette condition n'est valable que si les valeurs de ces types sont codées et envoyées par l'intermédiaire du fournisseur de service sous-jacent. La notation TTCN traite donc les objets de données de type SET OF de la même manière que les objets de type SEQUENCE OF, c'est-à-dire en se référant aux composantes par des numéros, le numéro *i* renvoyant toujours au *i*ème champ déclaré pour ce type.

Une fois reçu(e) la primitive ASP, l'unité PDU ou le message CM, l'appel de la composante d'index *i* donnera toujours la même valeur. Aucune opération TTCN ne modifie l'ordre des éléments d'un type SET OF.

DÉFINITION SYNTAXIQUE:

715 ArrayRef ::= Dot "[" ComponentNumber "]"
716 ComponentNumber ::= Expression

EXEMPLE 79 – Référence à un ensemble de composantes:

```
Array_type ::= SEQUENCE OF {BOOLEAN}
```

Si var2 est de type ASN.1 Array_type, les expressions suivantes pourraient être écrites afin de référer à la première expression booléenne de la séquence:

var2.[0]

var1.[1-1]

15.10.2.4 Références à un bit

La référence à un bit peut être utilisée pour faire référence à une composante d'un objet de données du type BITSTRING. A cette fin, les objets de données de type BITSTRING sont supposés être définis comme SEQUENCE OF {BOOLEAN}. Une référence à un bit peut donc être élaborée en utilisant la notation d'index comme pour les références à des ensembles. Le bit à l'extrême gauche a un indice de zéro. La résolution d'une expression utilisée comme indice dans une référence à un bit aura pour résultat un entier non négatif. Par ailleurs, si certains bits d'une chaîne de bits BITSTRING sont associés à un identificateur (bits nommés), cet identificateur peut être utilisé pour référer au bit.

DÉFINITION SYNTAXIQUE:

717 BitRef ::= Dot (BitIdentifier | "[" BitNumber "]")
718 BitIdentifier ::= Identifier
719 BitNumber ::= Expression

EXEMPLE 80 – Références à un bit:

```
B_type ::= BITSTRING {ack(0),poll(3)}
```

Cette expression désigne le type B_type BITSTRING où le bit zéro est appelé "ack" (accusé de réception) et le bit trois "poll" (interrogation).

Si b_str est du type ASN.1 B_type, on peut écrire:

b_str.ack := TRUE

b_str.[2] := FALSE

Il est à noter que b_str.poll := TRUE et b_str.[3] := TRUE sont des expressions qui toutes deux affectent la valeur TRUE au bit "poll".

15.10.3 Références à des objets de données définis à l'aide de tables

La syntaxe définie en 15.10.2.2 servira aussi à former des références à des enregistrements désignant des composantes des primitives ASP, des unités PDU, des messages CM et des éléments des types structurés définis sous forme tabulaire. Le chaînage des primitives ASP, des unités PDU, des messages CM et des types structurés sous forme tabulaire aura le même effet sur les références aux enregistrements que pour ceux définis en notation ASN.1.

Lorsqu'un paramètre, un champ ou un élément est défini de manière à inclure un élément comme une vraie sous-structure d'un type structuré à définition tabulaire, la référence à l'élément de cette sous-structure sera formée en faisant suivre la référence à l'enregistrement du paramètre, du champ ou de l'élément par un point puis par l'identificateur de l'élément de cette structure.

Si une structure est utilisée dans un développement de macro, il sera fait référence à ses éléments comme si cette structure avait été développée dans la structure qui s'y réfère.

Si un paramètre, un champ ou un élément est défini comme étant du métatype **PDU**, aucune référence ne sera faite aux champs de cette sous-structure.

15.10.4 Affectations

15.10.4.1 Introduction

Des événements de test peuvent être associés à une liste d'affectations et à un qualificateur. Les affectations sont séparées par des virgules et la liste est mise entre parenthèses.

DÉFINITION SYNTAXIQUE:

701 AssignmentList ::= "(" Assignment {Comma Assignment } ")"
702 Assignment ::=DataObjectReference ":=" Expression

Dans une affectation, le membre droit prendra une valeur du même type que le membre gauche.

Une affectation a pour effet de donner à la variable d'un test élémentaire ou d'une suite de tests (ou au paramètre de primitive ASP ou au champ d'unité PDU) la valeur de l'expression. Cette expression ne contiendra aucune variable non valuée.

Toutes les affectations sont effectuées dans l'ordre de leur apparition, c'est-à-dire de gauche à droite.

EXEMPLE 81 – Utilisation des affectations sur des lignes d'événement:

```
(X:=1)
  (Y:=2)
    L!A (Y:=0, X:=Y, A.field1:=y)
      L?B (Y:=B.field2, X:=X+1)
```

Lorsque la transmission de l'unité PDU A est réussie, les variables de test élémentaire X et Y ainsi que le champ field1 de l'unité A ont un contenu nul. A la réception de l'unité PDU B, la variable Y se voit affecter le contenu du champ field2 de l'unité B et la variable de test élémentaire X est incrémentée.

15.10.4.2 Règles d'affectation pour les types chaînes

Les règles suivantes s'appliquent aux affectations faisant intervenir des types de chaînes avec restriction de longueur:

a) si la chaîne d'origine est plus longue que la longueur définie pour le type de chaîne de destination, elle sera tronquée à droite à la longueur maximale du type de la chaîne de destination;

b) si la chaîne d'origine est plus courte que la longueur définie pour le type de la chaîne de destination, elle sera alignée à gauche et complétée par des caractères de remplissage jusqu'à correspondre à la taille maximale du type de la chaîne de destination.

Les caractères de remplissage sont les suivants:

– " " (blanc) pour toutes les chaînes de caractères CharacterString;

– "0" (zéro) pour les chaînes BITSTRING, HEXSTRING et OCTETSTRING.

Lorsqu'une variable de type chaîne non liée (c'est-à-dire d'une longueur arbitraire) est utilisée à gauche d'une affectation, elle sera limitée à la valeur du côté droit sans remplissage. Le remplissage ne doit être utilisé que lorsque la variable est du type chaîne de longueur fixe.

15.10.5 Qualificateurs

Un événement pourra être qualifié par une expression booléenne entre crochets placée à sa suite. L'interprétation de cette qualification est que la déclaration n'est exécutée que si l'événement concorde et si le qualificateur prend la valeur "vrai" (TRUE).

Si un qualificateur et une affectation sont associés au même événement, le qualificateur apparaîtra en premier, chacun de ses termes étant évalué avec les valeurs en vigueur avant l'exécution de l'affectation.

DÉFINITION SYNTAXIQUE:

681 Qualifier :: = "[" Expression "]"

15.10.6 Lignes d'événement comportant des affectations et des qualificateurs

Un événement peut être associé à une affectation, à un qualificateur ou aux deux. Dans le premier cas, l'affectation n'est exécutée que si l'événement concorde. Dans le deuxième cas, l'événement ne peut concorder que si le qualificateur prend la valeur "vrai" (TRUE). Dans le dernier cas, l'événement ne concorde que si le qualificateur prend la valeur "vrai", et l'affectation n'est exécutée que si l'événement concorde.

Si un événement "réception" est spécifié avec un qualificateur, et qu'un événement qui s'est réalisé concorde potentiellement avec l'événement spécifié, le qualificateur sera évalué dans le contexte de l'événement intervenu. Si le qualificateur fait référence à des paramètres de primitive ASP ou à des champs d'unité PDU, les valeurs de ces paramètres ou de ces champs seront extraites de l'événement intervenu.

Les règles d'utilisation des affectations dans les événements sont les suivantes:

a) dans un événement SEND, toutes les affectations sont effectuées *après* le calcul du qualificateur et *avant* la transmission de la primitive ASP ou de l'unité PDU;

b) dans les événements SEND, les affectations sont permises pour les champs de la primitive ou de l'unité PDU en cours transmise;

c) dans un événement RECEIVE, les affectations sont effectuées *après* que la réalisation de l'événement s'est produite et ne peuvent concerner les champs de la primitive ASP ou de l'unité PDU qui vient d'être reçue.

Une affectation dans la partie comportementale portant sur une contrainte de paramètre de primitive ASP, de champ d'unité PDU ou d'élément de structure provoque le remplacement des valeurs de contrainte au niveau de la ligne d'événement SEND.

EXEMPLE 82 – Utilisation d'un événement SEND qualifié:

```
PARTIAL_TREE
!A[X=3]
!B
```

Dans le traitant de ces événements SEND concurrents, le testeur n'envoie A que si la variable X vaut 3. Sinon il envoie B.

Il est possible d'utiliser l'événement OTHERWISE avec des qualificateurs et des affectations. Si on utilise un qualificateur, son expression booléenne devient une condition supplémentaire à l'acceptation d'un événement entrant. Si on utilise une déclaration d'affectation, cette affectation n'intervient que si toutes les conditions de concordance de l'événement OTHERWISE sont respectées.

EXEMPLE 83 – Utilisation de l'événement OTHERWISE, des qualificateurs et des affectations:

```
PARTIAL_TREE(PCO:XSAP; PCO2; YSAP)
PCO1?A                                                          PASS
PCO2?B                                                          INCONC
PCO1?C                                                          PASS
PCO2?OTHERWISE [X <>2] (Motif:= "X différent de 2")             FAIL
PCO2?OTHERWISE (Motif:= "X égal à 2 mais l'événement n'est pas B")  FAIL
```

Si on suppose qu'aucun événement n'est reçu au point PCO1, la réception de l'événement B au point PCO2 lorsque X = 2 donne un verdict non concluant. La réception d'un autre événement au point PCO2 lorsque X <> 2 donne un verdict d'échec et affecte la valeur "X différent de 2" à la variable motif de type chaîne de caractères. Si un événement reçu au point PCO2 ne répond à aucun de ces scénarios, c'est le dernier événement OTHERWISE qui concorde.

Les événements auxquels participent des messages CM présents à des points PC peuvent aussi être associés à une affectation, à un qualificateur ou à ces deux types d'éléments, de la même manière que pour les unités PDU, ci-dessus.

EXEMPLE 84 – Messages CM associés à un qualificateur:

```
A_CP!A_CM                [X=2]
```

15.11 Pseudo-événements

Les affectations, les qualificateurs et les temporisateurs peuvent se retrouver seuls sur une ligne de déclaration d'un arbre comportemental, sans événement associé. De telles expressions autonomes sont appelées pseudo-événements.

Un pseudo-événement a la signification suivante:

a) qualificateur seul: celui-ci est évalué s'il prend la valeur "vrai" (TRUE) et l'exécution se poursuit avec la déclaration comportementale suivante; s'il prend la valeur "faux" (FALSE), on passe à l'option suivante. S'il n'existe aucune autre option, il s'agit d'une erreur de test élémentaire;

b) affectations et temporisateurs: les affectations sont exécutées de gauche à droite et les opérations de temporisation sont exécutées de gauche à droite;

c) affectations et temporisateurs précédés d'un qualificateur: le qualificateur est d'abord évalué; s'il prend la valeur "vrai" (TRUE), les affectations et les opérations de temporisation sont exécutées.

15.12 Gestion des temporisateurs

15.12.1 Introduction

La gestion des temporisateurs est modélisée par un ensemble d'opérations, qui peuvent être combinées à des événements ou apparaître sous forme de pseudo-événements autonomes.

Les opérations de temporisation peuvent s'appliquer:

– à un temporisateur individuel, spécifié en faisant suivre l'opération de temporisation par le nom du temporisateur;

– à tous les temporisateurs, ce qui est spécifié en omettant le nom du temporisateur.

On suppose que les temporisateurs utilisés dans une suite de tests sont inactifs ou déclenchés. Tous les temporisateurs déclenchés sont automatiquement annulés à la fin de chaque test élémentaire. Il existe trois opérations de temporisation prédéfinies: START (déclenchement), CANCEL (annulation) et READTIMER (lecture de temporisation). Il est possible de spécifier plus d'une opération de temporisateur sur une même ligne d'événement si cela est nécessaire. Dans ce cas, les opérations sont séparées par des virgules.

Lorsqu'une opération de temporisation apparaît sur la même ligne de déclaration qu'un événement ou un qualificateur, elle est exécutée si et seulement si l'événement concorde et/ou si le qualificateur prend la valeur "vrai" (TRUE).

DÉFINITION SYNTAXIQUE:

727 TimerOps ::=TimerOp {CommaTimerOp}
728 TimerOps ::=StartTimer | CancelTimer | ReadTimer

15.12.2 Opération de déclenchement START

L'opération START sert à déclencher un temporisateur.

DÉFINITION SYNTAXIQUE:

729 StartTimer ::= START (TimerIdentifier)["("TimerValue ")"]
301 TimerIdentifier ::= Identifier
731 TimerValue ::= Expression

Le paramètre facultatif TimerValue (valeur de temporisation) doit être donné si aucune durée par défaut n'a été spécifiée lors de la déclaration du temporisateur ou si on souhaite affecter au temporisateur une heure d'expiration (c'est-à-dire une durée) différente de la valeur par défaut si celle-ci a été spécifiée.

Les valeurs de temporisation seront du type entier. Le rédacteur du test élémentaire s'assurera que le paramètre facultatif valeur de temporisation prend une valeur entière strictement positive. Si un temporisateur est déclenché avec une valeur négative ou nulle, il en résultera une erreur de test élémentaire.

Toute variable figurant dans l'expression spécifiant la valeur facultative de la temporisation sera valuée. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

Si une nouvelle durée est imposée à un temporisateur, cette nouvelle valeur ne s'appliquera qu'à l'instance courante du temporisateur: toutes les opérations START ultérieures de déclenchement de ce temporisateur ne spécifiant pas de durée adopteront la durée par défaut déclarée dans la section des déclarations de temporisation.

EXEMPLE 85 – Utilisation du temporisateur START:

Les Ti sont des identificateurs de temporisateurs et les Vi leurs valeurs de temporisation:
START T0
START T0 (V0)
START T1, START T2 (V2)

L'opération START peut s'appliquer à un temporisateur déjà déclenché: dans ce cas, le temporisateur est annulé, réinitialisé et redéclenché. Toutes les entrées qui s'y rapportent dans la liste de fins de temporisation en seront retirées.

15.12.3 Opération CANCEL (annulation)

L'opération CANCEL sert à arrêter un temporisateur déclenché.

DÉFINITION SYNTAXIQUE:

730 CancelTimer ::= CANCEL [TimerIdentifier| FormalParIdentifier]
301 TimerIdentifier::=Identifier
731 TimerValue ::=Expression

Un temporisateur annulé devient inactif. Si un événement TIMEOUT figure dans la liste de fins de temporisation, il en sera retiré. Si le nom du temporisateur est omis dans l'opération CANCEL, tous les temporisateurs en fonctionnement sont annulés et la liste de fins de temporisation est vidée de son contenu.

L'annulation d'un temporisateur inactif est une opération valide, mais n'a bien sûr aucun effet.

EXEMPLE 86 – Exemples d'utilisation du temporisateur CANCEL:

Les Ti sont des identificateurs de temporisateurs.
CANCEL
CANCEL T0
CANCEL T1, CANCEL T2
CANCEL T1, START T3

15.12.4 Opération READTIMER (lecture de temporisateur)

L'opération READTIMER sert à obtenir le temps écoulé depuis le déclenchement de la temporisation et à l'enregistrer dans la variable spécifiée de suite de tests ou de test élémentaire. Cette variable est du type entier. La valeur temporelle qui lui est affectée est exprimée dans l'unité de temps spécifiée pour ce temporisateur lors de sa déclaration. Par convention, l'application de l'opération READTIMER à un temporisateur inactif renvoie la valeur zéro.

DÉFINITION SYNTAXIQUE:

732 ReadTimer ::= READTIMER (TimerIdentifier| FormalParIdentifier) "("DataObjectReference")"
301 TimerIdentifier ::= Identifier

EXEMPLE 87 – Utilisation du temporisateur READTIMER:

```
:
START TimerName(TimeVal)
      ?EVENT_A
            +Tree_A
      ?EVENT_B
            +Tree_B
      ?EVENT_C
            READTIMER TimerNAME(CurrTime)
      ?TIMEROUT TimerName
```

Si EVENT_C est reçu avant l'expiration de la temporisation TimerName, le temps écoulé depuis son déclenchement est enregistré dans la variable de test élémentaire ou de suite de tests CurrTime. Le comportement de Tree_C pourra utiliser cette valeur.

EXEMPLE 88 – Utilisation combinée de l'opération READTIMER et d'autres opérations de temporisation:

READTIMER T1 (PASSED_TIME), CANCEL T1
READTIMER T1 (V1), START NEW_TIMER (V1)

15.13 Construction ATTACH (rattachement)

15.13.1 Introduction

Des arbres peuvent être rattachés à d'autres à l'aide de la construction ATTACH définie par la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

696 Attach ::= "+" TreeReference [ActualParList]
698 TreeReference ::= TestStepIdentifier | TreeIdentifier
699 ActualParList ::= "(" ActualPar{Comma ActualPar} ")"
700 ActualPar ::= Value | PCO_Identifier | CP_Identifier | TimerIdentifier

Les variables de suite de tests et de test élémentaire sont communes à l'arbre de rattachement (l'arbre principal) et à l'arbre rattaché, c'est-à-dire que toute modification apportée à une telle variable dans un arbre rattaché sera également effective dans l'arbre principal. Chaque construction de rattachement d'arbre occupera seule une ligne de déclaration complète.

15.13.2 Visibilité des arbres après rattachement

Les descriptions comportementales peuvent contenir plus d'un arbre. Toutefois, seul l'arbre *principal* de la description comportementale est accessible depuis l'extérieur de celle-ci. Tous les arbres imbriqués sont considérés comme des modules de test locaux de cette description comportementale et donc inaccessibles depuis l'extérieur.

A noter que seuls les tests élémentaires sont directement exécutables, les modules de test n'étant exécutés que s'ils sont rattachés à un test élémentaire, soit directement soit par l'intermédiaire d'un ou plusieurs autres modules de test. Les tests élémentaires ne peuvent pas être rattachés.

Une référence à un arbre peut être un identificateur de module de test ou un identificateur d'arbre:

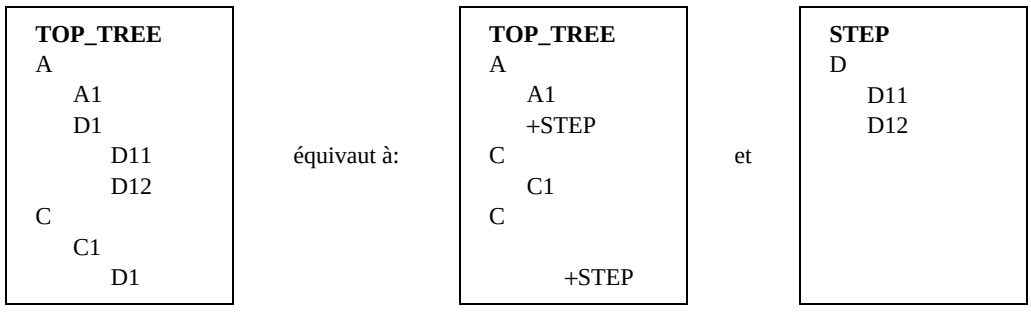
a) l'identificateur de module de test indiquant le rattachement d'un module de test situé dans la bibliothèque des modules de test; ce module de test est désigné par son identificateur unique;

b) l'identificateur d'arbre étant le nom de l'un des arbres figurant dans la description comportementale courante; il s'agit donc du rattachement d'un arbre local.

15.13.3 Règles de base du rattachement des arbres

Il est possible de détacher certaines parties d'un arbre comportemental pour former des arbres comportementaux distincts, c'est-à-dire des modules de test. Les points d'où un module de test a été séparé de l'arbre d'origine sont indiqués par le symbole de rattachement (+) suivi du nom attribué à ce module de test.

EXEMPLE 89 – Fractionnement d'un grand arbre en deux plus petits:



Cette opération peut s'effectuer non seulement sur l'arbre comportemental principal (l'arbre racine) du test élémentaire, mais également sur les modules de test qui en ont été détachés. L'arbre rattaché est un arbre local ou un élément de la bibliothèque des modules de test.

Le rattachement d'arbre peut être défini d'une manière plus générale qu'une simple insertion de modules de test:

– un arbre rattaché ne contient pas nécessairement des trajets complets allant jusqu'à des déclarations qui constitueront les feuilles de l'arbre auquel il est rattaché (son *arbre d'appel*): certaines "branches", communes à tous les trajets de l'arbre rattaché, pourront être spécifiées dans l'arbre d'appel, c'est-à-dire comme des comportements ultérieurs au point de rattachement;

– certaines déclarations (même du niveau supérieur) du module de test rattaché peuvent prendre elles-mêmes la forme +SOME_SUBTREE (+sous-arbre donné), permettant ainsi le rattachement d'autres modules de test;

– il est possible de paramétrer des modules de test rattachés.

15.13.4 Signification du rattachement d'arbre

15.13.4.1 La liste suivante définit la sémantique opératoire du rattachement d'arbre:

a) la ligne de rattachement (par exemple, +STEP) de l'arbre de comportement (par exemple, TOP_TREE) constitue formellement l'une des déclarations optionnelles (par exemple, A_i) de l'ensemble ordonné:

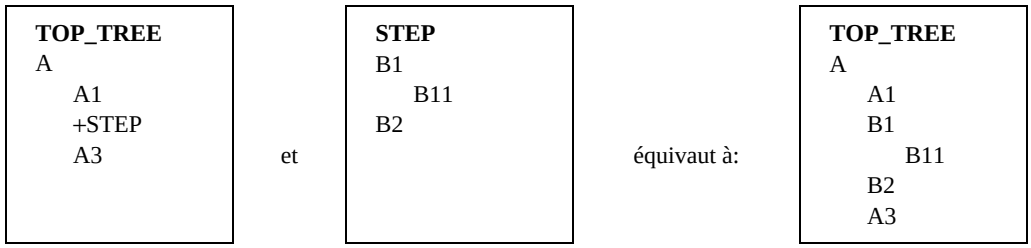
$$(A_1, ..., A_i, ..., A_n)$$

Le rattachement de STEP en cette position signifie qu'on développe TOP_TREE en insérant les déclarations optionnelles du niveau d'indentation supérieur (par exemple, B_1, …, B_m) du module de test STEP dans cette séquence, pour former ainsi une nouvelle séquence d'options:

$$(A_1, ..., A_{(i-1)}, B_1, ..., B_m, A_{(i+1)}, ..., A_n)$$

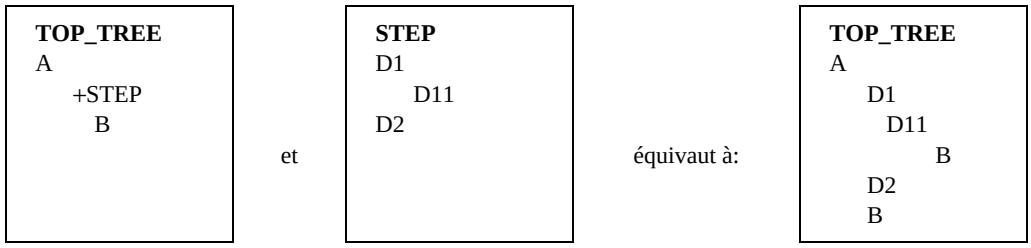
Tous les comportements de niveau d'indentation inférieur dépendant des comportements B seront rattachés en même temps qu'eux;

EXEMPLE 90 – Développement d'un module de test:



b) tous les comportements consécutifs à la ligne +STEP dans cet arbre, deviennent des comportements consécutifs à toutes les feuilles du STEP rattaché après développement;

EXEMPLE 91 – Comportement consécutif à une construction de rattachement:



c) lorsqu'on utilise une liste de paramètres effectifs dans une construction ATTACH, chacun des paramètres formels se verra substituer le paramètre effectif correspondant par simple substitution textuelle. Cette substitution s'effectue conformément aux règles de visibilité suivantes:

1) les paramètres effectifs de la construction ATTACH à un arbre local ne sont substitués aux paramètres formels correspondants qu'à l'intérieur de cet arbre local;

2) les paramètres effectifs de la construction ATTACH à l'arbre racine d'un module de test sont substitués à toutes les occurrences des paramètres formels correspondants dans l'arbre racine et dans tous les arbres locaux à l'intérieur du module de test;

3) lorsqu'un arbre paramétré est rattaché:

   i) le nombre des paramètres effectifs est le même que celui des paramètres formels;

   ii) chaque paramètre actuel prend une valeur du type du paramètre formel correspondant;

   iii) les paramètres formels et effectifs des modules de test doivent être utilisés de manière qu'il n'y ait que de la notation TTCN qui soit créée par substitution textuelle.

EXEMPLE 92 – Substitution de paramètres:

```
TOP_TREE(L:NSAP;U:TSAP)
L!CONreq(M:=1)
    +STEP(U,M,2)
```

et

```
STEP (PCO:TSAP;X,Y:INTEGER)
PCO?CONind(X:Y)
```

équivaut à:

```
TOP_TREE(L:NSAP;U:TSAP)
L!CONreq(M:=1)
    U?CONind(M:=2)
```

EXEMPLE 93 – Règles de visibilité applicables à la substitution de paramètres:

Comportement dynamique de module de test

Nom de module de test	**:** TEST_STEP_1(X,Y:INTEGER)
Groupe	**:** TTCN_EXAMPLES/PARAMS/STEPS/
Objectif	**:** illustration des règles de visibilité en substitution de paramètres
Comportement par défaut	**:**
Commentaires	**:**

n°	Etiquette	Description comportementale	Réf. des contraintes	Verdict	Commentaires
1		?A	A1		
2		+TEST_STEP_2(X)			
3		+LOCAL(5)			
		LOCAL(F:INTEGER)			
4		!B	B1		
5		(TC_VAR:=F+Y)		PASS	

Commentaires détaillés:

lorsque TEST_STEP1 est rattaché par un arbre d'appel, toutes les occurrences des paramètres formels X et Y figurant dans la totalité du module de test (y compris dans l'arbre LOCAL) sont remplacées par les paramètres effectifs d'appel. A noter que les paramètres formels X et Y ne sont pas remplacés automatiquement par les paramètres effectifs dans TEST_STEP2. Cependant, la valeur du paramètre effectif se substitue à celle du paramètre formel X dans la construction ATTACH "+TEST_STEP2(X)", ce qui aboutit à ce que la valeur du paramètre effectif X (dans TEST_STEP1) se substitue au paramètre formel du TEST_STEP2 quel que soit le nom qui lui est donné dans la déclaration de ce dernier module. A noter enfin que le paramètre effectif 5 (constante) se substitue au paramètre formel "F" lors du rattachement de l'arbre LOCAL. Cette substitution n'intervient que dans l'arbre local.

15.13.5 Transfert de contraintes paramétrées

Il est possible de transférer des contraintes sous forme de paramètres aux modules de test. Si la contrainte possède une liste de paramètres formels, elle sera transférée avec une liste de paramètres effectifs. Les paramètres effectifs de la contrainte seront déjà valués au point de rattachement.

EXEMPLE 94 – Transfert d'une contrainte paramétrée:

Supposons que la contrainte C1 possède un seul paramètre formel de type INTEGER, TOP_TREE, rattache STEP et transfère C1 sous forme de paramètre. A noter que la référence à la contrainte dans STEP n'est pas paramétrée:

```
TOP_TREE
    :
    +STEP(C1(3))
    :
```

```
STEP(PARA:A_PDU)
    :
    !A_PDU              PAR
    :
```

15.13.6 Rattachement récursif d'arbre

Le rattachement d'arbre fonctionnant récursivement (STEP pouvant contenir une ligne +SOME_OTHER_TREE), la sémantique de développement d'un arbre peut ne jamais aboutir à un arbre exempt de lignes de rattachement.

EXEMPLE 95 – Rattachement récursif autorisé d'un arbre:

```
TOP_TREE
A
+STEP
    B
```

et

```
STEP
C
        D
    +TOP_TREE
```

un développement équivaut à:

```
TOP_TREE
A
C
   D
       B
+TOP_TREE
  B
```

Un arbre ne peut se rattacher lui-même, directement ou indirectement, à son niveau supérieur d'indentation.

NOTE – Il n'est pas nécessaire de développer ni un module de test qui ne sera pas exécuté, ni des déclarations optionnelles situées au-delà du niveau d'indentation courant tant qu'une option de ce niveau n'aura pas été sélectionnée.

EXEMPLE 96 – Rattachement récursif non autorisé d'un arbre:

```
TOP_TREE
A
    +STEP
      B
```

et

```
STEP
C
  +TOP_TREE
D
```

un développement équivaut à:

```
TOP_TREE
A
 C
+TOP_TREE
 B
 C
  B
```

15.13.7 Rattachement d'arbre et comportements par défaut

Les comportements par défaut dans un arbre seront développés avant tout rattachement de celui-ci (voir 15.18.2).

NOTE – L'utilisation simultanée dans une description comportementale de rattachements d'arbre et de comportements par défaut nécessite une attention particulière.

15.14 Etiquettes et construction GOTO (saut)

Toute ligne de déclaration de l'arbre comportemental peut recevoir une étiquette dans la colonne étiquettes.

NOTE 1 – Chaque fois qu'une entrée étiquetée est exécutée dans l'arbre comportemental, il faut enregistrer son étiquette qui sera consignée dans le journal de conformité de façon à pouvoir l'associer à l'enregistrement de l'exécution de cette entrée.

Il est possible dans un arbre comportemental de spécifier une construction de saut GOTO vers une étiquette à condition que cette étiquette soit associée à la première proposition d'un ensemble de propositions concurrentes dont l'une forme un nœud antécédent du point depuis lequel l'instruction de saut GOTO sera exécutée. Seuls sont autorisés les sauts à l'intérieur d'un même arbre, c'est-à-dire à l'intérieur d'un arbre racine de test élémentaire, d'un arbre de module de test, d'un arbre de comportement par défaut ou d'un arbre local. Par conséquent, toute étiquette utilisée dans une construction GOTO se trouvera dans l'arbre même où est située cette construction. Les sauts vers le premier niveau d'indentation des arbres locaux, des modules de test et des comportements par défaut ne sont pas autorisés.

Une instruction de saut GOTO ne doit pas spécifier une étiquette avant une construction ACTIVATE qui est un nœud antécédent de l'instruction GOTO.

La construction GOTO est spécifiée par une flèche (→) ou le mot clé GOTO, suivis du nom de l'étiquette, et occupera une ligne de déclaration indépendante dans l'arbre comportemental.

DÉFINITION SYNTAXIQUE:

695 GoTo ::= ("→"| GOTO) Label

Une étiquette est unique dans un arbre. Si une construction GOTO est exécutée, le test élémentaire procède au traitement de l'ensemble d'options auquel l'étiquette se réfère.

Les constructions GOTO sont toujours inconditionnelles et donc toujours exécutées.

NOTE 2 – Il est possible de placer une expression booléenne comme antécédent immédiat d'une construction de saut GOTO pour obtenir l'effet d'un saut conditionnel.

EXEMPLE 97 – Utilisation de la construction de saut GOTO:

Comportement dynamique de test élémentaire					
Nom du test élémentaire : GOTO_EX1					
Groupe : TTCN_EXAMPLES/GOTO_EXAMPLE1/					
Objectif : illustrer l'utilisation d'étiquettes avec la construction GOTO					
Comportement par défaut :					
Commentaires :					
n°	**Etiquette**	**Description comportementale**	**Réf. des contraintes**	**Verdict**	**Commentaires**
1	LA	?A	A1		
2	LB	?B	B1		
3	LB2	+B-tree			
4	LC	?C	C1		
5	LD	[D=1]			
6		→LA			
7	LE	[E=1]			
8	LF	!F	F1	FAIL	
Commentaires détaillés: cet exemple illustre un saut vers l'étiquette LA. Il est également possible de sauter de cette position vers LB ou LD, mais pas vers LB2 ou LF (car les ensembles correspondants d'options ne contiennent pas de nœud antécédent du point de saut), ni vers LC ou LE (car ces étiquettes ne sont pas les premières d'un ensemble d'options).					

15.15 Construction REPEAT (répétition)

Le présent sous-paragraphe décrit un mécanisme utilisable dans les descriptions comportementales pour exécuter un module de test de manière itérative. Cette construction obéit à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

697 Repeat **::= REPEAT** TreeReference [ActualParList]UNTIL Qualifier

La référence d'arbre TreeReference pointera vers un arbre local ou à un module de test défini dans la bibliothèque des modules de test. Se reporter au 15.13 pour les règles de rattachement. La signification de la construction REPEAT est la suivante: tout d'abord, l'arbre désigné par la référence d'arbre est exécuté. Puis le qualificateur est évalué. S'il prend la valeur "vrai" (TRUE), la construction de répétition REPEAT est achevée. Sinon, l'exécution de l'arbre reprend, suivie de l'évaluation du qualificateur. Ce traitement se répète jusqu'à ce que le qualificateur prenne la valeur "vrai" (TRUE).

Une construction REPEAT peut toujours être exécutée; elle constituera normalement la dernière option d'une série de déclarations TTCN du même niveau d'indentation, ainsi que le prévoit 15.9.5.3 a).

NOTE – Il est recommandé d'utiliser si possible la construction de répétition REPEAT de préférence à la construction GOTO.

EXEMPLE 98 – Utilisation de la construction de répétition REPEAT (voir aussi l'Appendice I)**:**

Comportement dynamique de test élémentaire					
Nom du test élémentaire : RPT_EX1					
Groupe : TTCN_EXAMPLES/REPEAT_EXAMPLE1/					
Objectif : illustrer l'utilisation de la construction REPEAT					
Comportement par défaut :					
Commentaires :					
n°	**Etiquette**	**Description comportementale**	**Réf. des contraintes**	**Verdict**	**Commentaires**
1		(FLAG:=FALSE)			
2		!A	A1		
3		REPEAT STEP1 (FLAG)UNTIL(FLAG)			
4		!D	D1	PASS	
		STEP1 (F:BOOLEAN)			
		?B(F:=TRUE)			
5		?C(F:=FALSE)	B1		
6			C1		
Commentaires détaillés: cet exemple décrit un test pouvant recevoir un nombre quelconque d'événements C au point PCO du testeur inférieur jusqu'à la réception du message B attendu.					

15.16 Référence aux contraintes

15.16.1 Objectif de la colonne référence aux contraintes

Cette colonne permet de se référer à une contrainte particulière imposée à une primitive ASP, à une unité PDU ou à un message CM. Ces contraintes sont définies dans la partie "contraintes" (voir les paragraphes 12, 13 et 14). La référence à une contrainte sera présente avec des événements SEND (envoi), IMPLICIT SEND (envoi implicite) et RECEIVE (réception). Elle est facultative lorsqu'une primitive ASP ou un message CM n'a pas de paramètres, ou si une unité PDU ne compte pas de champs; elle n'accompagnera aucune autre sorte de déclaration TTCN.

Une entrée dans la colonne référence aux contraintes peut être une référence à une contrainte effective, le symbole AnyValue ("?"), ou un paramètre formel dont le paramètre effectif est une référence à une contrainte ou le symbole AnyValue. Si AnyValue est utilisé à la place d'une référence à une contrainte, cela indique une contrainte dont la valeur n'a aucune portée, soit l'équivalent d'une contrainte ayant AnyOrNone ("*") dans chaque paramètre, champ ou élément.

Une référence à une contrainte effective obéit à la syntaxe suivante:

DÉFINITION SYNTAXIQUE:

670 ConsRef ::= ConstraintIdentifier [ActualCrefParList]
671 ActualCrefParList ::= "("ActualCrefPar { CommaActualCrefPar} ")"
672 ActualCrefPar ::= Value

EXEMPLE 99 – Référence à une contrainte sans liste de para.mètres:

	N_SAP?CR_PDU	CRI		

15.16.2 Transfert de paramètres dans des références à des contraintes

Une référence à une contrainte peut avoir une liste de paramètres facultatifs permettant de manipuler des valeurs de contraintes particulières depuis l'arbre comportemental.

La liste des paramètres effectifs respectera les conditions suivantes:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur du type du paramètre formel correspondant ou un symbole dont la valeur peut correspondre à ce type formel.

Si une contrainte est transférée en tant que paramètre effectif et qu'elle soit déclarée avec une liste de paramètres formels, elle comportera également une liste de paramètres effectifs (éventuellement imbriquée). Toutes les variables apparaissant dans la liste de paramètres seront valuées au moment de l'utilisation de cette contrainte. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

EXEMPLE 100 – Référence à une contrainte avec une liste de paramètres:

	N_SAP?N_DATAreq	D1(P1,CR1(P2))		

Où D1 est une contrainte portant sur la primitive de demande N_DATA (demande de données réseau) et comportant deux paramètres (les paramètres effectifs P1 et CR1) et où CR1 est une contrainte comportant un paramètre (le paramètre effectif P2).

15.16.3 Contraintes, qualificateurs et affectations

Si un événement est qualifié et comporte aussi une référence à une contrainte, on considère qu'il concorde si et seulement si le qualificateur *et* la contrainte sont respectés.

Si un événement est suivi d'une affectation et comporte une référence à une contrainte ou un qualificateur, on l'interprétera comme suit: l'affectation sera exécutée si et seulement si l'événement se produit conformément à la définition donnée ci-dessus.

15.17 Verdicts

15.17.1 Introduction

Dans les tables de comportement dynamique, les entrées de la colonne verdict sont:

– soit un résultat préliminaire, indiqué entre parenthèses;

– soit un verdict final explicite.

Aucune entrée de l'un de ces types n'apparaîtra sur une ligne vide ou dans les déclarations TTCN suivantes:

a) construction de rattachement ATTACH;

b) construction de répétition REPEAT;

c) construction de saut GOTO;

d) envoi implicite IMPLICIT SEND.

DÉFINITION SYNTAXIQUE:

```
674  Verdict ::= Pass| Fail| Inconclusive| Result
675  Pass ::= PASS| P| "("PASS")"| "("P")"
676  Fail ::= FAIL| F| "("FAIL")"| "("F")"
677  Inconclusive ::= INCONC| I| "("INCONC")"| "("I")"
678  Result ::= R
```

NOTE – Pendant l'exécution d'un test élémentaire, chaque fois que dans l'arbre comportemental un verdict est associé à une entrée exécutée, il sera consigné dans le journal de conformité de façon à être associé à l'enregistrement de cette entrée dans l'arbre comportemental.

15.17.2 Résultats préliminaires

La variable prédéfinie R du type prédéfini R_TYPE sert dans chaque test élémentaire à enregistrer tout résultat intermédiaire. Ces valeurs sont des identificateurs prédéfinis pour lesquels, par conséquent, la différence minuscules-majuscules est significative.

R peut s'utiliser partout où on peut utiliser d'autres variables de test élémentaire, sauf dans le membre gauche d'une déclaration d'affectation. Il s'agit donc d'une variable accessible en lecture seulement, à l'exception des modifications apportées à sa valeur dans la colonne verdict (selon les spécifications ci-dessous).

Un résultat préliminaire spécifié dans la colonne verdict aura l'une des trois valeurs suivantes:

a) **(P)** ou **(PASS)**, signifiant que l'objectif du test est atteint sous un certain aspect;

b) **(I)** ou **(INCONC)**, signifiant que quelque chose a rendu le test élémentaire non concluant par rapport à un aspect de l'objectif du test;

c) **(F)** ou **(FAIL)**, signifiant qu'une erreur de protocole s'est produite ou qu'un certain aspect de l'objectif du test a échoué.

NOTE 1 – PASS ou P, FAIL ou F, INCONC ou I sont des mots clés qui ne s'utilisent que dans les colonnes verdict. Les identificateurs prédéfinis *pass, fail, inconc* et *none* sont des valeurs représentant le contenu possible de la variable prédéfinie R. Ces identificateurs prédéfinis ne s'utilisent que pour tester la variable R dans des lignes comportementales.

Chaque fois qu'un résultat préliminaire est consigné à la suite de l'exécution de l'entrée correspondante dans l'arbre comportemental, la valeur de la variable prédéfinie R du test élémentaire est modifiée conformément au Tableau 7.

NOTE 2 – L'ordre de priorité (croissant) est donc: N, P, I, F. Même si R a la valeur *fail*, il peut être utile d'enregistrer un résultat préliminaire P ou I pour consigner dans le journal de conformité que P ou I peut convenir pour certains aspects de l'objectif du test, bien que ces valeurs ne modifient pas celle de R.

Tableau 7/X.292 – Calcul de la valeur de la variable R

Valeur courante de R	Entrée dans la colonne verdict		
	(PASS)	(INCONC)	(FAIL)
None	pass	inconc	fail
Pass	pass	inconc	fail
Inconc	inconc	inconc	fail
Fail	fail	fail	fail

15.17.3 Verdict final

S'il faut le spécifier, un verdict final explicite dans la colonne verdict, prendra une des valeurs suivantes:

a) **P** ou **PASS**, signifiant qu'il faut consigner un verdict de succès;

b) **I** ou **INCONC**, signifiant qu'il faut consigner un verdict non concluant;

c) **F** ou **FAIL**, signifiant qu'il faut consigner un verdict d'échec;

d) la variable prédéfinie R, signifiant que la valeur de R est le verdict final, sauf si elle est égale à *none*, auquel cas on enregistre une erreur de test élémentaire à la place du verdict final.

Tableau 8/X.292 – Détermination du verdict final R

Valeur courante de R	Entrée dans la colonne verdict			
	(PASS)	(INCONC)	(FAIL)	R
None	pass	inconc	fail	*error*
Pass	pass	inconc	fail	pass
Inconc	*error*	inconc	fail	inconc
Fail	*error*	*error*	fail	fail

Chaque fois qu'un verdict final explicite est spécifié pendant l'exécution d'un test élémentaire, celui-ci prend fin. Pour être conforme aux dispositions de la Recommandation X.291, un verdict final explicite n'est spécifié que si le test élémentaire a repris un état stable approprié (par exemple, l'état de test au repos).

NOTE 1 – Il est nécessaire que la spécification d'un verdict final explicite puisse mettre fin à un test élémentaire, lorsqu'on atteint par exemple un état stable dans un module de test rattaché et qu'un comportement ultérieur est spécifié dans l'arbre d'appel.

Si on atteint la feuille de l'arbre comportemental sans qu'un verdict final explicite soit spécifié, le verdict final est alors déterminé comme dans le cas d) ci-dessus (c'est-à-dire comme si on avait introduit la variable R dans la colonne verdict).

S'il faut enregistrer un verdict final explicite autre que R, il faut alors le comparer à la valeur de R pour déterminer s'ils sont cohérents. Si la valeur de R est *fail*, un verdict final **PASS** ou **INCONC** est alors considéré comme incohérent; si la valeur de R est *inconc*, un verdict final **PASS** est considéré comme incohérent. Chacune de ces incohérences constitue une erreur de test élémentaire.

NOTE 2 – Dans ce cas, on consigne le message "erreur de test élémentaire" dans le journal de conformité.

15.17.4 Verdicts et déclaration OTHERWISE (sinon)

Une déclaration OTHERWISE (sinon) n'aboutira pas à un verdict de succès; elle devrait plutôt se terminer par un verdict d'échec car elle correspond généralement à un événement de test non valide.

15.17.5 Affectation de verdict en notation TTCN concomitante

En notation TTCN concomitante, le verdict final est affecté par la composante MTC, soit explicitement dans la colonne de verdict, soit implicitement comme conséquence de la fin de la composante MTC. Les résultats de tests préliminaires sont tenus à jour dans la variable de résultat global, à laquelle la composante MTC peut accéder comme variable de test élémentaire R. La variable de résultat global est mise à jour chaque fois qu'un résultat préliminaire ou un verdict est enregistré dans la colonne de verdict par une ligne comportementale de composante MTC concordante. Si la composante MTC se termine sans affecter de verdict explicite, le verdict sera établi comme si R avait été placée dans la colonne de verdict [voir 15.17.3 d)].

En outre, chaque composante PTC doit enregistrer au moins un résultat préliminaire. Ce résultat préliminaire est tenu à jour dans sa variable de résultat local, à laquelle la composante PTC peut accéder comme variable de test élémentaire R. Lorsqu'un résultat préliminaire est affecté par une composante PTC, par toute entrée dans la colonne de verdict d’une ligne comportementale de composante PTC concordante (que l'entrée soit entre parenthèses ou non), sa variable de résultat local et la variable de résultat global sont mises à jour au moyen de l'algorithme spécifié en 15.17.2. Dans une composante PTC, une entrée dans la colonne de verdict qui n'est pas encadrée par des parenthèses n'est pas un verdict final, mais elle doit entraîner la fin de la composante PTC si cette ligne comportementale concorde.

La fin de la composante MTC avant celle de toutes les composantes PTC doit entraîner une erreur de test élémentaire.

Lorsque la composante MTC utilise la variable R dans une expression booléenne ou une affectation, elle accède à la variable de résultat global. Lorsqu'une composante PTC utilise la variable R dans une expression booléenne ou une affectation, elle accède à la variable de résultat local. La composante MTC peut aussi accéder à une variable de résultat local qui lui est propre en utilisant la variable de test élémentaire prédéfinie MTC_R plutôt que R. MTC_R est du type prédéfini R_TYPE. La variable MTC_R est mise à jour chaque fois qu'un résultat préliminaire ou un verdict est enregistré dans la colonne de verdict par une ligne comportementale de composante MTC concordante, mais elle n'est pas touchée par les résultats préliminaires de composantes PTC. La variable MTC_R ne doit pas être utilisée dans la colonne de verdict.

La valeur d'une variable de résultat local d'une composante PTC peut seulement être communiquée à une autre composante de test par l'intermédiaire de messages CM. La valeur des variables locale ou globale de la composante MTC peut seulement être communiquée à une composante PTC par l'intermédiaire de messages CM.

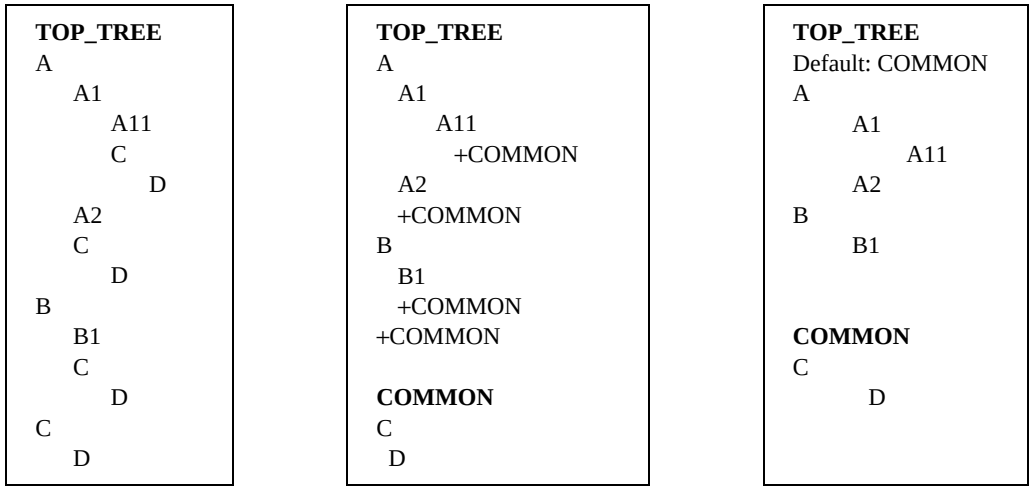
15.18 Signification des comportements par défaut

15.18.1 Introduction

On utilise très souvent un comportement par défaut pour privilégier un ensemble de trajets intéressants dans un test en déclarant comme comportements par défaut les déclarations optionnelles communes moins intéressantes (ainsi que leurs comportements consécutifs).

Le même effet pourrait être obtenu, mais avec une concision moindre, en rattachant un module de test (par exemple +DEFAULT) comme dernière option générale supplémentaire. Contrairement au rattachement d'arbre, le comportement par défaut se développe en de nombreux points de l'arbre auquel il est associé. Cette propriété impose une utilisation attentive des comportements par défaut.

EXEMPLE 101 – Identification d'un arbre de comportement par défaut:



1: ensemble complet des déclarations optionnelles

2: rattachement explicite d'arbre

3: un comportement par défaut donne le même résultat que 2

Aucun comportement par défaut n'est spécifié pour un autre, c'est-à-dire qu'un comportement par défaut ne peut avoir lui-même de comportement par défaut. Le rattachement d'arbres ne sera pas utilisé dans des arbres de comportement par défaut, c'est-à-dire qu'à des arbres de comportement par défaut ne peuvent pas se rattacher des modules de test. Les tests élémentaires et les modules de test ne peuvent être désignés comme des comportements par défaut.

L'exécution d'un test élémentaire ne nécessite pas le développement des comportements par défaut dans tous les arbres qui s'y réfèrent. Cela apparaît dans la description opératoire de la signification des comportements par défaut: quand une concordance est recherchée dans une séquence d'options (ce qui peut nécessiter des tentatives répétées) et qu'aucune n'est trouvée, on essaie également le premier niveau d'indentation des déclarations optionnelles du comportement par défaut. Si aucune d'elles ne concorde, on répète l'opération avec de nouvelles valeurs de temporisateurs et de files d'attente à tous les points PCO concernés. Si une concordance est trouvée dans le comportement par défaut, on le poursuit en ce point.

Pour s'assurer qu'aucun comportement ultérieur n'interviendra après l'exécution d'un comportement par défaut, l'exécution d'une feuille d'un arbre de comportement par défaut, autre qu'une déclaration RETURN, entraînera la fin du test élémentaire. Afin de provoquer cette fin, dans un arbre de comportement par défaut, on affecte un verdict final "R" dans la colonne de verdict de chaque feuille de l'arbre de comportement par défaut pour laquelle aucun verdict ou résultat préliminaire n'est inscrit dans la colonne de verdict et, pour chaque feuille dont la colonne de verdict comporte un résultat préliminaire, ce résultat est implicitement transformé en verdict final.

15.18.2 Références aux comportements par défaut

Les comportements des tests élémentaires et des modules de test font référence à une liste de comportements par défaut dans la bibliothèque des comportements par défaut, par l'intermédiaire de l'entrée de comportement par défaut dans l'en-tête de la table.

DÉFINITION SYNTAXIQUE:

631 DefaultReference ::= DefaultIdentifier [ActualParList]

Chaque référence à cette liste localise un comportement par défaut au moyen de son identificateur unique. L'identificateur DefaultIdentifier doit être une référence à un comportement par défaut défini dans la bibliothèque des comportements par défaut.

Des paramètres peuvent être associés aux comportements par défaut. La liste des paramètres effectifs doit satisfaire aux conditions suivantes:

a) le nombre de paramètres effectifs doit être le même que le nombre de paramètres formels;

b) chaque paramètre effectif doit prendre pour valeur un élément du type formel correspondant;

c) toutes les variables apparaissant dans la liste de paramètres doivent être liées lorsque la contrainte est appelée.

EXEMPLE 102 – Référence au comportement par défaut:

102.1

Comportement dynamique de test élémentaire					
Nom du test élémentaire : DEF_EX1					
Groupe : TTCN_EXAMPLES/DEFAULT_EXAMPLE1/					
Objectif : illustrer l'utilisation des comportements par défaut					
Comportement par défaut : DEF1(L)					
Commentaires : l'arbre de l'exemple ** peut être divisé dans ce test élémentaire avec le comportement par défaut DEF1.					
n°	**Etiquette**	**Description comportementale**	**Réf. des contraintes**	**Verdict**	**Commentaires**
1		L!CONNECTrequest	CR1		Demande…
2		L?CONNECTconfirm	CC1		…Confirmation
3		L!DATArequest	DTR1		Envoi de données
4		L?DATAindication	DTI1		Réception de données
5		L!DISCONNECTrequest	DSC1	PASS	Acceptation

102.2

Comportement dynamique de comportement par défaut					
Nom du comportement par défaut : DEF_EX1					
Groupe : TTCN_EXAMPLES/DEFAULTS_LIB/DEFAULT_1/					
Objectif : illustration d'un comportement par défaut simple					
Commentaires : l'arbre de l'exemple ** peut être divisé dans ce test élémentaire avec le comportement par défaut DEF1.					
n°	**Etiquette**	**Description comportementale**	**Réf. des contraintes**	**Verdict**	**Commentaires**
1		X?DISCONNECTindication	DSC2	INCONC	Prématurе

NOTE – Sur le plan syntaxique, le comportement par défaut de la deuxième des tables ci-dessus rattache l'indication X?DISCONNECTindication comme proposition pour chacune des déclarations L! et L? de la première table. Le rattachement de l'arbre de comportement comme proposition pour une déclaration L! qui produit toujours un résultat positif est toutefois sans signification.

15.18.3 Déclaration RETURN

La déclaration RETURN est une extension des capacités de la description du comportement par défaut. Une déclaration RETURN ne doit être utilisée que dans un arbre de comportement par défaut. Elle aura la syntaxe suivante.

Lorsque le développement par défaut d'un arbre est effectué, l'exécution d'une déclaration RETURN entraîne la continuation du traitement à la première proposition de l'ensemble de propositions qui a entraîné la tentative d'exécution du comportement par défaut.

15.18.4 Déclaration ACTIVATE

La déclaration ACTIVATE permet l'activation d'un ensemble de comportements par défaut. Au lieu d'être implicitement actifs pendant la durée d'un test élémentaire, les comportements par défaut peuvent être activés de manière sélective par la déclaration ACTIVATE. La tentative d'exécution du comportement par défaut ainsi activé est effectuée dans l'ordre spécifié par la déclaration ACTIVATE, par exemple ACTIVATE (Def_1, Def_2) entraînera l'exécution de Def_1 avant celle de Def_2 lorsque le comportement par défaut est requis.

Le comportement par défaut spécifié dans une déclaration ACTIVATE a priorité sur tout comportement par défaut actif, y compris un comportement par défaut spécifié dans un test élémentaire ou un en-tête de module de test.

Une déclaration ACTIVATE dont la liste de références à des comportements par défaut est vide, par exemple ACTIVATE(), désactive tout comportement par défaut.

15.18.5 Comportements par défaut et rattachement d'arbre

Lorsqu'on utilise un rattachement d'arbre, il est important de bien comprendre comment les comportements par défaut se raccrochent tant à l'arbre d'appel qu'au module de test rattaché. Pour éviter les effets collatéraux masqués, les comportements par défaut applicables au module de test rattaché sont définis comme étant ceux qui sont spécifiés dans la table définissant ce module de test. Ainsi, si le module de test est défini dans la bibliothèque des modules de test, les comportements par défaut qui lui sont applicables sont spécifiés dans l'en-tête de la table comportementale du module de test. Ou alors, si le module de test est défini localement dans la même table comportementale que l'arbre d'appel, les mêmes comportements par défaut s'appliquent à la fois à l'arbre d'appel et au module de test rattaché.

Pour éviter la multiplication des insertions de comportements par défaut dans un ensemble d'options, le comportement par défaut spécifié pour un arbre donné ne s'applique pas au niveau d'indentation supérieur des déclarations optionnelles de cet arbre, sauf s'il s'agit de l'arbre racine d'un test élémentaire.

Pour développer correctement un arbre, il est nécessaire de développer les comportements par défaut à la fois:

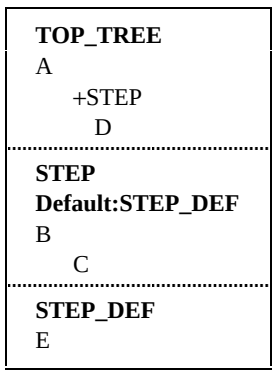
a) avant de développer l'arbre en tant qu'arbre rattaché;

b) avant de développer les modules de test rattachés de cet arbre.

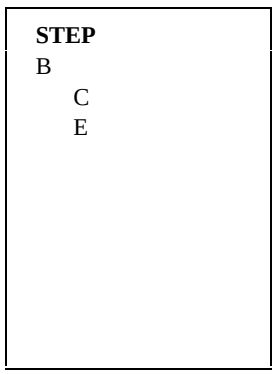
Le développement des comportements par défaut s'effectue donc localement pour un arbre simple et comprend le rattachement de l'arbre de comportement par défaut au bas de chacun des ensembles d'options de cet arbre (à l'exception du niveau supérieur d'indentation pour les arbres autres que l'arbre racine d'un test élémentaire).

Les règles de développement des comportements par défaut s'appliquent de la même manière à un ensemble d'options contenant un événement OTHERWISE.

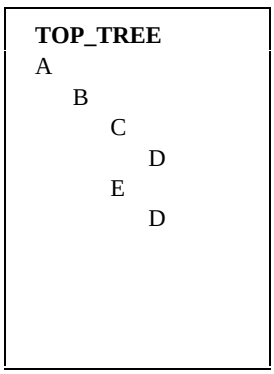
EXEMPLE 103 – Caractère local d'un comportement par défaut par rapport à un module de test:



1: TOP_TREE rattache STEP, qui a le comportement par défaut STEP_DEF

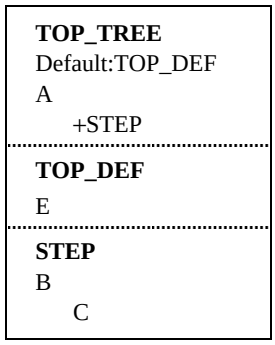


2: développement de STEP_DEF dans STEP

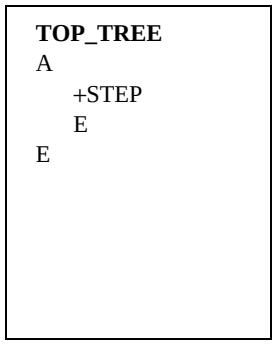


3: développement de STEP dans TOP_TREE

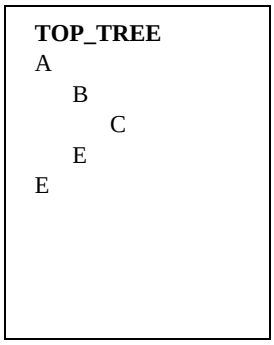
EXEMPLE 104 – Caractère local d'un comportement par défaut par rapport à un arbre d'appel:



1: TOP_TREE rattache STEP. Le comportement par défaut de TOP_TREE est TOP_DEF

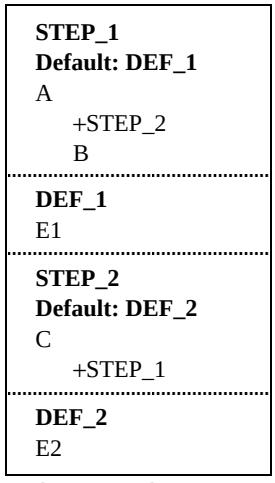


2: développement de TOP_DEF dans TOP_TREE

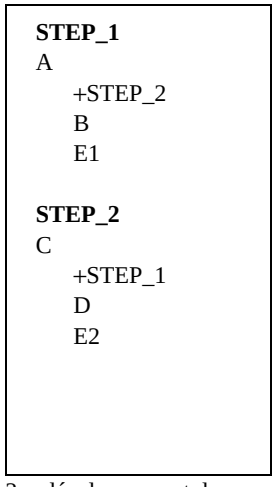


3: développement de STEP dans TOP_TREE

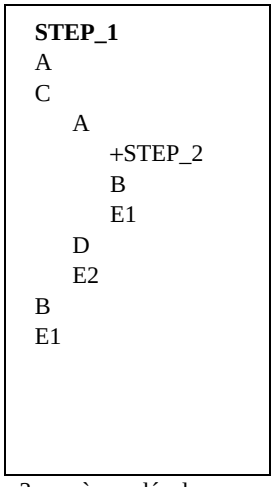
EXEMPLE 105 – Cas d'un rattachement cyclique d'arbre:



1: STEP_1 et STEP_2 sont rattachés l'un à l'autre. Le comportement par défaut de STEP_1 est DEF_1, celui de STEP_2 est DEF_2



2: développement de DEF_1 dans STEP_1 et de DEF_2 dans STEP_2



3: après un développement de STEP_2 sans comportement par défaut et un développement de STEP_1 sans comportement par défaut

NOTE – Il est déconseillé d'utiliser ces rattachements cycliques.

15.18.6 Rattachements d'arbre, comportements par défaut, opérations Activate et Return

Si l'opération ACTIVATE est utilisée dans un test élémentaire, la sémantique des comportements par défaut et des rattachements d'arbre ne peut être décrite que dynamiquement et non de manière statique. En effet, la sémantique opératoire des comportements par défaut de l'Annexe B est spécifiée sous forme de développement dynamique d'arbre, un niveau à la fois.

Selon ce modèle sémantique dynamique, la spécification d'une liste de comportements par défaut dans l'en-tête équivaut à faire précéder l'arbre comportemental par une opération ACTIVATE de cette liste d'arbres de comportement par défaut. Dans un module de test, le fait de placer une liste de comportements par défaut dans l'en-tête équivaut à placer une opération ACTIVATE de cette liste d'arbres de comportement par défaut entre chaque déclaration du premier niveau de déclarations par défaut et son comportement subséquent. Si un module de test pour lequel aucun comportement par défaut n'est spécifié dans l'en-tête est rattaché, les opérations ACTIVATE engagées n'ont pas de paramètres et elles désactivent donc tous les comportements par défaut.

Comme un comportement subséquent à un rattachement d'arbre prend ses valeurs par défaut dans le contexte de l'arbre d'appel plutôt que dans le module de test rattaché, le rattachement d'arbre suppose l'insertion d'une opération ACTIVATE après chaque nœud de feuille non terminal (c'est-à-dire un nœud qui n'affecte pas de verdict) pour rétablir la valeur des comportements par défaut à celle du contexte dans lequel le rattachement a eu lieu. Dans le cas où le nœud de feuille est une opération RETURN, cela suppose que l'opération ACTIVATE doit avoir lieu avant l'opération RETURN afin de garantir qu'elle a un résultat avant le saut au contexte externe.

L'effet de la combinaison de comportements par défaut et du rattachement d'arbre comportemental est illustré par l'Exemple 106 de test élémentaire.

EXEMPLE 106 – Exemple de test élémentaire X-Def1 servant à illustrer la signification des comportements par défaut:

Comportement dynamique de test élémentaire

Nom du test élémentaire : X-Def1
Groupe :
Objectif :
Comportement par défaut : D1, D2

E	Description comportementale	RéfC	V
	X +T1 Y Z +T2		

Comportement dynamique de test élémentaire

Nom du test élémentaire : T1
Groupe :
Objectif :
Comportement par défaut : D3, D4

E	Description comportementale	RéfC	V
	A B C		

Comportement dynamique de test élémentaire

Nom du test élémentaire : T2
Groupe :
Objectif :
Comportement par défaut :

E	Description comportementale	RéfC	V
	D E F		

Cet exemple de test élémentaire équivaut à l'Exemple 107, dans lequel la liste de comportements par défaut dans l'en-tête de test élémentaire a été remplacée par une opération ACTIVATE de la même liste de comportements par défaut comme première déclaration TTCN de l'arbre comportemental.

EXEMPLE 107 – Autre spécification possible de l'exemple de test élémentaire X-Def1 faisant appel à une déclaration ACTIVATE:

Comportement dynamique de test élémentaire

Nom du test élémentaire : X-Def1
Groupe :
Objectif :
Comportement par défaut :

E	Description comportementale	RéfC	V
	ACTIVATE(D1,D2) X +T1 Y Z +T2		

Le traitement d'une opération ACTIVATE établit le contexte par défaut courant. Le passage au niveau suivant d'options rattache la liste d'arbres comportementaux par défaut dans le contexte par défaut courant au niveau suivant d'options.

Ainsi, l'évaluation du test élémentaire de l'Exemple 107 pourrait s'effectuer de la manière illustrée dans la Figure 8. En premier, la déclaration ACTIVATE(D1,D2) est évaluée afin d'établir le contexte par défaut à D1 et D2. Ensuite, en supposant que X concorde, D1 et D2 sont rattachés au même niveau d'options que T1. Lorsque T1 est développé, ACTIVATE(D3,D4) est inséré après le premier niveau de propositions de ce module de test et ACTIVATE(D1,D2) est inséré après les deux nœuds de feuille afin de rétablir le contexte par défaut avant que le comportement suivant, Y, soit atteint. En supposant que A concorde alors, les comportements par défaut D1 et D2 sont rattachés de manière redondante au même niveau d'options que la déclaration ACTIVATE; cela s'explique parce que le contexte courant par défaut est toujours adjoint au niveau suivant d'options, indistinctement, même si le niveau suivant d'options consiste en une construction ou en un pseudo-événement correspondant. Lorsque la nouvelle déclaration ACTIVATE est évaluée, le contexte par défaut devient un contexte applicable au module de test T1. Ensuite, si B concorde, l'évaluation passe à la proposition ACTIVATE explicite qui ramène le contexte par défaut à un contexte applicable à l'arbre racine.

L'Exemple 108 est un autre exemple de test élémentaire. Celui-ci combine des comportements par défaut spécifiés dans des en-têtes avec une déclaration ACTIVATE et un rattachement d'arbre.

EXEMPLE 108 – Exemple de test élémentaire X-Def2 servant à illustrer la signification des comportements par défaut et de la déclaration ACTIVATE:

Comportement dynamique de test élémentaire			
Nom du test élémentaire : X-Def2 **Groupe** : **Objectif** : **Comportement par défaut** : D1			
E	**Description comportementale**	**RéfC**	**V**
	X ACTIVATE(D2) +T S +T S		

Comportement dynamique de module de test			
Nom du test élémentaire : T **Groupe** : **Objectif** : **Comportement par défaut** : D3			
E	**Description comportementale**	**RéfC**	**V**
	Y Z		

Le déroulement de l'évaluation de ce test élémentaire est illustré par la Figure 9. Cette figure montre le déroulement de l'évaluation suivant les deux chemins principaux du test élémentaire et elle indique que le contexte par défaut applicable à la première déclaration S est déterminé par la déclaration ACTIVATE, alors que le contexte par défaut applicable à la deuxième déclaration S est déterminé par les comportements par défaut spécifiés dans l'en-tête du test élémentaire; aucun de ces contextes par défaut pour les déclarations S n'est touché par les rattachements à des arbres précédents.

La Figure 9 commence en montrant l'effet du développement du rattachement de T au premier niveau d'options ainsi que de l'adjonction des comportements par défaut initiaux. Si X concorde, l'évaluation se poursuit par l'intermédiaire de la déclaration ACTIVATE(D2) jusqu'à la deuxième occurrence du rattachement de T, le contexte par défaut passant à D2 et le rattachement de D2 étant adjoint au même niveau d'options que T. T est ensuite développé, les deux déclarations ACTIVATE étant insérées pour établir le contexte par défaut du module de test et le contexte par défaut de l'arbre racine étant ensuite rétabli. Ces changements apportés au contexte par défaut sont ensuite montrés dans les deux étapes suivantes de l'évaluation, en supposant que le premier Y concorde, puis Z. Le résultat est S, une option possible au rattachement de D2 étant évaluée dans le contexte par défaut D2.

Le chemin de remplacement montré dans la Figure 9 commence par la concordance de Y au lieu de celle de X. Cela entraîne la continuation dans le contexte par défaut D3, alors que si Z concorde, le contexte par défaut est rétabli à D1. Ainsi, la progression le long de ce chemin mène à S, une option au rattachement de D1 étant évaluée dans le contexte par défaut D1.

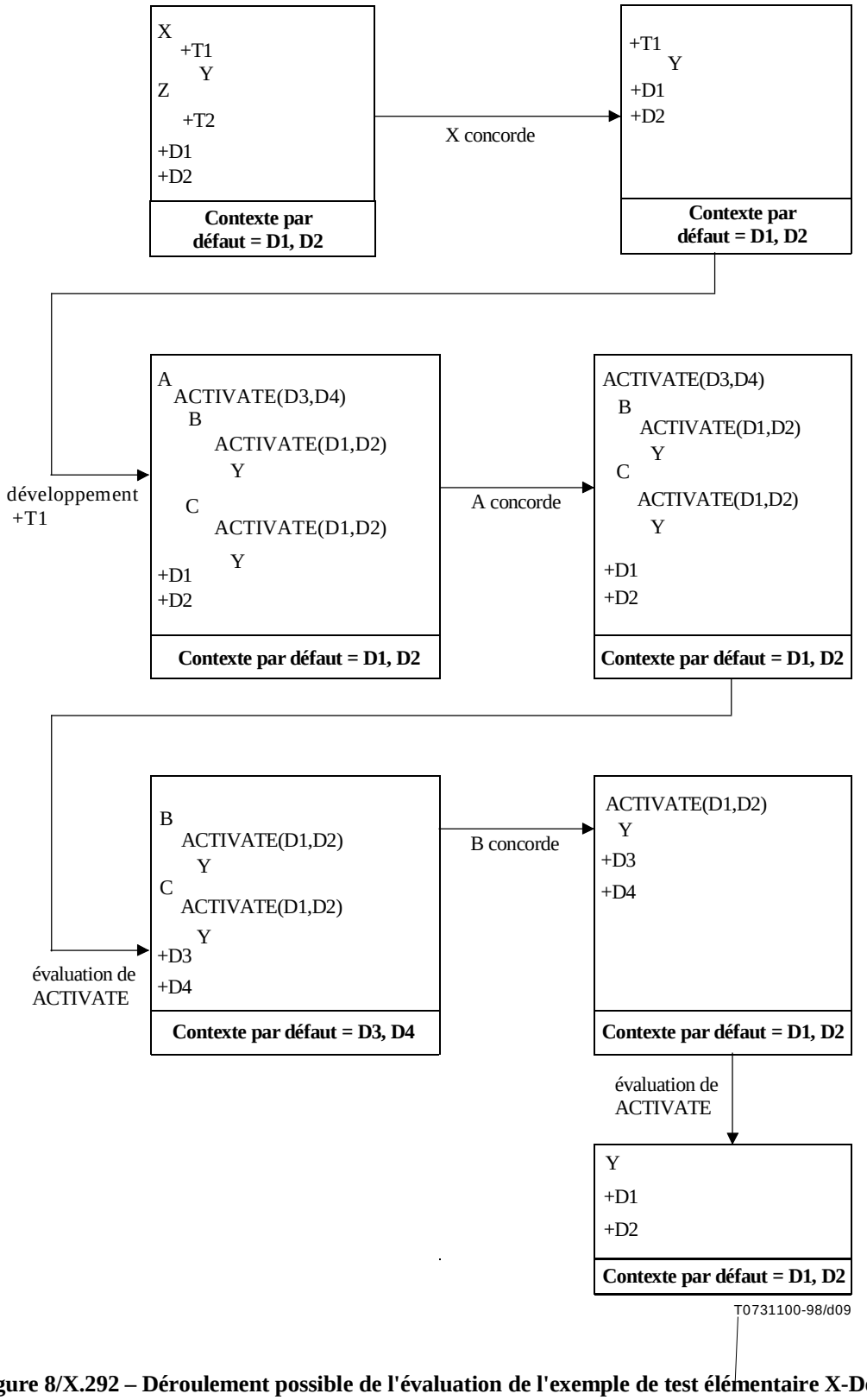


Figure 8/X.292 – Déroulement possible de l'évaluation de l'exemple de test élémentaire X-Def1

Le déroulement de l'évaluation des exemples de tests élémentaires des Figures 8 et 9 n'a pas montré le développement d'arbres de comportement par défaut. Lorsque l'arbre de comportement par défaut est développé, cet arbre ou tout arbre de comportement local associé contient une construction RETURN, ce qui équivaut à la mise en place d'une étiquette au début de l'ensemble courant d'options, chaque construction RETURN étant remplacée par une déclaration ACTIVATE, afin de rétablir le contexte par défaut de l'arbre d'appel, suivie d'une construction GOTO qui mène vers cette nouvelle étiquette.

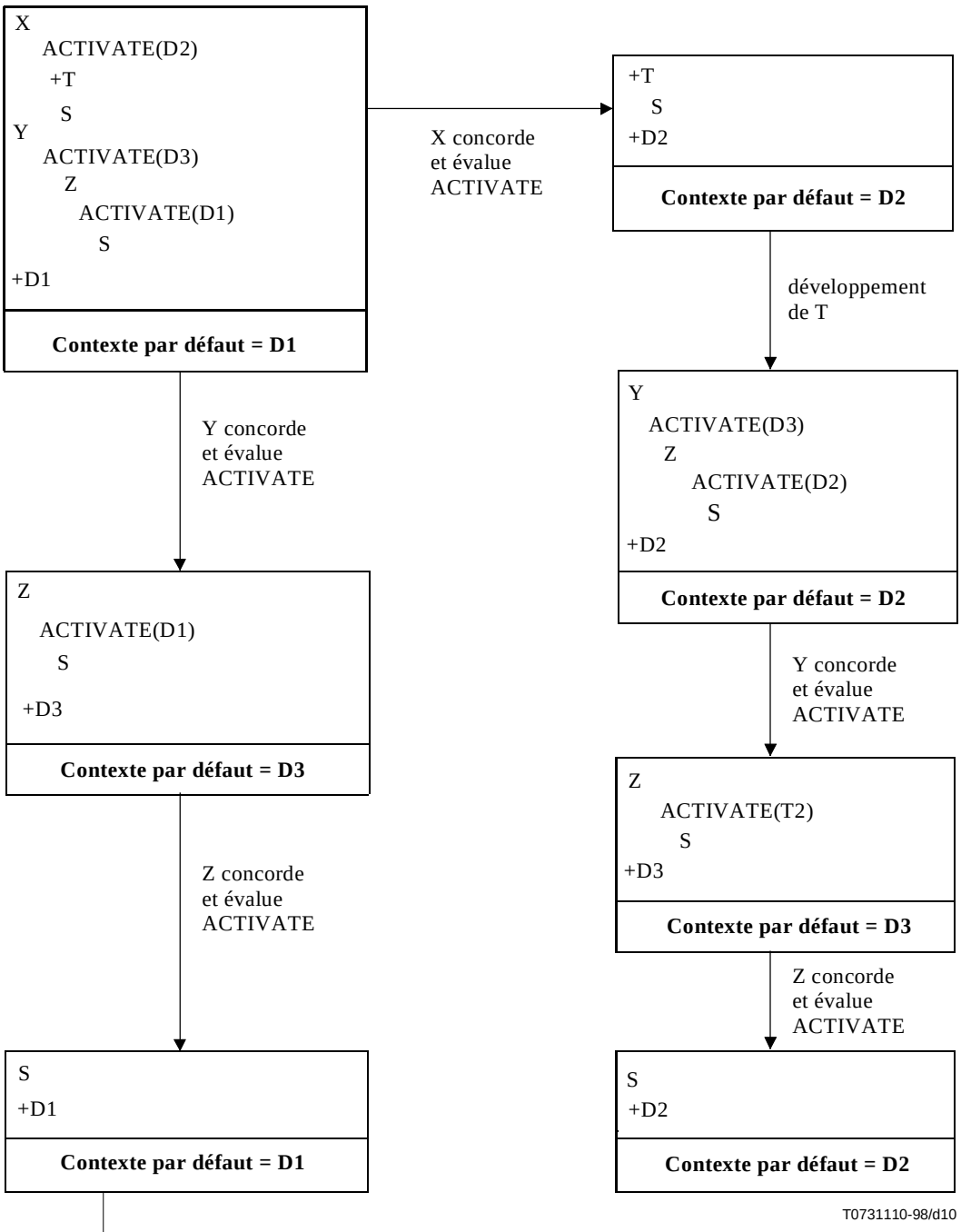


Figure 9/X.292 – Déroulement possible de l'évaluation de l'exemple de test élémentaire X-Def2

Tous les nœuds de feuille, autres que RETURN, d'un arbre de comportement par défaut dans lequel tous les sous-arbres locaux ont été rattachés n'ont pas de comportement ultérieur et ils affecteront donc un verdict ou résulteront en une erreur de test élémentaire.

L'Exemple 109 de test élémentaire illustre ce processus.

EXEMPLE 109 – Exemple de test élémentaire X-Def3 servant à illustrer la signification des comportements par défaut et de la déclaration RETURN:

Comportement dynamique de test élémentaire			
Nom du test élémentaire : X-Def3 **Groupe :** **Objectif :** **Comportement par défaut : D1**			
E	**Description comportementale**	**RéfC**	**V**
	X Y		P

Comportement dynamique par défaut			
Nom du comportement par défaut : D1 **Nom du test élémentaire :** **Objectif :**			
E	**Description comportementale**	**RéfC**	**V**
	C D RETURN E		F

Le déroulement de l'évaluation de cet exemple de test élémentaire est illustré par la Figure 10. L'arbre de comportement par défaut D1 est d'abord rattaché au premier niveau d'options de l'arbre racine. D1 est alors développé. Comme D1 comporte une déclaration RETURN, il s'agit d'un développement assez complexe. L'événement supérieur au niveau des déclarations optionnelles où le rattachement se produit est désigné au moyen de l'étiquette unique E. Comme l'arbre de comportement de rattachement est par défaut, son propre contexte par défaut interne est vide car les comportements par défaut n'ont pas leur propre comportement par défaut. Par conséquent, une déclaration ACTIVATE sans arguments est insérée après le premier niveau d'options de l'arbre rattaché. En outre, la déclaration RETURN est remplacée par une déclaration ACTIVATE pour rétablir le contexte par défaut à D1, suivie au niveau suivant par GOTO L. A ce point, lorsque cet arbre développé est évalué, si C concorde, il passe à la déclaration ACTIVATE() avec le rattachement redondant du contexte par défaut D1. L'évaluation de la déclaration ACTIVATE() a pour effet que le contexte par défaut est vidé. Alors, si D concorde, la déclaration ACTIVATE(D1) est évaluée pour rétablir le contexte par défaut à D1. Cela amène à la déclaration GOTO ainsi qu'à un autre rattachement par défaut du contexte par défaut D1. La déclaration GOTO est évaluée puis le traitement retourne à l'état dans lequel l'étiquette L a été ajoutée. L'évaluation se poursuit de manière itérative dans cette boucle jusqu'à ce que X, suivi de Y, concorde, pour l'affectation d'un verdict de succès, ou que C, suivi de E, concorde, pour l'affectation d'un verdict d'échec.

15.18.7 Comportements par défaut et opération CREATE

Le comportement par défaut n'est pas hérité par les modules de test utilisés dans une opération CREATE, c'est-à-dire par les modules de test qui suivent leur description de comportement en parallèle avec la composante MTC. Par conséquent, le domaine d'application du comportement par défaut en notation TTCN concomitante est toujours local du point de vue de la composante MTC ou PTC.

Dans les cas où un module de test est utilisé dans une opération CREATE, le comportement par défaut spécifié dans l'en-tête du module de test doit être appliqué au premier niveau d'indentation. Cette utilisation des comportements par défaut est conforme à l'application des comportements par défaut aux tests élémentaires.

15.18.8 Comportements par défaut et messages CM

Le comportement par défaut est appliqué à un ensemble de déclarations optionnelles qui ne reçoivent que des messages CM. Cela peut avoir pour conséquence que les unités PDU qui arrivent avant la réception des messages CM exécutés, ou que les unités PDU qui sont déjà dans la file d'attente du point PCO mais qui ne sont pas encore reçues, soient retirées du point PCO. Afin de prévenir le retrait des unités PDU de la file d'attente du point PCO, la construction NO_DEFAULTS doit être précisée en tant qu'événement précédant immédiatement l'ensemble de déclarations optionnelles qui reçoivent seulement le ou les messages CM.

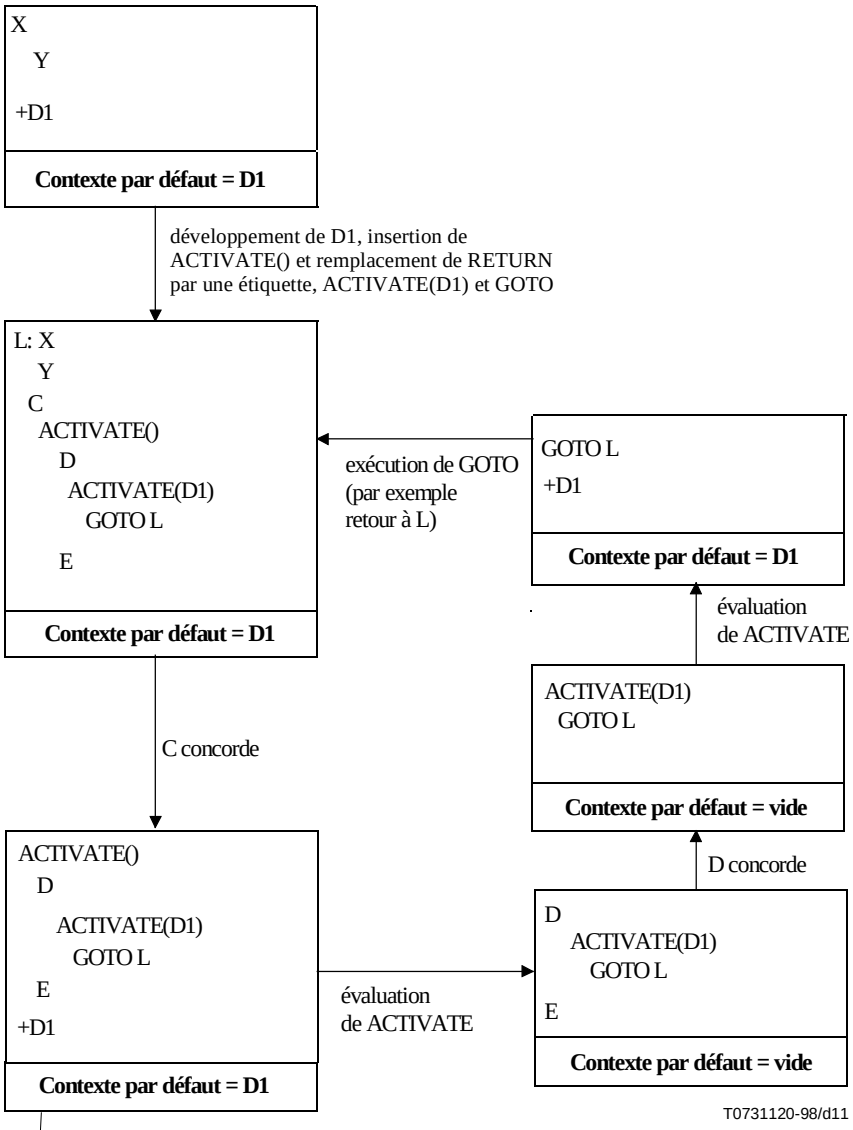


Figure 10/X.292 – Déroulement possible de l'évaluation du test élémentaire X-Def3

16 Suite de page

16.1 Suite de page dans les tables en notation TTCN

Lorsqu'une table TTCN est trop longue pour tenir sur une seule page, on utilise le mécanisme suivant:

a) on imprime les mots "suite page suivante" *après* la ligne de la table à laquelle la coupure intervient;

b) on imprime les mots "suite de la page précédente" *avant* la suite de la table sur la page suivante.

Il est possible de couper les tables en un endroit quelconque, c'est-à-dire dans l'en-tête, dans le corps ou dans le cadre de bas de page. Dans tous les cas, les titres des sections (par exemple, les en-têtes des colonnes) sont répétés sur la nouvelle page. Il n'est pas obligatoire de reproduire l'en-tête complet.

EXEMPLE 110 – Suite d'une table de paramètres d'une suite de tests:

Déclarations des paramètres de la suite de tests			
Nom du paramètre	**Type**	**Réf. PICS/PIXIT**	**Commentaires**
PAR1	INTEGER	Question aa du PICS	
PAR2	BOOLEAN	Question bb du PICS	
PAR3	IA5String	Question cc du PIXIT	

suite page suivante page n

suite de la page précédente page n + 1

Déclarations des paramètres de la suite de tests			
Nom du paramètre	**Type**	**Réf. PICS/PIXIT**	**Commentaires**
PAR4	BOOLEAN	Question dd du PICS	
PAR5	HEXSTRING	Question ee du PICS	

16.2 Suite de page des tables de comportement dynamique

Lorsqu'il est nécessaire de couper une table de comportement dynamique, on utilise l'un des deux mécanismes suivants:

a) la modularisation,

c'est-à-dire la spécification d'une partie de l'arbre comportemental comme un module de test (non local) de bibliothèque, ce qui permet de modulariser l'arbre et de réduire, pour le formulaire courant, le nombre des comportements à celui qui peut figurer sur une seule page;

b) le mécanisme de suite de page,

c'est-à-dire, dans le cas des tables de comportement dynamique, la présentation des informations supplémentaires suivantes qui facilitent l'alignement des niveaux d'indentation:

1) l'impression, avant les mots "suite page suivante", du niveau d'indentation (entre crochets) de la dernière déclaration TTCN avant la coupure de page;

2) sur la nouvelle page, l'impression, après les mots "suite de la page précédente", du niveau d'indentation (entre crochets) de la première déclaration TTCN de la suite de la table.

Lorsque des tests élémentaires sont longs, il peut être nécessaire d'adopter un niveau d'indentation apparent différent de celui déclaré. Dans ce cas, le niveau d'indentation déclaré, indiqué entre crochets, sera aligné avec une valeur d'indentation choisie pour la première ligne de déclaration de la suite de la table. D'autres indications concernant les niveaux d'indentation peuvent également être données pour en faciliter encore l'alignement.

Annexe A

Syntaxe et sémantique statique de la notation TTCN

A.1 Introduction

La présente annexe définit la syntaxe et la sémantique statique de la notation TTCN. Cette notation se présente sous deux formes, une forme graphique (TTCN.GR) et une forme exploitable par machine (TTCN.MP). Pour l'utilisateur, la forme graphique de la notation TTCN, la notation TTCN.GR, a l'avantage d'une interprétation visuelle facile à comprendre. Toutefois, cette forme de notation ne se prête pas facilement au traitement par machine. La notation TTCN.MP répond à cette question et poursuit les objectifs suivants:

a) offrir une syntaxe formelle destinée à la notation TTCN dans la forme Backus-Naur (BNF);

b) servir de syntaxe de transfert;

c) faciliter la génération automatique des suites de tests exécutables (ETS) à partir des suites de tests abstraites (ATS);

d) autre traitement informatique.

NOTE – La génération automatique des suites ETS n'entre pas dans le cadre de la présente Recommandation.

La présente annexe définit également la sémantique statique des notations TTCN.GR et TTCN.MP.

A.2 Conventions appliquées à la description de la syntaxe

A.2.1 Métanotation syntaxique

Le Tableau A.1 définit la métanotation utilisée pour spécifier l'extension de la forme de la grammaire BNF à la notation TTCN (dorénavant appelée "forme BNF").

En métanotation, la concaténation lie plus étroitement que l'opérateur option. Par conséquent, "abc def | ghi jkl" est l'équivalent de "(abc def) | (ghi jkl)".

Tableau A.1/X.292 – Métanotation syntaxique de la notation TTCN.MP

::=	est par définition
abc xyz	abc suivis de xyz
|	ption
[abc]	0 ou 1 instance d'abc
{abc}	0 ou plusieurs instances d'abc
{abc}+	1 ou plusieurs instances d'abc
(...)	groupement textuel
abc	le symbole non terminal abc
abc	un symbole terminal abc
"abc"	un symbole terminal abc

A.2.2 Définitions de la syntaxe de la notation TTCN.MP

A.2.2.1 Les tables complètes définies en notation TTCN.GR sont représentées en notation TTCN.MP par des productions du type:

$Begin_KEYWORD .... $End_KEYWORD

EXEMPLE A.1 – TS_PARdcls ::= **$Begin_TS_PARdcls** {TS_PARdcl}+ **$End_TS_PARdcls**

Normalement, ces productions comprennent au moins un champ obligatoire.

A.2.2.2 Tant les ensembles de lignes d'une table que les lignes individuelles (à savoir les ensembles de champs dans une table) sont représentés par des productions du type:

$KEYWORD .... $End_KEYWORD

Begin n'apparaît pas dans le mot clé ouvrant.

EXEMPLE A.2 – TS_PARdcl ::= **$TS_PARdcl** TS_PARid TS_PARtype PICS_PIXIT [Comment]
$End_TS_PARdcl

A.2.2.3 Les champs individuels d'une ligne sont représentés par:

$KEYWORD ....

Il n'y a pas de mot clé fermant.

EXEMPLE A.3 – TS_ParId ::= **$TS_ParId** TS_ParIdentifier

EXEMPLE A.4 – TS_ParIdentifier ::= Identifier

A.2.2.4 Les ensembles de tables, jusqu'à et y compris la suite de tests, sont représentés par des productions du type:

$KEYWORD .... $End_KEYWORD

EXEMPLE A.5 – ASP_TypeDefs ::= **$ASP_TypeDefs** [TTCN_ASP_TypeDefs] [ASN1_ASP_TypeDefs]
$End_ASP_TypeDefs

A.2.2.5 Pour toutes les autres productions définissant des symboles non terminaux, le membre droit de l'affectation ne comportera de mot clé ni à son début, ni à sa fin.

EXEMPLE A.6 – TimerIdentifier ::= Identifier

A.2.2.6 Lors de l'analyse syntaxique de propositions en notation TTCN.MP, la présence dans un identificateur de tout symbole dont l'insertion est interdite à cet endroit peut signifier la fin de l'identificateur. Lorsqu'il est nécessaire d'insérer un caractère sans signification à la fin d'un identificateur afin de le séparer d'un autre identificateur ou mot clé (par exemple lorsqu'un identificateur est suivi d'un mot clé comme **BY** ou **OR**), les séparateurs "espace" et "tabulation" sont les séparateurs recommandés.

A.3 Productions syntaxiques en notation TTCN.MP dans la forme BNF

A.3.1 Spécification de notation TTCN

1 TTCN_Specification ::= TTCN_Module | Suite

A.3.2 Module TTCN

2 TTCN_Module ::= **$TTCN_Module** TTCN_ModuleId TTCN_ModuleOverviewPart [TTCN_ModuleImportPart] [DeclarationsPart] [ConstraintsPart] [DynamicPart] **$End_TTCN_Module**

3 **TTCN_ModuleId ::=$TTCN_ModuleId** TTCN_ModuleIdentifier

4 TTCN_ModuleIdentifier ::= Identifier

A.3.2.1 Partie présentation générale de module TTCN

5 TTCN_ModuleOverviewPart ::= **$TTCN_ModuleOverviewPart** TTCN_ModuleExports [TTCN_ModuleStructure] [TestCaseIndex] [TestStepIndex] [DefaultIndex] **$End_TTCN_ModuleOverviewPart**

A.3.2.1.1 Table d'exportation de module TTCN

6 TTCN_ModuleExports ::= **$Begin_TTCN_ModuleExports** TTCN_ModuleId [TTCN_ModuleRef] [TTCN_ModuleObjective] [StandardsRef] [PICSref] [PIXITref] [TestMethods] [Comment] ExportedObjects [Comment] **$End_TTCN_ModuleExports**

7 **TTCN_ModuleRef ::= $TTCN_ModuleRef** BoundedFreeText

8 TTCN_ModuleObjective ::= **$TTCN_ModuleObjective** BoundedFreeText

9 ExportedObjects ::= **$ExportedObjects** {ExportedObject} **$End_ExportedObjects**

10 **ExportedObject ::= $ExportedObject** ObjectId ObjectType [SourceInfo] [Comment] **$End_ExportedObject**

11 **ObjectId ::= $ObjectId** ObjectIdentifier

12 ObjectIdentifier ::= Identifier | ObjectTypeReference

13 ObjectTypeReference ::= Identifier "[" Identifier "]"

/* SÉMANTIQUE STATIQUE – Le premier identificateur est NamedNumber ou Enumeration et l'identificateur entre crochets est le nom du type correspondant. */

14 ObjectType ::= **$ObjectType** TTCN_ObjectType

15 TTCN_ObjectType ::= **SimpleType_Object** | **StructType_Object** | **ASN1_Type_Object** | **TS_Op_Object** | **TS_Proc_Object** | **TS_Par_Object** | **SelectExpr_Object** | **TS_Const_Object** | **TS_Var_Object** | **TC_Var_Object** | **PCO_Type_Object** | **PCO_Object** | **CP_Object** | **Timer_Object** | **TComp_Object** | **TCompConfig_Object** | **TTCN_ASP_Type_Object** | **ASN1_ASP_Type_Object** | **TTCN_PDU_Type_Object** | **ASN1_PDU_Type_Object** | **TTCN_CM_Type_Object** | **ASN1_CM_Type_Object** | **EncodingRule_Object** | **EncodingVariation_Object** | **InvalidFieldEncoding_Object** | **Alias_Object** | **StructTypeConstraint_Object** | **ASN1_TypeConstraint_Object** | **TTCN_ASP_Constraint_Object** | **ASN1_ASP_Constraint_Object** | **TTCN_PDU_constraint_Object** | **ASN1_PDU_Constraint_Object** | **TTCN_CM_Constraint_Object** | **ASN1_CM_Constraint_Object** | **TestCase_Object** | **TestStep_Object** | **Default_Object** | **NamedNumber_Object** | **Enumeration_Object**

16 **SourceInfo ::= $SourceInfo** (SourceIdentifier | ObjectDirective)
/* SÉMANTIQUE STATIQUE – SourceIdentifier est le nom de l'objet source original. */

17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier

18 ObjectDirective ::= Omit | **EXTERNAL**

A.3.2.1.2 Structure de module TTCN

19 TTCN_ModuleStructure ::= **$Begin_TTCN_ModuleStructure** Structure&Objectives [Comment] **$End_TTCN_ModuleStructure**

A.3.2.2 Partie importation de module TTCN

20 TTCN_ModuleImportPart ::= **$TTCN_ModuleImportPart** [ExternalObjects] [ImportDeclarations] **$End_TTCN_ModuleImportPart**

A.3.2.2.1 Objets externes

21 ExternalObjects ::= **$Begin_ExternalObjects** [ExternalGroupId] {ExternalObject}+ [Comment] **$End_ExternalObjects**

22 **ExternalGroupId ::= $ExternalGroupId** ExternalGroupIdentifier

23 ExternalGroupIdentifier ::= Identifier

24 ExternalObject ::= **$ExternalObject** ExternalObjectId ObjectType [Comment] **$End_ExternalObject**

25 **ExternalObjectId ::= $ExternalObjectId** ExternalObjectIdentifier

26 ExternalObjectIdentifier ::= ObjectIdentifier | TS_OpId&ParList | ConsId&ParList | TestStepId&ParList

A.3.2.2.2 Déclarations d'importation

27 ImportDeclarations ::= **$ImportDeclarations** {ImportsOrGroup}+ **$End_ImportDeclarations**

28 **ImportsOrGroup ::= Imports | ImportsGroup**

29 ImportsGroup ::= **$ImportsGroup** ImportsGroupId {ImportsOrGroup}+ **$End_ImportsGroup**

30 ImportsGroupId ::= **$ImportsGroupId** ImportsGroupIdentifier

31 Imports ::= **$Begin_Imports** SourceId [ImportsGroupRef] [SourceRef] [StandardsRef] [Comment] ImportedObjects [Comment] **$End_Imports**

32 **SourceId ::= $SourceId** SourceIdentifier

33 ImportsGroupRef ::= **$ImportsGroupRef** ImportsGroupReference

34 ImportsGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ImportsGroupIdentifier "/"}

35 ImportsGroupIdentifier ::= Identifier

36 SourceRef ::= **$SourceRef** BoundedFreeText

37 ImportedObjects ::= **$ImportedObjects** {ImportedObject}+ **$End_ImportedObjects**

38 **ImportedObject ::= $ImportedObject** ObjectId ObjectType [SourceInfo] [Comment] **$End_ImportedObject**

A.3.3 Suite de tests

39 Suite ::= **$Suite** SuiteId SuiteOverviewPart [ImportPart] DeclarationsPart ConstraintsPart DynamicPart **$End_Suite**
/* SÉMANTIQUE STATIQUE – L'identificateur SuiteId doit être identique à SuiteId déclaré dans la table TestSuiteStructure (Suite Structure). */

40 SuiteId ::= **$SuiteId** SuiteIdentifier

41 SuiteIdentifier ::= Identifier

A.3.3.1 Aperçu général de la suite de tests

42 SuiteOverviewPart::= **$SuiteOverviewPart** [TestSuiteIndex] SuiteStructure TestCaseIndex [TestStepIndex] [DefaultIndex] [TestSuiteExports] **$End_SuiteOverviewPart**

A.3.3.2 Indice de la suite de tests

43 TestSuiteIndex ::= **$Begin_TestSuiteIndex** {ObjectInfo} [Comment] **$End_TestSuiteIndex**

A.3.3.2.1 Informations sur l'objet importé

44 ObjectInfo ::= **$ObjectInfo** ObjectId ObjectType SourceId OrigObjectId [PageNum] [Comment] **$End_ObjectInfo**
45 PageNum ::= $PageNum PageNumber
46 PageNumber ::= Number
47 OrigObjectId ::= **$OrigObjectId** ObjectIdentifier

A.3.3.3 Structure de la suite de tests

48 SuiteStructure ::= **$Begin_SuiteStructure** SuiteId StandardsRef PICSref PIXITref TestMethods [Comment] Structure&Objectives [Comment] **$End_SuiteStructure**
49 StandardsRef ::= **$StandardsRef** BoundedFreeText
50 PICSref ::= **$PICSref** BoundedFreeText
51 PIXITref ::= **$PIXITref** BoundedFreeText
52 TestMethods ::= **$TestMethods** BoundedFreeText
53 Comment ::= **$Comment** [BoundedFreeText]
54 Structure&Objectives ::= **$Structure&Objectives** {Structure&Objective} **$End_Structure&Objectives**
55 Structure&Objective ::= **$Structure&Objective** TestGroupRef SelExprId Objective **$End_Structure&Objective**
56 SelExprId ::=**$SelectExprId** [SelectExprIdentifier]

A.3.3.4 Indice des tests élémentaires

57 TestCaseIndex ::= **$Begin_TestCaseIndex** {[CollComment] CaseIndex}+ [Comment] **$End_TestCaseIndex**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
58 CollComment ::= **$CollComment** [BoundedFreeText]
59 CaseIndex ::= **$CaseIndex** TestGroupRef TestCaseId SelExprId Description **$End_CaseIndex**
/* SÉMANTIQUE STATIQUE – Les tests élémentaires doivent être énumérés dans l'ordre où ils apparaissent dans la partie dynamique. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe de tests doit être fournie pour le premier test élémentaire de chaque groupe de modules de test. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe de tests doit être fournie pour chaque test élémentaire qui suit immédiatement un groupe de modules de test. */
60 Description ::= **$Description** BoundedFreeText

A.3.3.5 Indice des modules de test

61 TestStepIndex ::= **$Begin_TestStepIndex** {[CollComment] StepIndex} [Comment] **$End_TestStepIndex**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
62 StepIndex ::= **$StepIndex** TestStepRef TestStepId Description **$End_StepIndex**
/* SÉMANTIQUE STATIQUE – L'identificateur de module de test TestStepId ne comportera pas de liste de paramètres formels. */
/* SÉMANTIQUE STATIQUE – Les modules de test doivent être énumérés dans l'ordre où ils apparaissent dans la partie dynamique. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe de modules de test doit être fournie pour le premier module de test de chaque groupe de modules de test. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe de modules de test doit être fournie pour chaque module de test qui suit immédiatement un groupe de modules de test. */

A.3.3.6 Indice par défaut

63 DefaultIndex ::= **$Begin_DefaultIndex** {[CollComment] DefIndex} [Comment] **$End_DefaultIndex**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
64 DefIndex ::= **$DefIndex** DefaultRef DefaultId Description **$End_DefIndex**
/* SÉMANTIQUE STATIQUE – L'identificateur par défaut DefaultId ne comportera pas de liste de paramètres formels. */
/* SÉMANTIQUE STATIQUE – Les indices par défaut seront énumérés dans l'ordre où ils apparaissent dans la partie dynamique. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe par défaut sera fournie pour le premier indice par défaut de chaque groupe de comportements par défaut. */
/* SÉMANTIQUE STATIQUE – Une référence explicite de groupe par défaut sera fournie pour chaque indice par défaut qui suit immédiatement un groupe de comportements par défaut. */

A.3.3.7 Table d'exportation de suite de tests

65 TestSuiteExports::= **$Begin_TestSuiteExports** ExportedObjects [Comment] **$End_TestSuiteExports**

A.3.3.8 Partie importation

66 ImportPart ::= **$ImportPart** ImportDeclarations **$End_ImportPart**

A.3.3.9 Partie déclarative

67 DeclarationsPart ::= **$DeclarationsPart** Definitions Parameterization&Selection Declarations ComplexDefinitions **$End_DeclarationsPart**

A.3.3.10 Définitions

A.3.3.10.1 Généralités

68 Definitions ::= [TS_TypeDefs] [EncodingDefs] [TS_OpDefs] [TS_ProcDefs]

A.3.3.10.2 Définitions de type de suite de tests

69 TS_TypeDefs ::= **$TS_TypeDefs** [SimpleTypeDefsOrGroup] [StructTypeDefs] [ASN1_TypeDefs] [ASN1_TypeRefsOrGroup] **$End_TS_TypeDefs**

A.3.3.10.3 Définitions de type simple

70 SimpleTypeDefsOrGroup ::= SimpleTypeDefs | SimpleTypeGroup
71 SimpleTypeGroup ::= **$SimpleTypeGroup** SimpleTypeGroupId {SimpleTypeDefsOrGroup}+ **$End_SimpleTypeGroup**
72 SimpleTypeGroupId ::= **$SimpleTypeGroupId** SimpleTypeGroupIdentifier
73 SimpleTypeDefs ::= **$Begin_SimpleTypeDefs** [SimpleTypeGroupRef] {[CollComment] SimpleTypeDef}+ [Comment] **$End_SimpleTypeDefs**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
74 SimpleTypeGroupRef ::= **$SimpleTypeGroupRef** SimpleTypeGroupReference
75 SimpleTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {SimpleTypeGroupIdentifier "/"}
76 SimpleTypeGroupIdentifier ::= Identifier
77 SimpleTypeDef ::= **$SimpleTypeDef** SimpleTypeId SimpleTypeDefinition [PDU_FieldEncoding] [Comment] **$End_SimpleTypeDef**
78 SimpleTypeId ::= **$SimpleTypeId** SimpleTypeIdentifier
79 SimpleTypeIdentifier ::= Identifier
80 SimpleTypeDefinition ::= **$SimpleTypeDefinition** Type&Restriction
/* SÉMANTIQUE STATIQUE – Il n'y aura pas de références récursives (directes ou indirectes) à Type&Restriction. */
81 Type&Restriction ::= Type [Restriction]
/* SÉMANTIQUE STATIQUE – Le type sera PredefinedType ou SimpleType. */
82 Restriction ::= LengthRestriction | IntegerRange | SimpleValueList
/* SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par Restriction formera un sous-ensemble vrai des valeurs du type de base. */
83 LengthRestriction ::= SingleTypeLength | RangeTypeLength
/* SÉMANTIQUE STATIQUE – L'attribut LengthRestriction (restriction de longueur) ne sera fourni que si le type de base est un type chaîne [c'est-à-dire BITSTRING (chaîne binaire), HEXSTRING (chaîne hexadécimale), OCTETSTRING (chaîne d'octets) ou CharacterString (chaîne de caractères)] ou dérivé d'un type chaîne. */
84 SingleTypeLength ::= "[" Number "]"
85 RangeTypeLength ::= "[" LowerTypeBound To UpperTypeBound "]"
/* SÉMANTIQUE STATIQUE – LowerTypeBound sera un nombre non négatif. */
/* SÉMANTIQUE STATIQUE – La valeur de LowerTypeBound sera inférieure à celle de UpperTypeBound. */
86 IntegerRange ::= "(" LowerTypeBound To UpperTypeBound ")"
/* SÉMANTIQUE STATIQUE – La valeur de LowerTypeBound sera inférieure à celle de UpperTypeBound. */
87 LowerTypeBound ::= [Minus] Number | Minus **INFINITY**
88 UpperTypeBound ::= [Minus] Number | **INFINITY**
89 To ::= **TO** | ".."
90 SimpleValueList ::= "(" [Minus] LiteralValue {Comma [Minus] LiteralValue} ")"
/* SÉMANTIQUE STATIQUE – Si la valeur Minus (signe moins) est utilisée dans l'attribut SimpleValueList (liste des valeurs simples), la valeur littérale (LiteralValue) doit être un nombre. */
/* SÉMANTIQUE STATIQUE – Les valeurs LiteralValues doivent appartenir au type de base et former un sous-ensemble vrai des valeurs définies par le type de base. */

A.3.3.10.4 Définitions de type structuré

91 StructTypeDefs ::= **$StructTypeDefs** {StructTypeDefOrGroup}+ **$End_StructTypeDefs**
92 StructTypeDefOrGroup ::= StructTypeDef | StructTypeGroup
93 StructTypeGroup ::= **$StructTypeGroup** StructTypeGroupId {StructTypeDefOrGroup}+ **$End_StructTypeGroup**
94 StructTypeGroupId ::= **$StructTypeGroupId** StructTypeGroupIdentifier
95 StructTypeDef ::= **$Begin_StructTypeDef** StructId [StructTypeGroupRef] [EncVariationId] [Comment] ElemDcls [Comment] **$End_StructTypeDef**
96 StructId ::= **$StructId** StructId&FullId
97 StructId&FullId ::= StructIdentifier [FullIdentifier]

98 FullIdentifier ::= "(" BoundedFreeText ")"
/* SÉMANTIQUE STATIQUE – Certains objets en notation TTCN permettent l'abréviation des dénominations telles qu'elles apparaissent dans la norme de protocole appropriée. Si une abréviation est utilisée, l'identificateur complet FullIdentifier doit figurer dans la déclaration de l'objet. */
99 StructIdentifier ::= Identifier
100 StructTypeGroupRef ::= **$StructTypeGroupRef** StructTypeGroupReference
101 StructTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {StructTypeGroupIdentifier "/"}
102 StructTypeGroupIdentifier ::= Identifier
103 ElemDcls ::= **$ElemDcls** {ElemDcl}+ **$End_ElemDcls**
104 ElemDcl ::= **$ElemDcl** ElemId ElemType [PDU_FieldEncoding] [Comment] **$End_ElemDcl**
105 ElemId ::= **$ElemId** ElemId&FullId
106 ElemId&FullId ::= ElemIdentifier [FullIdentifier]
107 ElemIdentifier ::= Identifier
108 ElemType ::= **$ElemType** Type&Attributes
/* SÉMANTIQUE STATIQUE – Il ne peut y avoir de références récursives (ni directement ni indirectement) dans l'attribut Types&Attributes (types et attributs). */
/* SÉMANTIQUE STATIQUE – Un type d'élément de structure doit être un type prédéfini PredefinedType, un identificateur de type de suite de tests TS_TypeIdentifier, un identificateur d'unité PDU PDU_Identifier ou une unité PDU. */

A.3.3.10.5 Définitions de type en notation ASN.1

109 ASN1_TypeDefs ::= **$ASN1_TypeDefs** {ASN1_TypeDefOrGroup}+ **$End_ASN1_TypeDefs**
110 ASN1_TypeDefOrGroup ::= ASN1_TypeDef | ASN1_TypeGroup
111 ASN1_TypeGroup ::= **$ASN1_TypeGroup** ASN1_TypeGroupId {ASN1_TypeDefOrGroup}+ **$End_ASN1_TypeGroup**
112 ASN1_TypeGroupId ::= **$ASN1_TypeGroupId** ASN1_TypeGroupIdentifier
113 ASN1_TypeDef ::= **$Begin_ASN1_TypeDef** ASN1_TypeId [ASN1_TypeGroupRef] [EncVariationId] [Comment] ASN1_TypeDefinition [Comment] **$End_ASN1_TypeDef**
114 ASN1_TypeId ::= **$ASN1_TypeId** ASN1_TypeId&FullId
115 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
116 ASN1_TypeIdentifier ::= Identifier
117 ASN1_TypeGroupRef ::= **$ASN1_TypeGroupRef** ASN1_TypeGroupReference
118 ASN1_TypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_TypeGroupIdentifier "/"}
119 ASN1_TypeGroupIdentifier ::= Identifier
120 ASN1_TypeDefinition ::= **$ASN1_TypeDefinition** ASN1_Type&LocalTypes **$End_ASN1_TypeDefinition**
121 ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}
/* SÉMANTIQUE STATIQUE – Les types mentionnés provenant de la définition de type en notation ASN.1, ASN1_Type, doivent être définis soit dans d'autres tables de définition de type en notation ASN.1, soit encore par référence dans la table des références à un type en notation ASN.1, soit enfin localement (c'est-à-dire les types locaux en notation ASN.1 ASN1_LocalTypes) dans la même table après la première définition de type. */
/* SÉMANTIQUE STATIQUE – Les types ASN1_LocalTypes ne doivent pas être utilisés dans d'autres parties de la suite de tests. */
122 ASN1_Type ::= Type
/* RÉFÉRENCE – Où Type est un symbole non terminal défini dans la Recommandation X.680:
Type ::= BuiltinType | ReferencedType | ConstrainedType
Aux fins de la notation TTCN, la production suivante dans la Recommandation X.680:
SubtypeElements ::= SingleValue | ConstrainedSubtype | ValueRange | PermittedAlphabet | SizeConstraint | TypeConstraint | InnerTypeConstraint
est redéfinie comme suit:
SubtypeElements ::= SingleValue | ConstrainedSubtype | ValueRange | PermittedAlphabet | SizeConstraint | TypeConstraint | InnerTypeConstraint | ASN1_Encoding
Cela signifie que ASN1_Encoding peut être appliqué à tout endroit où une contrainte de type TypeConstraint peut être appliquée à l'ensemble d'un type ASN1_Type ou de tout type ASN.1 dans le type ASN1_Type ou dans un ensemble (SET OF) ou une séquence (SEQUENCE OF) de types [en mettant ASN1_Encoding entre parenthèses immédiatement après le mot clé SET ou SEQUENCE – à l'opposé d'une contrainte de taille (SizeConstraint) à un tel emplacement, les parenthèses sont requises car il n'y a pas d'argument de compatibilité amont permettant leur omission].

Aux fins de la notation TTCN, les productions suivantes dans la Recommandation X.680:
BuiltinValue ::=
BitStringType |
BooleanType |
CharacterStringType |
ChoiceType |
ChoiceType |
EmbeddedPDUType |
EnumeratedType |
ExternalType |
InstanceOfType |
IntegerType |

NullType |
ObjectClassFieldType |
OctetStringType |
RealType |
SequenceOfType |
SetType |
SetOfType |
TaggedType

ReferencedType ::=
DefinedType |
UsefulType |
SelectionType |
TypeFromObject |
ValueSetFromObjects

DefinedType ::=
Externaltypereference |
Typereference |
ParameterizedType|
ParameterizedValueSetType

Elements ::=
SubtypeElements |
ObjectSetElements |
"("ElementSetSpec")"

sont redéfinies comme suit:

BuiltinValue ::=
BitStringType |
BooleanType |
CharacterStringType |
ChoiceType |
EmbeddedPDUType |
EnumeratedType |
ExternalType |
IntegerType |
NullType |
ObjectIdentifierType |
RealType |
SequenceOfType |
SequenceOfType |
SetType |
SetOfType |
TaggedType

ReferencedType ::=
DefinedType |
UsefulType |
SelectionType |

DefinedType ::=
Externaltypereference |
Typereference

Elements ::=
SubtypeElements |
"("ElementSetSpec")"*/

/* SÉMANTIQUE STATIQUE – Chaque référence terminale de type utilisée dans la production Type sera une des suivantes: une référence à un type ASN1_LocalType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier. */

/* SÉMANTIQUE STATIQUE – Les définitions de type en notation ASN.1 utilisées en notation TTCN ne doivent pas faire de référence à un type externe, comme défini dans la Recommandation X.680. */

123 ASN1_LocalType ::= Typeassignment

/* RÉFÉRENCE – Où Typeassignment (affectation de type) est un symbole non terminal défini dans la Recommandation X.680. */

/* SÉMANTIQUE STATIQUE – Les définitions de type en notation ASN.1 utilisées en notation TTCN n'utiliseront pas de références à un type externe, comme défini dans la Recommandation X.680. */

A.3.3.10.6 Définitions de type ASN.1 par référence

124 ASN1_TypeRefsOrGroup ::= ASN1_TypeRefs | ASN1_TypeRefsGroup
125 ASN1_TypeRefsGroup ::= **$ASN1_TypeRefsGroup** ASN1_TypeRefsGroupId {ASN1_TypeRefsOrGroup}+ **$End_ASN1_TypeRefsGroup**
126 ASN1_TypeRefsGroupId ::= **$ASN1_TypeRefsGroupId** ASN1_TypeGroupIdentifier
127 ASN1_TypeRefs ::= **$Begin_ASN1_TypeRefs** [ASN1_TypeRefsGroupRef] {[CollComment] ASN1_TypeRef}+ [Comment] **$End_ASN1_TypeRefs**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
128 ASN1_TypeRefsGroupRef ::= **$ASN1_TypeRefsGroupRef** ASN1_TypeGroupReference
129 ASN1_TypeRef ::= **$ASN1_TypeRef** ASN1_TypeId ASN1_TypeReference ASN1_ModuleId [EncVariationId] [Comment] **$End_ASN1_TypeRef**
/* SÉMANTIQUE STATIQUE – L'identificateur ASN1_TypeId ne doit pas être déclaré en faisant usage d'un identificateur FullIdentifier. */
130 ASN1_TypeReference ::= **$ASN1_TypeReference** TypeReference
131 TypeReference ::**=** typereference
/* RÉFÉRENCE – Où la référence type TypeReference est définie dans la Recommandation X.680. */
/* SÉMANTIQUE STATIQUE – Si la définition de type ASN.1 fait référence à un autre type dans le même module ASN.1, le type de référence est importé implicitement (de la même manière que pour un module TTCN). */
132 ASN1_ModuleId ::= **$ASN1_ModuleId** ASN1_ModuleIdentifier
133 ASN1_ModuleIdentifier ::**=** ModuleIdentifier
/* RÉFÉRENCE – Où ModuleIdentifier est un identificateur non terminal défini dans la Recommandation X.680. */
/* SÉMANTIQUE STATIQUE – L'identificateur ModuleIdentifier doit être unique dans le domaine considéré. */

A.3.3.10.7 Définitions d'opérations de suite de tests

134 TS_OpDefs ::= **$TS_OpDefs** {TS_OpDefOrGroup}+ **$End_TS_OpDefs**
135 TS_OpDefOrGroup ::= TS_OpDef | TS_OpDefGroup
136 TS_OpDefGroup ::= **$TS_OpDefGroup** TS_OpDefGroupId {TS_OpDefOrGroup}+ **$End_TS_OpDefGroup**
137 TS_OpDefGroupId ::= **$TS_OpDefGroupId** TS_OpDefGroupIdentifier
138 TS_OpDefGroupIdentifier ::= Identifier
139 TS_OpDef ::= **$Begin_TS_OpDef** TS_OpId [TS_OpGroupRef] TS_OpResult [Comment] TS_OpDescription [Comment] **$End_TS_OpDef**
140 TS_OpId ::= **$TS_OpId** TS_OpId&ParList
141 TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]
/* SÉMANTIQUE STATIQUE – Le type de paramètre formel d'opération de suite de tests TS_OpIdentifier doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou ASP_Identifier, ou encore le métatype **PDU**. */
142 TS_OpIdentifier ::= Identifier
143 TS_OpGroupRef ::= **$TS_OpGroupRef** TS_OpGroupReference
144 TS_OpGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_OpGroupIdentifier "/"}
145 TS_OpGroupIdentifier ::= Identifier
146 TS_OpResult ::= **$TS_OpResult** TypeOrPDU
/* SÉMANTIQUE STATIQUE – TypeOrPDU doit être le type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou un identificateur ASP_Identifier, ou encore le métatype **PDU**. */
147 TS_OpDescription ::= **$TS_OpDescription** BoundedFreeText

A.3.3.10.8 Définitions des procédures des opérations des suites de tests

148 TS_ProcDefs ::= **$TS_ProcDefs** {TS_ProcDefOrGroup}+ **$End_TS_ProcDefs**
149 TS_ProcDefOrGroup ::= TS_ProcDef | TS_ProcDefGroup
150 TS_ProcDefGroup ::= **$TS_ProcDefGroup** TS_ProcDefGroupId {TS_ProcDefOrGroup}+ **$End_TS_ProcDefGroup**
151 TS_ProcDefGroupId ::= **$TS_ProcDefGroupId** TS_ProcDefGroupIdentifier
152 TS_ProcDefGroupIdentifier ::= Identifier
153 TS_ProcDef ::= **$Begin_TS_ProcDef** TS_ProcId [TS_ProcGroupRef] TS_ProcResult [Comment] TS_ProcDescription [Comment] **$End_TS_ProcDef**
/* EXIGENCE LEXICALE – Les commentaires peuvent être imbriqués dans une description TS_ProcDescription en les encadrant au moyen de "/*" et de "*/", mais ils ne peuvent pas être emboîtés. Ils peuvent être transmis en notation TTCN.MP, mais ils doivent être supprimés avant l'analyse lexicale de propositions en notation TTCN.MP. */
154 TS_ProcId ::= **$TS_ProcId** TS_ProcId&ParList
155 TS_ProcId&ParList ::= TS_ProcIdentifier [FormalParList]
/* SÉMANTIQUE STATIQUE – Le type de paramètre formel d'opération de suite de tests doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou un identificateur ASP_Identifier, ou encore le métatype **PDU**. */
156 TS_ProcIdentifier ::= Identifier
157 TS_ProcGroupRef ::= **$TS_ProcGroupRef** TS_ProcGroupReference
158 TS_ProcGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ProcGroupIdentifier "/"}
159 TS_ProcGroupIdentifier ::= Identifier

160 TS_ProcResult ::= **$TS_ProcResult** TypeOrPDU
/* SÉMANTIQUE STATIQUE – TypeOrPDU doit être le type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou un identificateur ASP_Identifier, ou encore le métatype **PDU**. */
161 TS_ProcDescription ::= **$TS_ProcDescription** TS_OpProcDef **$End_TS_ProcDescription**
162 TS_OpProcDef ::= [VarBlock] ProcStatement
/* NOTE – Les commentaires sont permis dans une définition TS_OpProcDef, où ils commenceront par "/*" et se termineront par "*/", mais il est supposé qu'ils seront supprimés avant l'analyse de la syntaxe. Par conséquent, la forme BNF n'inclut pas la syntaxe de tels commentaires imbriqués. */
163 VarBlock ::= **VAR** VarDcls **ENDVAR**
164 VarDcls ::= {VarDcl SemiColon}
165 VarDcl ::= [**STATIC**] VarIdentifiers Colon TypeOrPDU [Colon Value]
166 VarIdentifiers ::= VarIdentifier {Comma VarIdentifier}
167 VarIdentifier ::= Identifier
168 ProcStatement ::= ReturnValueStatement | Assignment | IfStatement | WhileLoop | CaseStatement | ProcBlock
169 ReturnValueStatement ::= **RETURNVALUE** Expression
170 IfStatement ::= **IF** Expression **THEN** {ProcStatement SemiColon}+ [**ELSE** {ProcStatement SemiColon}+] **ENDIF**
171 WhileLoop ::= **WHILE** Expression **DO** {ProcStatement SemiColon}+ **ENDWHILE**
172 CaseStatement ::= **CASE** Expression **OF** {CaseClause SemiColon}+ [**ELSE** {ProcStatement SemiColon}+] **ENDCASE**
173 CaseClause ::= IntegerLabel Colon ProcStatement
174 IntegerLabel ::= Number | TS_ParIdentifier | TS_ConstIdentifier
175 ProcBlock ::= **BEGIN** {ProcStatement SemiColon}+ **END**

A.3.3.11 Paramétrage et sélection

A.3.3.11.1 Généralités

176 Parameterization&Selection ::= [TS_ParDclsOrGroup] [SelectExprDefsOrGroup]

A.3.3.11.2 Déclarations de paramètres de suite de tests

177 TS_ParDclsOrGroup ::= TS_ParDcls | TS_ParDclsGroup
178 TS_ParDclsGroup ::= **$TS_ParDclsGroup** TS_ParDclsGroupId {TS_ParDclsOrGroup}+ **$End_TS_ParDclsGroup**
179 TS_ParDclsGroupId ::= **$TS_ParDclsGroupId** TS_ParDclsGroupIdentifier
180 TS_ParDclsGroupIdentifier ::= Identifier
181 TS_ParDcls ::= **$Begin_TS_ParDcls** [TS_ParGroupRef] {[CollComment] TS_ParDcl}+ [Comment] **$End_TS_ParDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
182 TS_ParGroupRef ::= **$TS_ParGroupRef** TS_ParGroupReference
183 TS_ParGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ParGroupIdentifier "/"}
184 TS_ParGroupIdentifier ::= Identifier
185 TS_ParDcl ::= **$TS_ParDcl** TS_ParId TS_ParType [TS_ParDefault] PICS_PIXITref [Comment] **$End_TS_ParDcl**
186 TS_ParId ::= **$TS_ParId** TS_ParIdentifier
187 TS_ParIdentifier ::= Identifier
188 TS_ParType ::= **$TS_ParType** TypeOrPDU
/* SÉMANTIQUE STATIQUE – TypeOrPDU doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou ASP_Identifier, ou encore le métatype **PDU**. */
189 TS_ParDefault ::= **$TS_ParDefault** [DefaultValue]
190 DefaultValue ::= Expression
/* SÉMANTIQUE STATIQUE – Le paramètre DefaultValue ne doit pas contenir de variables TS_Variables ou TC_Variables et il doit se résoudre en une valeur constante. */
/* SÉMANTIQUE OPÉRATOIRE – L'évaluation du paramètre DefaultValue doit produire un élément dont le type correspond au type déclaré. */
191 PICS_PIXITref ::= **$PICS_PIXITref** BoundedFreeText

A.3.3.11.3 Définitions de l'expression de sélection de test élémentaire

192 SelectExprDefsOrGroup ::= SelectExprDefs | SelectExprDefsGroup
193 SelectExprDefsGroup ::= **$SelectExprDefsGroup** SelectExprDefsGroupId {SelectExprDefsOrGroup}+ **$End_SelectExprDefsGroup**
194 SelectExprDefsGroupId ::= **$SelectExprDefsGroupId** SelectExprDefsGroupIdentifier
195 SelectExprDefsGroupIdentifier ::= Identifier
196 SelectExprDefs ::= **$Begin_SelectExprDefs** [SelectExprGroupRef] {[CollComment] SelectExprDef}+ [Comment] **$End_SelectExprDefs**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
197 SelectExprGroupRef ::= **$SelectExprGroupRef** SelectExprGroupReference
198 SelectExprGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {SelectExprGroupIdentifier "/"}
199 SelectExprGroupIdentifier ::= Identifier
200 SelectExprDef ::= **$SelectExprDef** SelectExprId SelectExpr [Comment] **$End_SelectExprDef**
201 SelectExprId ::= **$SelectExprId** SelectExprIdentifier

202 SelectExprIdentifier ::= Identifier

203 SelectExpr ::= **$SelectExpr** SelectionExpression

204 SelectionExpression ::= Expression

/* SÉMANTIQUE STATIQUE – L'expression de sélection (SelectionExpression) ne doit contenir que les valeurs LiteralValues, des identificateurs de paramètre de suite de tests (TS_ParIdentifiers), des identificateurs de constante de suite de tests (TS_ConstIdentifiers) et des identificateurs d'expression de sélection (SelectExprIdentifiers). */

/* SÉMANTIQUE OPÉRATOIRE – L'expression (SelectionExpression) doit prendre une valeur booléenne spécifique. */

/* SÉMANTIQUE STATIQUE – L'expression ne peut se rapporter de manière récursive (ni directement, ni indirectement) à l'identificateur SelExprIdentifier qu'elle définit. */

A.3.3.12 Déclarations

A.3.3.12.1 Généralités

205 Declarations ::= [TS_ConstDclsOrGroup] [TS_ConstRefsOrGroup] [TS_VarDclsOrGroup] [TC_VarDclsOrGroup] [PCO_TypeDclsOrGroup] [PCO_DclsOrGroup] [CP_DclsOrGroup] [TimerDclsOrGroup] [TCompDclsOrGroup TCompConfigDcls]

/* SÉMANTIQUE STATIQUE – Les points PCO seront facultatifs. */

A.3.3.12.2 Déclarations de constantes de suite de tests

206 TS_ConstDclsOrGroup ::= TS_ConstDcls | TS_ConstDclsGroup

207 TS_ConstDclsGroup ::= **$TS_ConstDclsGroup** TS_ConstDclsGroupId {TS_ConstDclsOrGroup}+ **$End_TS_ConstDclsGroup**

208 TS_ConstDclsGroupId ::= **$TS_ConstDclsGroupId** TS_ConstDclsGroupIdentifier

209 TS_ConstDclsGroupIdentifier ::= Identifier

210 TS_ConstDcls ::= **$Begin_TS_ConstDcls** [TS_ConstGroupRef] {[CollComment] TS_ConstDcl}+ [Comment] **$End_TS_ConstDcls**

/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */

211 TS_ConstGroupRef ::= **$TS_ConstGroupRef** TS_ConstGroupReference

212 TS_ConstGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ConstGroupIdentifier "/"}

213 TS_ConstGroupIdentifier ::= Identifier

214 TS_ConstDcl ::= **$TS_ConstDcl** TS_ConstId TS_ConstType TS_ConstValue [Comment] **$End_TS_ConstDcl**

215 TS_ConstId ::= **$TS_ConstId** TS_ConstIdentifier

216 TS_ConstIdentifier ::= Identifier

217 TS_ConstType ::= **$TS_ConstType** Type

/* SÉMANTIQUE STATIQUE – Le type ne sera pas un type structuré, le type PDU, le type ASP ou le type CM exprimé sous forme tabulaire. */

218 TS_ConstValue ::= **$TS_ConstValue** DeclarationValue

219 DeclarationValue ::= Expression

/* SÉMANTIQUE STATIQUE – La valeur de déclaration (DeclarationValue) ne contiendra pas de variables TS_Variables ou TC_Variables et elle doit être résolue en une valeur constante. */

/* SÉMANTIQUE OPÉRATOIRE – La valeur prise par DeclarationValue sera conforme à son type déclaré. */

A.3.3.12.3 Déclarations de constante de suite de tests par référence

220 TS_ConstRefsOrGroup ::= TS_ConstRefs | TS_ConstRefsGroup

221 TS_ConstRefsGroup ::= **$TS_ConstRefsGroup** TS_ConstRefsGroupId {TS_ConstRefsOrGroup}+ **$End_TS_ConstRefsGroup**

222 TS_ConstRefsGroupId ::= **$TS_ConstRefsGroupId** TS_ConstRefsGroupIdentifier

223 TS_ConstRefsGroupIdentifier ::= Identifier

224 TS_ConstRefs ::= **$Begin_TS_ConstRefs** [TS_ConstRefsGroupRef] {[CollComment] TS_ConstRef}+ [Comment] **$End_TS_ConstRefs**

/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */

225 TS_ConstRefsGroupRef ::= **$TS_ConstRefsGroupRef** TS_ConstGroupReference

226 TS_ConstRef ::= **$TS_ConstRef** TS_ConstId TS_ConstType ASN1_ValueReference ASN1_ModuleId [Comment] **$End_TS_ConstRef**

/* SÉMANTIQUE STATIQUE – Le type de TS_ConstType sera soit un type PredefinedType ou un type ASN1_Type importé par référence provenant du module auquel l'identificateur ASN1_ModuleId fait référence. */

227 ASN1_ValueReference ::= **$ASN1_ValueReference** ValueReference

228 ValueReference ::= valuereference

/* RÉFÉRENCE – valuereference est une référence non terminale définie dans la Recommandation X.680. */

/* SÉMANTIQUE STATIQUE – La valeur doit correspondre à celle d'un élément du type TS_ConstType. */

A.3.3.12.4 Déclarations de variables de suite de tests

TS_VarDclsOrGroup ::= TS_VarDcls | TS_VarDclsGroup
TS_VarDclsGroup ::= **$TS_VarDclsGroup** TS_VarDclsGroupId {TS_VarDclsOrGroup}+ **$End_TS_VarDclsGroup**
TS_VarDclsGroupId ::= **$TS_VarDclsGroupId** TS_VarDclsGroupIdentifier
TS_VarDclsGroupIdentifier ::= Identifier
TS_VarDcls ::= **$Begin_TS_VarDcls** [TS_VarGroupRef] {[CollComment] TS_VarDcl}+ [Comment] **$End_TS_VarDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
TS_VarGroupRef ::= **$TS_VarGroupRef** TS_VarGroupReference
TS_VarGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS"VarGroupIdentifier "/"}
TS_VarGroupIdentifier ::= Identifier
TS_VarDcl ::= **$TS_VarDcl** TS_VarId TS_VarType TS_VarValue [Comment] **$End_TS_VarDcl**
TS_VarId ::= **$TS_VarId** TS_VarIdentifier
TS_VarIdentifier ::= Identifier
TS_VarType ::= **$TS_VarType** TypeOrPDU
/* SÉMANTIQUE STATIQUE – Le type TypeOrPDU sera un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou ASP_Identifier, ou encore le métatype **PDU**. */
TS_VarValue ::= **$TS_VarValue** [DeclarationValue]

A.3.3.12.5 Déclarations de variables de test élémentaire

TC_VarDclsOrGroup ::= TC_VarDcls | TC_VarDclsGroup
TC_VarDclsGroup ::= **$TC_VarDclsGroup** TC_VarDclsGroupId {TC_VarDclsOrGroup}+ **$End_TC_VarDclsGroup**
TC_VarDclsGroupId ::= **$TC_VarDclsGroupId** TC_VarDclsGroupIdentifier
TC_VarDclsGroupIdentifier ::= Identifier
TC_VarDcls ::= **$Begin_TC_VarDcls** [TC_VarGroupRef] {[CollComment] TC_VarDcl}+ [Comment] **$End_TC_VarDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
TC_VarGroupRef ::= **$TC_VarGroupRef** TC_VarGroupReference
TC_VarGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TC_VarGroupIdentifier "/"}
TC_VarGroupIdentifier ::= Identifier
TC_VarDcl ::= **$TC_VarDcl** TC_VarId TC_VarType TC_VarValue [Comment] **$End_TC_VarDcl**
TC_VarId ::= **$TC_VarId** TC_VarIdentifier
TC_VarIdentifier ::= Identifier
TC_VarType ::= **$TC_VarType** TypeOrPDU
/* SÉMANTIQUE STATIQUE – Le type TypeOrPDU sera un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou ASP_Identifier, ou encore le métatype **PDU**. */
TC_VarValue ::= **$TC_VarValue** [DeclarationValue]

A.3.3.12.6 Déclarations de type PCO

PCO_TypeDclsOrGroup ::= PCO_TypeDcls | PCO_TypeDclsGroup
PCO_TypeDclsGroup ::= **$PCO_TypeDclsGroup** PCO_TypeDclsGroupId {PCO_TypeDclsOrGroup}+ **$End_PCO_TypeDclsGroup**
PCO_TypeDclsGroupId ::= **$PCO_TypeDclsGroupId** PCO_TypeDclsGroupIdentifier
PCO_TypeDclsGroupIdentifier ::= Identifier
PCO_TypeDcls ::= **$Begin_PCO_TypeDcls** [PCO_TypeGroupRef] {[CollComment] PCO_TypeDcl}+ [Comment] **$End_PCO_TypeDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
PCO_TypeGroupRef ::= **$PCO_TypeGroupRef** PCO_GroupReference
PCO_TypeDcl ::= **$PCO_TypeDcl** PCO_TypeId RoleOrComment **$End_PCO_TypeDcl**
PCO_TypeId ::= **$PCO_TypeId** PCO_TypeIdentifier
PCO_TypeIdentifier ::= Identifier
RoleOrComment ::= P_Role [Comment] | Comment
/* NOTE – Comme il faut que le rôle de chaque type PCO_Type dans une table de déclaration de type PCO soit spécifié dans la colonne rôle ou commentaires, au moins un des éléments P_Role ou Comment doit être présent. */

A.3.3.12.7 Déclarations de points de contrôle et d'observation (PCO)

PCO_DclsOrGroup ::= PCO_Dcls | PCO_DclsGroup
PCO_DclsGroup ::= **$PCO_DclsGroup** PCO_DclsGroupId {PCO_DclsOrGroup}+ **$End_PCO_DclsGroup**
PCO_DclsGroupId ::= **$PCO_DclsGroupId** PCO_DclsGroupIdentifier
PCO_DclsGroupIdentifier ::= Identifier

269 PCO_Dcls ::= **$Begin_PCO_Dcls** [PCO_GroupRef] {[CollComment] PCO_Dcl}+ [Comment] **$End_PCO_Dcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
/* SÉMANTIQUE STATIQUE – Conformément à la Recommandation X.290, le nombre de points PCO correspondra à la méthode de test utilisée. */
270 PCO_GroupRef ::= **$PCO_GroupRef** PCO_GroupReference
271 PCO_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PCO_GroupIdentifier "/"}
272 PCO_GroupIdentifier ::= Identifier
273 PCO_Dcl ::= **$PCO_Dcl** PCO_Id PCO_TypeId&MuxValue [P_Role] [Comment] **$End_PCO_Dcl**
274 PCO_Id ::= **$PCO_Id** PCO_Identifier
275 PCO_Identifier ::= Identifier
276 PCO_TypeId&MuxValue ::= **$PCO_TypeId** PCO_TypeIdentifier ["(" MuxValue ")"]
277 MuxValue ::= TS_ParIdentifier
278 P_Role ::= **$PCO_Role** [PCO_Role]
279 PCO_Role **::= UT** | **LT**

A.3.3.12.8 Déclarations de points de contrôle (CP)

280 CP_DclsOrGroup ::= CP_Dcls | CP_DclsGroup
281 CP_DclsGroup ::= **$CP_DclsGroup** CP_DclsGroupId {CP_DclsOrGroup}+ **$End_CP_DclsGroup**
282 CP_DclsGroupId ::= **$CP_DclsGroupId** CP_DclsGroupIdentifier
283 CP_DclsGroupIdentifier ::= Identifier
284 CP_Dcls ::= **$Begin_CP_Dcls** [CP_GroupRef] {[CollComment] CP_Dcl}+ [Comment] **$End_CP_Dcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
285 CP_GroupRef ::= **$CP_GroupRef** CP_GroupReference
286 CP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {CP_GroupIdentifier "/"}
287 CP_GroupIdentifier ::= Identifier
288 CP_Dcl ::= **$CP_Dcl** CP_Id [Comment] **$End_CP_Dcl**
289 CP_Id ::= **$CP_Id** CP_Identifier
290 CP_Identifier ::= Identifier

A.3.3.12.9 Déclarations de temporisation

291 TimerDclsOrGroup ::= TimerDcls | TimerDclsGroup
292 TimerDclsGroup ::= **$TimerDclsGroup** TimerDclsGroupId {TimerDclsOrGroup}+ **$End_TimerDclsGroup**
293 TimerDclsGroupId ::= **$TimerDclsGroupId** TimerDclsGroupIdentifier
294 TimerDclsGroupIdentifier ::= Identifier
295 TimerDcls ::= **$Begin_TimerDcls** [TimerGroupRef] {[CollComment] TimerDcl}+ [Comment] **$End_TimerDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
296 TimerGroupRef ::= **$TimerGroupRef** TimerGroupReference
297 TimerGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TimerGroupIdentifier "/"}
298 TimerGroupIdentifier ::= Identifier
299 TimerDcl ::= **$TimerDcl** TimerId Duration Unit [Comment] **$End_TimerDcl**
300 TimerId ::= **$TimerId** TimerIdentifier
301 TimerIdentifier ::= Identifier
302 Duration ::= **$Duration** [DeclarationValue]
/* SÉMANTIQUE OPÉRATOIRE – DeclarationValue aura une valeur de type INTEGER (entier), positive et non nulle. */
303 Unit ::= **$Unit** TimeUnit
304 TimeUnit ::= **ps** | **ns** | **us** | **ms** | **s** | **min**
/* SÉMANTIQUE STATIQUE – Si une temporisation est dérivée d'une déclaration de conformité d'implémentation de protocole (PICS), des informations supplémentaires sur l'implémentation de protocole destinée au test (PIXIT), la déclaration de temporisation doit alors spécifier les mêmes unités que l'entrée PICS/PIXIT. */

A.3.3.12.10 Déclarations de composantes de tests

305 TCompDclsOrGroup ::= TCompDcls | TCompDclsGroup
306 TCompDclsGroup ::= **$TCompDclsGroup** TCompDclsGroupId {TCompDclsOrGroup}+ **$End_TCompDclsGroup**
307 TCompDclsGroupId ::= **$TCompDclsGroupId** TCompDclsGroupIdentifier
308 TCompDclsGroupIdentifier ::= Identifier
309 TCompDcls ::= **$Begin_TCompDcls** [TCompGroupRef] {[CollComment] TCompDcl}+ [Comment] **$End_TCompDcls**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
310 TCompGroupRef ::= **$TCompGroupRef** TCompGroupReference
311 TCompGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TCompGroupIdentifier "/"}
312 TCompGroupIdentifier ::= Identifier
313 TCompDcl ::= **$TCompDcl** TCompId C_Role NumOf_PCOs NumOf_CPs [Comment] **$End_TCompDcl**
314 TCompId ::= **$TCompId** TCompIdentifier
315 TCompIdentifier::= Identifier

316 C_Role ::= **$TCompRole** TCompRole
317 TCompRole **::= MTC** | **PTC**
318 NumOf_PCOs ::= **$NumOf_PCOs** Num_PCOs
319 Num_PCOs ::= Number
320 NumOf_CPs ::= **$NumOf_CPs** Num_CPs
321 Num_CPs ::= Number

A.3.3.12.11 Déclarations de configuration de composantes de test

322 TCompConfigDcls ::= **$TCompConfigDcls** {TCompConfigDclOrGroup}+ **$End_TCompConfigDcls**
323 TCompConfigDclOrGroup ::= TCompConfigDcl | TCompConfigDclGroup
324 TCompConfigDclGroup ::= **$TCompConfigDclGroup** TCompConfigDclGroupId {TCompConfigDclOrGroup}+ **$End_TCompConfigDclGroup**
325 TCompConfigDclGroupId ::= **$TCompConfigDclGroupId** TCompConfigDclGroupIdentifier
326 TCompConfigDclGroupIdentifier ::= Identifier
327 TCompConfigDcl ::= **$Begin_TCompConfigDcl** TCompConfigId [TCompConfigGroupRef] [Comment] TCompConfigInfos [Comment] **$End_TCompConfigDcl**
328 TCompConfigId ::= **$TCompConfigId** TCompConfigIdentifier
329 TCompConfigIdentifier ::= Identifier
330 TCompConfigGroupRef ::= **$TCompConfigGroupRef** TCompConfigGroupReference
331 TCompConfigGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TCompConfigGroupIdentifier "/"}
332 TCompConfigGroupIdentifier ::= Identifier
333 TCompConfigInfos ::= **$TCompConfigInfos** {TCompConfigInfo}+ **$End_TCompConfigInfos**
/* SÉMANTIQUE STATIQUE – Il y aura exactement une information TCompConfigInfos pour une composante de test dont le rôle TCompRole est **MTC**. */
334 TCompConfigInfo ::= **$TCompConfigInfo** TCompUsed PCOs_Used CPs_Used [Comment] **$End_TCompConfigInfo**
335 TCompUsed ::= **$TCompUsed** TCompIdentifier
336 PCOs_Used ::= **$PCOs_Used** [PCO_List]
337 PCO_List ::= PCO_Identifier {Comma PCO_Identifier}
/* SÉMANTIQUE STATIQUE – Le nombre de points PCO dans la liste PCO_List doit être identique au nombre indiqué dans la déclaration de composantes de test. */
/* SÉMANTIQUE STATIQUE – Un identificateur PCO_Identifier ne doit pas être utilisé plus d'une fois dans une même configuration de composantes de test. */
338 CPs_Used ::= **$CPs_Used** [CP_List]
339 CP_List ::= CP_Identifier {Comma CP_Identifier}
/* SÉMANTIQUE STATIQUE – Pour une composante PTC, le nombre de points CP dans la liste CP_List doit être identique au nombre indiqué dans la déclaration de composantes de test. */
/* SÉMANTIQUE STATIQUE – Pour une composante MTC, le nombre de points CP dans la liste CP_List ne doit pas être supérieur au nombre indiqué dans la déclaration de composantes de test. */
/* SÉMANTIQUE STATIQUE – Un identificateur CP_Identifier ne doit pas figurer plus d'une fois dans une liste CP_List donnée. */
/* SÉMANTIQUE STATIQUE – Chacun des identificateurs CP_Identifier utilisés dans une configuration de composantes de test doit figurer dans la liste CP_List d'exactement deux composantes de test de cette configuration. */

A.3.3.13 Définitions de type de primitive ASP, d'unité PDU et de message CM

A.3.3.13.1 Généralités

340 ComplexDefinitions ::= [ASP_TypeDefs] [PDU_TypeDefs] [CM_TypeDefs] [AliasDefsOrGroup]
/* SÉMANTIQUE STATIQUE – Les unités PDU seront facultatives. */

A.3.3.13.2 Définitions de type de primitive ASP

341 ASP_TypeDefs ::= **$ASP_TypeDefs** [TTCN_ASP_TypeDefs] [ASN1_ASP_TypeDefs] [ASN1_ASP_TypeDefsByRefOrGroup] **$End_ASP_TypeDefs**

A.3.3.13.3 Définitions de type de primitive ASP sous forme tabulaire

342 TTCN_ASP_TypeDefs ::= **$TTCN_ASP_TypeDefs** {TTCN_ASP_TypeDefOrGroup}+ **$End_TTCN_ASP_TypeDefs**
343 TTCN_ASP_TypeDefOrGroup ::= TTCN_ASP_TypeDef | TTCN_ASP_TypeDefGroup
344 TTCN_ASP_TypeDefGroup ::= **$TTCN_ASP_TypeDefGroup** TTCN_ASP_TypeDefGroupId {TTCN_ASP_TypeDefOrGroup}+ **$End_TTCN_ASP_TypeDefGroup**
345 TTCN_ASP_TypeDefGroupId ::= **$TTCN_ASP_TypeDefGroupId** ASP_GroupIdentifier
346 TTCN_ASP_TypeDef ::= **$Begin_TTCN_ASP_TypeDef** ASP_Id [ASP_GroupRef] PCO_Type [Comment] [ASP_ParDcls] [Comment] **$End_TTCN_ASP_TypeDef**
347 ASP_Id ::= **$ASP_Id** ASP_Id&FullId
348 ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]

349 ASP_Identifier ::= Identifier
/* SÉMANTIQUE STATIQUE – L'identificateur peut être l'identificateur AliasIdentifier à la condition qu'il soit utilisé dans la colonne de comportement d'une table de comportement (par exemple dans une description comportementale). */
350 ASP_GroupRef ::= **$ASP_GroupRef** ASP_GroupReference
351 ASP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASP_GroupIdentifier "/"}
352 ASP_GroupIdentifier ::= Identifier
353 PCO_Type ::= **$PCO_Type** [PCO_TypeIdentifier]
/* SÉMANTIQUE STATIQUE – S'il n'y a pas de table de déclaration de type PCO_Type, l'identificateur PCO_TypeIdentifier doit appartenir à un des types PCO figurant dans la table de déclaration de point PCO. */
/* SÉMANTIQUE STATIQUE – Si un seul point PCO est défini dans une suite de tests, l'identificateur PCO_TypeIdentifier est facultatif. */
354 ASP_ParDcls ::= **$ASP_ParDcls** {ASP_ParDcl} **$End_ASP_ParDcls**
355 ASP_ParDcl ::= **$ASP_ParDcl** ASP_ParId ASP_ParType [Comment] **$End_ASP_ParDcl**
356 ASP_ParId ::= **$ASP_ParId** ASP_ParIdOrMacro
357 ASP_ParIdOrMacro ::= ASP_ParId&FullId | MacroSymbol
/* SÉMANTIQUE STATIQUE – Le symbole MacroSymbol ne doit être utilisé qu'en combinaison avec une référence à un type Structured Type. */
358 ASP_ParId&FullId ::= ASP_ParIdentifier [FullIdentifier]
359 ASP_ParIdentifier ::= Identifier
360 ASP_ParType ::= **$ASP_ParType** Type&Attributes
/* SÉMANTIQUE STATIQUE – Le type doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou une unité **PDU**. */

A.3.3.13.4 Définitions de type de primitive ASP en notation ASN.1

361 ASN1_ASP_TypeDefs ::= **$ASN1_ASP_TypeDefs** {ASN1_ASP_TypeDefOrGroup} **$End_ASN1_ASP_TypeDefs**
362 ASN1_ASP_TypeDefOrGroup ::= ASN1_ASP_TypeDef | ASN1_ASP_TypeDefGroup
363 ASN1_ASP_TypeDefGroup ::= **$ASN1_ASP_TypeDefGroup** ASN1_ASP_TypeDefGroupId {ASN1_ASP_TypeDefOrGroup}+ **$End_ASN1_ASP_TypeDefGroup**
364 ASN1_ASP_TypeDefGroupId ::= **$ASN1_ASP_TypeDefGroupId** ASN1_ASP_GroupIdentifier
365 ASN1_ASP_TypeDef ::= **$Begin_ASN1_ASP_TypeDef** ASP_Id [ASN1_ASP_GroupDef] PCO_Type [Comment] [ASN1_TypeDefinition] [Comment] **$End_ASN1_ASP_TypeDef**
366 ASN1_ASP_GroupRef ::= **$ASN1_ASP_GroupRef** ASN1_ASP_GroupReference
367 ASN1_ASP_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_ASP_GroupIdentifier "/"}
368 ASN1_ASP_GroupIdentifier ::= Identifier

A.3.3.13.5 Définitions de type de primitive ASP en notation ASN.1 par référence

369 ASN1_ASP_TypeDefsByRefOrGroup ::= ASN1_ASP_TypeDefsByRef | ASN1_ASP_TypeDefsByRefGroup
370 ASN1_ASP_TypeDefsByRefGroup ::= **$ASN1_ASP_TypeDefsByRefGroup** ASN1_ASP_TypeDefsByRefGroupId {ASN1_ASP_TypeDefsByRefOrGroup}+ **$End_ASN1_ASP_TypeDefsByRefGroup**
371 ASN1_ASP_TypeDefsByRefGroupId ::= **$ASN1_ASP_TypeDefsByRefGroupId** ASN1_ASP_GroupIdentifier
372 ASN1_ASP_TypeDefsByRef ::= **$Begin_ASN1_ASP_TypeDefsByRef** [ASN1_ASP_DefsByRefGroupRef] {[CollComment] ASN1_ASP_TypeDefByRef}+ [Comment] **$End_ASN1_ASP_TypeDefsByRef**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
373 ASN_ASP_DefsByRefGroupRef ::= **$ASN1_ASP_DefsByRefGroupRef** ASN1_ASP_GroupReference
374 ASN1_ASP_TypeDefByRef ::= **$ASN1_ASP_TypeDefByRef** ASP_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [Comment] **$End_ASN1_ASP_TypeDefByRef**
/* SÉMANTIQUE STATIQUE – L'identificateur ASP_Id ne doit pas être déclaré en faisant usage de l'identificateur FullIdentifier. */

A.3.3.13.6 Définitions de types d'unité PDU

375 PDU_TypeDefs ::= **$PDU_TypeDefs** [TTCN_PDU_TypeDefs] [ASN1_PDU_TypeDefs] [ASN1_PDU_TypeDefsByRefOrGroup] **$End_PDU_TypeDefs**

A.3.3.13.7 Définitions de types d'unité PDU sous forme tabulaire

376 TTCN_PDU_TypeDefs ::= **$TTCN_PDU_TypeDefs** {TTCN_PDU_TypeDefOrGroup}+ **$End_TTCN_PDU_TypeDefs**
377 TTCN_PDU_TypeDefOrGroup ::= TTCN_PDU_TypeDef | TTCN_PDU_TypeDefGroup
378 TTCN_PDU_TypeDefGroup ::= **$TTCN_PDU_TypeDefGroup** TTCN_PDU_TypeDefGroupId {TTCN_PDU_TypeDefOrGroup}+ **$End_TTCN_PDU_TypeDefGroup**
379 TTCN_PDU_TypeDefGroupId ::= **$TTCN_PDU_TypeDefGroupId** PDU_GroupIdentifier
380 TTCN_PDU_TypeDef ::= **$Begin_TTCN_PDU_TypeDef** PDU_Id [PDU_GroupRef] PCO_Type [PDU_EncodingId] [EncVariationId] [Comment] [PDU_FieldDcls] [Comment] **$End_TTCN_PDU_TypeDef**
/* SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant seulement incluse dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif. */
381 PDU_Id ::= **$PDU_Id** PDU_Id&FullId
382 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]

PDU_Identifier ::= Identifier
/* SÉMANTIQUE STATIQUE – L'identificateur peut être l'identificateur AliasIdentifier à la condition qu'il soit utilisé dans la colonne de comportement d'une table comportementale (c'est-à-dire dans une description comportementale). */
PDU_GroupRef ::= **$PDU_GroupRef** PDU_GroupReference
PDU_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PDU_GroupIdentifier "/"}
PDU_GroupIdentifier ::= Identifier
PDU_EncodingId ::= **$PDU_EncodingId** [EncodingRuleIdentifier]
PDU_FieldDcls ::= **$PDU_FieldDcls** {PDU_FieldDcl} **$End_PDU_FieldDcls**
PDU_FieldDcl ::= **$PDU_FieldDcl** PDU_FieldId PDU_FieldType [PDU_FieldEncoding] [Comment] **$End_PDU_FieldDcl**
PDU_FieldId ::= **$PDU_FieldId** PDU_FieldIdOrMacro
PDU_FieldIdOrMacro ::= PDU_FieldId&FullId | MacroSymbol
/* SÉMANTIQUE STATIQUE – Le symbole MacroSymbol ne doit être utilisé qu'en combinaison avec une référence à un type structuré. */
MacroSymbol ::= "<-"
PDU_FieldId&FullId ::= PDU_FieldIdentifier [FullIdentifier]
PDU_FieldIdentifier ::= Identifier
PDU_FieldType ::= **$PDU_FieldType** Type&Attributes
/* SÉMANTIQUE STATIQUE – Type doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou une unité **PDU**. */
Type&Attributes ::= (Type [LengthAttribute]) | **PDU**
/* SÉMANTIQUE OPÉRATOIRE – L'ensemble des valeurs définies par l'attribut LengthAttribute doit être un sous-ensemble vrai des valeurs du type de base. */
/* SÉMANTIQUE STATIQUE – L'attribut LengthAttribute ne sera indiqué que si le type de base est un type chaîne (à savoir BITSTRING, HEXSTRING, OCTETSTRING ou CharacterString) ou s'il en dérive. */
LengthAttribute ::= SingleLength | RangeLength
SingleLength ::= "[" Bound "]"
Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
/* SÉMANTIQUE OPÉRATOIRE – L'attribut Bound (borne) prendra une valeur non négative de type INTEGER (entier) ou la valeur INFINITY (infini). */
RangeLength ::= "[" LowerBound To UpperBound "]"
/* SÉMANTIQUE OPÉRATOIRE – La valeur de LowerBound (borne inférieure) doit être inférieure à celle de UpperBound (borne supérieure). */
LowerBound ::= Bound
UpperBound ::= Bound | **INFINITY**

A.3.3.13.8 Définitions de types d'unité PDU en notation ASN.1

ASN1_PDU_TypeDefs ::= **$ASN1_PDU_TypeDefs** {ASN1_PDU_TypeDefOrGroup} **$End_ASN1_PDU_TypeDefs**
ASN1_PDU_TypeDefOrGroup ::= ASN1_PDU_TypeDef | ASN1_PDU_TypeDefGroup
ASN1_PDU_TypeDefGroup ::= **$ASN1_PDU_TypeDefGroup** ASN1_PDU_TypeDefGroupId {ASN1_PDU_TypeDefOrGroup}+ **$End_ASN1_PDU_TypeDefGroup**
ASN1_PDU_TypeDefGroupId ::= **$ASN1_PDU_TypeDefGroupId** ASN1_PDU_GroupIdentifier
ASN1_PDU_TypeDef ::= **$Begin_ASN1_PDU_TypeDef** PDU_Id [ASN1_PDU_GroupRef] PCO_Type [PDU_EncodingId] [EncVariationId] [Comment] [ASN1_TypeDefinition] [Comment] **$End_ASN1_PDU_TypeDef**
/* SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant incluse seulement dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif. */
ASN1_PDU_GroupRef ::= **$ASN1_PDU_GroupRef** ASN1_PDU_GroupReference
ASN1_PDU_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_PDU_GroupIdentifier "/"}
ASN1_PDU_GroupIdentifier ::= Identifier

A.3.3.13.9 Définitions de types d'unité PDU en notation ASN.1 par référence

ASN1_PDU_TypeDefsByRefOrGroup ::= ASN1_PDU_TypeDefsByRef | ASN1_PDU_TypeDefsByRefGroup
ASN1_PDU_TypeDefsByRefGroup ::= **$ASN1_PDU_TypeDefsByRefGroup** ASN1_PDU_TypeDefsByRefGroupId {ASN1_PDU_TypeDefsByRefOrGroup}+ **$End_ASN1_PDU_TypeDefsByRefGroup**
ASN1_PDU_TypeDefsByRefGroupId ::= **$ASN1_PDU_TypeDefsByRefGroupId** ASN1_PDU_GroupIdentifier
ASN1_PDU_TypeDefsByRef ::= **$Begin_ASN1_PDU_TypeDefsByRef** [ASN1_PDU_DefsByRefGroupRef] {[CollComment] ASN1_PDU_TypeDefByRef}+ [Comment] **$End_ASN1_PDU_TypeDefsByRef**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
ASN1_PDU_DefsByRefGroupRef ::= **$ASN1_PDU_DefsByRefGroupRef** ASN1_PDU_GroupReference
ASN1_PDU_TypeDefByRef ::= **$ASN1_PDU_TypeDefByRef** PDU_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [PDU_EncodingId] [EncVariationId] [Comment] **$End_ASN1_PDU_TypeDefByRef**
/* SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant seulement incluse dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif. */
/* SÉMANTIQUE STATIQUE – L'identificateur PDU_Id ne doit pas être déclaré avec un identificateur FullIdentifier. */

A.3.3.13.10 Définitions de type de message CM

417 CM_TypeDefs ::= **$CM_TypeDefs** [TTCN_CM_TypeDefs] [ASN1_CM_TypeDefs] **$End_CM_TypeDefs**

A.3.3.13.11 Définitions de type de message CM tabulaire

418 TTCN_CM_TypeDefs ::= **$TTCN_CM_TypeDefs** {TTCN_CM_TypeDefOrGroup}+ **$End_TTCN_CM_TypeDefs**
419 TTCN_CM_TypeDefOrGroup ::= TTCN_CM_TypeDef | TTCN_CM_TypeDefGroup
420 TTCN_CM_TypeDefGroup ::= **$TTCN_CM_TypeDefGroup** TTCN_CM_TypeDefGroupId
{TTCN_CM_TypeDefOrGroup}+ **$End_TTCN_CM_TypeDefGroup**
421 TTCN_CM_TypeDefGroupId ::= **$TTCN_CM_TypeDefGroupId** CM_GroupIdentifier
422 TTCN_CM_TypeDef ::= **$Begin_TTCN_CM_TypeDef** CM_Id [CM_GroupRef] [Comment] [CM_ParDcls] [Comment]
$End_TTCN_CM_TypeDef
423 CM_Id ::= **$CM_Id** CM_Identifier
424 CM_Identifier ::= Identifier
425 CM_GroupRef ::= **$CM_GroupRef** CM_GroupReference
426 CM_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {CM_GroupIdentifier "/"}
427 CM_GroupIdentifier ::= Identifier
428 CM_ParDcls ::= **$CM_ParDcls** {CM_ParDcl} **$End_CM_ParDcls**
429 CM_ParDcl ::= **$CM_ParDcl** CM_ParId CM_ParType [Comment] **$End_CM_ParDcl**
430 CM_ParId ::= **$CM_ParId** CM_ParIdOrMacro
431 CM_ParIdOrMacro ::= CM_ParIdentifier | MacroSymbol
/* SÉMANTIQUE STATIQUE – Le symbole MacroSymbol ne doit être utilisé qu'en combinaison avec une référence à un type structuré. */
432 CM_ParIdentifier ::= Identifier
433 CM_ParType ::= **$CM_ParType** Type&Attributes

A.3.3.13.12 Définitions de type de message CM en notation ASN.1

434 ASN1_CM_TypeDefs ::= **$ASN1_CM_TypeDefs** {ASN1_CM_TypeDefOrGroup}+ **$End_ASN1_CM_TypeDefs**
435 ASN1_CM_TypeDefOrGroup ::= ASN1_CM_TypeDef | ASN1_CM_TypeDefGroup
436 ASN1_CM_TypeDefGroup ::= **$ASN1_CM_TypeDefGroup** ASN1_CM_TypeDefGroupId
{ASN1_CM_TypeDefOrGroup}+ **$End_ASN1_CM_TypeDefGroup**
437 ASN1_CM_TypeDefGroupId ::= **$ASN1_CM_TypeDefGroupId** ASN1_CM_GroupIdentifier
438 ASN1_CM_TypeDef ::= **$Begin_ASN1_CM_TypeDef** CM_Id [ASN1_CM_GroupRef] [Comment]
[ASN1_TypeDefinition] [Comment] **$End_ASN1_CM_TypeDef**
439 ASN1_CM_GroupRef ::= **$ASN1_CM_GroupRef** ASN1_CM_GroupReference
440 ASN1_CM_GroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_CM_GroupIdentifier "/"}
441 ASN1_CM_GroupIdentifier ::= Identifier

A.3.3.13.13 Définitions des variations de codage

442 EncodingDefs ::= **$EncodingDefs** [EncodingDefinitionsOrGroup] [EncodingVariations] [InvalidFieldEncodingDefs]
$End_EncodingDefs

A.3.3.13.13.1 Définitions du codage

443 EncodingDefinitionsOrGroup ::= EncodingDefinitions | EncodingDefinitionsGroup
444 EncodingDefinitionsGroup ::= **$EncodingDefinitionsGroup** EncodingDefinitionsGroupId
{EncodingDefinitionsOrGroup}+ **$End_EncodingDefinitionsGroup**
445 EncodingDefinitionsGroupId ::= **$EncodingDefinitionsGroupId** EncodingGroupIdentifier
446 EncodingDefinitions ::= **$Begin_EncodingDefinitions** [EncodingGroupRef] {[CollComment] EncodingDefinition}+
[Comment] **$End_EncodingDefinitions**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
447 EncodingGroupRef ::= **$EncodingGroupRef** EncodingGroupReference
448 EncodingGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {EncodingGroupIdentifier "/"}
449 EncodingGroupIdentifier ::= Identifier
450 EncodingDefinition ::= **$EncodingDefinition** EncodingRuleId EncodingRef EncodingDefault [Comment]
$End_EncodingDefinition
/* SÉMANTIQUE OPÉRATOIRE – Il ne peut y avoir plus d'un identificateur EncodingRuleIdentifier dont le comportement par défaut EncodingDefault contient une expression DefaultExpression dont la valeur est TRUE. */
451 EncodingRuleId ::= **$EncodingRuleId** EncodingRuleIdentifier
452 EncodingRuleIdentifier ::= Identifier
453 EncodingRef ::= **$EncodingRef** EncodingReference
454 EncodingReference ::= BoundedFreeText
455 EncodingDefault ::= **$EncodingDefault** [DefaultExpression]
456 DefaultExpression ::= Expression
/* SÉMANTIQUE STATIQUE – L'expression DefaultExpression ne doit contenir que des valeurs LiteralValues, des identificateurs TS_ParIdentifiers et des identificateurs TS_ConstIdentifiers. */

A.3.3.13.13.2 Variations de codage

457 EncodingVariations ::= **$EncodingVariations** {EncodingVariationSetOrGroup}+ **$End_EncodingVariations**
458 EncodingVariationSetOrGroup ::= EncodingVariationSet | EncodingVariationSetGroup
459 EncodingVariationSetGroup ::= **$EncodingVariationSetGroup** EncodingVariationSetGroupId {EncodingVariationSetOrGroup}+ **$End_EncodingVariationSetGroup**
460 EncodingVariationSetGroupId ::= **$EncodingVariationSetGroupId** EncVariationGroupIdentifier
461 EncodingVariationSet ::= **$Begin_EncodingVariationSet** EncodingRuleId [EncVariationGroupRef] Encoding_TypeList [Comment] EncodingVariationList [Comment] **$End_EncodingVariationSet**
462 EncVariationGroupRef ::= **$EncVariationGroupRef** EncVariationGroupReference
463 EncVariationGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {EncVariationGroupIdentifier "/"}
464 EncVariationGroupIdentifier ::= Identifier
465 EncodingVariationList ::= **$EncodingVariationList** {EncodingVariation}+ **$End_EncodingVariationList**
466 Encoding_TypeList ::= **$Encoding_TypeList** [TypeList]
467 TypeList ::=Type {Comma Type}
/* SÉMANTIQUE STATIQUE – Le type ne doit pas être ASP_Identifier, PDU_Identifier ou StructIdentifier, car ces types peuvent être codés par des règles de codage mais non par codage de champ. */
468 EncodingVariation ::= **$EncodingVariation** EncodingVariationId VariationRef VariationDefault [Comment] **$End_EncodingVariation**
/* SÉMANTIQUE OPÉRATOIRE – Il ne peut y avoir plus d'un identificateur EncodingIdentifier ayant une valeur VariationDefault contenant une expression DefaultExpression dont l'évaluation donne la valeur TRUE. */
469 EncodingVariationId ::= **$EncodingVariationId** EncVariationId&ParList
470 EncVariationId&ParList ::= EncVariationIdentifier [FormalParList]
471 EncVariationIdentifier ::= Identifier
472 VariationRef ::= **$VariationRef** VariationReference
473 VariationReference ::= BoundedFreeText
474 VariationDefault ::= **$VariationDefault** [DefaultExpression]

A.3.3.13.13.3 Définitions de codage de champs non valides

475 InvalidFieldEncodingDefs ::= **$InvalidFieldEncodingDefs** {InvalidFieldEncodingDefOrGroup}+ **$End_InvalidFieldEncodingDefs**
476 InvalidFieldEncodingDefOrGroup ::= InvalidFieldEncodingDef | InvalidFieldEncodingGroup
477 InvalidFieldEncodingGroup ::= **$InvalidFieldEncodingGroup** InvalidFieldEncodingGroupId {InvalidFieldEncodingOrGroup}+ **$End_InvalidFieldEncodingGroup**
478 InvalidFieldEncodingGroupId ::= **$InvalidFieldEncodingGroupId** InvalidFieldEncodingGroupIdentifier
479 InvalidFieldEncodingDef ::= **$Begin_InvalidFieldEncodingDef** InvalidFieldEncodingId [InvalidFieldEncodingGroupRef] Encoding_TypeList [Comment] InvalidFieldEncodingDefinition [Comment] **$End_InvalidFieldEncodingDef**
480 InvalidFieldEncodingId ::= **$InvalidFieldEncodingId** InvalidFieldEncodingId&ParList
481 InvalidFieldEncodingId&ParList ::= InvalidFieldEncodingIdentifier [FormalParList]
482 InvalidFieldEncodingIdentifier ::= Identifier
483 InvalidFieldEncodingGroupRef ::= **$InvalidFieldEncodingGroupRef** InvalidFieldEncodingGroupReference
484 InvalidFieldEncodingGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {InvalidFieldEncodingGroupIdentifier "/"}
485 InvalidFieldEncodingGroupIdentifier ::= Identifier
486 InvalidFieldEncodingDefinition ::= **$InvalidFieldEncodingDefinition** TS_OpProcDef **$End_InvalidFieldEncodingDefinition**
/* SÉMANTIQUE OPÉRATOIRE – La définition TS_OpProcDef doit produire un résultat BitString qui doit être interprété dans le sens où le bit de poids fort du codage est transmis en premier. */

A.3.3.13.14 Définitions des alias (pseudonymes)

487 AliasDefsOrGroup ::= AliasDefs | AliasDefsGroup
488 AliasDefsGroup ::= **$AliasDefsGroup** AliasDefsGroupId {AliasDefsOrGroup}+ **$End_AliasDefsGroup**
489 AliasDefsGroupId ::= **$AliasDefsGroupId** AliasDefsGroupIdentifier
490 AliasDefsGroupIdentifier ::= Identifier
491 AliasDefs ::= **$Begin_AliasDefs** [AliasGroupRef] {[CollComment] AliasDef}+ [Comment] **$End_AliasDefs**
/* NOTE – Les commentaires collectifs peuvent être utilisés dans cette table conformément aux indications de la Figure 2. */
492 AliasGroupRef ::= **$AliasGroupRef** AliasGroupReference
493 AliasGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {AliasGroupIdentifier "/"}
494 AliasGroupIdentifier ::= Identifier
495 AliasDef ::= **$AliasDef** AliasId ExpandedId [Comment] **$End_AliasDef**
496 AliasId ::= **$AliasId** AliasIdentifier

497 AliasIdentifier ::= Identifier
/* SÉMANTIQUE STATIQUE – Un identificateur d'alias (AliasIdentifier) ne sera utilisé que sur une ligne de déclaration d'une description de comportement. */
/* SÉMANTIQUE STATIQUE – L'identificateur AliasIdentifier ne sera utilisé que pour un identificateur ASP_Identifier ou PDU_Identifier valide. */
498 ExpandedId ::= **$ExpandedId** Expansion
499 Expansion ::= ASP_Identifier | PDU_Identifier

A.3.3.14 Partie contraintes

500 ConstraintsPart ::= **$ConstraintsPart** [TS_TypeConstraints] [ASP_Constraints] [PDU_Constraints] [CM_Constraints] **$End_ConstraintsPart**

A.3.3.15 Déclarations de contraintes relatives au type de suite de tests

501 TS_TypeConstraints ::= **$TS_TypeConstraints** [StructTypeConstraints] [ASN1_TypeConstraints] **$End_TS_TypeConstraints**

A.3.3.16 Déclarations de contraintes de type structuré

502 StructTypeConstraints ::= **$StructTypeConstraints** {StructTypeConstraintOrGroup}+ **$End_StructTypeConstraints**
503 StructTypeConstraintOrGroup ::= StructTypeConstraint | StructTypeConstraintGroup
504 StructTypeConstraintGroup ::= **$StructTypeConstraintGroup** StructTypeConstraintGroupId {StructTypeConstraintOrGroup}+ **$End_StructTypeConstraintGroup**
505 StructTypeConstraintGroupId ::= **$StructTypeConstraintGroupId** StructTypeConstraintGroupIdentifier
506 StructTypeConstraint ::= **$Begin_StructTypeConstraint** ConsId [StructTypeConstraintGroupRef] StructId DerivPath [EncVariationId] [Comment] ElemValues [Comment] **$End_StructTypeConstraint**
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur de structure (Struct_Id) ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */
507 StructTypeConstraintGroupRef ::= **$StructTypeConstraintGroupRef** StructTypeConstraintGroupReference
508 StructTypeConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {StructTypeConstraintGroupIdentifier "/"}
509 StructTypeConstraintGroupIdentifier ::= Identifier
510 EncVariationId ::= **$EncVariationId** [EncVariationCall]
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]
512 ElemValues ::= **$ElemValues** {ElemValue}+ **$End_ElemValues**
513 ElemValue ::= **$ElemValue** ElemId ConsValue [PDU_FieldEncoding] [Comment] **$End_ElemValue**
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur d'élément (ElemId) ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Les valeurs d'élément paramétré dans une contrainte de base ne seront ni modifiées ni explicitement omises dans une contrainte modifiée. */
514 PDU_FieldEncoding ::= **$PDU_FieldEncoding** [PDU_FieldEncodingCall]
515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall
516 InvalidFieldEncodingCall ::= InvalidFieldEncodingIdentifier (ActualParList | "(" ")")

A.3.3.17 Déclarations de contraintes de type en notation ASN.1

517 ASN1_TypeConstraints ::= **$ASN1_TypeConstraints** {ASN1_TypeConstraintOrGroup}+ **$End_ASN1_TypeConstraints**
518 ASN1_TypeConstraintOrGroup ::= ASN1_TypeConstraint | ASN1_TypeConstraintGroup
519 ASN1_TypeConstraintGroup ::= **$ASN1_TypeConstraintGroup** ASN1_TypeConstraintGroupId {ASN1_TypeConstraintOrGroup}+ **$End_ASN1_TypeConstraintGroup**
520 ASN1_TypeConstraintGroupId ::= **$ASN1_TypeConstraintGroupId** ASN1_TypeConstraintGroupIdentifier
521 ASN1_TypeConstraint ::= **$Begin_ASN1_TypeConstraint** ConsId [ASN1_TypeConstraintGroupRef] ASN1_TypeId DerivPath [EncVariationId] [Comment] ASN1_ConsValue [Comment] **$End_ASN1_TypeConstraint**
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASN1_TypeId ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */
522 ASN1_TypeConstraintGroupRef ::= **$ASN1_TypeConstraintGroupRef** ASN1_TypeConstraintGroupReference
523 ASN1_TypeConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_TypeConstraintGroupIdentifier "/"}
524 ASN1_TypeConstraintGroupIdentifier ::= Identifier

A.3.3.18 Déclarations des contraintes de primitive ASP

525 ASP_Constraints ::= **$ASP_Constraints** [TTCN_ASP_Constraints] [ASN1_ASP_Constraints] **$End_ASP_Constraints**

A.3.3.19 Déclarations des contraintes de primitive ASP sous forme tabulaire

526 TTCN_ASP_Constraints ::= **$TTCN_ASP_Constraints** {TTCN_ASP_ConstraintOrGroup}+ **$End_TTCN_ASP_Constraints**

527 TTCN_ASP_ConstraintOrGroup ::= TTCN_ASP_Constraint | TTCN_ASP_ConstraintGroup

528 TTCN_ASP_ConstraintGroup ::= **$TTCN_ASP_ConstraintGroup** TTCN_ASP_ConstraintGroupId {TTCN_ASP_ConstraintOrGroup}+ **$End_TTCN_ASP_ConstraintGroup**

529 TTCN_ASP_ConstraintGroupId ::= **$TTCN_ASP_ConstraintGroupId** ASP_ConstraintGroupIdentifier

530 TTCN_ASP_Constraint ::= **$Begin_TTCN_ASP_Constraint** ConsId [ASP_ConstraintGroupRef] ASP_Id DerivPath [Comment] [ASP_ParValues] [Comment] **$End_TTCN_ASP_Constraint**

/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_Id ne sera pas utilisé. */

/* SÉMANTIQUE STATIQUE – Si une primitive ASP comporte une sous-structure, la spécification de contrainte d'une primitive ASP aura la même structure que celle de la définition de type de cette primitive ASP. */

/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */

531 ASP_ConstraintGroupRef ::= **$ASP_ConstraintGroupRef** ASP_ConstraintGroupReference

532 ASP_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASP_ConstraintGroupIdentifier "/"}

533 ASP_ConstraintGroupIdentifier ::= Identifier

534 ASP_ParValues ::= **$ASP_ParValues** {ASP_ParValue} **$End_ASP_ParValues**

535 ASP_ParValue ::= **$ASP_ParValue** ASP_ParId ConsValue [Comment] **$End_ASP_ParValue**

/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_ParId ne sera pas utilisé. */

/* SÉMANTIQUE STATIQUE – Si une définition de primitive ASP fait référence à un type structuré en tant que sous-structure d'un paramètre (c'est-à-dire avec un nom de paramètre), la contrainte correspondante aura le même nom de paramètre dans la position correspondante de la colonne nom du paramètre de la contrainte et la valeur sera une référence à une contrainte s'appliquant à ce paramètre (c'est-à-dire s'appliquant à cette sous-structure conformément à la définition du type structuré). */

/* SÉMANTIQUE STATIQUE – Si une définition de primitive ASP fait référence à un paramètre spécifié comme étant du métatype PDU, alors, dans une contrainte correspondante, la valeur de ce paramètre sera spécifiée comme étant le nom d'une contrainte d'unité PDU ou d'un paramètre formel. */

/* SÉMANTIQUE STATIQUE – Dans une contrainte, on ne pourra utiliser des contraintes structurées par développement de macro que si la définition de primitive ASP correspondante renvoie aussi au même type structuré par développement de macro. */

/* SÉMANTIQUE STATIQUE – Les valeurs de paramètres de primitive ASP paramétrée dans une contrainte de base ne seront ni modifiées ni explicitement omises dans une contrainte modifiée. */

A.3.3.20 Déclarations de contraintes de primitive ASP en notation ASN.1

536 ASN1_ASP_Constraints ::= **$ASN1_ASP_Constraints** {ASN1_ASP_ConstraintOrGroup}+ **$End_ASN1_ASP_Constraints**

537 ASN1_ASP_ConstraintOrGroup ::= ASN1_ASP_Constraint | ASN1_ASP_ConstraintGroup

538 ASN1_ASP_ConstraintGroup ::= **$ASN1_ASP_ConstraintGroup** ASN1_ASP_ConstraintGroupId {ASN1_ASP_ConstraintOrGroup}+ **$End_ASN1_ASP_ConstraintGroup**

539 ASN1_ASP_ConstraintGroupId ::= **$ASN1_ASP_ConstraintGroupId** ASN1_ASP_ConstraintGroupIdentifier

540 ASN1_ASP_Constraint ::= **$Begin_ASN1_ASP_Constraint** ConsId [ASN1_ASP_ConstraintGroupRef] ASP_Id DerivPath [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_ASP_Constraint**

/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_Id ne sera pas utilisé. */

/* SÉMANTIQUE STATIQUE – Si une primitive ASP comporte une sous-structure, les contraintes s'appliquant aux primitives ASP du même type auront une structure ASN.1 compatible (c'est-à-dire comportant éventuellement certains groupements). */

/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que la contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */

541 ASN1_ASP_ConstraintGroupRef ::= **$ASN1_ASP_ConstraintGroupRef** ASN1_ASP_ConstraintGroupReference

542 ASN1_ASP_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_ASP_ConstraintGroupIdentifier "/"}

543 ASN1_ASP_ConstraintGroupIdentifier ::= Identifier

A.3.3.21 Déclarations de contraintes d'unité PDU

544 PDU_Constraints ::= **$PDU_Constraints** [TTCN_PDU_Constraints] [ASN1_PDU_Constraints] **$End_PDU_Constraints**

A.3.3.22 Déclarations de contraintes d'unité PDU en notation tabulaire

545 TTCN_PDU_Constraints ::= **$TTCN_PDU_Constraints** {TTCN_PDU_ConstraintOrGroup}+
$End_TTCN_PDU_Constraints

546 TTCN_PDU_ConstraintOrGroup ::= TTCN_PDU_Constraint | TTCN_PDU_ConstraintGroup

547 TTCN_PDU_ConstraintGroup ::= **$TTCN_PDU_ConstraintGroup** TTCN_PDU_ConstraintGroupId
{TTCN_PDU_ConstraintOrGroup}+ **$End_TTCN_PDU_ConstraintGroup**

548 TTCN_PDU_ConstraintGroupId ::= **$TTCN_PDU_ConstraintGroupId** PDU_ConstraintGroupIdentifier

549 TTCN_PDU_Constraint ::= **$Begin_TTCN_PDU_Constraint** ConsId [PDU_ConstraintGroupRef] PDU_Id DerivPath
[EncRuleId] [EncVariationId] [Comment] [PDU_FieldValues] [Comment] **$End_TTCN_PDU_Constraint**
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_Id ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Si une unité PDU comporte une sous-structure, les contraintes s'appliquant aux unités PDU du même type auront la même structure. */
/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que la contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */

550 PDU_ConstraintGroupRef ::= **$PDU_ConstraintGroupRef** PDU_ConstraintGroupReference

551 PDU_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {PDU_ConstraintGroupIdentifier "/"}

552 PDU_ConstraintGroupIdentifier ::= Identifier

553 EncRuleId ::= **$EncRuleId** [EncodingRuleIdentifier]

554 ConsId ::= **$ConsId** ConsId&ParList

555 ConsId&ParList ::= ConstraintIdentifier [FormalParList]

556 ConstraintIdentifier ::= Identifier

557 DerivPath ::= **$DerivPath** [DerivationPath]

558 DerivationPath ::= {ConstraintIdentifier Dot}+
/* SÉMANTIQUE STATIQUE – Si une définition de contrainte est une modification d'une contrainte existante, le nom de cette dernière figurera dans l'entrée du chemin de dérivation de la table. */
/* SÉMANTIQUE STATIQUE – Le premier identificateur de contrainte (ConstraintIdentifier) dans le chemin de dérivation (DerivationPath) doit être un identificateur de contrainte de base. */
/* SÉMANTIQUE STATIQUE – Le chemin de dérivation sera la liste complète des contraintes présentées dans l'ordre d'application des modifications qu'elles apportent à la contrainte de base. */
/* SÉMANTIQUE STATIQUE – Il n'y aura pas de blanc entre l'identificateur ConstraintIdentifier et la production Dot. */

559 PDU_FieldValues ::= **$PDU_FieldValues** {PDU_FieldValue} **$End_PDU_FieldValues**

560 PDU_FieldValue ::= **$PDU_FieldValue** PDU_FieldId ConsValue [PDU_FieldEncoding] [Comment]
$End_PDU_FieldValue
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_FieldId ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Si une définition d'unité PDU fait référence à un type structuré en tant que sous-structure d'un champ (c'est-à-dire avec un nom de champ), la contrainte correspondante aura le même nom de champ dans la position correspondante de la colonne nom du champ de la contrainte et la valeur sera une référence à une contrainte s'appliquant à ce champ (c'est-à-dire s'appliquant à cette sous-structure conformément à la définition du type structuré). */
/* SÉMANTIQUE STATIQUE – Si une définition d'unité PDU fait référence à un champ spécifié comme étant du métatype PDU, alors, dans une contrainte correspondante, la valeur de ce champ sera spécifiée comme étant le nom d'une contrainte d'unité PDU ou d'un paramètre formel. */
/* SÉMANTIQUE STATIQUE – On ne pourra utiliser dans une contrainte des contraintes structurées par développement de macro que si la définition d'unité PDU correspondante renvoie aussi au même type structuré par développement de macro. */
/* SÉMANTIQUE STATIQUE – Les valeurs de champ d'unité PDU paramétrée dans une contrainte de base ne seront ni modifiées, ni explicitement omises dans une contrainte modifiée. */

561 ConsValue ::= **$ConsValue** ConstraintValue&Attributes
/* SÉMANTIQUE OPÉRATOIRE – ConsValue prendra pour valeur un élément du type spécifié pour le paramètre de primitive ASP, pour le champ d'unité PDU ou pour l'élément de structure. Ces valeurs peuvent inclure des spécificateurs de concordance compatibles avec le type spécifié. */

562 ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
/* NOTE – Dans la syntaxe de la notation ASN.1, la valeur DefinedValue permet d'accéder aux attributs et valeurs d'une contrainte (ConstraintValue&Attributes). Voir la référence à la production 739 pour les valeurs. */
/* SÉMANTIQUE STATIQUE – La valeur de contrainte ConstraintValue respectera toutes les restrictions définies pour le paramètre de primitive ASP, pour le champ d'unité PDU ou pour le type d'élément de structure, notamment les plages de valeurs, les listes de valeurs, les restrictions alphabétiques et les restrictions de longueur, de même que les restrictions définies par l'attribut de valeur ValueAttributes. */
/* SÉMANTIQUE OPÉRATOIRE – Aucune spécification de longueur définie pour le paramètre de primitive ASP ou pour le type de champ d'unité PDU dans les déclarations de type de suite de tests ne doit contrevenir aux spécifications de longueur figurant dans la définition de type de primitive ASP ou d'unité PDU. */
/* SÉMANTIQUE STATIQUE – Ni les variables de suite de tests ni celles de tests élémentaires ne doivent être utilisées dans les contraintes, à moins d'être passées en tant que paramètres effectifs, auquel cas une valeur leur sera attribuée et elles ne seront plus modifiées. */

563 ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
/* SÉMANTIQUE STATIQUE – Lorsque l'expression ConstraintExpression est utilisée dans une contrainte, ses termes ne doivent pas contenir les identificateurs TS_VarIdentifier ou TC_VarIdentifier. */

564 ConstraintExpression ::= Expression
/* SÉMANTIQUE OPÉRATOIRE – L'expression ConstraintExpression prendra pour valeur un élément du type spécifié. */

565 MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange | SuperSet | SubSet | Permutation
/* NOTE – Aucun spécificateur de concordance n'est considéré comme valeur spécifique. */

566 Complement ::= **COMPLEMENT** ValueList

567 Omit ::= Dash | **OMIT**
/* SÉMANTIQUE STATIQUE – Dans les contraintes en notation ASN.1, l'attribut Omit (omission) ne sera utilisé que pour les paramètres de primitive ASP et champs d'unité PDU déclarés être de type OPTIONAL (facultatifs) ou DEFAULT (par défaut). */

568 AnyValue ::= "?"

569 AnyOrOmit ::= "*"

570 ValueList ::= "(" ConstraintValue&Attributes {Comma ConstraintValue&Attributes} ")"
/* SÉMANTIQUE STATIQUE – Chaque attribut ConstraintValue&Attributes (attributs et valeur de contrainte) sera du type déclaré pour le paramètre de primitive ASP, pour le champ d'unité PDU ou pour l'élément de structure dans lequel la liste de valeurs ValueList est utilisée. */

571 ValueRange ::= "(" ValRange ")"
/* SÉMANTIQUE STATIQUE – L'attribut d'intervalle de valeurs (ValueRange) ne sera utilisé que pour les paramètres de primitive ASP, les champs d'unité PDU et les éléments de structure de type INTEGER (entier). */
/* SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par l'intervalle ValueRange sera un sous-ensemble vrai des valeurs autorisées par le type déclaré du paramètre de primitive ASP, du champ d'unité PDU ou de l'élément de structure. */

572 ValRange ::= (LowerRangeBound To UpperRangeBound)
/* SÉMANTIQUE OPÉRATOIRE – La limite inférieure d'intervalle (LowerRangeBound) sera inférieure à la limite supérieure (UpperRangeBound). */

573 LowerRangeBound ::= ConstraintExpression | Minus **INFINITY**
/* SÉMANTIQUE OPÉRATOIRE – L'expression ConstraintExpression prendra une valeur de type INTEGER (entier). */

574 UpperRangeBound ::= ConstraintExpression | **INFINITY**
/* SÉMANTIQUE OPÉRATOIRE – L'expression ConstraintExpression prendra une valeur de type INTEGER (entier). */

575 SuperSet ::= **SUPERSET** "(" ConstraintValue&Attributes ")"
/* SÉMANTIQUE STATIQUE – L'argument de l'attribut SuperSet (surensemble), c'est-à-dire ConstraintValue&Attributes, sera du type SET OF (ensemble de). */

576 SubSet ::= **SUBSET** "(" ConstraintValue&Attributes ")"
/* SÉMANTIQUE STATIQUE – L'argument de l'attribut SubSet (sous-ensemble), c'est-à-dire ConstraintValue&Attributes, sera du type SET OF. */

577 Permutation ::= **PERMUTATION** ValueList
/* SÉMANTIQUE STATIQUE – Dans les contraintes en notation ASN.1, la déclaration IF_PRESENT ne sera utilisée que pour les paramètres de primitive ASP et les champs d'unité PDU qui ont été déclarés OPTIONAL. */
/* SÉMANTIQUE STATIQUE – L'attribut Permutation ne sera utilisé qu'à l'intérieur d'une valeur de type SEQUENCE OF (séquence de). */
/* SÉMANTIQUE STATIQUE – La liste ValueList sera du type spécifié par SEQUENCE OF. */

578 ValueAttributes ::= [ValueLength] [**IF_PRESENT**] [ASN1_Encoding]
/* SÉMANTIQUE STATIQUE – Dans les contraintes ASN.1, la déclaration IF_PRESENT ne sera utilisée que pour les paramètres de primitive ASP et champs d'unité PDU déclarés de type OPTIONAL ou DEFAULT. */
/* SÉMANTIQUE STATIQUE – Le codage en notation ASN.1 (ASN1_Encoding) ne sera utilisé que pour les attributs de valeur dans les contraintes de type en notation ASN.1 et les contraintes d'unité PDU en notation ASN.1. */

579 ASN1_Encoding ::= **ENC** PDU_FieldEncodingCall

580 ValueLength ::= SingleValueLength | RangeValueLength
/* SÉMANTIQUE STATIQUE – L'attribut de longueur ValueLength ne sera utilisé que pour les paramètres de primitive ASP, les champs d'unité PDU et les éléments de structure qui sont déclarés BITSTRING, HEXSTRING, OCTETSTRING, CharacterString, SEQUENCE OF ou SET OF. */
/* SÉMANTIQUE STATIQUE – L'attribut ValueLength ne sera utilisé qu'en combinaison avec les mécanismes suivants: SpecificValue (valeur spécifique), Complement (complément), Omit (omission), AnyValue (valeur quelconque), AnyOrOmit (quelconque ou omission), AnyOrNone (quelconque ou aucun) et Permutation (permutation). */
/* SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par l'attribut ValueLength sera un sous-ensemble vrai des valeurs autorisées par le type déclaré des paramètres de primitive ASP, des champs d'unité PDU et des éléments de structure. */

581 SingleValueLength ::= "[" ValueBound "]"

582 ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
/* SÉMANTIQUE OPÉRATOIRE – L'attribut ValueBound (limite de valeur) prendra une valeur de type INTEGER (entier) non négative. */

583 RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
/* SÉMANTIQUE OPÉRATOIRE – La limite inférieure (LowerValueBound) sera inférieure à la limite supérieure (UpperValueBound). */

584 LowerValueBound ::= ValueBound

585 UpperValueBound ::= ValueBound | **INFINITY**

A.3.3.23 Déclarations de contraintes d'unité PDU en notation ASN.1

586 ASN1_PDU_Constraints ::= **$ASN1_PDU_Constraints** {ASN1_PDU_ConstraintOrGroup}+ **$End_ASN1_PDU_Constraints**
587 ASN1_PDU_ConstraintOrGroup ::= ASN1_PDU_Constraint | ASN1_PDU_ConstraintGroup
588 ASN1_PDU_ConstraintGroup ::= **$ASN1_PDU_ConstraintGroup** ASN1_PDU_ConstraintGroupId {ASN1_PDU_ConstraintOrGroup}+ **$End_ASN1_PDU_ConstraintGroup**
589 ASN1_PDU_ConstraintGroupId ::= **$ASN1_PDU_ConstraintGroupId** ASN1_PDU_ConstraintGroupIdentifier
590 ASN1_PDU_Constraint ::= **$Begin_ASN1_PDU_Constraint** ConsId [ASN1_PDU_ConstraintGroupRef] PDU_Id DerivPath [EncRuleId] [EncVariationId] [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_PDU_Constraint**
/* SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_Id ne sera pas utilisé. */
/* SÉMANTIQUE STATIQUE – Si une unité PDU comporte une sous-structure, les contraintes s'appliquant à ces unités PDU auront une structure ASN.1 compatible (c'est-à-dire comportant éventuellement certains groupements). */
/* SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste. */
591 ASN1_PDU_ConstraintGroupRef ::= **$ASN1_PDU_ConstraintGroupRef** ASN1_PDU_ConstraintGroupReference
592 ASN1_PDU_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_PDU_ConstraintGroupIdentifier "/"}
593 ASN1_PDU_ConstraintGroupIdentifier ::= Identifier
594 ASN1_ConsValue ::= **$ASN1_ConsValue** ConstraintValue&AttributesOrReplace **$End_ASN1_ConsValue**
595 ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement {Comma Replacement}
596 Replacement ::= **REPLACE** ReferenceList **BY** ConstraintValue&Attributes | **OMIT** ReferenceList
/* SÉMANTIQUE STATIQUE – L'attribut Replacement (remplacement) ne sera utilisé que si l'attribut DerivPath est spécifié. */
/* SÉMANTIQUE STATIQUE – Les valeurs Replacement paramétrées dans une contrainte de base ne seront ni modifiées ni explicitement omises dans une contrainte modifiée. */
597 ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}

A.3.3.24 Déclarations de contraintes de message CM

598 CM_Constraints ::= **$CM_Constraints** [TTCN_CM_Constraints] [ASN1_CM_Constraints] **$End_CM_Constraints**

A.3.3.25 Déclarations de contraintes de message CM en notation tabulaire

599 TTCN_CM_Constraints ::= **$TTCN_CM_Constraints** {TTCN_CM_ConstraintOrGroup}+ **$End_TTCN_CM_Constraints**
600 TTCN_CM_ConstraintOrGroup ::= TTCN_CM_Constraint | TTCN_CM_ConstraintGroup
601 TTCN_CM_ConstraintGroup ::= **$TTCN_CM_ConstraintGroup** TTCN_CM_ConstraintGroupId {TTCN_CM_ConstraintOrGroup}+ **$End_TTCN_CM_ConstraintGroup**
602 TTCN_CM_ConstraintGroupId ::= **$TTCN_CM_ConstraintGroupId** CM_ConstraintGroupIdentifier
603 TTCN_CM_Constraint ::= **$Begin_TTCN_CM_Constraint** ConsId [CM_ConstraintGroupRef] CM_Id DerivPath [Comment] [CM_ParValues] [Comment] **$End_TTCN_CM_Constraint**
604 CM_ConstraintGroupRef ::= **$CM_ConstraintGroupRef** CM_ConstraintGroupReference
605 CM_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {CM_ConstraintGroupIdentifier "/"}
606 CM_ConstraintGroupIdentifier ::= Identifier
607 CM_ParValues ::= **$CM_ParValues** {CM_ParValue} **$End_CM_ParValues**
608 CM_ParValue ::= **$CM_ParValue** CM_ParId ConsValue [Comment] **$End_CM_ParValue**

A.3.3.26 Déclarations de contraintes de message CM en notation ASN.1

609 ASN1_CM_Constraints ::= **$ASN1_CM_Constraints** {ASN1_CM_ConstraintOrGroup}+ **$End_ASN1_CM_Constraints**
610 ASN1_CM_ConstraintOrGroup ::= ASN1_CM_Constraint | ASN1_CM_ConstraintGroup
611 ASN1_CM_ConstraintGroup ::= **$ASN1_CM_ConstraintGroup** ASN1_CM_ConstraintGroupId {ASN1_CM_ConstraintOrGroup}+ **$End_ASN1_CM_ConstraintGroup**
612 ASN1_CM_ConstraintGroupId ::= **$ASN1_CM_ConstraintGroupId** ASN1_CM_ConstraintGroupIdentifier
613 ASN1_CM_Constraint ::= **$Begin_ASN1_CM_Constraint** ConsId [ASN1_CM_ConstraintGroupRef] CM_Id DerivPath [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_CM_Constraint**
614 ASN1_CM_ConstraintGroupRef ::= **$ASN1_CM_ConstraintGroupRef** ASN1_CM_ConstraintGroupReference
615 ASN1_CM_ConstraintGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_CM_ConstraintGroupIdentifier "/"}
616 ASN1_CM_ConstraintGroupIdentifier ::= Identifier

A.3.3.27 Partie dynamique

617 DynamicPart ::= **$DynamicPart** [TestCases] [TestStepLibrary] [DefaultsLibrary] **$End_DynamicPart**

A.3.3.28 Tests élémentaires

618 TestCases ::= **$TestCases** {TestGroup | TestCase}+ **$End_TestCases**
619 TestGroup ::= **$TestGroup** TestGroupId {TestGroup | TestCase}+ **$End_TestGroup**
620 TestGroupId ::= **$TestGroupId** TestGroupIdentifier
621 TestGroupIdentifier ::= Identifier
622 TestCase ::= **$Begin_TestCase** TestCaseId TestGroupRef TestPurpose [Configuration] DefaultsRef [Comment] BehaviourDescription [Comment] **$End_TestCase**
623 TestCaseId ::= **$TestCaseId** TestCaseIdentifier
624 TestCaseIdentifier ::= Identifier
625 TestGroupRef ::= **$TestGroupRef** TestGroupReference
626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
/* SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/". */
627 TestPurpose ::= **$TestPurpose** BoundedFreeText
628 Configuration ::= **$Configuration** TCompConfigIdentifier
629 DefaultsRef::= **$DefaultsRef** [DefaultRefList]
630 DefaultRefList ::= DefaultReference {Comma DefaultReference}
631 DefaultReference ::= DefaultIdentifier [ActualParList]

A.3.3.29 Bibliothèque des modules de test

632 TestStepLibrary ::= **$TestStepLibrary** {TestStepGroup | TestStep}+ **$End_TestStepLibrary**
633 TestStepGroup ::= **$TestStepGroup** TestStepGroupId {TestStepGroup | TestStep}+ **$End_TestStepGroup**
634 TestStepGroupId ::= **$TestStepGroupId** TestStepGroupIdentifier
635 TestStepGroupIdentifier ::= Identifier
636 TestStep ::= **$Begin_TestStep** TestStepId TestStepRef Objective DefaultsRef [Comment] BehaviourDescription [Comment] **$End_TestStep**
637 TestStepId ::= **$TestStepId** TestStepId&ParList
638 TestStepId&ParList ::= TestStepIdentifier [FormalParList]
639 TestStepIdentifier ::= Identifier
640 TestStepRef ::= **$TestStepRef** TestStepGroupReference
641 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
/* SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/". */
642 Objective ::= **$Objective** BoundedFreeText

A.3.3.30 Bibliothèque des comportements par défaut

643 DefaultsLibrary ::= **$DefaultsLibrary** {DefaultGroup | Default}+ **$End_DefaultsLibrary**
644 DefaultGroup ::= **$DefaultGroup** DefaultGroupId {DefaultGroup | Default}+ **$End_DefaultGroup**
645 DefaultGroupId ::= **$DefaultGroupId** DefaultGroupIdentifier
646 Default ::= **$Begin_Default** DefaultId DefaultRef Objective [Comment] BehaviourDescription [Comment] **$End_Default**
/* SÉMANTIQUE STATIQUE – La description BehaviourDescription n'utilisera pas le rattachement à un arbre, sauf à des arbres locaux (c'est-à-dire que les arbres de comportement par défaut ne rattacheront pas de modules de test). */
647 DefaultRef ::= **$DefaultRef** DefaultGroupReference
648 DefaultId ::= **$DefaultId** DefaultId&ParList
649 DefaultId&ParList ::= DefaultIdentifier [FormalParList]
650 DefaultIdentifier ::= Identifier
651 DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}
/* SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/". */
652 DefaultGroupIdentifier ::= Identifier

A.3.3.31 Descriptions de comportement

653 BehaviourDescription ::= **$BehaviourDescription** RootTree {LocalTree} **$End_BehaviourDescription**
654 RootTree ::= {BehaviourLine}+
655 LocalTree ::= Header {BehaviourLine}+
656 Header ::= **$Header** TreeHeader
657 TreeHeader ::= TreeIdentifier [FormalParList]
658 TreeIdentifier ::= Identifier
659 FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
660 FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
661 FormalParIdentifier ::= Identifier
662 FormalParType ::= Type | PCO_TypeIdentifier | **PDU** | **CP** | **TIMER**
/* SÉMANTIQUE STATIQUE – Dans une opération de suite de tests ou dans une opération de codage, FormalParType ne doit pas être un type de point PCO ou le mot clé CP. */
/* SÉMANTIQUE STATIQUE – Si un paramètre formel est de type **PDU**, ce paramètre formel ne doit pas être mentionné avec une référence à une composante (c'est-à-dire que l'on ne peut faire référence à des champs spécifiés de l'unité PDU). */

A.3.3.32 Lignes de comportement

663 BehaviourLine ::= **$BehaviourLine** LabelId Line Cref VerdictId [Comment] **$End_BehaviourLine**

664 Line ::= **$Line** Indentation StatementLine

665 Indentation ::= "[" Number "]"

/* SÉMANTIQUE STATIQUE – Les déclarations du premier niveau d'options multiples dans une description comportementale auront la valeur d'indentation zéro. */

/* SÉMANTIQUE STATIQUE – Les déclarations ayant un antécédent auront comme valeur d'indentation celle de l'option du niveau précédent plus un. */

666 LabelId ::= **$LabelId** [Label]

667 Label ::= Identifier

668 Cref ::= **$Cref** [ConstraintReference]

669 ConstraintReference ::= ConsRef | FormalParIdentifier | AnyValue

/* SÉMANTIQUE STATIQUE – La présence de l'attribut ConsRef sera concomitante des déclarations SEND (envoi), IMPLICIT SEND (envoi implicite) et RECEIVE (réception) et son type correspondra (sera le même ou sera un sous-ensemble) au type de la primitive ASP, de l'unité PDU ou du message CM spécifié dans la déclaration SEND, IMPLICIT_SEND ou RECEIVE. L'attribut ConstraintReference (référence à une contrainte) n'est pas nécessaire aux primitives ASP et aux messages CM dépourvus de paramètre, et aux unités PDU dépourvues de champ. Il n'apparaîtra dans aucun autre type de déclaration TTCN. */

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier renverra un attribut ConsRef. */

/* SÉMANTIQUE STATIQUE – L'attribut ConstraintReference donné en référence d'un événement SEND (envoi) ne contiendra pas de spécificateurs de concordance, sauf omit, sauf si ceux-ci sont explicitement affectés dans la ligne d'événement SEND. */

670 ConsRef ::= ConstraintIdentifier [ActualCrefParList]

671 ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"

/* SÉMANTIQUE STATIQUE – Voir la sémantique statique relative à la production 699. */

672 ActualCrefPar ::= Value

/* NOTE – Le paramètre Value permet d'accéder au spécificateur de concordance, aux identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, FormalParIdentifier ou ConsRef. */

673 VerdictId ::= **$VerdictId** [Verdict]

674 Verdict ::= Pass | Fail | Inconclusive | Result

/* SÉMANTIQUE STATIQUE – L'attribut Verdict n'apparaîtra pas dans les entrées des types suivants de l'arbre de comportement: entrées vides, constructions ATTACH, constructions REPEAT, constructions GOTO, déclarations IMPLICIT SEND ou RETURN. */

675 Pass ::= **PASS** | **P** | "(" **PASS** ")" | "(" **P** ")"

676 Fail ::= **FAIL** | **F** | "(" **FAIL** ")" | "(" **F** ")"

677 Inconclusive ::= **INCONC** | **I** | "(" **INCONC** ")" | "(" **I** ")"

678 Result ::= **R**

/* SÉMANTIQUE STATIQUE – L'identificateur R ne sera jamais utilisé dans le membre gauche d'une affectation. */

A.3.3.33 Déclarations en notation TTCN

679 StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList] [TimerOps]) | (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend

680 Event ::= Send | Receive | Otherwise | Timeout | Done

/* SÉMANTIQUE STATIQUE – Un événement Receive (réception), Otherwise (sinon) ou Timeout (fin de temporisation) ne peut être suivi que par d'autres événements Receive, Otherwise et Timeout dans l'ensemble des options de l'arbre complètement développé. En conséquence, les arbres par défaut ne comporteront que des événements Receive, Otherwise et Timeout au premier niveau d'indentation des options. */

681 Qualifier ::= "[" Expression "]"

/* SÉMANTIQUE OPÉRATOIRE – L'attribut Qualifier (qualificateur) aura une valeur de type BOOLEAN. */

682 Send ::= [PCO_Identifier | CP_Identifier | FormalParIdentifier] "!" (ASP_Identifier | PDU_Identifier | CM_Identifier)

/* SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier, CP_Identifier ou FormalParIdentifier ne seront présents que si la suite de tests utilise un seul point PCO et aucun point CP. */

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier renverra à un identificateur PCO_Identifier ou à un identificateur CP_Identifier. */

/* SÉMANTIQUE STATIQUE – Seuls les messages CM peuvent être échangés aux points CP, et seules les primitives ASP et les unités PDU peuvent être échangées aux points PCO. */

683 ImplicitSend ::= "<" **IUT** "!" (ASP_Identifier | PDU_Identifier) ">"

/* SÉMANTIQUE STATIQUE – L'attribut ImplicitSend ne sera utilisé que pour les méthodes de test à distance. */

684 Receive ::= [PCO_Identifier | CP_Identifier | FormalParIdentifier] "?" (ASP_Identifier | PDU_Identifier | CM_Identifier)

/* SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier, CP_Identifier ou FormalParIdentifier ne seront présents que si la suite de tests utilise un seul point PCO et aucun point CP. */

/* SÉMANTIQUE STATIQUE – Seuls les messages CM peuvent être échangés aux points CP, et seules les primitives ASP et les unités PDU peuvent être échangées aux points PCO. */

685 Otherwise ::= [PCO_Identifier | CP_Identifier | FormalParIdentifier] "?" **OTHERWISE**

/* SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier, CP_Identifier ou FormalParIdentifier ne seront présents que si la suite de tests utilise un seul point PCO et aucun point CP. */

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier ne peut être que de type point PCO ou point CP. */

686 Timeout ::= "?" **TIMEOUT** [TimerIdentifier | FormalParIdentifier]

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier ne peut être que de type TIMER (temporisateur). */

687 Done ::= "?" **DONE** "(" [TCompIdList] ")"

688 TCompIdList ::= TCompIdentifier {Comma TCompIdentifier}

689 Construct ::= GoTo | Attach | Repeat | Return | Activate | Create

690 Activate ::= **ACTIVATE** "(" [DefaultRefList] ")"

/* SÉMANTIQUE STATIQUE – La construction ACTIVATE ne peut être utilisée dans les tables de comportement par défaut. */

691 Return ::= **RETURN**

/* SÉMANTIQUE STATIQUE – La construction RETURN ne peut être utilisée, sauf dans les arbres de comportement par défaut (y compris les arbres locaux des tables de comportement par défaut). */

692 Create ::= **CREATE** "(" CreateList ")"

693 CreateList ::= CreateTComp {Comma CreateTComp}

694 CreateTComp ::= TCompIdentifier Colon TreeReference [ActualParList]

/* SÉMANTIQUE STATIQUE – Le rôle de l'identificateur TCompIdentifier ne sera pas un rôle MTC. */

695 GoTo ::= ("->" | **GOTO**) Label

/* SÉMANTIQUE STATIQUE – La colonne des étiquettes ne contiendra que des étiquettes mentionnées dans des déclarations GoTo. */

/* SÉMANTIQUE STATIQUE – L'étiquette Label sera associée à la première option d'un ensemble d'options multiples dont l'une constitue un nœud antécédent du point à partir duquel le renvoi GoTo intervient. */

/* SÉMANTIQUE STATIQUE – Le renvoi GoTo ne sera utilisé qu'à l'intérieur d'un même arbre, c'est-à-dire dans un arbre racine de test élémentaire, dans un arbre de module de test, dans un arbre par défaut ou dans un arbre local; par conséquent, chaque étiquette d'une construction GoTo se trouve dans l'arbre qui contient cette construction. */

/* SÉMANTIQUE STATIQUE – Aucune opération ACTIVATE ne peut être un nœud antécédent d'une construction GoTo sur la branche d'un arbre située entre une étiquette et la déclaration GoTo. */

/* SÉMANTIQUE STATIQUE – Aucune déclaration GoTo ne renverra au premier niveau d'option dans les arbres locaux, les modules de test et les arbres par défaut. */

696 Attach ::= "+" TreeReference [ActualParList]

/* SÉMANTIQUE STATIQUE – Une référence TreeReference ne se rattachera jamais à elle-même, ni directement ni indirectement, à son niveau supérieur d'indentation. */

/* SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que lenombre de paramètres formels. */

/* SÉMANTIQUE STATIQUE – Les paramètres formels et effectifs d'un module de test ne doivent servir qu'à réaliser une notation TTCN valide par substitution de texte. */

/* SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, ConsRef, MatchingSymbol, FormalParIdentifier, PCO_Identifier, CP_Identifier et les spécificateurs de concordance peuvent être transférés en tant que paramètres effectifs vers un arbre rattaché. */

697 Repeat ::= **REPEAT** TreeReference [ActualParList] **UNTIL** Qualifier

/* SÉMANTIQUE STATIQUE – Une référence TreeReference ne se rattachera jamais à elle-même, ni directement ni indirectement, à son niveau supérieur d'indentation. */

/* SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que le nombre de paramètres formels. */

/* SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, ConsRef, MatchingSymbol, FormalParIdentifier et les points PCO et CP peuvent être transférés vers l'arbre en tant que paramètres effectifs dans une déclaration REPEAT. */

698 TreeReference ::= TestStepIdentifier | TreeIdentifier

/* SÉMANTIQUE STATIQUE – L'identificateur d'arbre (TreeIdentifier) désignera un des arbres de la description comportementale courante, c'est-à-dire que les arbres locaux ne seront pas accessibles depuis l'extérieur de la description comportementale dans laquelle ils sont spécifiés. */

699 ActualParList ::= "(" ActualPar {Comma ActualPar} ")"

/* SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que le nombre de paramètres formels. */

/* SÉMANTIQUE OPÉRATOIRE – Chaque paramètre effectif prendra une valeur compatible avec le type du paramètre formel correspondant, ou dans le cas d'opérations prédéfinies, compatible avec les types pour lesquels l'opération est définie. */

/* SÉMANTIQUE STATIQUE – Si un paramètre est une contrainte paramétrée, cette contrainte sera transférée avec sa liste de paramètres effectifs. */

/* SÉMANTIQUE STATIQUE – Les paramètres effectifs seront liés à une valeur. */

/* SÉMANTIQUE STATIQUE – Si le type de paramètre formel est PDU, le paramètre effectif sera alors déclaré de type PDU ou de type **PDU** spécifique. */

700 ActualPar ::= Value | PCO_Identifier | CP_Identifier | TimerIdentifier

/* NOTE – Le paramètre Value permet d'accéder au spécificateur de concordance, aux identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, FormalParIdentifier ou ConsRef. */

A.3.3.34 Expressions

701 AssignmentList ::= "(" Assignment {Comma Assignment} ")"

702 Assignment ::= DataObjectReference ":=" Expression

/* SÉMANTIQUE STATIQUE – Sauf dans une définition de procédure ou une définition du codage, le membre gauche de l'affectation renverra toujours à un identificateur TS_VarIdentifier ou TC_VarIdentifier, à une référence à un champ de variable ou à une référence à un paramètre de primitive ASP ou de champ d'unité PDU à transmettre. */

/* SÉMANTIQUE STATIQUE – Dans la définition d'une procédure d'une opération TSOp ou EncodingOp, l'identificateur de l'objet de données apparaissant dans le membre gauche de l'affectation sera un identificateur VarIdentifier. */

/* SÉMANTIQUE STATIQUE – Une expression ne contiendra pas de variables non liées à une valeur. */

/* SÉMANTIQUE OPÉRATOIRE – L'expression du membre droit de l'affectation prendra une valeur explicite du même type que le membre gauche. */

703 Expression ::= SimpleExpression [RelOp SimpleExpression]

/* SÉMANTIQUE OPÉRATOIRE – S'il existe deux expressions simples (SimpleExpressions) liées par un opérateur relationnel RelOp, ces expressions prendront des valeurs spécifiques de types compatibles. */

/* SÉMANTIQUE OPÉRATOIRE – Si l'opérateur RelOp est "<" | ">" | ">=" | "<=", chaque expression simple prendra une valeur spécifique de type INTEGER (entier). */

/* SÉMANTIQUE STATIQUE – Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans les expressions arithmétiques. */

704 SimpleExpression ::= Term {AddOp Term}

/* SÉMANTIQUE OPÉRATOIRE – Chaque terme prendra une valeur spécifique. Si plus d'un terme existe et si l'opérateur additif AddOp est "OR" (ou), les termes seront du type BOOLEAN; si l'opérateur AddOp est "+" ou "–", les termes seront du type INTEGER. */

705 Term ::= Factor {MultiplyOp Factor}

/* SÉMANTIQUE OPÉRATOIRE – Chaque facteur prendra une valeur spécifique. Si plus d'un facteur existe et si l'opérateur multiplicatif MultiplyOp est "AND" (et), les facteurs seront du type BOOLEAN; si l'opérateur MultiplyOp est "*" ou "/", les facteurs seront du type INTEGER. */

706 Factor ::= [UnaryOp] Primary

/* SÉMANTIQUE OPÉRATOIRE – L'élément Primary (primaire) prendra une valeur spécifique. Si l'opérateur unaire UnaryOp existe et est "NOT" (non), l'élément primaire sera du type BOOLEAN; si l'opérateur UnaryOp est "+" ou "–", l'élément primaire sera du type INTEGER. */

707 Primary ::= Value | DataObjectReference | OpCall | SelectExprIdentifier | "(" Expression ")"

/* SÉMANTIQUE STATIQUE – L'identificateur d'expression de sélection (SelectExprIdentifier) ne sera utilisé que dans des expressions de sélection. */

/* NOTE – Le paramètre Value permet d'accéder au spécificateur de concordance, aux identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, FormalParIdentifier ou ConsRef. */

708 DataObjectReference ::= DataObjectIdentifier {ComponentReference}

/* SÉMANTIQUE STATIQUE – Les identificateurs de paramètres de primitive ASP et de champs d'unité PDU associés aux événements SEND et RECEIVE ne seront utilisés que pour renvoyer aux valeurs de ces paramètres et champs sur la ligne même de la déclaration. */

/* SÉMANTIQUE STATIQUE – Chacune des références ComponentReference désignera un paramètre de primitive ASP, un champ d'unité PDU, un élément de structure ou une valeur ASN.1 déclarée explicitement dans l'objet qui la précède immédiatement dans le membre droit de l'affectation. */

/* SÉMANTIQUE STATIQUE – L'identificateur DataObjectIdentifier (objet de données) ne peut être un identificateur VarIdentifier, sauf dans une définition de la procédure d'une opération TestSuiteOperation ou EncodingOperation. */

709 DataObjectIdentifier ::= TS_ParIdentifier | TS_ConstIdentifier | TS_VarIdentifier | TC_VarIdentifier | FormalParIdentifier | ASP_Identifier | PDU_Identifier | CM_Identifier | VarIdentifier

710 ComponentReference ::= RecordRef | ArrayRef | BitRef

/* SÉMANTIQUE STATIQUE – La référence d'enregistrement RecordRef sera utilisée pour désigner les composantes des types SEQUENCE (séquence), SET (ensemble) et CHOICE (choix) à l'exclusion de tout autre type ASN.1. */

/* SÉMANTIQUE STATIQUE – RecordRef sera utilisée pour faire référence aux paramètres de primitive ASP, aux champs d'unité PDU et aux éléments de structure sous forme tabulaire. */

/* SÉMANTIQUE STATIQUE – La référence de tableau ArrayRef sera utilisée pour désigner les composantes des types SEQUENCE OF (séquence de) et SET OF (ensemble de) à l'exclusion de tout autre type ASN.1. */

711 RecordRef ::= Dot (ComponentIdentifier | ComponentPosition)

/* SÉMANTIQUE STATIQUE – La forme RecordRef avec utilisation de l'identificateur ComponentIdentifier sera toujours utilisée pour faire référence aux composantes des types SEQUENCE, SET et CHOICE de l'ASN.1 quand un identificateur est déclaré pour cette composante. */

/* SÉMANTIQUE STATIQUE – La forme du type de RecordRef avec utilisation de l'identificateur ComponentIdentifier sera toujours utilisée pour faire référence aux paramètres de primitive ASP, aux champs d'unité PDU et aux éléments de structure déclarés sous forme tabulaire. */

/* SÉMANTIQUE STATIQUE – La forme de RecordRef avec utilisation de l'attribut de position de composante ComponentPosition sera toujours utilisée pour faire référence aux composantes des types SEQUENCE, SET et CHOICE de l'ASN.1 quand un identificateur n'est pas déclaré pour cette composante. */

/* SÉMANTIQUE STATIQUE – L'identificateur de structure (StructIdentifier) ne sera pas utilisé si la structure pertinente est utilisée comme macro. L'identificateur StructIdentifier et l'identificateur PDU PDU_Identifier ne seront pas compris dans RecordRef chaque fois qu'un paramètre, un champ ou un élément est concaténé à une unité PDU ou à une structure et que RecordRef devra identifier une composante de cette unité PDU ou de cette structure. */

/* SÉMANTIQUE STATIQUE – Lorsqu'une structure est utilisée comme développement de macro, il sera fait référence aux éléments de la structure comme si cette dernière avait été développée dans la primitive ASP ou l'unité PDU qui s'y réfère. */

/* SÉMANTIQUE STATIQUE – Si un paramètre, champ ou élément est défini en tant que métatype PDU, aucune référence aux champs de cette sous-structure ne sera faite. */

712 ComponentIdentifier ::= ASP_ParIdentifier | PDU_FieldIdentifier | CM_ParIdentifier | ElemIdentifier | ASN1_Identifier

713 ASN1_Identifier ::= Identifier

/* NOTE – L'identificateur ASN1_Identifier identifie un champ à l'intérieur d'un type SEQUENCE, SET ou CHOICE de l'ASN.1. */

/* SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier associé à une valeur nommée NamedValue ne sera pas utilisé à moins que cette valeur n'apparaisse dans un type SEQUENCE, SET ou CHOICE. */

/* SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier sera prévu pour identifier la variante dans un type CHOICE. */

/* SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier sera prévu chaque fois que la définition de valeur devient ambiguë parce que des valeurs OPTIONAL sont omises dans un type SEQUENCE. */

714 ComponentPosition ::= "(" Number ")"

715 ArrayRef ::= Dot "[" ComponentNumber "]"

716 ComponentNumber ::= Expression

/* SÉMANTIQUE OPÉRATOIRE – Le numéro de composant ComponentNumber prendra une valeur spécifique de type INTEGER (entier) non négative. */

717 BitRef ::= Dot (BitIdentifier | "[" BitNumber "]")

718 BitIdentifier ::= Identifier

/* NOTE – L'identificateur de bits BitIdentifier identifie un bit particulier dans une chaîne BIT STRING de l'ASN.1. */

719 BitNumber ::= Expression

/* SÉMANTIQUE OPÉRATOIRE – Le numéro de bit BitNumber prendra une valeur spécifique de type INTEGER (entier) non négative. */

720 OpCall ::= OpIdentifier (ActualParList | "(" ")")

/* SÉMANTIQUE STATIQUE – Voir la section sémantique statique relative à la production 699. */

721 OpIdentifier ::= TS_OpIdentifier | TS_ProcIdentifier | PredefinedOpIdentifier

722 PredefinedOpIdentifier ::= BIT_TO_INT | HEX_TO_INT | INT_TO_BIT | INT_TO_HEX | IS_CHOSEN | IS_PRESENT | LENGTH_OF | NUMBER_OF_ELEMENTS

723 AddOp ::= "+" | "–" | **OR**

/* SÉMANTIQUE OPÉRATOIRE – Les opérandes des opérateurs "+" et "–" seront de type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou dérivés de type INTEGER (des sous-intervalles). Les opérandes de l'opérateur OR seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN. */

724 MultiplyOp ::= "*" | "/" | **MOD** | **AND**

/* SÉMANTIQUE OPÉRATOIRE – Les opérandes des opérateurs "*", "/" et MOD (modulo) seront du type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou dérivés du type INTEGER (des sous-intervalles). Les opérandes de l'opérateur AND seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN. */

725 UnaryOp ::= "+" | "–" | **NOT**

/* SÉMANTIQUE OPÉRATOIRE – Les opérandes des opérateurs "+", "–" seront du type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou des dérivés du type INTEGER (des sous-intervalles). Les opérandes de l'opérateur NOT seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN. */

726 RelOp ::= "=" | "<" | ">" | "<>" | "≥" | "≤"

A.3.3.35 Opérations de temporisation

727 TimerOps ::= TimerOp {Comma TimerOp}

728 TimerOp ::= StartTimer | CancelTimer | ReadTimer

729 StartTimer ::= **START** (TimerIdentifier | FormalParIdentifier) ["(" TimerValue ")"]

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier prendra une valeur de type TIMER (temporisateur). */

730 CancelTimer ::= **CANCEL** [TimerIdentifier | FormalParIdentifier]

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier prendra une valeur de type TIMER (temporisateur). */

731 TimerValue ::= Expression

/* SÉMANTIQUE OPÉRATOIRE – La valeur du temporisateur Timervalue prendra une valeur de type INTEGER positive différente de zéro. */

732 ReadTimer ::= **READTIMER** (TimerIdentifier | FormalParIdentifier) "(" DataObjectReference ")"

/* SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier prendra une valeur de type TIMER (temporisateur). */

/* SÉMANTIQUE STATIQUE – La référence DataObjectReference renverra toujours à un identificateur TS_VarIdentifier, à un identificateur TC_VarIdentifier, à une référence au champ de variable. */

/* SÉMANTIQUE OPÉRATOIRE – La référence DataObjectReference prendra une valeur de type INTEGER. */

A.3.3.36 Types

733 TypeOrPDU ::= Type | **PDU**
734 Type ::= PredefinedType | ReferenceType

A.3.3.36.1 Types prédéfinis

735 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | **OBJECTIDENTIFIER** | **R_Type** | CharacterString
736 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString** | **T61String** | **ISO646String**

A.3.3.36.2 Types de référenciation

737 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier | CM_Identifier
/* SÉMANTIQUE STATIQUE – Exception faite des types prédéfinis, tous les types utilisés dans une suite de tests seront déclarés dans les définitions de type de suite de tests, de type de primitive ASP, de type d'unité PDU ou de type de message CM et recevront un nom pour y faire référence. */
738 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier

A.3.3.37 Valeurs

739 Value ::= LiteralValue | ASN1_Value [ASN1_Encoding]
/* RÉFÉRENCE – Où ASN1_Value est un type valeur non terminale, comme défini dans la Recommandation X.680. Aux fins de la notation TTCN, cette production est définie de la manière suivante dans la Recommandation X.680:
DefinedValue ::= Externalvaluereference | valuereference est redéfinie comme suit:
DefinedValue ::= ConstraintValue&Attributes | valuereference
Cela signifie que des références externes en notation ASN.1 ne sont pas permises en notation TTCN, mais l'ensemble des possibilités de ConstraintValue&Attributes en notation TTCN tel que défini dans la production 562 peut contenir des valeurs en notation ASN.1. Par conséquent, ces valeurs comprennent les expressions, les spécificateurs de concordance, les références aux contraintes, la longueur des valeurs, l'attribut IF_PRESENT, de même que les opérations de codage de champ en ASN.1. */
Aux fins de la notation TTCN, les productions suivantes figurent dans la Recommandation X.680:
BuiltinValue ::=
BitStringValue |
BooleanValue |
CharacterStringValue |
ChoiceValue |
EmbeddedPDUValue |
EnumeratedValue |
ExternalValue |
InstanceOfValue |
IntegerValue |
NullValue |
ObjectClassFieldValue |
ObjectIdentifierValue |
OctetStringValue |
RealValue |
SequenceOfValue |
SequenceOfValue |
SetValue |
SetOfValue |
TaggedValue

ReferencedValue ::=
DefinedValue |
ValueFromObject

sont redéfinies comme suit:

BuiltinValue ::=
BitStringValue |
BooleanValue |
CharacterStringValue |
ChoiceValue |
EmbeddedPDUValue |
EnumeratedValue |
ExternalValue |
IntegerValue |
NullValueValue |
ObjectIdentifiervalue |

OctetStringValue |
RealValue |
SequenceValue |
SequenceOfValue |
SetValue |
SetOfValue |
TaggedValue

ReferencedValue ::=
DefinedValue */

/* SÉMANTIQUE STATIQUE – Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans les expressions arithmétiques. */

740 LiteralValue ::= Number | BooleanValue | Bstring | Hstring | Ostring | Cstring | R_Value

741 Number ::= (NonZeroNum {Num}) | **0**

742 NonZeroNum ::= **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

743 Num ::= **0** | NonZeroNum

744 BooleanValue ::= **TRUE** | **FALSE**

745 Bstring ::= "_" {Bin | Wildcard} "_" **B**

746 Bin ::= **0** | **1**

747 Hstring ::= "_" {Hex | Wildcard} "_" **H**

748 Hex ::= Num | **A** | **B** | **C** | **D** | **E** | **F**

749 Ostring ::= "_" {Oct | Wildcard} "_" **O**

750 Oct ::= Hex Hex

751 Cstring ::= """ {Char | Wildcard | "\"} """

752 Char ::= */* RÉFÉRENCE – Caractère défini par le type chaîne de caractères approprié. */*

/* EXIGENCE LEXICALE – Si le type CharacterString comporte un guillemet ", ce caractère sera dédoublé dans la dénomination d'une valeur quelconque. */

753 Wildcard ::= AnyOne | AnyOrNone

754 AnyOne ::= "?"

/* SÉMANTIQUE STATIQUE – Le caractère générique AnyOne (= un caractère quelconque) ne sera utilisé que dans des valeurs de type chaîne, SEQUENCE OF et SET OF. */

755 AnyOrNone ::= "*"

/* SÉMANTIQUE STATIQUE – Le caractère générique "*" (AnyOrNone = zéro ou un caractère quelconque) ne sera utilisé que dans les valeurs de type chaîne, SEQUENCE OF et SET OF. */

756 R_Value ::= **pass** | **fail** | **inconc** | **none**

757 Identifier ::= Alpha{AlphaNum | Underscore | DoubleColon}

/* SÉMANTIQUE STATIQUE – Tous les identificateurs auxquels il est fait référence dans une suite de tests en notation TTCN seront soit déclarés explicitement dans la suite des tests, soit déclarés explicitement dans une définition de type en notation ASN.1 citée elle-même en référence par la suite de tests, ou seront des identificateurs prédéfinis en notation TTCN. */

/* SÉMANTIQUE STATIQUE – Les deux points (:) seront seulement utilisés dans les identificateurs qui ont été déclarés dans une table d'importation. Les identificateurs contenant les deux points (:) ne seront pas utilisés dans une table d'exportation. Les deux points servent à séparer le nom d'un module TTCN d'un identificateur qui a été initialement spécifié dans le module TTCN. */

758 Alpha ::= UpperAlpha | LowerAlpha

759 AlphaNum ::= Alpha | Num

760 UpperAlpha ::= **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z**

761 LowerAlpha ::= **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z**

762 ExtendedAlphaNum ::= */* RÉFÉRENCE – Caractère d'un jeu quelconque défini dans la Norme ISO/CEI 10646. */*

763 BoundedFreeText ::= *"/*"* FreeText *"*/"*

764 FreeText ::= {ExtendedAlphaNum}

/* EXIGENCE LEXICALE – Une chaîne FreeText (texte libre) ne contiendra pas le groupement "*/" à moins de faire précéder un tel groupement par une barre oblique inversée ("\"). */

A.3.3.38 Productions diverses

765 Comma ::= ","

766 Dot ::= "."

767 Dash ::= "-"

768 Minus ::= "-"

769 SemiColon ::= ";"

770 DoubleColon ::= Colon Colon

771 Colon ::= ":"

772 Underscore ::= "_"

A.4 Spécifications générales de la sémantique statique

A.4.1 Introduction

Les spécifications de la sémantique statique liées à des productions spécifiques de la forme BNF sont indiquées sous forme de commentaires à la suite des productions correspondantes, dans le format suivant:

/* STATIC SEMANTICS – ... */

Toutes les autres spécifications de la sémantique statique commune aux notations graphique TTCN.GR et informatisable TTCN.MP sont spécifiées dans le reste du A.4. La sémantique statique additionnelle appliquée en notation TTCN.MP est spécifiée au A.5.2.

A.4.2 Unicité des identificateurs

A.4.2.1 Dans certains cas, les suites de tests renvoient à des éléments définis dans d'autres Recommandations OSI. Les définitions de types peuvent notamment faire référence à des modules de définition de type ASN.1 selon la Recommandation X.680. Les noms provenant de ces modules (tels que les identificateurs de sous-champs à l'intérieur des définitions de types structurés ASN.1) peuvent être utilisés dans toute la suite de tests.

Etant donné que les règles concernant les identificateurs en ASN.1 et en notation TTCN ne sont pas totalement compatibles, les conventions suivantes s'appliquent:

a) les références aux types, les identificateurs de modules et les références aux valeurs qui figurent dans les différentes tables de définitions de type ASN.1 respecteront les conventions définies dans la Recommandation X.680 relative aux identificateurs;

b) dans les identificateurs utilisés dans les autres parties d'une suite de tests, les tirets (–) seront remplacés par des soulignés (_).

Dans certaines tables TTCN, une partie de la syntaxe ASN.1 peut être utilisée pour la définition de types. Dans ce cas, les identificateurs se conformeront aux règles de l'ASN.1 sauf que les tirets (–) ne seront pas utilisés. Les soulignés (_) peuvent être utilisés à leur place. Toutes les autres spécifications établies dans la Recommandation X.680 (par exemple, les identificateurs de type commenceront par une majuscule et les identificateurs de champs dans les définitions structurées de l'ASN.1 commenceront par une minuscule) s'appliquent aux suites de tests TTCN chaque fois que la notation ASN.1 est utilisée.

A.4.2.2 Tous les identificateurs des objets TTCN suivants seront uniques dans toute la suite de tests:

a) types de suite de tests;

b) opérations de suite de tests;

c) paramètres de suite de tests;

d) expressions de sélection de test élémentaire;

e) constantes de suite de tests;

f) variables de suite de tests;

g) variables de test élémentaire;

h) types de point PCO;

NOTE – Si la table ne contient pas de déclaration de type de point PCO, les types de point PCO sont implicitement déclarés dans la table, alors l'unicité renvoie à la signification du type de point PCO; le même type de point PCO ayant la même signification peut figurer plusieurs fois dans la table de déclaration.

i) points PCO;

j) points CP;

k) temporisations;

l) composantes de tests;

m) configurations des composantes de tests;

n) types de primitive ASP;

o) types d'unité PDU;

p) types de message CM;

q) types structurés;

r) règles de codage;

s) variations de codage;

t) codage de champ non valide;

u) alias (pseudonymes);

v) contraintes de primitive ASP;

w) contraintes d'unité PDU;

x) contraintes de message CM;

y) contraintes de structure;

z) tests élémentaires;

aa) modules de tests;

ab) arbres par défaut;

ac) noms des règles de codage;

ad) noms des variations de codage;

ae) noms de codage de champ non valide.

A.4.2.3 Toutes les références aux objets TTCN suivants seront uniques dans toute la suite de tests:

a) références à des groupes de tests;

b) références à des groupes de modules de tests;

c) références à des groupes de comportements par défaut.

A.4.2.4 Le Tableau A.2 énumère les mots réservés de la notation TTCN. Ces mots réservés ne doivent pas être utilisés comme identificateurs dans une suite de tests TTCN. Pour tous les mots réservés de la notation TTCN et les identificateurs TTCN, la différence majuscule/minuscule est significative.

A.4.2.5 Le Tableau A.3 énumère les mots réservés de la notation ASN.1. Ces mots réservés ne seront pas utilisés comme identificateurs dans une suite de tests TTCN.

A.4.2.6 Quand la notation ASN.1 est utilisée dans une suite de tests TTCN, les identificateurs ASN.1 de la liste suivante seront uniques dans toute la suite de tests, que la définition ASN.1 soit explicite ou implicite par référence:

a) identificateurs de type (*TypeIdentifiers*) d'une définition de type de l'ASN.1;

b) identificateurs apparaissant dans un type ENUMERATED (énuméré) de l'ASN.1 en tant que valeurs distinctives;

c) identificateurs apparaissant dans une liste de nombres nommés (*NamedNumberList*) d'un type INTEGER de l'ASN.1.

A.4.2.7 Les noms de paramètres de primitive ASP seront uniques dans la primitive ASP où ils sont déclarés. Les noms de champs d'unité PDU seront uniques dans l'unité PDU où ils sont déclarés. Les noms de paramètres de message CM seront uniques dans le message CM où ils sont déclarés.

A.4.2.8 Si un type structuré est utilisé comme développement de macro, les noms des éléments contenus dans le type structuré seront uniques dans chaque primitive ASP, unité PDU ou message CM où cette structure sera développée.

A.4.2.9 Les étiquettes utilisées dans un arbre seront uniques dans cet arbre (par exemple, arbre racine de test élémentaire, arbre de module de test, arbre par défaut, arbre local).

A.4.2.10 L'identificateur d'en-tête d'arbre utilisé pour les arbres locaux sera unique dans la description de comportement dynamique où ces arbres apparaissent et sera différent de tout identificateur ayant une signification unique dans toute la suite de tests.

NOTE – Cela signifie qu'un identificateur d'arbre local peut avoir le même nom qu'un identificateur d'arbre local dans une autre description comportementale, mais pas le même qu'un autre module de test dans la bibliothèque des modules de test.

A.4.2.11 Les noms de paramètres formels qui peuvent apparaître en option comme faisant partie des éléments suivants seront uniques dans cette liste de paramètres formels et ne coïncideront avec aucun autre identificateur ayant une signification unique dans toute la suite de tests:

a) définition des opérations de suite de tests;

b) en-tête d'un arbre local;

c) identificateur de module de test;

d) identificateur par défaut;

e) déclaration de contrainte paramétrée.

Tableau A.2/X.292 – Mots clés réservés de la notation TTCN

ACTIVATE	IA5String	pass
AND	IF	PDU
BEGIN	IF_PRESENT	PERMUTATION
BITSTRING	INCONC	PrintableString
BIT_TO_INT	inconc	ps
BOOLEAN	INFINITY	PTC
BY	INTEGER	R
CANCEL	INT_TO_BIT	READTIMER
CASE	INT_TO_HEX	REPEAT
COMPLEMENT	IS_CHOSEN	REPLACE
CP	IS_PRESENT	RETURN
CREATE	IUT	RETURNVALUE
DO	LT	R_Type
DONE	min	s
ELSE	MOD	START
ENC	ms	STATIC
END	MTC	SUPERSET
ENDCASE	NOT	SUBSET
ENDIF	ns	TeletexString
ENDVAR	OF	THEN
ENDWHILE	OMIT	TIMEOUT
F	OR	TIMER
FAIL	OTHERWISE	TO
fail	P	TRUE
FALSE	LENGTH_OF	UNTIL
GeneralString	none	µs
GOTO	NUMBER_OF_ELEMENTS	UT
GraphicString	NumericString	VAR
HEXSTRING	OCTETSTRING	VideotexString
HEX_TO_INT	OBJECTIDENTIFIER	VisibleString
I	PASS	WHILE

A.4.2.12 Un nom de paramètre formel figurant dans la liste des paramètres formels d'un en-tête d'arbre local supplantera, dans le champ de visibilité de cette liste locale, un nom identique de paramètre formel figurant dans la liste des paramètres formels du module de test où l'arbre local est défini.

A.4.2.13 En notation TTCN concomitante, les points PCO et CP utilisés dans un test élémentaire seront seulement ceux qui ont été définis dans la configuration de la composante de test de ce test élémentaire.

A.4.2.14 Chaque identificateur utilisé dans la définition de la procédure d'une opération de suite de tests sera le:

a) nom d'une variable déclarée localement;

b) nom d'un type utilisé dans une déclaration de variable;

c) nom d'un paramètre formel déclaré dans la liste des paramètres formels d'une opération;

d) nom d'une opération de suite de tests.

Tableau A.3/X.292 – Mots réservés de la notation ASN.1

ABSENT	EXTERNAL	OPTIONAL
ABSTRACT SYNTAX	FALSE	PDV
ALL	FROM	PRESENT
APPLICATION	GeneralString	PRIVATE
AUTOMATIC	GeneralizedTime	PrintableString
BEGIN	GraphicString	REAL
BIT	IA5String	SEQUENCE
BMPString	IDENTIFIER	SET
BOOLEAN	IMPLICIT	SIZE
CHARACTER	IMPORT	STRING
CHOICE	INCLUDES	SYNTAX
CLASS	INSTANCE	T61String
COMPONENT	INTEGER	TRUE
COMPONENTS	INTERSECTION	TeletexString
CONSTRAINED	ISO646String	TYPEIDENTIFIER
DEFAULT	MAX	UNION
DEFINITIONS	MIN	UNIQUE
EMBEDDED	NULL	UNIVERSAL
END	NumericString	Universalstring
ENUMERATED	OBJECT	UTCTime
EXCEPT	ObjectDescriptor	VideotexString
EXPLICIT	OCTET	VisibleString
EXPORT	OF	WITH

NOTE – Le Tableau A.3 contient un certain nombre de mots clés définis qui ne sont pas pris en charge dans la présente Recommandation. Ces mots clés ont été réservés pour faciliter l'intégration ultérieure à la notation TTCN de fonctions en ASN.1 1997.

Le cadre des noms des paramètres formels et des noms des variables déclarés localement est la définition de la procédure d'une opération de suite de tests. Par conséquent, les valeurs de tous les autres types d'identificateur ne sont pas directement accessibles à partir de la définition de la procédure d'une opération de suite de tests. Ces valeurs sont accessibles seulement lorsqu'elles sont transmises comme paramètres effectifs d'une opération de suite de tests.

A.4.2.15 Les contraintes de type structuré, de primitive ASP, d'unité PDU et de message CM en notation TTCN ne peuvent être spécifiées à l'aide des tableaux en notation ASN.1 (c'est-à-dire les contraintes de type en notation ASN.1, les contraintes de primitive en notation ASN.1, les contraintes d'unité PDU en notation ASN.1 ou les contraintes de message CM en notation ASN.1). De même, les contraintes de type en notation ASN.1, de primitive ASP en notation ASN.1, d'unité PDU en notation ASN.1 et de message CM en notation ASN.1 ne peuvent être spécifiées à l'aide de tableaux en notation TTCN (c'est-à-dire les contraintes de type structuré en notation TTCN, les contraintes de primitive ASP en notation TTCN, les contraintes d'unité PDU en notation TTCN ou les contraintes de message CM en notation TTCN).

NOTE – Cependant, lors du chaînage de primitives ASP ou d'unités PDU à d'autres unités PDU, la primitive ASP ou l'unité PDU qui les contient peut être, par exemple, spécifiée en notation TTCN tabulaire, alors que l'unité PDU qu'elles contiennent peut être spécifiée en notation ASN.1.

A.5 Différences entre la notation TTCN.GR et la notation TTCN.MP

A.5.1 Différences syntaxiques

Les différences syntaxiques entre notations informatisable TTCN.MP et graphique TTCN.GR sont énumérées ci-dessous:

a) la notation TTCN.MP utilise des mots clés comme délimiteurs entre les entrées, tandis que la notation TTCN.GR utilise des encadrés;

b) la notation TTCN.MP utilise une dénotation numérique explicite des niveaux d'indentation des événements de test, tandis que cet ordre d'imbrication est inhérent à l'aspect graphique de la notation TTCN.GR;

c) la notation TTCN.MP comporte une occurrence supplémentaire de l'identificateur de suite, afin de faciliter l'identification des suites de tests abstraites (ATS) en méthode automatisée;

d) en notation TTCN.MP, les descriptions comportementales des tests élémentaires sont explicitement regroupées par l'insertion de l'identificateur du groupe de tests en tête des descriptions comportementales des tests élémentaires appartenant à ce groupe; cette information reproduit celle qui figure dans l'index des tests élémentaires et dans les références de groupe de tests des descriptions comportementales des tests élémentaires;

e) les tables de structure de suites de tests, d'index des tests élémentaires, d'index de modules de tests et d'index par défaut exigent un numéro de page par entrée. Etant donné que de tels numéros n'ont pas de signification sémantique en traitement machine, ils ne sont pas pris en compte dans la notation TTCN.MP;

f) la notation TTCN.GR peut gérer les formulaires tant simples que compacts, utilisés pour les contraintes de primitive ASP et d'unité PDU et pour les tests élémentaires; pour le format tabulaire simple, la notation TTCN ne peut gérer que la forme BNF et la représentation d'un certain nombre de tables simples en format compact TTCN.GR est un problème de sortie; quand on établit le mappage entre une table compacte de contraintes et la notation TTCN.MP (c'est-à-dire le format simple), les champs supprimés par les modifications seront omis;

g) les symboles "/*" et "*/" ouvrant et fermant les chaînes de texte libre borné BoundedFreeText en notation TTCN.MP n'apparaîtront pas en notation TTCN.GR;

h) il y a deux positions possibles pour la colonne étiquettes des tableaux de description comportementale en notation TTCN.GR et une seule pour les étiquettes en notation TTCN.MP;

i) les suites de pages et de lignes n'apparaissent qu'en notation TTCN.GR et n'existent pas en notation TTCN.MP;

j) la numérotation des pages et des lignes n'apparaît qu'en notation TTCN.GR et n'existe pas en notation TTCN.MP;

k) si des références de groupes en notation TTCN.GR sont utilisées avec des définitions, des déclarations ou des contraintes pour indiquer un groupement hiérarchique d'objets, alors chaque identificateur de groupe correspondant est inséré avant la syntaxe d'un groupe de tableaux auxquels cet identificateur appartient, et la syntaxe de l'identificateur de groupe et les groupes de tableaux qui suivent sont délimités par les mots clés en notation TTCN.MP correspondant au type d'objet.

A.5.2 Eléments additionnels de sémantique statique en notation TTCN.MP

On trouvera ci-dessous une liste d'éléments additionnels de sémantique statique utilisés en notation TTCN.MP:

a) en notation TTCN.MP, les déclarations du premier niveau d'options n'ayant pas de prédécesseur dans l'arbre racine ou l'arbre local auquel elles appartiennent ont la valeur d'indentation zéro; les déclarations ayant un prédécesseur ont une valeur d'indentation égale à celle de l'option du niveau précédent plus un;

b) en notation TTCN.MP, l'information de structure de suite de tests, représentée sous la forme d'identificateurs de groupe de tests précédant les descriptions comportementales des tests élémentaires, sera la même que celle qui est définie par la partie de la structure de suite de tests correspondant aux groupes de tests et celle qui est définie par l'index des tests élémentaires.

A.6 Liste des numéros de production dans la forme BNF

A.6.1 Introduction

Le présent sous-paragraphe constitue un index alphabétique des productions en forme BNF figurant dans l'Annexe A. Le présent index donne pour chaque production un renvoi sous forme de numéro de production (et non de numéro de page).

A.6.2 Index des productions

A

B

C

D

E

F

G

H

Q

R

S

T

U

V

W

Annexe B

Sémantique opératoire de la notation TTCN

B.1 Introduction

L'Annexe A décrit la syntaxe de la notation TTCN au moyen de règles de production en forme BNF et de restrictions relatives aux productions considérées, dont l'application peut être vérifiée soit de façon statique, soit de façon dynamique.

La présente annexe définit la sémantique de la notation TTCN en décrivant une procédure abstraite qui exécute des suites de tests TTCN valides sur le plan de la syntaxe. Cette procédure met en marche, pour chaque test élémentaire, une "machine TTCN" abstraite qui évalue le test par la création, le développement et l'interprétation d'un "arbre d'évaluation", en traitant un niveau (ensemble ordonné d'options à une certaine position dans l'arbre) à la fois. Dans l'exécution de la notation TTCN concomitante, des machines TTCN additionnelles sont mises en marche, une pour chaque composante PTC créée. Ces machines fonctionnent de la même façon que la machine TTCN principale, qui exécute alors la composante de test principale. Les points PCO et CP nécessaires, reliant les machines TTCN à leur environnement et entre elles, sont présumés déjà existants et initialement vides.

La procédure abstraite (EVALUATE_TEST_SUITE) et les machines TTCN (EVALUATE_TEST_CASE, EVALUATE_TEST_COMPONENT) sont décrites au B.5. L'arbre d'évaluation a la forme d'un arbre de comportement TTCN, mais enrichi de composantes additionnelles. Dans une machine TTCN, il est initialement réglé de façon à correspondre à l'arbre racine de test élémentaire ou de module de test indiqué, ou à l'arbre local. Pendant l'exécution du test élémentaire, l'arbre d'évaluation est développé, et le "contrôle" est généralement transféré vers le bas dans l'arbre d'évaluation, sauf dans l'exécution de constructions GOTO et RETURN, le contrôle étant alors transféré vers le haut.

Les composantes d'arbre additionnelles, introduites pour des raisons techniques, sont les suivantes: chaque nœud (option) possède, en plus de la ligne StatementLine dénotée, une valeur booléenne IsDefault, indiquant si le nœud découle d'une table de comportement par défaut; chaque niveau possède, en plus de la liste dénotée de lignes StatementLines, une valeur booléenne IsExpanded, indiquant si le niveau a déjà été développé.

Il n'est pas nécessaire qu'une machine TTCN réelle soit construite de façon que son fonctionnement interne soit exactement identique à celui de la machine abstraite. La sémantique opératoire TTCN définit seulement ce que devrait être le comportement externe d'une machine TTCN réelle, c'est-à-dire par rapport aux files d'attente de points PCO et CP, aux temporisateurs et à la liste de temporisateurs, et à l'information de fin de composante de test. Les détails de l'implémentation ne sont pas pertinents.

B.2 Priorité

La sémantique opératoire appliquée à la notation TTCN est présentée dans les sous-paragraphes qui suivent sous forme d'une combinaison de pseudo-code et de langage naturel. Lorsqu'elles chevauchent, ces deux notations visent à prendre des significations identiques. Par contre, si le pseudo-code et le langage naturel se contredisent, il s'agit d'une erreur qui doit être portée à la connaissance de l'organisme de normalisation sous forme de rapport d'anomalie. Dans ce cas, en attendant que l'anomalie ait été corrigée par l'organisme de normalisation, le pseudo-code prend le pas sur le langage naturel.

B.3 Traitement des erreurs de test élémentaire

Dans le corps de la présente Recommandation, ainsi que dans l'Annexe A et dans la présente annexe, on décrit des conditions qui conduisent à la détection d'erreurs de test élémentaire. L'observation d'une erreur de test élémentaire doit être consignée dans le journal de conformité et conduire à l'abandon du test élémentaire.

Sans qu'il en soit fait mention explicitement dans ce qui suit, une erreur de test élémentaire est toujours détectée de façon dynamique lorsque l'évaluation d'une partie quelconque d'une expression ne donne pas une valeur définie. Les expressions sont évaluées, entre autres, dans l'application d'affectations, de qualificateurs et de contraintes.

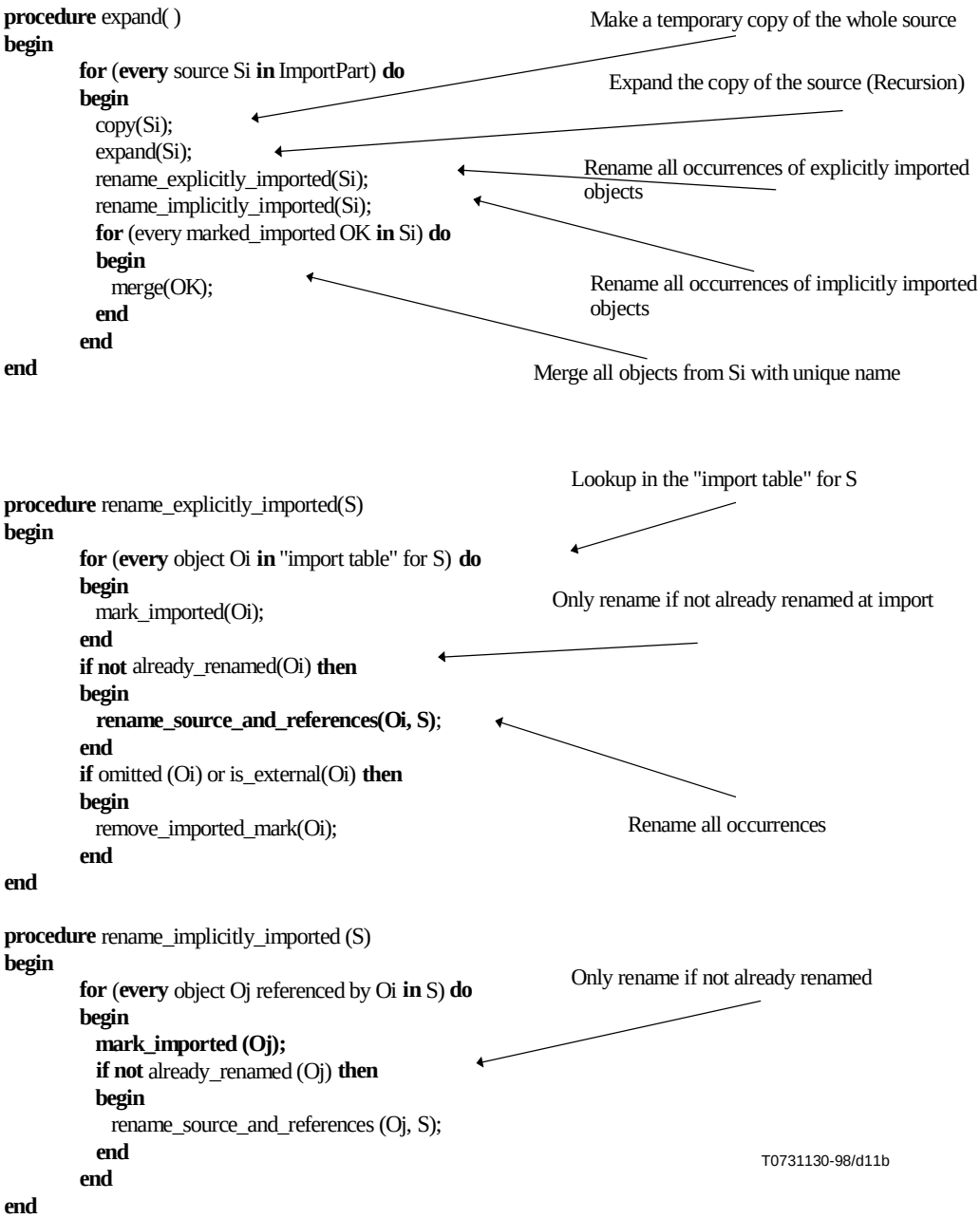
B.4 Conversion d'une suite de tests modulaire en une suite de tests développée équivalente

Cet algorithme ne traite pas les cas d'erreur. Il nécessite des objets qui sont uniques dans le cadre de leur définition et de leur utilisation.

Dans la conversion d'une suite de tests modulaire en une suite de tests développée, il est nécessaire de renommer certains objets TTCN importés (afin d'éviter les conflits entre noms). Dans ce processus de renommage, deux options sont possibles:

a) le nom d'origine est conservé tel que défini dans la déclaration/définition de l'objet;

b) le nouveau nom est construit par concaténation de l'identificateur du module et du nom d'origine de l'objet. Les deux éléments doivent être séparés par deux traits de soulignement, par exemple ModuleA_ConnectionRequest.



Le principe de cet algorithme consiste à faire une copie temporaire de chaque objet source, à faire un développement de la copie, puis à marquer chaque objet à importer et, finalement, à incorporer chaque objet marqué dans la suite d'importation.

Dans le processus de développement des sources importées, tous les objets importés implicitement et explicitement sont renommés sous la forme Module::Identifier, s'ils n'ont pas déjà été renommés au moment de l'importation. Chaque module doit avoir un identificateur unique. Dans la suite de tests développée, tous les objets importés implicitement et explicitement sont nettement reconnaissables et, comme chaque module doit posséder un nom unique, les conflits entre noms sont impossibles.

B.5 Sémantique opératoire de la notation TTCN

B.5.1 Introduction

Les arbres de comportement de la notation TTCN sont évalués un niveau d'options à la fois. A chaque niveau, des comportements par défaut sont adjoints, des constructions de rattachement sont développées et des constructions REPEAT sont remplacées. Il en résulte un ensemble d'options qui peuvent être évaluées en vue de déterminer laquelle offre une bonne concordance et détermine par conséquent à quel ensemble d'options il faut passer ensuite. Les conditions de concordance d'une déclaration TTCN dépendent de ce qui est codé sur la ligne de comportement, et elles sont décrites dans le présent texte sémantique.

B.5.2 Notation en pseudo-code

B.5.2.1 Introduction

La sémantique de la notation TTCN est définie à l'aide d'une méthode fonctionnelle simple qui explique l'exécution d'une description de comportement de test élémentaire TTCN, comprenant le développement par paliers d'un arbre d'évaluation et l'exécution des nœuds de cet arbre. Les fonctions utilisées ont pour but de mieux faire comprendre la sémantique de la notation TTCN et ne sont pas destinées à être associées à un quelconque modèle d'exécution ou langage de programmation de haut niveau. Elles ne sont pas censées être des méthodes directes destinées à exécuter des déclarations TTCN.

Les mots clés de pseudo-code sont imprimés en gras, par exemple **procedure**, **function**, **begin**, **end**, **if**, **then**, **else**. Dans l'en-tête de leur définition, les noms de procédure, de processus et de fonction, sont mis en évidence par l'emploi du gras, afin de faciliter la recherche. Pour la même raison, le type de données d'une fonction est mis en évidence. Autrement, les types de données ne sont pas traités explicitement.

B.5.2.2 Procédures et fonctions

Bon nombre de déclarations sont des appels de procédures **procedure**. Des expressions de fonctions **function** peuvent être utilisées chaque fois qu'une valeur du type associé est requise. Elles obtiennent leur valeur (et prennent immédiatement fin) au moyen d'une déclaration **return**, suivie d'une expression de valeur.

Les paramètres de procédure et de fonction sont généralement des "paramètres polyvalents", c'est-à-dire des paramètres formels sur lesquels on peut effectuer une "lecture" et une "écriture". En particulier, les fonctions peuvent avoir des "effets secondaires" et sont essentiellement des "procédures avec une valeur". Les variables dans le corps d'une procédure ou d'une fonction qui ne sont ni des paramètres formels ni l'une quelconque des variables globales susmentionnées sont des variables locales de ce corps, sans déclaration explicite.

Des précautions sont prises pour:

– que les paramètres ne soient lus que lorsqu'ils ont une valeur définie;

– que les termes ne soient utilisés comme des paramètres effectifs que lorsque la procédure ou la fonction n'affecte pas une valeur au paramètre formel correspondant, c'est-à-dire lorsque ce paramètre est purement un paramètre d'entrée.

B.5.2.3 Processus

Les **processus** se comportent comme des procédures, sauf qu'ils sont exécutés chacun sur une machine TTCN distincte. Ils ne sont pas exécutés en mode emboîtement. Dans un processus, des objets de données globaux peuvent être déclarés, de façon à être disponibles dans toutes les procédures et fonctions appelées dans le processus sans être transférés explicitement sous forme de paramètres. Le fait d'éviter les longues listes de paramètres simplifie la lecture du pseudo-code. Evidemment, il existe des instances d'objets globaux indépendamment dans chaque processus (machine TTCN). Il n'y a pas de relation entre les objets globaux de différents processus.

Dans la présente annexe, les objets énumérés ci-dessous sont traités comme des objets globaux dans chaque processus:

– EvaluationTree, du test élémentaire (ou de la composante de test principale) ou de la composante de test parallèle;

– CurrentLevel, à développer ou à mettre en concordance;

– Defaults, le contexte par défaut courant, utilisé dans le développement par défaut;

– Snapshot, la vue fixée temporairement de l'environnement;

– ReturnLevel, à prendre en considération après l'exécution d'une déclaration RETURN;

– ReturnDefaults, le contexte par défaut du niveau ReturnLevel;

– SendObject, la primitive ASP, l'unité PDU ou le message CM à envoyer ensuite;

– ReceiveObject, la primitive ASP, l'unité PDU ou le message CM reçu en dernier.

Ainsi, chaque machine TTCN a son propre arbre EvaluationTree, etc.

D'autres objets, cependant, sont accessibles à partir de tous les processus. L'état pertinent de l'"environnement de la procédure EVALUATE_TEST_SUITE", c'est-à-dire le contenu des points PCO et CP pertinents, ainsi que les listes des temporisateurs arrivés à expiration, les valeurs des temporisateurs et la liste des composantes de test parallèle terminées, sont présumés globalement accessibles à partir de toutes les composantes de test, et il n'est pas nécessaire de les transférer explicitement sous forme de paramètres. De même, les paramètres de suite de tests, les constantes de suite de tests et les variables de suite de tests sont présumés accessibles à partir de tous les processus de test élémentaire ou de composante de test.

B.5.2.4 Langage naturel dans le pseudo-code

Certaines parties du pseudo-code sont écrites en langage naturel, afin de limiter la complexité de la présente annexe. Ces parties sont délimitées par les marques /# et #/. Elles représentent des déclarations, des détails relatifs aux boucles FOR, ou des expressions de pseudo-code et on suppose qu'elles sont exécutées ou évaluées, lorsqu'elles apparaissent.

Les commentaires purs, destinés au lecteur, qui n'ont pas à être exécutés ou évalués par une machine TTCN, sont délimités par les marques (* et *).

B.5.2.5 Niveaux et options

Un niveau visité dans un arbre représente à la fois une position dans l'arbre et l'ensemble ordonné d'options à ce niveau.

Une option visitée dans un arbre détermine une position de niveau dans l'arbre (voir LEVEL_OF dans B.5.25). L'option représente simultanément une position à ce niveau, une ligne BehaviourLine, une ligne StatementLine, etc.

Ainsi, les niveaux et les options dans un arbre sont des pointeurs, mais l'éclatement des objets de données sur lesquels ils pointent est effectué implicitement.

B.5.3 Exécution d'une suite de tests

B.5.3.1 Introduction

La suite de tests est exécutée dans la procédure principale, EVALUATE_TEST_SUITE. Chaque composante de test principale (test élémentaire dans le cas d'un test non concomitant) est exécutée sur une machine TTCN abstraite exécutant une procédure EVALUATE_TEST_CASE. Chaque composante de test parallèle est exécutée sur une machine TTCN indépendante, exécutant une procédure EVALUATE_TEST_COMPONENT.

```
procedure EVALUATE_TEST_SUITE(TestSuiteId)
        (* Cette procédure introduit des noms uniques pour tous les arbres TTCN, y compris les sous-arbres locaux. Elle établit des
           objets de données propres à une suite de tests et évalue chaque test élémentaire dont les expressions de sélection
           prennent la valeur TRUE. *)
begin
        for /# every Test Case, Test Step or Default behaviour table Table in TestSuiteId #/ do
        begin
          /#    Rename all local trees of Table such that they become unique throughout the test suite and different from any Test
                Case,
                Test Step or Default behaviour table name in the Test Suite. #/;
          /#    Rename accordingly in Table all references to local trees in attachments. #/;
          /#    Every node in every behaviour tree gets a new Boolean component "IsDefault".
                This component is set to TRUE for all nodes in Default Dynamic Behaviour Tables
                and FALSE for all nodes in all other tables. #/;
        end;
```

```
        for /# every Default behaviour table Table in TestSuiteId #/ do
        begin
          /#    For each leaf of the behaviour tree which does not have an entry in the verdict column assign the verdict
                R. #/
          /#    or each leaf of the behaviour table which has a preliminary result assigned, change the preliminary result to a
                verdict by removing the parentheses around it. #/
        end;
        Evaluated := /# empty list of Test Case Identifiers #/;
        /#  Set values of Test Suite Parameters, Test Suite Constants, and, where to be initialized, of Test Suite Variables #/;
        for /# every Test Case Identifier TCId of TestSuiteId that is not yet in Evaluated #/ do (* dans n'importe quel ordre *)
        begin
          SelEx := /# conjunction of the selection expressions of all test groups containing Test Case TCId (directly or via lower
          groups) #/;
          if EVALUATE_BOOLEAN(SelEx) then
                start process EVALUATE_TEST_CASE (TCId);
          /# add TCId to the list Evaluated #/;
        end
end
```

B.5.4 Exécution d'un test élémentaire

B.5.4.1 Exécution d'un test élémentaire – Pseudo-code

```
process EVALUATE_TEST_CASE(TestCaseId)
        (*  Ce processus initialise l'arbre EvaluationTree à partir de l'arbre racine de test élémentaire et le contexe par défaut à partir
            des références de comportement par défaut énumérées avec la description de comportement d'un test élémentair. Il
            transfère le contrôle au niveau le plus élevé des options et appelle leur évaluation. *)
global EvaluationTree, CurrentLevel, Defaults, Snapshot, ReturnLevel, ReturnDefaults, SendObject, ReceiveObject;
begin
        /#  Initialize Test Case Variables, global R and MTC_R, PCOs, CPs, Timers, and the Timeout List of TestCaseId. #/;
        EvaluationTree := ROOT_TREE(TestCaseId);
          (*  L'arbre EvaluationTree est un arbre fini croissant construit en insérant ensemble et en étendant des copies d'arbres
              provenant de la description de comportement d'un test élémentaire et des bibliothèques des modules de test et des
              comportements par défaut. Une composante IsExpanded est ajoutée à chaque niveau. *)
          CurrentLevel := FIRST_LEVEL(EvaluationTree) ;
          (*  niveau représente à la fois une position dans un arbre et l'ensemble ordonné d'options à cette position. *)
          ReturnLevel := CurrentLevel;
          Defaults := DEF_REF_LIST(TestCaseId);
          ReturnDefaults := Defaults;
          EVALUATE_LEVELS ();
          (*  Y compris, par appels emboîtés, l'évaluation de tous les niveaux subséquents pertinents de l'arbre d'évaluation
              croissant. *)
end

procedure EVALUATE_LEVELS ()
        (*  Cette procédure étend et évalue d'abord le niveau CurrentLevel, lequel est le niveau d'options actif courant de l'arbre
            EvaluationTree. Defaults fournit le contexte par défaut actif courant. Les options contenues dans le niveau CurrentLevel
            sont traitées suivant leur ordre d'apparition, si nécessaire par boucles répétées. L'option CurrentAlternative est la variable
            de la boucle FOR, représentant l'option courante au niveau CurrentLevel. Grâce au mécanisme d'image instantanée, dans
            chaque boucle d'essais de mise en concordance par l'intermédiaire du niveau CurrentLevel, l'état de l'environnement
            considéré ne change pas, de sorte qu'un caractère instantané est attribué à chacune de ces boucles.
            Sauf dans le cas des erreurs de test élémentaire détectées dynamiquement, l'évaluation du niveau CurrentLevel comprend
            l'évaluation réussi d'une option. Viennent ensuite l'affectation d'un verdict et l'évaluation du niveau suivant et ainsi, par
            déduction, de tous les niveaux auxquels le contrôle est ensuite transféré. *)
begin
          if NOT IS_EXPANDED() then
            (*  Par cette condition, nous évitons de développer de façon répétitive les niveaux qui sont des cibles de constructions
                GOTO. *)
            EXPAND_CURRENT_LEVEL ();
          (*  Maintenant, le niveau courant est exempt de constructions REPEAT et de rattachements, et il comprend les valeurs par
              défaut nécessaires. *)
          repeat
            (*  ... exécutant des boucles au niveau courant, en essayant de faire concorder une option. *)
            TAKE_SNAPSHOT();
            (*  ... de la ou des files d'attente de points PCO et CP entrants, de la liste des fins de temporisation pertinente et de l'état
                de fin de toute autre composante de test. *)
            for /# every CurrentAlternative in CurrentLevel, in the given order #/          do
            (*  essai de mise en concordance avec l'option courante. Remarquer qu'une option visitée dans un arbre détermine une
                position de niveau dans l'arbre et représente, selon le contexte dans lequel elle est utilisée, une position à ce niveau,
                une ligne BehaviourLine, une ligne StatementLine, etc. *)
```

```
        begin
          if EVALUATE_EVENT_LINE (CurrentAlternative) then
          (*  En l'absence d'erreurs de test élémentaire, la composante de test ou le test élémentaire prend fin dans l'appel
              EVAL_VERDICT_ENTRY ou GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT de l'instance récursive la
              plus intérieure de la procédure EVALUATE_LEVELS, par exemple s'il y a un verdict final ou s'il n'y a pas de
              niveau suivant. Alors, la bouche FOR sera abandonnée également. *)
          begin
            if /# Alternative has a verdict column entry VerdictEntry #/ then
              EVAL_VERDICT_ENTRY(VerdictEntry);
            GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT(CurrentAlternative);
            EVALUATE_LEVELS();
          end
        end
        until SNAPSHOT_FIXED();
        (*  SNAPSHOT_FIXED renvoie la valeur TRUE si Snapshot ne peut plus changer. *)
        LOG(TEST_CASE_ERROR);
        STOP_TEST_CASE();
end
```

B.5.4.2 Exécution d'un test élémentaire ou d'une composante de test – Description en langage naturel

Etape 1 Dans l'arbre racine, l'évaluation commence au niveau d'indentation le plus bas numériquement (en notation TTCN.MP), c'est-à-dire le plus à gauche (en notation TTCN.GR).

Etape 2 Développer le niveau courant de façon à inclure explicitement toutes les valeurs par défaut, et à remplacer tous les rattachements à un arbre, dans la mesure du nécessaire, ainsi que toutes les constructions REPEAT, par leurs développements.

Etape 3 Prendre une image instantanée des files d'attente des points PCO et CP et de la liste des fins de temporisation.

NOTE 1 – Le fait de prendre une image instantanée ne supprime pas un événement de la file d'attente d'un point PCO ou CP.

Considérer la première ligne de comportement au niveau courant des options.

Etape 4 Evaluer la déclaration TTCN à la ligne de comportement courante.

L'évaluation de chaque type de déclaration TTCN est spécifiée dans la sémantique opératoire propre à ce type de déclaration.

Etape 5 Si l'évaluation de la déclaration TTCN aboutit à une concordance positive, passer à l'Etape 6.

Sinon, s'il reste d'autres options dans l'ensemble courant d'options, passer à la ligne de comportement suivante de cet ensemble et aller à l'Etape 4.

S'il n'existe pas d'autres options et que, cependant, toutes les files d'attente de points PCO et CP se rapportant à cet ensemble d'options contiennent au moins un événement et que tous les temporisateurs intervenant dans les déclarations de fin de temporisation de l'ensemble d'options figurent dans la liste de fins de temporisation, interrompre le test élémentaire et indiquer *test case error* (erreur de test élémentaire).

NOTE 2 – Dans ces conditions, aucune option de l'ensemble ne pourra jamais concorder.

Dans tous les autres cas, c'est-à-dire lorsqu'il n'y a plus d'options et que l'image instantanée suivante pourrait donner un portrait différent, passer à l'Etape 3.

Etape 6 Si un verdict préliminaire est codé, procéder comme en B.5.23.2.

Etape 7 Si un nœud feuille de l'arbre ou un nœud avec verdict final est atteint, passer à l'Etape 8**.**

Sinon, déterminer et prendre le niveau suivant à évaluer et passer à l'Etape 2.

Etape 8 Utiliser le verdict final, ou, s'il n'est pas spécifié, la valeur courante de la variable R du résultat préliminaire comme verdict final du test élémentaire, comme indiqué en B.5.23.2 et B.5.25.

B.5.5 Développement d'un ensemble d'options

B.5.5.1 Introduction

Le présent sous-paragraphe définit la marche à suivre pour développer un ensemble d'options en vue de déterminer quelle option concorde.

Cette opération est effectuée en quatre étapes:

a) sauvegarde du contexte par défaut, pour un niveau étiqueté;

b) rattachement de l'ensemble courant d'arbres de comportement par défaut;

c) développement des arbres rattachés, au besoin, de façon récursive, jusqu'à ce qu'il ne reste plus d'options de rattachement dans l'ensemble;

d) développement des constructions REPEAT, en les remplaçant par un sous-arbre dans lequel les rattachements à un arbre et les constructions GOTO apparaissent dans les niveaux inférieurs seulement.

```
procedure EXPAND_CURRENT_LEVEL ()
begin
    if /# CurrentLevel has a label #/ then
      SAVE_DEFAULTS ();
    APPEND_DEFAULTS ();
    EXPAND_ATTACHMENTS (EvaluationTree, CurrentLevel, Defaults);
    (*   Le niveau CurrentLevel est maintenant exempt de rattachements à un arbre. *)
    EXPAND_REPEATS ();
    /# Component IsExpanded of CurrentLevel #/ := TRUE;
end
```

B.5.5.2 Sauvegarde des valeurs par défaut

```
procedure SAVE_DEFAULTS ()
begin
        /# Replace CurrentLevel and its subsequent behaviour in the EvaluationTree by ACTIVATE (Defaults), followed by
CurrentLevel
        and its subsequent behaviour, with the label of the former CurrentLevel moved to the ACTIVATE line. #/;
        /# Consider new ACTIVATE line as the CurrentLevel #/;
end
```

B.5.5.3 Développement des constructions REPEAT

Si *RepeatedTree* indique une référence TreeReference particulière avec sa liste ActualParList, si *Condition* indique une expression booléenne particulière et si *label* indique une étiquette qui n'est utilisée nulle part ailleurs, alors "REPEAT *RepeatedTree* UNTIL [*Condition*]" peut être remplacé par:

```
              [TRUE]
label           +RepeatedTree
                        [NOT (Condition)]
                            -> label
                        [Condition]
                            :
```

Les lignes décrivant le comportement subséquent de la construction REPEAT suivent [*Condition*] dans ce développement, avec une indentation additionnelle de un niveau.

```
procedure EXPAND_REPEATS ()
begin
        for /# every alternative A in CurrentLevel, in the given order #/ do
        begin
          if /# A is of the form REPEAT RepeatedTree UNTIL [Condition] #/ then
          begin
            Subsequent := SUBSEQUENT_BEHAVIOUR_TO (EvaluationTree,A);
            Label := NEW_LABEL ();
            (*   Créer une étiquette qui n'a été utilisée ni dans la suite de tests (réétiquetée) ni dans l'arbre EvaluationTree. *)
            Expansion :=    MAKE_TREE ( "[TRUE]",
                        MAKE_TREE ( Label: "+" RepeatedTree,
                          MAKE_TREE ( "[NOT(" Condition ")]",
                                "->" Label,
                          MAKE_TREE ( "[" Condition "]",
                                  Subsequent,
                                  )),
                              ),
                          );
            REPLACE_ALT_TREE (EvaluationTree, CurrentLevel, A, Expansion);
          end
        end
end
```

B.5.5.4 Adjonction des comportements par défaut

Pendant l'évaluation d'un test élémentaire, il existe à chaque niveau d'options une liste courante de références à un arbre par défaut. Cette liste provient soit de la liste contenue dans la table de comportement dynamique appropriée, soit de la construction ACTIVATE la plus récemment évaluée. L'adjonction de valeurs par défaut s'effectue en ajoutant, pour chaque élément de la liste courante de valeurs par défaut, la construction "+ DefaultReference" à la fin de l'ensemble d'options.

```
procedure APPEND_DEFAULTS ()
begin
    for /# every D in Defaults, in the given order #/ do
    begin
      APPEND_TO_LEVEL (EvaluationTree, CurrentLevel, "+" D);
      (* L'arbre EvaluationTree et le niveau CurrentLevel sont mis à jour par l'adjonction du rattachement de D au niveau
         CurrentLevel. *)
    end
end
```

B.5.5.5 Développement des arbres rattachés

Le développement des arbres rattachés s'effectue en remplaçant d'abord la construction de rattachement + *TestStep* par l'arbre ou, le cas échéant, par l'arbre racine de la construction *TestStep*, puis, si un comportement est spécifié avec indentation à la suite de la construction Attach, en insérant avec indentation ce comportement après chaque feuille de l'arbre rattaché. Etant donné que la liste de références à un arbre par défaut propre aux arbres rattachés peut être contenue dans l'en-tête de la table de comportement dynamique d'un module de test, le développement du rattachement à un arbre doit être fait de telle façon que, s'il y a concordance d'un événement quelconque au premier niveau d'options de l'arbre rattaché, le contexte par défaut soit modifié, et si un nœud feuille de l'arbre rattaché est atteint sans qu'un verdict soit affecté, le contexte par défaut de l'arbre d'appel soit rétabli avant l'évaluation du comportement subséquent. Ces modifications du contexte par défaut sont décrites de la façon la plus simple lorsqu'on considère l'insertion de constructions ACTIVATE appropriées aux endroits pertinents. Si l'arbre rattaché est en réalité un arbre par défaut, il n'y aura pas de références par défaut dans l'en-tête, de sorte que les constructions ACTIVATE qui sont insérées au moment de l'entrée dans cet arbre n'auront aucun paramètre et désactiveront par conséquent toutes les valeurs par défaut dans la plage de la visibilité de l'arbre par défaut.

Le développement des arbres rattachés au niveau Level s'effectue en appliquant la procédure suivante:

```
procedure EXPAND_ATTACHMENTS (Tree, Level, OuterDefaults)
begin
    for /# every alternative A in Level in Tree, in the given order #/ do
    begin
      if /# A is an ATTACH construct, i.e. of the form "+" AttachedTreeId ActualParList #/ then
      begin
        Subsequent := SUBSEQUENT_BEHAVIOUR_TO (Tree,A);
        AttachedTree := ROOT_TREE (AttachedTreeId);
        REPLACE_PARAMETERS (AttachedTreeId, AttachedTree, ActualParList);
        (* Remplacement des paramètres formels de l'arbre AttachedTree par les paramètres effectifs spécifiés dans la liste
           ActualParList, par substitution textuelle *)
        RELABEL(AttachedTree);
        NewDefaults := DEF_REF_LIST(AttachedTreeId);
        NewLevel := FIRST_LEVEL(AttachedTree);
        EXPAND_ATTACHMENTS (AttachedTree, NewLevel, NewDefaults);
        EXPAND_SUBTREE (AttachedTree, Subsequent, NewDefaults, OuterDefaults);
        (* c'est-à-dire:  Insertion de ACTIVATE(NewDefaults) au-dessous du premier niveau de l'arbre AttachedTree et
                          rattachement de ACTIVATE(OuterDefaults) et de Subsequent à chaque nœud feuille de l'arbre
                          AttachedTree *)
        REPLACE_ALT_TREE(Tree, Level, A, AttachedTree);
      end
    end
end
```

```
procedure EXPAND_SUBTREE (SubTree, Subsequent, InnerDefaults, OuterDefaults)
        (*  Cette procédure insère d'abord ACTIVATE(InnerDefaults) au-dessous du premier niveau du sous-arbre SubTree, puis
            rattache ACTIVATE(OuterDefaults) et Subsequent à chaque nœud feuille du sous-arbre SubTree. *)
begin
        Level := FIRST_LEVEL(SubTree);
        for /# every alternative A of Level in SubTree #/ do
        begin
          SubOfA := SUBSEQUENT_BEHAVIOUR_TO (SubTree, A);
          ActTree := MAKE_TREE(A,
                MAKE_TREE("ACTIVATE(" InnerDefaults ")",
                    SubOfA, ), );
          REPLACE_ALT_TREE(SubTree, Level, A, ActTree);
        end
        for /# every leaf A in SubTree #/ do
        begin
          LeafTree :=   MAKE_TREE (A,
              MAKE_TREE ( "ACTIVATE(" OuterDefaults ")",
                    Subsequent, ), );
          REPLACE_ALT_TREE(SubTree, LEVEL_OF(SubTree, A), A, LeafTree);
        end
end
```

Le développement des arbres rattachés est aussi expliqué en 15.13.

B.5.6 Evaluation d'une ligne d'événement

B.5.6.1 Pseudo-code

```
function EVALUATE_EVENT_LINE(Alternative): BOOLEAN
        (*  Cette fonction appelle les fonctions EVALUATE_EVENT, EVALUATE_PSEUDO_EVENT ou
            EVALUATE_CONSTRUCT, selon le type de ligne StatementLine que constitue l'option courante. *)
begin
        case    STATEMENT_LINE_TYPE_OF(Alternative) of
        begin
          EVENT:          if  EVALUATE_EVENT (Alternative)          then return TRUE; else return FALSE;
          PSEUDO_EVENT:   if  EVALUATE_PSEUDO_EVENT (Alternative)   then return TRUE; else return FALSE;
            CONSTRUCT:    (*  Les seules constructions possibles sont maintenant GoTo, Return, Activate, Create. *)
                          if  EVALUATE_CONSTRUCT (Alternative)       then return TRUE; else return FALSE;
        end
end
```

B.5.6.2 Description en langage naturel

Evaluer la déclaration TTCN sur la ligne de comportement courante, d'après le type de déclaration, c'est-à-dire selon qu'il s'agit d'un événement, d'un pseudo-événement ou d'une construction. L'évaluation de chaque type de déclaration TTCN est spécifiée dans la sémantique opératoire pour ce type de déclaration TTCN dans les sous-paragraphes qui suivent.

B.5.7 Fonctions appliquées aux événements TTCN

B.5.7.1 Fonctions appliquées aux événements TTCN – Pseudo-code

```
function EVALUATE_EVENT(Alternative): BOOLEAN
        (*  Cette fonction appelle les événements SEND, RECEIVE, OTHERWISE, TIMEOUT, DONE ou IMPLICIT SEND,
            selon le type d'événement que constitue l'option courante. *)
begin
        case EVENT_TYPE_OF(Alternative) of
        begin
          SEND:           if  SEND (Alternative)           then return TRUE; else return FALSE;
          RECEIVE:        if  RECEIVE (Alternative)        then return TRUE; else return FALSE;
          OTHERWISE:      if  OTHERWISE (Alternative)      then return TRUE; else return FALSE;
          TIMEOUT:        if  TIMEOUT (Alternative)        then return TRUE; else return FALSE;
          DONE:           if  DONE (Alternative)           then return TRUE; else return FALSE;
          IMPLICIT_SEND:  if  IMPLICIT_SEND (Alternative)  then return TRUE; else return FALSE;
        end
end
```

B.5.7.2 Fonctions appliquées aux événements TTCN – Description en langage naturel

Si la déclaration TTCN est un événement, elle sera évaluée comme spécifié en B.5.8 s'il s'agit d'un événement SEND, en B.5.9 s'il s'agit d'un événement RECEIVE, en B.5.10 s'il s'agit d'un événement OTHERWISE, en B.5.11 s'il s'agit d'un événement TIMEOUT, en B.5.12 s'il s'agit d'un événement DONE, ou en B.5.13 s'il s'agit d'un événement IMPLICIT SEND.

B.5.8 Exécution de l'événement SEND

B.5.8.1 Exécution de l'événement SEND – Pseudo-code

```
function SEND ( SendLine ): BOOLEAN
begin
        /# Read   PCOorCPidentifier,
                  ASPorPDUorCMidentifier,
                  Qualifier,
                  Assignments,
                  TimerOperations,
                  ConstraintsReference        from SendLine #/;
        if EVALUATE_BOOLEAN (Qualifier) then
        begin
          BUILD_SEND_OBJECT (ASPorPDUorCMidentifier, ConstraintsReference );
          EXECUTE_ASSIGNMENTS (Assignment);
          SEND_EVENT (PCOorCPidentifier, ConstraintReference);
          TIMER_OPS (TimerOperations);
          LOG(PCOorCPidentifier, SendObject);
          return TRUE;
        end
        else return FALSE;
end

procedure BUILD_SEND_OBJECT (ASPorPDUorCMidentifier, ConstraintsReference)
begin
        SendObject := /#   an instance of ASPorPDUorCMidentifier whose parameters/fields have the values specified by
                           ConstraintsReference #/;
end

procedure SEND_EVENT (PCOorCPidentifier, ConstraintsReference)
begin
        /#  Encode SendObject according to applicable encoding rules and variations,
            see ConstraintsReference and associated type definitions #/;
        /#  Put encoded SendObject at the end of OUTPUT_Q(PCOorCPidentifier) #/;
end
```

B.5.8.2 Exécution de l'événement SEND – Description en langage naturel

Le contenu de la primitive ASP, de l'unité PDU ou du message CM, comme spécifié dans l'entrée de référence de contraintes nommées, doit être envoyé. On notera qu'en présence d'un qualificateur, l'événement SEND n'est exécuté que si ce qualificateur a la valeur TRUE.

Etape 1 S'il existe un qualificateur, il est évalué avant tout autre traitement:

- si le qualificateur prend la valeur FALSE, l'événement SEND n'est pas exécuté;
- si le qualificateur prend la valeur TRUE, passer à l'Etape 2.

Etape 2 Créer une primitive ASP, une unité PDU ou un message CM, comme spécifié dans la référence de contraintes nommées.

Si la caractéristique de chaînage dynamique a été utilisée, la valeur spécifiée dans l'entrée de référence de contraintes sera affectée au paramètre ou au champ correspondant de la primitive ASP, de l'unité PDU ou du message CM à envoyer.

L'utilisation de la caractéristique de chaînage dynamique a pour effet de mettre en mémoire une copie de la contrainte nommée dans le paramètre ou champ nommé de la primitive ASP, de l'unité PDU ou du message CM constitué à des fins de comparaison. La structure définie pour la référence aux contraintes associées est utilisée pour ce paramètre ou ce champ nommé.

Etape 3 S'il y a une déclaration d'affectation, cette affectation sera effectuée comme indiqué en B.5.16, en particulier en changeant peut-être la primitive ASP, l'unité PDU ou le message CM à envoyer.

Etape 4 La primitive ASP, l'unité PDU ou le message CM est maintenant entièrement renseigné conformément aux spécifications données. Le testeur inférieur (LT) ou le testeur supérieur (UT) code les unités PDU (mais pas les primitives ASP ni les messages CM, à part les unités PDU imbriquées dans ces éléments), conformément aux règles de codage applicables. Le testeur inférieur (LT) ou le testeur supérieur (UT) enverra la primitive ASP, avec ses unités PDU codées imbriquées, ou l'unité PDU codée. Si un point PCO ou CP a été déclaré, la primitive ASP, l'unité PDU ou le message CM doit être envoyé au niveau de ce point PCO ou CP. Si le point PCO n'a pas été déclaré, c'est-à-dire si le test utilise un seul point PCO, la primitive ASP ou l'unité PDU est envoyée à partir du point PCO inférieur, parce qu'on ne peut pas établir de point CP implicite.

Etape 5 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne de comportement, la ou les opérations de temporisation seront effectuées comme indiqué en B.5.17.

Etape 6 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées en B.5.24.2:

- le point PCO ou CP au niveau duquel l'événement SEND a eu lieu;
- la primitive ASP, l'unité PDU ou le message CM entièrement défini qui a été envoyé.

B.5.9 Exécution de l'événement RECEIVE

B.5.9.1 Exécution de l'événement RECEIVE – Pseudo-code

```
function RECEIVE( ReceiveLine ): BOOLEAN
begin
        /# Read   PCOorCPidentifier,
                  ASPorPDUorCMidentifier,
                  Qualifier,
                  Assignments,
                  TimerOperations,
                  ConstraintsReference        from ReceiveLine #;
        if /# INPUT_Q (PCOorCPidentifier) is not empty #/ then
        begin
          if ( OBJECT_MATCHES(PCOorCPidentifier, ASPorPDUorCMidentifier, ConstraintsReference)
              AND EVALUATE_BOOLEAN (Qualifier) ) then
          begin
              EXECUTE_ASSIGNMENTS (Assignments);
              TIMER_OPS (TimerOperations);
              REMOVE_OBJECT (PCOorCPidentifier);
              LOG(PCOorCPidentifier, ReceiveObject);
              return TRUE;
          end
          else return FALSE;
        end
        else return FALSE;
end

function OBJECT_MATCHES (PCOorCPidentifier, ASPorPDUorCMidentifier, ConstraintsReference): BOOLEAN
begin
        ReceiveObject := /# copy of encoded object at head of INPUT_Q(PCOorCPidentifier) #/;
        if /#  ReceiveObject can be decoded according to applicable encoding rules and variations,
               as given by ConstraintsReference and associated type definitions #/ then
        begin
          /# decode it, to yield new version of ReceiveObject #/;
          if   ( /# ReceiveObject is of type ASPorPDUorCMidentifier #/
                 AND
                 /# parameters/fields of ReceiveObject have values matching the ConstraintsReference #/ ) then
               return TRUE;
          else return FALSE;
        end
        else return FALSE;
end

procedure REMOVE_OBJECT (PCOorCPidentifier),
begin
  /# remove object at head of INPUT_Q(PCOorCPidentifier) #/;
end
```

B.5.9.2 Exécution de l'événement RECEIVE – Description en langage naturel

Etape 1 Si l'image instantanée prise au début de l'itération courante du contrôle de concordance pour ce niveau d'options montre qu'il *n'y a pas* de primitive ASP, d'unité PDU ou de message CM entrant, cet événement RECEIVE ne peut pas être mis en concordance.

Sinon, passer à l'Etape 2.

Etape 2 Si un point PCO ou CP a été déclaré, la primitive ASP, l'unité PDU ou le message CM aura été reçu au niveau de ce point PCO ou CP. Si le point PCO n'a pas été déclaré, c'est-à-dire si la suite de tests utilise un seul point PCO, la primitive ASP ou l'unité PDU aura été reçue au niveau de ce point PCO inférieur. Remarquer qu'on ne peut pas établir de point CP implicite.

Etape 3 Les unités PDU entrantes sont décodées conformément aux règles de codage applicables. Une copie est faite de l'unité PDU entrante décodée, ou de la primitive ASP entrante ou du message CM entrant avec des unités PDU emboîtées décodées.

Etape 4 Si le qualificateur, éventuellement en utilisant des valeurs de l'objet de données entrant, prend la valeur FALSE, l'événement RECEIVE ne peut pas concorder. Sinon, passer à l'Etape 5.

Etape 5 Une copie du modèle de la primitive ASP, de l'unité PDU ou du message CM prévu est constituée, en utilisant la structure définie dans la déclaration de primitive ASP, d'unité PDU ou de message CM plus les valeurs, mécanismes de concordance et références aux contraintes chaînées spécifiées dans la référence aux contraintes nommées.

On compare cette copie à la primitive ASP, à l'unité PDU ou au message CM entrant, et à ses unités PDU décodées ou à l'unité PDU décodée, en vue de déterminer si l'événement RECEIVE peut concorder comme spécifié. Passer à l'Etape 6 seulement si l'événement RECEIVE ne concorde pas.

Etape 6 La primitive ASP, l'unité PDU ou le message CM entrant qui vient d'être mis en concordance sera retiré de la file d'attente du point PCO ou CP entrant et sera ignoré.

Etape 7 S'il y a des déclarations d'affectation, elles seront exécutées comme indiqué en B.5.16.2.

Etape 8 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne de comportement, elles seront exécutées comme indiqué en B.5.17.

Etape 9 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées en B.5.24.2:

- le point PCO ou CP au niveau duquel l'événement RECEIVE a eu lieu;
- la primitive ASP, l'unité PDU ou le message CM reçu, avec sa définition complète.

B.5.10 Exécution de l'événement OTHERWISE

B.5.10.1 Exécution de l'événement OTHERWISE – Pseudo-code

```
function OTHERWISE ( OtherwiseLine): BOOLEAN
begin
        /# Read   PCOorCPidentifier,
                  Qualifier,
                  Assignments,
                  TimerOperations          from OtherwiseLine #/;
        if   ( /# INPUT_Q (PCOorCPidentifier) is not empty #/
               AND EVALUATE_BOOLEAN (Qualifier) ) then
        begin
          EXECUTE_ASSIGNMENTS (Assignments);
          TIMER_OPS (TimerOperations);
          REMOVE_OBJECT (PCOorCPidentifier);
          LOG(PCOidentifier, ReceivedObject);
          return TRUE;
        end
        else return FALSE;
end
```

B.5.10.2 Exécution de l'événement OTHERWISE – Description en langage naturel

Le testeur acceptera toute donnée entrante qu'il a été impossible de décoder ou qui n'a pas concordé avec une option antérieure à cet événement OTHERWISE. On notera qu'en présence d'un qualificateur, l'événement OTHERWISE ne concorde que si le qualificateur a la valeur TRUE.

Etape 1 Si le qualificateur a la valeur FALSE, l'événement OTHERWISE ne peut concorder. Sinon, passer à l'Etape 2.

Etape 2 Si l'image instantanée prise au début de l'itération courante du contrôle de concordance pour ce niveau d'options montre qu'il n'y a pas de primitive ASP, d'unité PDU ou de message CM entrant, cet événement OTHERWISE ne peut pas être mis en concordance.

Sinon, passer à l'Etape 3.

Etape 3 Si un point PCO a été déclaré, la primitive ASP ou l'unité PDU aura été reçue au niveau de ce point PCO. Si un point CP a été déclaré, le message CM aura été reçu au niveau de ce point CP. Si le point PCO n'a pas été déclaré, c'est-à-dire si le test utilise un seul point PCO, la primitive ASP ou l'unité PDU aura été reçue au niveau du point PCO inférieur, parce qu'on ne peut pas établir de point CP implicite.

Etape 4 La primitive ASP, l'unité PDU ou le message CM entrant sera retiré de la file d'attente du point PCO ou CP entrant et sera ignoré.

Etape 5 S'il y a des déclarations d'affectation, elles seront exécutées comme indiqué en B.5.16.2.

Etape 6 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne de comportement, elles seront exécutées comme indiqué en B.5.17.

Etape 7 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées en B.5.24.2:

- le point PCO ou CP au niveau duquel l'événement OTHERWISE a eu lieu;
- la primitive ASP, l'unité PDU ou le message CM reçu.

B.5.11 Exécution de l'événement TIMEOUT

B.5.11.1 Exécution de l'événement TIMEOUT – Pseudo-code

```
function TIMEOUT ( TimeoutLine ): BOOLEAN
begin
        /# Read   TimerIdentifier,
                  Qualifier,
                  Assignments,
                  TimerOperations          from TimeoutLine #/;
        if EVALUATE_BOOLEAN (Qualifier) then
        begin
          if TIMER_EXPIRED (TimerIdentifier) then
          begin
            EXECUTE_ASSIGNMENTS (Assignments);
            TIMER_OPS (TimerOperations);
            LOG(TimerIdentifier);
            return TRUE;
          end
          else return FALSE;
        end
        else return FALSE;
end
```

```
function TIMER_EXPIRED (TimerIdentifier): BOOLEAN
begin
        if /# TimerIdentifier is not empty #/ then
        begin
          if /# timeout notification from TimerIdentifier is in copy of timeout list in Snapshot #/ then
          begin
            /# delete timeout notification from TimerIdentifier in actual timeout list #/;
            /# stop and reset the timer TimerIdentifier #/;
          return TRUE;
          end
          else return FALSE;
        end
        else (* L'identificateur TimerIdentifier n'est pas spécifié *)
        begin
          if /# any timeout notification is in copy of timeout list in Snapshot #/ then
          begin
            /# stop and reset all timers mentioned in actual timeout list#/;
            /# delete all timeout notifications in actual timeout list #/;
            return TRUE;
          end
          else return FALSE;
        end
end
```

B.5.11.2 Exécution de l'événement TIMEOUT – Description en langage naturel

Le testeur vérifiera si la temporisation nommée est arrivée à expiration. (Si aucun nom de temporisateur n'est donné, le testeur vérifiera si une temporisation *quelconque* est arrivée à expiration.) On notera qu'en présence d'un qualificateur, l'événement TIMEOUT ne concordera que si ce qualificateur a la valeur TRUE.

Etape 1 S'il y a un qualificateur, il sera évalué avant tout autre traitement.

- Si le qualificateur a la valeur FALSE, l'événement TIMEOUT ne concorde pas.
- Si le qualificateur a la valeur TRUE, passer à l'Etape 2.

Etape 2 Voir si l'une quelconque des temporisations explicitement ou implicitement nommées dans l'événement TIMEOUT est arrivée à expiration après déclenchement.

- Si aucun identificateur de temporisateur n'est spécifié, le testeur recherchera une fin de temporisation *quelconque*. Dans l'affirmative, tous les temporisateurs arrivés à expiration sont réinitialisés (et laissés à l'arrêt). Les entrées de fin de temporisation sont supprimées de la liste de fins de temporisation.
- Si un identificateur de temporisateur est spécifié, le testeur vérifiera si ce temporisateur a été déclenché et s'il est arrivé à expiration. Si oui, ce temporisateur est réinitialisé (et laissé à l'arrêt). L'entrée de fin de temporisation est supprimée de la liste de fins de temporisation.
- Si aucun temporisateur n'est arrivé à expiration, l'événement TIMEOUT ne peut pas concorder, c'est-à-dire que le système passe à l'option suivante.

Etape 3 S'il y a une déclaration d'affectation, elle est exécutée comme indiqué en B.5.16.2.

Etape 4 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne de comportement, ces opérations sont exécutées comme indiqué en B.5.17.

Etape 5 Consigner dans le journal de conformité les informations spécifiées en B.5.24, ainsi que le nom du temporisateur qui est arrivé à expiration.

B.5.12 Exécution de l'événement DONE

B.5.12.1 Exécution de l'événement DONE – Pseudo-code

```
function DONE (DoneLine): BOOLEAN
begin
        /# Read   TCompList,
                  Qualifier,
                  Assignments,
                  TimerOperations          from DoneLine #/;
        if EVALUATE_BOOLEAN (Qualifier) AND      ALL_TERMINATED(TCompList) then
        begin
          EXECUTE_ASSIGNMENTS (Assignments);
          TIMER_OPS (TimerOperations);
          LOG(TCompList);
          return TRUE;
        end
        else return FALSE;
end

function ALL_TERMINATED(TCompList): BOOLEAN
begin
        if   TCompList =/# EmptyList #/ then
             TCompList := /# list of all created Parallel Test Components #/;
        for /# every TComp in TCompList #/ do
        begin
          if /# TComp has not terminated in the Snapshot #/  then
            return FALSE;
        end
        return TRUE;
end
```

B.5.12.2 Exécution de l'événement DONE – Description en langage naturel

L'état de fin de la liste de composantes de test donnée doit être vérifié. Si toutes les composantes données ont pris fin (au moment de la dernière image SNAPSHOT), l'événement concorde, à condition que le qualificateur ait aussi la valeur TRUE.

Etape 1 S'il y a un qualificateur, il sera évalué avant tout autre traitement.

- Si le qualificateur a la valeur FALSE, l'événement DONE ne peut pas être exécuté avec succès.
- Si le qualificateur a la valeur TRUE, passer à l'Etape 2.

Etape 2 Si toutes les composantes de test figurant dans la liste TCompList avaient pris fin au moment de la dernière image SNAPSHOT, passer à l'Etape 3; sinon, cet événement DONE ne peut pas concorder.

Etape 3 S'il y a une déclaration d'affectation, elle est exécutée comme indiqué en B.5.16.

Etape 4 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne de comportement, ces opérations sont exécutées comme indiqué en B.5.17.

Etape 5 Consigner dans le journal de conformité les informations spécifiées en B.5.24, ainsi que la liste TCompList.

B.5.13 Exécution de l'événement IMPLICIT SEND

B.5.13.1 Exécution de l'événement IMPLICIT SEND – Pseudo-code

```
function IMPLICIT_SEND (Alternative): BOOLEAN
begin
        /# Execute IMPLICIT_SEND according to natural language description #/;
        return TRUE;
end
```

B.5.13.2 Exécution de l'événement IMPLICIT SEND – Description en langage naturel

L'implémentation sous test (IUT) est amenée à faire le nécessaire pour envoyer le contenu de la primitive ASP ou de l'unité PDU, comme spécifié dans l'entrée de référence aux contraintes de l'option.

Si la caractéristique de chaînage dynamique a été utilisée, la valeur spécifiée dans l'entrée de référence aux contraintes sera affectée au paramètre ou champ approprié de la primitive ASP ou de l'unité PDU à envoyer.

L'événement IMPLICIT SEND est toujours exécuté avec succès.

B.5.14 Exécution d'un pseudo-event

B.5.14.1 Exécution d'un pseudo-event – Pseudo-code

```
function EVALUATE_PSEUDO_EVENT ( PseudoEventLine ): BOOLEAN
begin
        /# Read   Qualifier,
                  Assignments,
                  TimerOperations          from PseudoEventLine #/;
        if EVALUATE_BOOLEAN (Qualifier) then
        begin
          EXECUTE_ASSIGNMENTS (Assignments);
          TIMER_OPS (TimerOperations);
          LOG( );
          return TRUE;
        end
        else return FALSE;
end
```

B.5.14.2 Exécution de PSEUDO-EVENTS – Description en langage naturel

Si la déclaration TTCN est un pseudo-événement, elle sera évaluée comme spécifié en B.5.15 s'il s'agit d'une expression booléenne, en B.5.16 s'il s'agit d'une déclaration d'affectation, en B.5.17 s'il s'agit d'une opération de temporisation (START, CANCEL ou READTIMER).

Après l'achèvement du pseudo-événement, consigner dans le journal de conformité les informations spécifiées en B.5.24.

B.5.15 Exécution des expressions BOOLEAN (booléennes)

B.5.15.1 Exécution des expressions BOOLEAN – Pseudo-code

```
function EVALUATE_BOOLEAN(Qualifier): BOOLEAN
begin
        if /# Qualifier is empty #/ then
          return TRUE;
        else
        begin
          if /# Qualifier evaluates to TRUE #/ then
            return TRUE;
          else return FALSE;
        end
end
```

B.5.15.2 Exécution des expressions BOOLEAN – Description en langage naturel

Une expression booléenne (c'est-à-dire un qualificateur) spécifie une condition qui doit être testée. Cette condition est soit vraie (TRUE), soit fausse (FALSE). Une expression booléenne peut être déclarée comme partie de ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, TIMEOUT ou OTHERWISE) ou comme ligne de déclaration à part entière (c'est-à-dire comme pseudo-événement).

Etape 1 L'expression booléenne est évaluée pour déterminer si la condition spécifiée est TRUE ou FALSE. Les règles normales de la logique booléenne s'appliquent, ainsi que les règles de priorité spécifiées en 11.4.2.1.

B.5.16 Exécution des ASSIGNMENTS

B.5.16.1 Exécution des ASSIGNMENTS – Pseudo-code

```
procedure EXECUTE_ASSIGNMENTS (AssignmentList)
begin
        for /# every assignment CurrentAssignment in AssignmentList, in the given order #/ do
        begin
          /# Execute CurrentAssignment #/;
        end
end
```

B.5.16.2 Exécution des ASSIGNMENTS – Description en langage naturel

La liste des affectations est évaluée de la gauche vers la droite. Dans chaque affectation, la variable du membre gauche de la déclaration doit prendre la valeur de l'expression du membre droit de cette déclaration. Cette expression est évaluée en respectant la priorité indiquée dans le Tableau 3.

Si l'affectation est exécutée dans une ligne Send, le membre gauche peut représenter une composante de primitive ASP, d'unité PDU ou de message CM, faisant référence à l'objet à envoyer. Si l'affectation est exécutée dans une ligne Receive, l'expression peut faire référence à des composantes de la primitive ASP, de l'unité PDU ou du message CM à recevoir.

B.5.17 Exécution des opérations TIMER

B.5.17.1 Exécution des opérations TIMER – Pseudo-code

```
procedure TIMER_OPS (TimerOperations)
begin
        for /# every TimerOperation in TimerOperations #/ do
        case TIMER_OP_TYPE_OF(TimerOperation) of
        begin
          START_TIMER:          START_TIMER(TimerOperation);
          CANCEL_TIMER:         CANCEL_TIMER(TimerOperation);
          READ_TIMER:           READ_TIMER(TimerOperation);
        end
end

procedure START_TIMER (TimerOperation)
begin
        /# perform as in B.5.17.2 #/;
end

procedure CANCEL_TIMER (TimerOperation)
begin
        /# perform as in B.5.17.3 #/;
end

procedure READ_TIMER (TimerOperation)
begin
        /# perform as in B.5.17.4 #/;
end
```

B.5.17.2 Exécution de l'opération START_TIMER – Description en langage naturel

Etape 1 Si le temporisateur est déjà déclenché, annuler la temporisation et passer à l'Etape 2. Sinon, passer directement à l'Etape 2.

Etape 2 La temporisation doit être déclenchée avec une valeur initiale indiquant qu'aucun temps ne s'est écoulé. Toute entrée relative à ce temporisateur dans la liste des fins de temporisation est supprimée de la liste.

B.5.17.3 Exécution de l'opération CANCEL_TIMER – Description en langage naturel

L'opération CANCEL_TIMER spécifie l'arrêt d'un ou de plusieurs temporisateurs.

Etape 1 Déterminer le nom du ou des temporisateurs à annuler:

- si aucun identificateur de temporisation n'est spécifié, annuler *toutes* les temporisations;
- si un identificateur de temporisation est spécifié, annuler la temporisation correspondant à cet identificateur.

Etape 2 Le descripteur d'état du ou des temporisateurs nommés ou impliqués est mis à "not running" (arrêté). Le ou les temporisateurs sont réinitialisés à zéro. Si la liste de fins de temporisation contient une ou plusieurs entrées concernant ce ou ces temporisateurs, ces entrées sont supprimées de la liste.

B.5.17.4 Exécution de l'opération READ_TIMER – Description en langage naturel

L'opération READ_TIMER spécifie que le temps compté par un temporisateur déclenché doit être mémorisé dans une variable. Le temporisateur n'est pas interrompu.

Etape 1 Lire la valeur du temporisateur dont le nom est spécifié. Si le temps écoulé correspond à *n* des unités déclarées pour ce type de temporisateur, mémoriser *n* dans la variable nommée.

Si le temporisateur est à l'arrêt, la variable nommée sera mise à zéro.

B.5.18 Fonctions appliquées aux constructions TTCN

B.5.18.1 Fonctions appliquées aux constructions TTCN – Pseudo-code

```
function EVALUATE_CONSTRUCT (Construct): BOOLEAN
        (* Etant donné que l'arbre EvaluationTree est développé au niveau CurrentLevel, les constructions REPEAT et ATTACH
           sont absentes dans ce cas-ci. *)
begin
        case CONSTRUCT_TYPE_OF(Construct) of
        begin
          ACTIVATE:     ACTIVATE(Construct);
          CREATE:       CREATE (Construct);
          GOTO:         (*  pas d'action, voir GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT *);
          RETURN:       (*  pas d'action, voir GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT *);
        end
        return TRUE;
end
```

B.5.18.2 Fonctions appliquées aux constructions TTCN – Description en langage naturel

Si la déclaration TTCN est une construction TTCN, elle sera évaluée comme spécifié en B.5.19 s'il s'agit d'une construction ACTIVATE, comme spécifié en B.5.20 s'il s'agit d'une construction CREATE, comme spécifié en B.5.21 s'il s'agit d'une construction GOTO, ou comme spécifié en B.5.22 s'il s'agit d'une construction RETURN. On n'a pas à traiter de constructions REPEAT, car elles ont toutes été remplacées au niveau CurrentLevel.

Les constructions TTCN seront toujours exécutées avec succès.

B.5.19 Exécution de la construction ACTIVATE

B.5.19.1 Exécution de la construction ACTIVATE – Pseudo-code

```
procedure ACTIVATE ( ActivateLine )
begin
    /#  Read     DefRefList    from ActivateLine #/;
    Defaults:=DefRefList;
    LOG(DefRefList);
end
```

B.5.19.2 Exécution de la construction ACTIVATE – Description en langage naturel

Remplacer le contexte par défaut courant par la liste DefaultRefList qui apparaît comme paramètre de la construction ACTIVATE.

Etape 1 Remplacer le contexte par défaut par DefaultRefList.

Etape 2 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées en B.5.24:

- la liste DefaultRefList.

B.5.20 Exécution de la construction CREATE

B.5.20.1 Exécution de l'événement CREATE – Pseudo-code

```
procedure CREATE ( CreateLine ): BOOLEAN
begin
        /#  Read   CreateList    from CreateLine #/;
        for /# every (TCompIdentifier,    TreeReference,    ActualParList) drawn from CreateList #/ do
        begin
          start process EVALUATE_TEST_COMPONENT(TCompIdentifier, TreeReference, ActualParList);
          (*  Lancement de l'évaluation concomitante de la référence TreeReference. *)
          LOG(TCompIdentifier,TreeReference, ActualParList);
        end
end

process EVALUATE_TEST_COMPONENT(TCompId, TreeReference, ActualParList)
        (*  Ce processus initialise l'arbre EvaluationTree à partir de l'arbre racine de module de test approprié ou de l'arbre local et
            le contexte par défaut à partir des références de comportement par défaut figurant dans la table de comportement
            correspondante. Il transfère le contrôle au niveau le plus élevé des options et appelle leur évaluation. *)
global EvaluationTree, CurrentLevel, Defaults, Snapshot, ReturnLevel, ReturnDefaults, SendObject, ReceiveObject;
begin
   /#  Initialize the local instances of Test Case Variables, local R, Timers, and the Timeout List of TCompId. #/;
   EvaluationTree := ROOT_TREE(TreeReference);
     (*  L'arbre EvaluationTree est un arbre fini croissant construit en insérant ensemble et en étendant des copies d'arbres provenant
         de la description de comportement d'un test élémentaire et des bibliothèques des modules de test et des comportements par
         défaut. Une composante IsExpanded est ajoutée à chaque niveau. *)
   REPLACE_PARAMETERS (TreeReference, EvaluationTree, ActualParList);
     CurrentLevel := FIRST_LEVEL(EvaluationTree) ;
     (*  Un niveau représente à la fois une position dans un arbre et l'ensemble ordonné d'options à cette position. *)
   ReturnLevel := CurrentLevel;
   Defaults := DEF_REF_LIST(TreeReference);
   ReturnDefaults := Defaults;
   EVALUATE_LEVELS ();
   (*  Y compris, par appels emboîtés, l'évaluation de tous les niveaux pertinents subséquents de l'arbre d'évaluation croissant. *)
end
```

B.5.20.2 Exécution de l'événement CREATE – Description en langage naturel

L'évaluation de la composante de test donnée doit être lancée.

Etape 1 L'évaluation de l'identificateur TCompIdentifier, lié à la référence TreeReference, est lancée, les paramètres de la liste ActualParList remplaçant les paramètres formels par substitution textuelle dans la référence TreeReference. Toutes les variables du test élémentaire, la variable de résultat local R, les temporisateurs et la liste de fins de temporisation locale sont fournis de nouveau uniquement pour fin d'utilisation par cette composante de test.

Etape 2 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées en B.5.24:

- l'identificateur TCompIdentifier;
- la référence TreeReference;
- la liste ActualParList.

B.5.21 Exécution de la construction GOTO

Le contrôle est transféré à l'ensemble d'options dont la colonne des étiquettes contient l'étiquette cible spécifiée. L'exécution se poursuit alors à ce nouveau niveau.

En pseudo-code, la construction GOTO est exécutée en tant que partie intégrante de la procédure GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT.

B.5.22 Exécution de la construction RETURN

Le contrôle est transféré à l'ensemble d'options à partir duquel les valeurs par défaut ont été entrées la dernière fois. L'exécution se poursuit alors à ce nouveau niveau.

En pseudo-code, la construction RETURN est exécutée en tant que partie intégrante de la procédure GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT.

B.5.23 Verdict

B.5.23.1 Verdict – Pseudo-code

```
procedure EVAL_VERDICT_ENTRY (VerdictEntry)
begin
        /# Expand VerdictEntry to full word, e.g. (P) becomes (PASS) #/;
        if /# VerdictEntry is a preliminary verdict "("PrelimVerdict")" #/ then
        begin
          UPDATE_PRELIM ( PrelimVerdict, /# local R, or MTC_R in case of Main Test Component #/);
          UPDATE_PRELIM ( PrelimVerdict, /# global R #/);
        end
        else (*  L'entrée VerdictEntry est un verdict final. *)
        begin
          if /# Current process is EVALUATE_TEST_CASE #/ then
          begin
            EXCLUDE_INCOMPATIBLE_ENTRY ( VerdictEntry, /# global R #/);
            LOG(VerdictEntry);
            /# assign final verdict in main test component or test case #/;
            TERMINATE_TEST_CASE();
          end
          else (*  Le processus est EVALUATE_TEST_COMPONENT *)
          begin
            EXCLUDE_INCOMPATIBLE_ENTRY ( VerdictEntry, /# local R #/);
            UPDATE_PRELIM          ( VerdictEntry, /# global R #/);
            stop process;
          end
        end
end

process EXCLUDE_INCOMPATIBLE_ENTRY (Entry, RVal)
begin
        if ( ( Entry = "R" AND /# RVal = none #/ ) OR
             (Entry = "PASS" AND /# Rval = inconc #/ ) OR
             (Entry = "PASS" AND /# Rval = fail #/ ) OR
             (Entry = "INCONC" AND /# Rval = fail #/ ) ) then
        begin
          LOG(TestCaseError);
          STOP_TEST_CASE();
          return FALSE;
        end
        else return TRUE;
end

procedure UPDATE_PRELIM (PrelimVerdict, ResultVar)
begin
        if (  ResultVar = none OR
             (ResultVar = pass AND PrelimVerdict <> PASS) OR
             (ResultVar = inconc AND PrelimVerdict = FAIL) ) then
        begin
          /# replace value of ResultVar by PrelimVerdict in lower case letters #/;
          LOG("("PrelimVerdict")");
        end
end
```

B.5.23.2 VERDICT – Description en langage naturel

Si un verdict est codé, l'exécuter:

- si le verdict est préliminaire, c'est-à-dire mis entre parenthèses, les variables de résultat local et de résultat global seront mises à jour conformément à l'algorithme de verdict du sous-paragraphe 15.17.2. Remarquer que dans la composante de test principale, la variable locale R est représentée par MTC_R. Le verdict déclaré est consigné dans le journal de conformité;

- si le verdict est R, la valeur courante de la variable R (la seule variable R ou la variable R globale) sera utilisée, dans la notation TTCN non concomitante ou dans la composante de test principale, comme verdict du test élémentaire. Si R est mis à la valeur "néant", signaler une erreur de test élémentaire;
- si le verdict est PASS (succès), INCONC (non concluant) ou FAIL (échec), le verdict déclaré sera pris, dans la notation TTCN non concomitante ou dans la composante de test principale, comme verdict final du test élémentaire. Si le verdict final contredit la variable R locale ou globale, signaler une erreur TestCaseError;
- dans les composantes de test parallèles, un verdict final R, PASS, INCONC ou FAIL est utilisé pour mettre à jour la variable globale R comme un verdict préliminaire. Le verdict déclaré est consigné dans le journal de conformité. Un verdict final met fin à l'évaluation de la composante de test.

B.5.24 Journal de conformité (log)

B.5.24.1 Journal LOG – Pseudo-code

```
procedure LOG( /# any number of arguments #/ )
begin
        /# log the line number of the event line (if any) #/;
        /# log the label associated with the event line (if any) #/;
        /# log the arguments passed to LOG #/;
        /# log the assignment(s) made (if any) #/;
        /# log the timer operation(s) performed (if any) #/;
        /# log current time #/; (*  temps courant, absolu ou relatif *)
end
```

B.5.24.2 Journal de conformité – Description en langage naturel

Consigner les informations suivantes dans le journal de conformité:

- numéro de la ligne d'événement (le cas échéant);
- étiquette associée à la ligne d'événement (le cas échéant);
- autres arguments définis ailleurs dans la présente annexe associés à la ligne d'événement (le cas échéant), par exemple le verdict final ou préliminaire, ou l'objet de données envoyé ou reçu;
- affectations effectuées (le cas échéant);
- opérations de temporisation exécutées (le cas échéant);
- horodatage.

B.5.25 Fonctions et procédures de traitement d'arbre

Afin de faciliter la recherche, les procédures et fonctions sont définies par ordre alphabétique.

```
procedure APPEND_TO_LEVEL (Tree,Level,Alternative)
begin
        /# Update Level and Tree by appending Alternative as new last alternative in Level in Tree #/;
end

function FIRST_LEVEL (Tree): LEVEL
begin
        return  /# the set of alternatives at the first level of indentation of Tree, i.e. the numerically lowest (in TTCN.MP),
                i.e. the leftmost (in TTCN.GR), level of indentation of the root tree #/;
end
```

```
procedure GOTO_NEXT_LEVEL_OR_STOP_WITH_VERDICT(Alternative)
begin
        (*  Chercher le niveau suivant à évaluer, le cas échéant. *)
        if /# Alternative is of the type "GOTO Label" or "-> Label" #/ then
          CurrentLevel := /# the unique level labelled with Label #/;
        else if /# Alternative is of the type "RETURN" #/ then
        begin
          CurrentLevel := ReturnLevel;
          Defaults := ReturnDefaults;
        end
        else if /# Alternative is a leaf of EvaluationTree #/; (*  mais pas une construction RETURN ou GOTO *) then
          EVAL_VERDICT_ENTRY("R"); (*  Arrêt de l'exécution du processus. *)
        else
          CurrentLevel := /# set of alternatives at next level of indentation below Alternative #/;
        (*  Sauvegarder les informations pour les déclarations RETURN à venir. *)
        if /# Component IsDefault of CurrentLevel #/ = FALSE then
        begin
          ReturnLevel := CurrentLevel;
          ReturnDefault := Default;
        end
end

function IS_EXPANDED (): BOOLEAN
begin
        return /# Component IsExpanded of CurrentLevel #/;
end

function LEVEL_OF (Tree, Alternative): LEVEL
begin
        return /# the level in Tree of which this Alternative is a member #/;
end

function MAKE_TREE (Statement, Tree1, Tree2): TREE
begin
        return /# the following tree:
            Statement
              Tree1
            Tree2                                #/   ;
        (*  L'arbre Tree1 et l'arbre Tree2 peuvent être vides, ce qui est représenté par une position de paramètre vide dans l'appel de
            MAKE_TREE. *)
end

function NEW_LABEL ( ): LABEL
begin
        return /# a label which has not yet been used in the execution of this Test Component, nor in the (relabelled) Test Suite #/ ;
              (*  Réalisable au moyen de compteurs et de noms de composantes de test *)
end

procedure RELABEL (Tree)
begin
        for /# each label L originally occurring in Tree #/ do
          begin
            NewLabel := NEW_LABEL();
            for /# each occurrence of L in Tree, in the label column or as the target of a GOTO #/ do
              begin
                /# replace L by NewLabel #/;
              end
          end
end

procedure REPLACE_ALT_TREE (Tree, Level, A, ReplacementTree)
begin
        (*  A est une option du niveau Level, lequel est un niveau de l'arbre Tree *)
        /#  In Tree, replace the subtree of Tree consisting of
            A and SUBSEQUENT_BEHAVIOUR_TO (Tree, A) by ReplacementTree,
            with all values of IsDefault in ReplacementTree set to the IsDefault-value of A,
            and all values of IsExpanded of levels in ReplacementTree set to FALSE. #/;
end
```

```
procedure REPLACE_PARAMETERS (TreeId, Tree, ActualParList)
begin
        /#  Replace the formal parameters in Tree by the actual parameters specified in ActualParList,
            doing so by textual substitution in Tree, using the formal parameter list accessible via TreeId. #/;
end

function ROOT_TREE (TreeId): TREE
begin
        return  /#  its root tree if TreeId denotes a Test Case or Test Step or Default Behaviour Table –
                    otherwise the local tree with this name. Each level gets a new Boolean component
                    "IsExpanded", initialized with value FALSE, indicating that this level has not yet been expanded. #/;
end

function SUBSEQUENT_BEHAVIOUR_TO (Tree, Alternative): TREE
begin
        return /# the subtree below Alternative in Tree #/;
        (*  Il s'agirait de l'arbre Tree3 si l'arbre Tree a la forme:
            Tree1
              Tree2
              Alternative
                Tree3
              Tree4
            Tree5                       *)
end
```

B.5.26 Fonctions diverses utilisées par le pseudo-code

```
function CONSTRUCT_TYPE_OF(Construct): CONSTRUCT_TYPE
begin
        return /# ACTIVATE, CREATE, GOTO, or RETURN, as appropriate #/;
end

function DEF_REF_LIST(TreeReference): DEFAULT_REF_LIST
begin
        return /# the default reference list in the header of the corresponding table in the case of a test step in the test step library, or
      the empty list in the case of default behaviour, or in the case of a local tree attachment the current value of Defaults (i.e. the
      currently active defaults in the calling tree) #/;
end

function EVENT_TYPE_OF(Alternative): EVENT_TYPE
begin
        return /# SEND, RECEIVE, OTHERWISE, TIMEOUT, DONE, or IMPLICIT_SEND, as appropriate #/;
end

function INPUT_Q(PCOorCPidentifier): QUEUE
begin
        if /# PCOorCPidentifier is empty #/ then
          return /# default PCO input queue #/;
        else return /# input queue identified by PCOorCPidentifier #/;
end

function OUTPUT_Q(PCOorCPidentifier): QUEUE
begin
        if /# PCOorCPidentifier is empty #/ then
          return /# default PCO output queue #/;
        else return /# output queue identified by PCOorCPidentifier #/;
end

function SNAPSHOT_FIXED (): BOOLEAN
begin
        if /# all relevant PCO and CP queue(s) have some event(s) on them and all relevant timers have expired #/ then
          return TRUE;
        else return FALSE;
end

function STATEMENT_LINE_TYPE_OF(Alternative): STATEMENT_LINE_TYPE
begin
        return /# EVENT, PSEUDO_EVENT, or CONSTRUCT, as appropriate #/;
end
```

```
procedure STOP_TEST_CASE()
begin
        /# stop all running processes #/;
end

procedure TAKE_SNAPSHOT()
        (*  Une image instantanée des files d'attente des points PCO et CP entrants, de la liste pertinente des fins de temporisation et
            de l'état de fin de toutes autres composantes de test est prise. Le fait de prendre une image instantanée ne supprime pas
            un événement d'un point PCO ou CP ni de la liste des fins de temporisation. *)
begin
        /# save current PCO and CP input queues in Snapshot #/;
        /# save current timeout list in Snapshot #/;
        /# save current list of terminated Test Components in Snapshot #/;
end

procedure TERMINATE_TEST_CASE()
begin
        if /# any Parallel Test Component processes are still running #/ then
          LOG(TEST_CASE_ERROR);
        STOP_TEST_CASE();
end

function TIMER_OP_TYPE_OF(Alternative): TIMER_OP_TYPE
begin
        return /# START_TIMER, CANCEL_TIMER, or READ_TIMER, as appropriate #/;
end
```

Annexe C

Modules TTCN

C.1 Introduction

Un module TTCN doit comprendre les sections suivantes, dans l'ordre indiqué:

a) partie aperçu général du module TTCN;

b) partie importation;

c) partie déclaration;

d) partie contraintes;

e) partie dynamique.

C.2 Partie aperçu général du module TTCN

C.2.1 Introduction

La partie aperçu général du module TTCN d'un module sert à fournir les informations requises pour l'utilisation du module par d'autres modules ou suites de tests. Elle comprend les éléments suivants:

a) table d'exportation de module TTCN;

b) structure du module TTCN;

c) index des tests élémentaires;

d) index des modules de test;

e) index des comportements par défaut.

C.2.2 Table d'exportation de module TTCN

Le formulaire de table d'exportation de module TTCN identifie le module et fournit des informations relatives à l'objectif global du module TTCN (par exemple bibliothèque des contraintes pour un protocole particulier).

Si un type de point PCO est donné comme étant un objet exporté dans la table d'exportation, il doit être défini dans la table facultative des types de points PCO.

Le nom de l'objet source d'origine doit être donné si l'objet est importé.

Si l'objet est déclaré comme étant un objet externe (objet externe explicite) ou s'il est un objet omis dans l'objet source importé (objet externe implicite), le mot clé EXTERNAL est donné à la place du nom de l'objet source.

L'exportation d'un objet de type Enumeration ou NamedNumber nécessite que le type correspondant soit donné. Les autres objets qui sont définis dans le type correspondant ne sont pas exportés en même temps. Ils sont cependant exportés implicitement et d'autres objets exportés peuvent y faire référence. Le nom du type est donné sous forme d'un suffixe de l'objet encadré par des crochets.

Les informations suivantes doivent être fournies dans la table d'exportation de module TTCN:

a) nom du module TTCN;

b) description de l'objectif du module;

c) référence complète du module TTCN;

d) références aux normes de base pertinentes, le cas échéant;

e) référence au formulaire PICS, le cas échéant;

f) référence au formulaire PIXIT, le cas échéant;

g) indication de la ou des méthodes de test, le cas échéant;

h) autres informations pouvant aider à comprendre le module TTCN, à inclure sous forme d'un commentaire;

i) liste d'objets exportés,

   contenant les informations suivantes pour chaque objet exporté:

   1) nom de l'objet,

      si l'objet est de type NamedNumber ou Enumeration, le type correspondant doit être indiqué sous forme d'un suffixe ajouté au nom de l'objet encadré par des crochets;

   2) type de l'objet;

   3) nom de l'objet source d'origine si l'objet est importé, ou directive d'objet EXTERNAL;

   4) numéro de page,

      indiquant l'emplacement de l'objet dans le module (aucun numéro de page ne sera fourni pour les objets importés).

Ces informations doivent être fournies dans le format de Formulaire C.1, ci-dessous.

Table d'exportation de module TTCN

Nom du module TTCN	:	*TTCN_ModuleIdentifier*
Objectif	:	*[FreeText]*
Réf. au module TTCN	:	*[FreeText]*
Réf. aux normes	:	*[FreeText]*
Réf. de déclaration PICS	:	*[FreeText]*
Réf. d'informations PIXIT	:	*[FreeText]*
Méthodes(s) de test	:	*[FreeText]*
Commentaire	:	*[FreeText]*

Nom d'objet	Type d'objet	Nom de la source	n° de page	Commentaires
⋮	⋮	⋮	⋮	⋮
ObjectIdentifier	*TTCN_ObjectType*	*[SourceIdentifier / ObjectDirective]*	*Number*	*[FreeText]*
⋮	⋮	⋮	⋮	⋮

Commentaires détaillés: *[FreeText]*

Formulaire C.1 – Table d'exportation de module TTCN

DÉFINITION SYNTAXIQUE:

4 TTCN_ModuleIdentifier ::= Identifier

12 ObjectIdentifier ::= Identifier | ObjectTypeReference

15 TTCN_ObjectType ::= **SimpleType_Object | StructType_Object | ASN1_Type_Object | TS_Op_Object | TS_Proc_Object | TS_Par_Object | SelectExpr_Object | TS_Const_Object | TS_Var_Object | TC_Var_Object | PCO_Type_Object | PCO_Object | CP_Object | Timer_Object | TComp_Object | TCompConfig_Object | TTCN_ASP_Type_Object | ASN1_ASP_Type_Object | TTCN_PDU_Type_Object | ASN1_PDU_Type_Object | TTCN_CM_Type_Object | ASN1_CM_Type_Object | EncodingRule_Object | EncodingVariation_Object | InvalidFieldEncoding_Object | Alias_Object | StructTypeConstraint_Object | ASN1_TypeConstraint_Object | TTCN_ASP_Constraint_Object | ASN1_ASP_Constraint_Object | TTCN_PDU_constraint_Object | ASN1_PDU_Constraint_Object | TTCN_CM_Constraint_Object | ASN1_CM_Constraint_Object | TestCase_Object | TestStep_Object | Default_Object | NamedNumber_Object | Enumeration_Object**

17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier

18 ObjectDirective ::= Omit | **EXTERNAL**

EXEMPLE C.1 – Table d'exportation de module TTCN

Table d'exportation de module TTCN				
Nom du module TTCN	**:** *TTCN_Module_A*			
Objectif	**:** illustrer l'utilisation de la table d'exportation de module TTCN			
Réf. au module TTCN	**:**			
Réf. aux normes	**:**			
Réf. de déclaration PICS	**:**			
Réf. d'informations PIXIT	**:**			
Méthode(s) de test	**:**			
Commentaire	**:**			
Nom d'objet	**Type d'objet**	**Nom de la source**	**nº de page**	**Commentaires**
String5	SimpleType_Object		3	
wait	Timer_Object	Module_B		
INTC	TTCN_PDU_Type_Object		13	
DEF1	Default_Object	TestSuite_1		
TC_2	TestCase_Object	TestSuite_2		
TC_3	TestCase_Object		33	
Preamble	TestStep_Object	EXTERNAL		

C.2.3 Structure du module TTCN

La structure du module TTCN contient une liste de groupes de tests du module (le cas échéant). Les informations suivantes doivent être fournies pour chaque groupe:

a) la référence de groupe d'un test,

   dans laquelle le premier identificateur peut être le nom du module, et chacun des identificateurs suivants représente la suite de l'ordonnancement conceptuel du module;

b) un identificateur facultatif de l'expression de sélection;

c) un objectif de groupe de tests;

d) un numéro de page (le numéro de page ne sera pas fourni dans le cas des groupes importés).

Ces informations doivent être fournies dans le format de Formulaire C.2, ci-dessous.

Structure de module TTCN			
Référence de groupe d'un test	**Référence de groupe d'un test**	**Objectif de groupe de tests**	**n° de page**
⋮ *TestGroupReference* ⋮	⋮ *TestGroupReference* ⋮	⋮ *FreeText* ⋮	⋮ *Number* ⋮
Commentaires détaillés: *[FreeText]*			

Formulaire C.2 – Structure de module TTCN

DÉFINITION SYNTAXIQUE:

626 TestGroupReference::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

La sémantique statique décrite en 10.2 "Structure de suite de tests" est applicable à la structure de module TTCN.

C.2.4 Index des tests élémentaires

La définition de l'index des tests élémentaires pour les modules est la même que la définition de l'index des tests élémentaires pour les suites de tests.

C.2.5 Index des modules de test

La définition de l'index des modules de test pour les modules est la même que la définition de l'index des modules de test pour les suites de tests.

C.2.6 Index des comportements par défaut

La définition de l'index des comportements par défaut pour les modules est la même que la définition de l'index des comportements par défaut pour les suites de tests.

C.3 Partie importation

C.3.1 Introduction

La partie importation d'un module sert à déclarer les objets qui ne sont pas explicitement définis mais qui ont été utilisés. Ces objets sont soit déclarés comme étant des objets externes, soit importés à partir d'autres objets sources. Cette partie comprend les éléments suivants:

a) objets externes;

b) table d'importation.

C.3.2 Objets externes

Le formulaire pour les objets externes contient la liste des objets auxquels font référence leurs identificateurs dans le module TTCN, mais qui ne sont ni importés ni définis explicitement. Un objet externe indique à l'importateur ce qu'il doit définir, lorsqu'il importe le module TTCN.

Les informations suivantes doivent être fournies pour chaque objet externe:

a) l'identificateur et les paramètres de l'objet,

les paramètres sont ajoutés lorsque l'objet est une opération de suite de tests, une contrainte ou un module de test;

b) le type d'objet;

c) un commentaire facultatif.

Ces informations doivent être fournies dans le format de Formulaire C.3, ci-dessous.

Objets externes		
Nom d'objet	**Type d'objet**	**Commentaires**
⋮ *Identifier | TS_OpId&ParList | ConsId&ParList | TestStepId&ParList* ⋮	⋮ *TTCN_ObjectType* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*		

Formulaire C.3 – Objets externes

DÉFINITION SYNTAXIQUE:

141 TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]

555 ConsEk&ParList ::= ConstraintIdentifier [FormalParList]

638 TestStepId&ParList ::=TestStepIdentifier [FormalParList]

15 TTCN_ObjectType ::= **SimpleType_Object | StructType_Object | ASN1_Type_Object | TS_Op_Object | TS_Proc_Object | TS_Par_Object | SelectExpr_Object | TS_Const_Object | TS_Var_Object | TC_Var_Object | PCO_Type_Object | PCO_Object | CP_Object | Timer_Object | TComp_Object | TCompConfig_Object | TTCN_ASP_Type_Object | ASN1_ASP_Type_Object | TTCN_PDU_Type_Object | ASN1_PDU_Type_Object | TTCN_CM_Type_Object | ASN1_CM_Type_Object | EncodingRule_Object | EncodingVariation_Object | InvalidFieldEncoding_Object | Alias_Object| StructTypeConstraint_Object | ASN1_TypeConstraint_Object | TTCN_ASP_Constraint_Object | ASN1_ASP_Constraint_Object | TTCN_PDU_constraint_Object | ASN1_PDU_Constraint_Object | TTCN_CM_Constraint_Object | ASN1_CM_Constraint_Object | TestCase_Object | TestStep_Object | Default_Object | NamedNumber_Object | Enumeration_Object**

EXEMPLE C.2 – Objets externes:

Objets externes		
Nom d'objet	**Type d'objet**	**Commentaires**
CRC(P:A_PDU) CONSTRAINT_A(acstr:t_CONNECT) TESTSTEP_A(I:INTEGER) DEF3	TS_Op_Object TTCN_PDU_Constraint_Object TestStep_Object Default_Object	

C.3.3 Table d'importation

La définition de la table d'importation pour les modules est la même que la définition de la table d'importation pour les suites de tests (voir 10.7).

Annexe D

Index de suite de tests

D.1 Introduction

L'index de suite de tests est une liste complète de tous les objets d'une suite de tests développée et elle résulte de la conversion d'une suite de tests modulaire en une suite de tests développée. Cette liste contient des informations relatives à chaque objet (par exemple nom de l'objet source/de la suite de tests, nom d'origine et numéro de page dans l'objet source de toute première origine).

D.2 Index de suite de tests

D.2.1 Introduction

L'index de suite de tests sert à fournir les informations requises pour tous les objets importés dans une suite de tests développée. Ces informations permettent de trouver facilement la définition d'un objet.

D.2.2 Index de suite de tests

Le formulaire d'index de suite de tests identifie tous les objets utilisés dans une suite de tests. Les informations suivantes doivent être fournies pour chaque objet:

a) le nom de l'objet,

c'est-à-dire le nom utilisé pour faire référence à l'objet (par exemple un nom généré);

b) le type de l'objet,

qui doit être le même que le type donné lorsque l'objet est défini;

c) le nom de l'objet source ou de la suite de tests,

où l'objet est défini;

d) le nom d'origine de l'objet,

c'est-à-dire le nom donné lorsque l'objet est explicitement défini;

e) un numéro de page facultatif,

indiquant l'emplacement de l'objet dans l'objet source d'origine.

Ces informations doivent être fournies dans le format de formulaire D.1, ci-dessous.

Index de suite de tests					
Nom d'objet	**Type d'objet**	**Nom de la source**	**Réf. à l'objet d'origine**	**n° de page**	**Commentaires**
⋮ *ObjectIdentifier* ⋮	⋮ *ObjectType* ⋮	⋮ *SourceIdentifier* ⋮	⋮ *[ObjectReference]* ⋮	⋮ *[Number]* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés: *[FreeText]*					

Formulaire D.1 – Index de suite de tests

Le numéro de page est donné lorsque l'objet source d'origine est Recommandation et qu'il n'y a aucune ambiguïté quant à l'emplacement de l'objet.

Annexe E

Formulaires compacts

E.1 Introduction

Sur option, il est possible de présenter un grand nombre de contraintes et de tests élémentaires sur une seule table. Cela peut être utile pour mettre en lumière les relations existant entre les contraintes simples ou les tests élémentaires simples. La présente annexe énonce les spécifications d'utilisation des formulaires compacts de contraintes et des formulaires compacts de tests élémentaires et en donne quelques exemples. Ces formulaires sont spécifiques et diffèrent des présentations générales du 7.3. Etant donné que les nouveaux formulaires ne sont qu'une autre manière de présenter les mêmes informations, aucune notation TTCN.MP n'y est associée. Les informations contenues dans une table compacte des contraintes ou une table compacte de tests élémentaires peuvent être traduites dans la notation TTCN.MP associée aux nombreuses tables de contraintes simples et aux nombreuses tables de tests élémentaires qui contiennent les mêmes informations.

E.2 Formulaires compacts pour les contraintes

E.2.1 Spécifications

Le regroupement de plusieurs tables de contrainte simple en une seule table compacte des contraintes est soumis aux conditions suivantes:

a) les contraintes ont le même type de primitive ASP, d'unité PDU, structuré ou ASN.1;

b) aucune information de codage n'est spécifiée dans un en-tête de table de contrainte simple ni dans la colonne de codage d'une de ces tables (du codage en notation ASN.1 spécifié en valeur ASN.1 peut cependant être spécifié dans les formulaires compacts);

c) il n'y a aucune entrée dans la colonne commentaires de l'une quelconque des tables de contrainte simple.

NOTE – Si les tables de contrainte simple ne comportent des commentaires que sur la ligne de commentaires détaillés en bas de page (c'est-à-dire si la colonne commentaires est vide), il est possible de regrouper plusieurs contraintes en un format compact. En ce cas, les commentaires détaillés particuliers des formulaires simples sont regroupés en un commentaire unique sur la ligne de commentaires détaillés en bas de page du formulaire compact.

E.2.2 Formulaires compacts pour les contraintes des primitives ASP

Lorsqu'une contrainte ne comporte que quelques paramètres ou que le nombre de contraintes est peu élevé, elles peuvent être présentées dans la version compacte du Formulaire E.1 de contraintes de primitive ASP, ci-dessous.

Déclarations de contraintes de primitive ASP					
Type de primitive ASP : *StructIdentifier*					
Nom de contrainte	**Chemin de dérivation**	**Nom de champ**			**Commentaires**
		ASP_ParIdentifier$_1$		*ASP_ParIdentifier*$_n$	
ConsId-&ParList$_1$	*Derivation-Path*$_1$	*ConstraintValue-&Attributes*$_{1,1}$		*ConstraintValue-&Attributes*$_{1,n}$	*[FreeText]*$_1$
ConsId-&ParList$_2$	*Derivation-Path*$_2$	*ConstraintValue-&Attributes*$_{2,1}$		*ConstraintValue-&Attributes*$_{2,n}$	*[FreeText]*$_2$
⋮	⋮	⋮		⋮	⋮
ConsId-&ParList$_m$	*Derivation-Path*$_m$	*ConstraintValue-&Attributes*$_{m,1}$		*ConstraintValue-&Attributes*$_{m,n}$	*[FreeText]*$_m$

Formulaire E.1 – Déclarations de contraintes de primitive ASP – Forme compacte

Ce formulaire est utilisé pour les primitives ASP et leurs paramètres tout comme le formulaire de déclarations de contraintes d'unité PDU est utilisé pour les unités PDU et leurs champs (voir E.2.3).

E.2.3 Formulaires compacts pour les contraintes d'unité PDU

E.2.3.1 Introduction

Lorsqu'une contrainte ne comporte que quelques champs ou que le nombre de contraintes est peu élevé, elles peuvent être présentées dans la version compacte du Formulaire E.2 de contraintes d'unité PDU, voir ci-dessous.

Dans le formulaire compact de contraintes, les colonnes représentent les champs, et les lignes représentent les différentes instances de contraintes d'unité PDU. S'il y a *n* champs dans la définition de type d'unité PDU, il y aura *n* colonnes nom de champ dans le formulaire compact des contraintes.

La colonne de chemin de dérivation est facultative: elle sera utilisée pour spécifier le chemin de dérivation des contraintes modifiées (voir 13.6). Une table compacte peut regrouper plusieurs contraintes de base (comme l'illustre l'exemple C.1) ou une contrainte de base et ses contraintes modifiées (comme dans l'exemple C.2). Lorsque des contraintes modifiées sont déclarées dans une table compacte, les champs non modifiés des contraintes modifiées sont représentés par des cases laissées en blanc à l'intersection de la ligne de la contrainte modifiée et de la colonne champ. Lorsqu'on réexprime une table compacte en notation TTCN.MP (c'est-à-dire en format simple), les champs vides (et donc hérités) seront omis. Les champs non spécifiés des contraintes modifiées sont laissés vides.

Déclarations de contraintes d'unité PDU					
Type d'unité PDU : *PDU_Identifier*					
Nom de contrainte	**Chemin de dérivation**	**Nom de champ**			**Commentaires**
		ASP_ParIdentifier$_1$		*ASP_ParIdentifier*$_n$	
ConsId-&ParList$_1$	*Derivation-Path*$_1$	*ConstraintValue-&Attributes*$_{1,1}$		*ConstraintValue-&Attributes*$_{1,n}$	*[FreeText]*$_1$
ConsId-&ParList$_2$	*Derivation-Path*$_2$	*ConstraintValue-&Attributes*$_{2,1}$		*ConstraintValue-&Attributes*$_{2,n}$	*[FreeText]*$_2$
⋮	⋮	⋮		⋮	⋮
ConsId-&ParList$_m$	*Derivation-Path*$_m$	*ConstraintValue-&Attributes*$_{m,1}$		*ConstraintValue-&Attributes*$_{m,n}$	*[FreeText]*$_m$

Formulaire E.2 – Déclarations de contraintes d'unité PDU – Forme compacte

EXEMPLE E.1 – Contraintes utilisant le formulaire compact des contraintes

E.1.1 Etant donné la déclaration de l'unité PDU_B:

Objets externes		
Nom d'unité PDU : PDU_B **Type de point PCO** : XSAP **Commentaire** :		
Nom de champ	**Type de champ**	**Commentaires**
FIELD1 FIELD2 FIELD3	INTEGER BOOLEAN IA5String	

E.1.2 les contraintes s'appliquant à l'unité PDU_B pourraient être représentées de la manière suivante sur le formulaire compact de contraintes:

Déclarations de contraintes d'unité PDU				
Type d'unité PDU : PDU_B				
Nom de contrainte	**Nom de champ**			**Commentaires**
	FIELD1	FIELD2	FIELD3	
CN1 CN2 CN3	3 (4,5,6) 0	TRUE FALSE ?	"une chaîne" "une chaîne" –	

La référence aux contraintes dans la partie dynamique pourrait comporter des entrées telles que PDU_B[CN1] et PDU_B[CN2].

E.1.3 Mécanisme d'héritage utilisant le formulaire compact des contraintes:

Déclarations de contraintes d'unité PDU						
Type d'unité PDU : PDU_A						
Nom de contrainte	**Chemin de dérivation**	**Nom de champ**				**Commentaires**
		FIELD1	FIELD2	FIELD3	FIELD2	
CN0		0	'FF'H	'00'B	TRUE	
CN1	CN0	1				
CN2	CN0. CN		–	?		

E.2.3.2 Contraintes compactes paramétrées

Les contraintes (en représentation) compactes peuvent aussi être paramétrées. En ce cas, la liste de paramètres sera ajoutée au nom de la contrainte et figurera dans la colonne du nom de contrainte des formulaires compacts des contraintes.

EXEMPLE E.2 – Contrainte compacte paramétrée

L'invocation des contraintes de l'unité PDU_X dans un module de test peut s'effectuer comme suit: S1, S2, S3, S4, S5(0), S5(1) ou S5(Var) où Var est une variable de test élémentaire ou de suite de tests:

Déclarations de containtes d'unité PDU			
Type d'unité PDU : PDU_X			
Nom de contrainte	**Nom de champ**		**Commentaires**
	P1	P2	
S1	0	0	
S2	0	1	
S3	1	0	
S4	1	1	
S5(A:INTEGER)	1	A	

E.2.4 Formulaires compacts utilisés pour les contraintes de type structuré

Pour les contraintes compactes de type structuré, on utilisera le Formulaire E.3, ci-dessous.

Déclarations de contraintes de type structuré					
Type de structure : *StructIdentifier*					
Nom de contrainte	**Chemin de dérivation**	**Nom de champ**			**Commentaires**
		ASP_ParIdentifier$_1$		*ASP_ParIdentifier*$_n$	
ConsId-&ParList$_1$	*Derivation-Path*$_1$	*ConstraintValue-&Attributes*$_{1,1}$		*ConstraintValue-&Attributes*$_{1,n}$	*[FreeText]*$_1$
ConsId-&ParList$_2$	*Derivation-Path*$_2$	*ConstraintValue-&Attributes*$_{2,1}$		*ConstraintValue-&Attributes*$_{2,n}$	*[FreeText]2*
⋮	⋮	⋮		⋮	⋮
ConsId-&ParList$_m$	*Derivation-Path*$_m$	*ConstraintValue-&Attributes*$_{m,1}$		*ConstraintValue-&Attributes*$_{m,n}$	*[FreeText]*$_m$

Formulaire E.3 – Déclarations de contraintes de type structuré (version compacte)

EXEMPLE E.3 – Utilisation des contraintes compactes de type structuré

L'unité PDU_Y comporte cinq champs: Y1 à Y5. Les champs Y1, Y2 et Y3 ont été combinés en un type structuré A. Dans les tables suivantes, la première montre les contraintes définies sur l'unité PDU_Y. Les deuxième et troisième tables renferment les mêmes informations que la dernière table.

Les deuxième et troisième tables montrent la spécification des contraintes du type structuré A à l'aide de formulaires de contrainte simple, tandis que la dernière table montre les contraintes du type structuré A au moyen du formulaire compact des contraintes. Dans les deux cas, le mécanisme de modification est utilisé.

Dans les tables suivantes, on voit que l'application de la contrainte YY1 impose aux champs Y1 à Y5 respectivement les valeurs 0,0,0,0,1, les valeurs des champs Y1 à Y3 étant calculées à partir du type structuré A en utilisant la contrainte A1. Si la contrainte YY2 est appliquée, les valeurs des champs Y1 à Y5 seront respectivement 0,3,0,1,0, les valeurs des champs Y1 à Y3 étant calculées à partir du type structuré A en utilisant la contrainte A2.

E.3.1 Table de contraintes d'unité PDU utilisant un type structuré (appelé A):

Déclarations de contraintes d'unité PDU				
Type d'unité PDU : PDU_Y				
Nom de contrainte	**Nom de champ**			**Commentaires**
	A	Y4	Y5	
YY1	A1	0	1	
YY2	A2	1	0	
YY3	A2	0	1	

E.3.2 A1 est une contrainte de base du type structuré A:

Déclaration de contrainte de type structuré		
Nom de contrainte : A1 **Type structuré** : A **Chemin de dérivation** : **Commentaires** :		
Nom d'élément	**Valeur d'élément**	**Commentaires**
Y1	0	
Y2	0	
Y3	0	

E.3.3 La contrainte A2 de type structuré est une contrainte modifiée dérivée de A1:

Déclaration de contrainte de type structuré		
Nom de contrainte : A2 **Type structuré** : A **Chemin de dérivation** : A1 **Commentaires** :		
Nom d'élément	**Valeur d'élément**	**Commentaires**
Y2	3	

E.3.4 Contraintes A1 et A2 du type structuré A sous forme compacte:

Déclaration de contrainte de type structuré					
Nom du type structuré : A					
Nom de contrainte	**Chemin de dérivation**	**Nom de champ**			**Commentaires**
		A	Y4	Y5	
A1 A2	 A1	0	0 3	0	

Lorsqu'on utilise les types structurés dans les déclarations de contraintes d'unité PDU, chaque nom de champ utilisé dans la définition du type structuré correspondra exactement au nom (ou nom abrégé, si le nom abrégé et le nom complet ont tous deux été définis) du champ d'unité PDU correspondant tel qu'il apparaît dans la définition originale du type d'unité PDU.

E.2.5 Formulaires compacts pour les contraintes ASN.1

Les Formulaires E.4, E.5 et E.6 seront utilisés pour les définitions sous forme compacte des contraintes de primitive ASP ASN.1, d'unité PDU ASN.1 et de type ASN.1:

Déclarations de contraintes de primitive ASP en notation ASN.1	
Type de primitive ASP : *ASP_Identifier*	
Nom de contrainte	**Valeur ASN.1**
ConsId&ParList$_1$	*ConstraintValue&Attributes*$_1$
. . .	. . .
ConsId&ParList$_m$	*ConstraintValue&Attributes*$_m$

Formulaire E.4 – Déclarations de contraintes de primitive ASP en notation ASN.1 – Forme compacte

Déclarations de contraintes d'unité PDU en notation ASN.1	
Type d'unité PDU : *ASP_Identifier*	
Nom de contrainte	**Valeur ASN.1**
ConsId&ParList$_1$	*ConstraintValue&Attributes*$_1$
. . .	. . .
ConsId&ParList$_m$	*ConstraintValue&Attributes*$_m$

Formulaire E.5 – Déclarations de contraintes d'unité PDU en notation ASN.1 – Forme compacte

Déclarations de contraintes de type en notation ASN.1	
Nom du type : *ASP_Identifier*	
Nom de contrainte	**Valeur ASN.1**
ConsId&ParList$_1$	*ConstraintValue&Attributes*$_1$
.	.
.	.
.	.
ConsId&ParList$_m$	*ConstraintValue&Attributes*$_m$

Formulaire E.6 – Déclarations de contraintes de type en notation ASN.1 – Forme compacte

E.3 Formulaire compact pour les tests élémentaires

E.3.1 Spécifications

Le regroupement de plusieurs tables simples de comportement dynamique de test élémentaire en une seule table compacte de comportements dynamiques de tests élémentaires n'est autorisé que lorsque les règles suivantes s'appliquent:

a) toutes les tables simples de comportement dynamique de test élémentaire appartiennent au même groupe de tests;

b) toutes les tables simples de comportement dynamique de test élémentaire ont le même arbre comportemental par défaut ou n'en possèdent pas du tout; il est recommandé de ne pas en avoir;

c) la description comportementale de chaque table simple de comportement dynamique de test élémentaire comporte une simple construction de rattachement ATTACH.

E.3.2 Formulaire compact pour les comportements dynamiques de tests élémentaires

Lorsqu'une série de tests élémentaires ont essentiellement le même comportement dynamique, les différences n'apparaissant que dans les contraintes en référence (par exemple, les tests utilisés de variation des paramètres des primitives ASP ou des unités PDU), ils peuvent être représentés dans la version compacte du Formulaire E.7 des comportements dynamiques de tests élémentaires, voir ci-dessous.

Comportements dynamiques de tests élémentaires			
Groupe : *TestGroupReference* **Comportement par défaut** : *DefaultReference*			
Nom du test élémentaire	**Objet**	**Rattachement de module de test**	**Commentaires**
⋮ *TestCaseIdentifier* ⋮	⋮ *FreeText* ⋮	⋮ *Attach* ⋮	⋮ *[FreeText]* ⋮
Commentaires détaillés:			

Formulaire E.7 – Comportements dynamiques de tests élémentaires (version compacte)

Chaque ligne du formulaire décrit un test élémentaire simple. La table de ce formulaire remplace à elle seule une série de tables de comportement dynamique de test élémentaire dans la partie comportementale de la suite de tests.

La dernière colonne comporte des commentaires concernant chacun des tests élémentaires en ce qui concerne leur rattachement.

Les tests élémentaires figurant dans le formulaire compact des tests élémentaires peuvent former un sous-ensemble de leur groupe et doivent apparaître dans l'ordre indiqué dans l'index des tests élémentaires.

EXEMPLE E.4 – Table compacte des tests élémentaires définissant une série de tests pour le service de transfert, accès et gestion de fichiers (FTAM):

Comportements dynamiques de tests élémentaires		
Groupe : R/BV/PV/LM/CR/OV **Comportement par défaut :**		
Nom du test élémentaire	**Objet**	**Rattachement de module de test**
OVERRIDE1	Omettre le paramètre de substitution quand le fichier existe.	+OVERRIDE (FCRERQ_001,FCRERP_001)
OVERRIDE2	Omettre le paramètre de subtitution quand le fichier n'existe pas.	+OVERRIDE (FCRERQ_002,FCRERP_002)

Appendice I

Exemples

I.1 Exemples de contraintes sous forme tabulaire

I.1.1 Définitions de primitive ASP et d'unité PDU

I.1.1.1 Définition de type plat

Définition de type d'unité PDU		
Nom d'unité PDU : R/BV/PV/LM/CR/OV **Type de point PCO** : **Commentaires** : illustration des mécanismes TTCN		
Nom de champ	**Type de champ**	**Commentaires**
Source Destination T_Class UserData	BITSTRING [4] BITSTRING [4] INTEGER0to4 IA5String	Longueur de 4 bits Longueur de 4 bits Défini comme type simple

I.1.1.2 Définition de type structuré

Définition de type d'unité PDU		
Nom d'unité PDU : R/BV/PV/LM/CR/OV **Type de point PCO** : **Commentaires** : illustration des mécanismes TTCN		
Nom de champ	**Type de champ**	**Commentaires**
T_Addresses T_Class UserData	T_AddressInfo INTEGER0to4 IA5String	 Défini comme type simple

Définition de type structuré		
Nom du type : T_CONNECT2		
Commentaires : peut être utilisé dans tous les exemples d'unité PDU de transport		
Nom d'élément	**Définition du type**	**Commentaires**
Source Destination	BITSTRING[4] BITSTRING[4]	Longueur de 4 bits Longueur de 4 bits

I.1.1.3 Type spécial d'unité PDU, pour permettre l'utilisation du chaînage (statique) des contraintes

Définition de type de primitive ASP		
Nom de primitive ASP : N_DATArequest		
Type de point PCO : N_SAP		
Commentaires : pour illustration seulement		
Nom de paramètre	**Type de paramètre**	**Commentaires**
CallingNetworkAddress CalledNetworkAddress ConnectionIdentifier Data	HEXSTRING HEXSTRING HEXSTRING PDU	 Pour permettre le chaînage des contraintes

I.1.2 Contraintes de primitive ASP/d'unité PDU

I.1.2.1 Contraintes de type plat

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_CLASS4_1		
Type d'unité PDU : T_CONNECT1		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
Source Destination T_Class UserData	TS_Par1 TS_Par2 4 "testing, testing"	

I.1.2.2 Contraintes de type structuré, faisant référence aux groupes de champs

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_CLASS4_2		
Type d'unité PDU : T_CONNECT2		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
T_Addresses T_Class UserData	WrongAddress 4 "one, two, three"	WrongAddress (fausse adresse) est une référence à une contrainte de type structuré.

Déclaration de contrainte de type structuré		
Nom de contrainte : WrongAddress		
Type structuré : T_AddressInfo		
Chemin de dérivation :		
Commentaires :		
Nom d'élément	**Valeur d'élément**	**Commentaires**
Source Destination	TS_Par1 '0000'B	

I.1.2.3 Chaînage, utile pour les unités PDU (emboîtées) dans des primitives ASP

Déclaration de contrainte de primitive ASP		
Nom de contrainte : N_DATAreq_With_T_CON_Class4_1		
Type de primitive ASP : N_DATArequest		
Chemin de dérivation :		
Commentaires : TCON_Class4_1 est une contrainte d'unité PDU (c'est-à-dire un chaînage).		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
CallingNetworkAddress CalledNetworkAddress ConnectionIdentifier Data	TS_Par3 TS_Par4 'ABCDEF'H TCON_Class4_1	

I.1.2.4 Contraintes paramétrées: il est possible de paramétrer des contraintes de types plat, structuré et chaîné. L'exemple suivant montre le paramétrage utilisé pour transférer une valeur:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_1(class:INTEGER)		
Type d'unité PDU : T_CONNECT1		
Chemin de dérivation :		
Commentaires : TCON_Class4_1 est une contrainte d'unité PDU (c'est-à-dire un chaînage).		
Nom de champ	**Valeur de champ**	**Commentaires**
Source Destination T_Class UserData	'1000'B ? class ?	 Class (classe) est un paramètre formel.

On peut faire référence à cette contrainte paramétrée depuis des tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous la forme suivante:

TCON_1(4) ou TCON_1(TCvariable)

Les valeurs de champ peuvent être des unités PDU (chaînées) entières:

Déclaration de contrainte de primitive ASP		
Nom de contrainte : N_DATAreq_With_T_CON(A_Constraint:T_CONNECT2)		
Type de primitive ASP : N_DATArequest		
Chemin de dérivation :		
Commentaires : TCON_Class4_1 est une contrainte d'unité PDU (c'est-à-dire un chaînage).		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
CallingNetworkAddress	TS_Par3	
CalledNetworkAddress	TS_Par4	
ConnectionIdentifier	'1234567'H	
Data	A_Constraint	A_Constraint est un paramètre formel.

Cette contrainte peut être appelée, par exemple, de la manière suivante:

N_DATAreq_With_TCON(TCON_Class4_2).

Comme le paramètre effectif est un nom de contrainte lui-même paramétrable, il est possible ainsi de représenter une profondeur arbitraire d'emboîtement des unités PDU.

I.1.2.5 Contraintes modifiées: il est possible d'utiliser des contraintes existantes et de les modifier pour en définir de nouvelles. Cela est vrai pour les contraintes de tous types: plat, structuré et paramétré.

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_CLASS0_1		
Type d'unité PDU : T_CONNECT1		
Chemin de dérivation : TCON_Class4_1		
Commentaires : La classe 0 est acceptable.		
Nom de champ	**Valeur de champ**	**Commentaires**
T_Class	0	

Des caractères génériques peuvent être utilisés pour les valeurs du type suivant:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_AnyClass		
Type d'unité PDU : T_CONNECT1		
Chemin de dérivation : TCON_Class4_1		
Commentaires : Toute classe (0..4) est acceptable.		
Nom de champ	**Valeur de champ**	**Commentaires**
T_Class	?	

Cette méthode n'est toutefois pas recommandée. Il est préférable d'utiliser la contrainte plus générale comme base.

Il est aussi possible de supprimer des champs entiers:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : TCON_Erroneous_NoClass		
Type d'unité PDU : T_CONNECT1		
Chemin de dérivation : TCON_Class4_1		
Commentaires : Aucune classe présente.		
Nom de champ	**Valeur de champ**	**Commentaires**
T_Class	–	T_Class omis

I.2 Exemples de contraintes ASN.1

I.2.1 Définitions de primitive ASP et d'unité PDU

I.2.1.1 Type plat

Définition de type d'unité PDU en notation ASN.1	
Nom d'unité PDU	**:** T_CONNECT1
Type de point PCO	**:**
Commentaires	**:**
Définition de type	
-- à la seule fin d'illustrer l'utilisation de la notation ASN.1 en notation TTCN	
SEQUENCE { source	BITSTRING (SIZE (4..4)),
Destination	BITSTRING (SIZE (4..4)),
t_Class	INTEGER (0..4)
userData	IA5String OPTIONAL
}	

I.2.1.2 Type structuré

Définition de type d'unité PDU en notation ASN.1	
Nom d'unité PDU	**:** T_CONNECT2
Type de point PCO	**:**
Commentaires	**:**
Définition de type	
-- à la seule fin d'illustrer l'utilisation de la notation ASN.1 en notation TTCN	
SEQUENCE { t_Addresses	T_AddressInfo
t_Class	INTEGER (0..4)
userData	IA5String
}	
-- on peut trouver le développement de T_AddressInfo dans une table séparée.	

Les productions connexes en notation ASN.1 qui sont normalement dans un seul module ASN.1 peuvent être réparties entre plusieurs tables TTCN:

Définition de type en notation ASN.1	
Nom du type	**:** T_AddressInfo
Commentaires	**:**
Définition de type	
SEQUENCE { source	BITSTRING (SIZE (4..4)),
destination	BITSTRING (SIZE (4..4)),
}	

I.2.1.3 Définition d'une primitive ASP

Définition de type de primitive ASP en notation ASN.1		
Nom de primitive ASP	**:**	N_DATArequest
Type de point PCO	**:**	N_SAP
Commentaires	**:**	

Définition de type

```
SEQUENCE    {    callingNetworkAddress    OCTETSTRING,    -- nombre pair d'octets
                 calledNetworkAddress     OCTETSTRING,    -- nombre pair d'octets
                 connectionIdentifier     OCTETSTRING,    -- nombre pair d'octets
                 Data                          T_PDUS
             }
```

Définition de type en notation ASN.1		
Nom du type	**:**	T_PDUS
Commentaires	**:**	

Définition de type

```
CHOICE      {    t1  T_CONNECT1
                 t2  T_CONNECT2
                 }
```

I.2.2 Contraintes de primitive ASP/d'unité PDU en notation ASN.1

I.2.2.1 Type plat

Déclaration de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	TCON_Class4_1
Type d'unité PDU	**:**	T_CONNECT1
Chemin de dérivation	**:**	T_CONNECT1
Commentaires	**:**	

Valeur de contrainte

```
{   source          TS_PAR1,
                    TS_PAR2,        -- l'identificateur de champ peut être omis.
    t_Class         4
    userData        "testing, testing"
}
```

I.2.2.2 Type structuré

Déclaration de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	TCON_Class4_2
Type d'unité PDU	**:**	T_CONNECT2
Chemin de dérivation	**:**	
Commentaires	**:**	
Valeur de contrainte		
{ t_Addresses t_Class userData }		WrongAddress, -- *référence à une contrainte de champ d'unité PDU* 4, "one, two, three"

Déclaration de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	WrongAddress
Type d'unité PDU	**:**	T_AddressInfo
Chemin de dérivation	**:**	
Commentaires	**:**	
Valeur de contrainte		
{ source TS_PAR1, destination }		 '0000'B

I.2.2.3 Chaînage d'une contrainte d'unité PDU

Déclaration de contrainte de primitive ASP en notation ASN.1		
Nom de contrainte	**:**	N_DATAreq_With_TCON_Class4_1
Type de primitive ASP	**:**	N_DATArequest
Chemin de dérivation	**:**	
Commentaires	**:**	
Valeur de contrainte		
{ callingNetworkAddress calledNetworkAddress connectionIdentifier data }		TS_PAR_3 TS_PAR_4 'ABCDEF'H t1 TCON_Class4_1 -- *chaînage vers une contrainte d'unité PDU*

I.2.2.4 Contraintes paramétrées: les contraintes ASN.1 peuvent être paramétrées tout comme les contraintes tabulaires TTCN, par exemple:

Déclaration de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	TCON_1(class:INTEGER)
Type d'unité PDU	**:**	T_CONNECT1
Chemin de dérivation	**:**	
Commentaires	**:**	

Valeur de contrainte			
{	source	’0000’B,	
	destination	?,	-- *caractère générique*
	t_Class	class,	-- *paramètre formel*
	userData	?	
}			

On peut faire référence à cette contrainte paramétrée depuis les tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous la forme:

TCON_1(4) ou TCON_1(TCvariable)

Un paramètre peut aussi représenter une unité PDU chaînée complète:

Déclaration de contrainte de primitive ASP en notation ASN.1		
Nom de contrainte	**:**	N_DATAreq_With_TCON(a_constraint:T_CONNECT2)
Type de primitive ASP	**:**	N_DATArequest
Chemin de dérivation	**:**	
Commentaires	**:**	

Valeur de contrainte		
{	callingNetworkAddress	TS_PAR_3
	calledNetworkAddress	TS_PAR_4
	connectionIdentifier	’1234567’H
	data	t2 a_constraint
		-- *a_constraint est un paramètre formel contenant une unité PDU complète.*
}		

On peut faire référence à cette contrainte depuis les tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous la forme:

N_DATAreq_With_TCON(TCON_Class4_2)

Comme le paramètre effectif est un nom de contrainte lui-même paramétrable, il est possible de représenter une profondeur arbitraire d'emboîtement.

I.2.2.5 Contraintes modifiées – De nouvelles contraintes peuvent être créées en modifiant des contraintes déjà définies à l'aide du mécanisme REPLACE (remplacement):

Déclaration de contrainte d'unité PDU en notation ASN.1	
Nom de contrainte	**:** TCON_Class0_1
Type d'unité PDU	**:** T_CONNECT1
Chemin de dérivation	**:** T_CON_Class4_1
Commentaires	**:**
Valeur de contrainte	
REPLACE t_Class BY 0	

Des caractères génériques peuvent aussi être utilisés comme suit:

Déclaration de contrainte d'unité PDU en notation ASN.1	
Nom de contrainte	**:** TCON_AnyClass
Type d'unité PDU	**:** T_CONNECT1
Chemin de dérivation	**:** T_CON_Class4_1
Commentaires	**:**
Valeur de contrainte	
REPLACE t_Class BY ?	

Pour spécifier les champs qui seront omis, on utilise le mécanisme OMIT (omission). Cela n'est permis que si le champ est déclaré OPTIONAL (facultatif):

Déclaration de contrainte d'unité PDU en notation ASN.1	
Nom de contrainte	**:** TCON_NoUserData
Type d'unité PDU	**:** T_CONNECT1
Chemin de dérivation	**:** TCON_Class4_1.TCON_AnyClass
Commentaires	**:**
Valeur de contrainte	
OMIT userData	

Il est possible de modifier les contraintes paramétrées ASN.1, mais on notera que les champs paramétrés eux-mêmes ne peuvent pas être remplacés:

Déclaration de contrainte d'unité PDU en notation ASN.1	
Nom de contrainte	**:** TCON_2(class:INTEGER)
Type d'unité PDU	**:** T_CONNECT1
Chemin de dérivation	**:** TCON_1
Commentaires	**:**
Valeur de contrainte	
REPLACE userData BY "CPS"	

I.2.3 Autres exemples de contraintes en notation ASN.1

I.2.3.1 Définition d'une unité PDU F-INITIALIZEresponse pour le service FTAM, à l'aide d'une table de définition de type unité PDU en notation ASN.1:

Définition de type d'unité PDU en notation ASN.1		
Nom d'unité PDU	**:**	F_INITIALIZEresponse
Type de point PCO	**:**	
Commentaires	**:**	
Définition de type		
SEQUENCE {		
state_result		State_Result DEFAULT success,
action_result		Action_Result DEFAULT success,
protocol_version		Protocol_Version DEFAULT { version_1},
implementation_information		Implementation_Information OPTIONAL,
presentation_context_management		[2] IMPLICIT BOOLEAN DEFAULT FALSE,
service_class		Service_Class DEFAULT { transfer_class },
functional_units		Functional_Units,
attribute_groups		Attribute_Groups DEFAULT { },
shared_ASE_information		Shared_ASE_Information OPTIONAL,
ftam_quality_of_service		FTAM_Quality_Of_Service,
contents_type_list		Contents_Type_List OPTIONAL,
diagnostic		Diagnostic OPTIONAL,
checkpoint_window		[8] IMPLICIT INTEGER DEFAULT 1
}		

Les champs de l'unité PDU (State_Result, Action_Result, etc.) sont déclarés dans les définitions de type ASN.1.

Le type Functional_Units est par exemple déclaré comme ceci:

Définition de type en notation ASN.1		
Nom du type	**:**	Functional_Units
Commentaires	**:**	
Définition de type		
[4] IMPLICIT BITSTRING		
{	read(2),	
	write(3),	
	file_access(4),	
	limited_file_management(5),	
	enhanced_file_management(6),	
	grouping(7),	
	fadu_locking(8),	
	recovery(9),	
	restart_data_transfer(10)	
}		

Une contrainte de base, F_INITrsp_001, s'appliquant à une réponse F_INITIALIZEresponse, est déclarée dans la partie contraintes:

Définition de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	F_INITrsp_001
Type d'unité PDU	**:**	F_INITIALIZEresponse
Chemin de dérivation	**:**	
Commentaires	**:**	
Valeur de contrainte		
{		
state_result		State_Result_001,
action_result		Action_Result_001,
protocol_version		Protocol_Version_001,
implementation_information		Implementation_Information_001,
presentation_context_management		FALSE,
service_class		Service_Class_001,
functional_units		Functional_Units_001,
attribute_groups		Attribute_Groups_001,
shared_ASE_information		Shared_ASE_Information_001,
ftam_quality_of_service		FTAM_Quality_Of_Service_001,
contents_type_list		Contents_Type_List_001,
diagnostic		Diagnostic_001,
checkpoint_window		1
}		

Une contrainte Functional_Units_001, appliquée au type Functional_Units, est déclarée dans une déclaration de contrainte de champ d'unité PDU en notation ASN.1:

Déclaration de contrainte de type en notation ASN.1		
Nom de contrainte	**:**	Functional_Units_001
Type structuré	**:**	Functional_Units
Chemin de dérivation	**:**	
Commentaires	**:**	
Valeur de contrainte		
'001'B – en écriture seulement		

Une deuxième contrainte, F_INITrsp_002, peut être créée en modifiant la contrainte de base, F_INIT_rsp001:

Déclaration de contrainte d'unité PDU en notation ASN.1		
Nom de contrainte	**:**	F_INITrsp_002
Type structuré	**:**	F_INITIALIZEresponse
Chemin de dérivation	**:**	F_INITrsp_001
Commentaires	**:**	
Valeur de contrainte		
OMIT	implementation_information,	
REPLACE	presentation_context_management	BY TRUE,
REPLACE	functional_units	BY Functional_Units_002
REPLACE	checkpoint_window	BY ?

où Functional_Units_002 est une déclaration de contrainte d'unité PDU en notation ASN.1.

I.3 Contraintes de base et contraintes modifiées

Supposons que nous ayons la définition de type d'unité PDU suivante:

Définition de type d'unité PDU		
Nom d'unité PDU : PDU_B		
Type de point PCO :		
Commentaires : ceci est la déclaration de l'unité de données de protocole PDU_B		
Nom de champ	**Type de champ**	**Commentaires**
FIELD1	INTEGER	
FIELD2	HEXSTRING	
FIELD3	BITSTRING	
FIELD4	BOOLEAN	

Une contrainte de base pour PDU_B pourrait être:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : C0		
Type d'unité PDU : PDU_B		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
FIELD1	0	
FIELD2	'FF'H	
FIELD3	'00'B	
FIELD4	TRUE	

Une contrainte modifiée C1 dérivée de la contrainte de base C0 pourrait être:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : C1		
Type d'unité PDU : PDU_B		
Chemin de dérivation : C0		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
FIELD1	1	Dans la contrainte de base C0, cette valeur de champ est 0.

Il est possible de poursuivre le processus de dérivation à partir de C1:

Déclaration de contrainte d'unité PDU		
Nom de contrainte : C2		
Type d'unité PDU : PDU_B		
Chemin de dérivation : C0.C1		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
FIELD2	–	Ce champ est omis.
FIELD3	?	Toute valeur licite est acceptée.

On fait référence à une contrainte modifiée dans un arbre de comportement par son nom.

I.4 Définitions de type utilisant des macroinstructions

Définition de type d'unité PDU à l'aide d'un symbole de macroinstruction:

Définition de type d'unité PDU		
Nom d'unité PDU : T_CONNECT3		
Type de point PCO :		
Commentaires : illustration du mécanisme des macroinstructions TTCN		
Nom de champ	**Type de champ**	**Commentaires**
<-	T_AddressGroup	
T_Class	INTEGER0to4	Défini comme type simple
UserData	IA5String	

Définition de type structuré		
Nom du type : T_AddressGroup		
Commentaires :		
Nom d'élément	**Définition de type**	**Commentaires**
Source	BITSTRING [4]	Longueur de 4 bits
Destination	BITSTRING [4]	Longueur de 4 bits

Déclaration de contrainte d'unité PDU		
Nom de contrainte : T_CON_Class4_3		
Type d'unité PDU : T_CONNECT3		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
<-	GoodAddress	Référence de déclaration de contrainte de type structuré
T_Class	4	
UserData	"one, two, three"	

Déclaration de contrainte de type structuré		
Nom de contrainte : GoodAddress		
Type structuré : T_AddressGroup		
Chemin de dérivation :		
Commentaires :		
Nom d'élément	**Définition d'élément**	**Commentaires**
Source	’0101’B	
Destination	’1111’B	

I.5 Utilisation de REPEAT (répétition)

Comportement dynamique de test élémentaire					
Nom du test élémentaire : RPT_EX2					
Groupe : TTCN_EXAMPLES/REPEAT_EXAMPLE2/					
Objectif : illustrer l'utilisation de REPEAT et la transmission de paramètre par substitution textuelle					
Comportement par défaut :					
Commentaires :					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1		(FLAG:=FALSE, COUNTER:=0)			
2		!A	A1		
3		REPEAT STEP2 (FLAG, COUNTER)			
		UNTIL [FLAG OR COUNTER=3]			
4		[FLAG]			
5		!D	D1	PASS	
6		[COUNTER=3]			
7		!E	E1	FAIL	
		STEP2 (F:BOOLEAN; NUMBER: INTEGER)			
8		?B (F:=TRUE)	B1		
9		?C (F:=FALSE, NUMBER:=NUMBER+1)	C1		
Commentaires détaillés:					
Cet exemple illustre comment l'exécution répétée de STEP2 (module 2) peut se terminer par la réception du message B ou de trois autres messages. Dans les lignes suivant la construction REPEAT, les expressions booléennes indiquent qu'il faut envoyer le message D si le message B est reçu et le message E si trois messages différents de B sont reçus. Cet exemple illustre également l'effet de la transmission de paramètre par substitution textuelle. Ainsi, F est remplacé par FLAG, et NUMBER est remplacé par COUNTER, de sorte que FLAG et COUNTER peuvent recevoir les résultats des affectations dans STEP2.					

I.6 Opérations de suite de tests

Utilisation d'une opération de suite de tests pour effectuer un total de contrôle:

Définition d'opération de suite de tests
Nom de l'opération : CRC(P:A_PDU)
Type de résultat : INTEGER
Commentaires :
Description
Calculer le total de contrôle de l'unité PDU P à l'aide de l'algorithme de contrôle de redondance cyclique (CRC) et le retourner.
NOTE – Dans une suite ATS réelle, cette opération serait décrite de manière plus détaillée.

Déclaration de contrainte d'unité PDU		
Nom de contrainte : CONS1		
Type d'unité PDU : A_PDU		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
:	:	
Checksum	?	
:	:	
L'affectation A_PDU.Checksum := CRC(CONS1) dans l'événement SEND approprié d'une description comportementale établira le total de contrôle dans la contrainte CONS1.		

I.7 Exemple d'une description générale de suite de tests

La table de structure de suite de tests ci-dessous définit une hiérarchie des groupes de tests et des tests élémentaires dans la suite considérée. Dans cette structure, des expressions de sélection de test sont identifiées qui régissent la sélection des groupes de tests et des tests élémentaires pour exécution. Par exemple, SELEXP_100 est référencé comme expression de contrôle pour la caractéristique X du protocole. Si la caractéristique X n'est pas prise en charge, aucun des tests élémentaires de la suite qui se trouvent dans le groupe de cette caractéristique ne sera sélectionné.

Structure de la suite de tests			
Nom de la suite : TEST_SUITE_A			
Réf. aux normes : Recommandations xxxx			
Réf. de déclaration PICS : Recommandation aaaa			
Réf. d'informations PIXIT : Recommandation bbbb			
Notation(s) des tests : Méthode de test DS			
Commentaires : Ceci n'est qu'un exemple.			
Référence de groupe d'un test	**Réf. de sélection**	**Objectif du groupe de tests**	**n° de page**
FEATURE_X	SELEXP_100	Caractéristique X facultative dans le test	50
FEATURE_A/ATTR_A		Attribut A obligatoire dans le test	50
FEATURE_A/ATTR_A/NEGOTIATION	SELEXP_101	Négociation de l'attribut A facultative dans le test	50
FEATURE_A/ATTR_A/USAGE		Usage de l'attribut A dans le test	60
FEATURE_A/ATTR_B e		Caractéristique Y obligatoire dans le test	80

Pour déterminer si la caractéristique X est prise en charge ou non, il faut évaluer SELEXP_100. Pour cela, on détermine si le paramètre de suite de tests dans SELEXP_100, c'est-à-dire TST_FX, a la valeur TRUE. Si tel est le cas, le traitement à l'intérieur du groupe continue. On notera que les tests concernant l'attribut A seront sélectionnés (aucune expression), mais que ceux qui concernent la caractéristique facultative de négociation de cet attribut ne le seront que si SELEXP_101 a la valeur TRUE.

Index de tests élémentaires				
Référence de groupe d'un test	**Identificateur des tests élémentaires**	**Référence de sélection**	**Objectif du groupe de tests**	**n° de page**
FEATURE_X/ATTR_A/NEGOTIATION	FX_ANEG_1	SELEXP_102	Demande de l'attr. A, nég. valide	50
	FX_ANEG_2	SELEXP_102	Demande de l'attr. A, nég. non val.	52
	FX_ANEG_3		Réception de l'attr. A, nég. non val.	54
	FX_ANEG_4		Réception de l'attr. A, nég. non val.	56
FEATURE_X/ATTR_A/USAGE	FX_AUSE_1	SELEXP_103	Usage de l'attribut A (VAL = 0)	60
	FX_AUSE_2		Réception de l'attribut A	62
	FX_AUSE_3		Réception de l'attribut A	64

Si la négociation de l'attribut A est prise en charge, les tests élémentaires de FX_ANEG_01 à FX_ANEG_04 sont tous candidats pour la sélection. Toutefois, les tests élémentaires "01" et "02" ne seront choisis que si l'expression complémentaire de sélection SELEXP_102 a la valeur TRUE. Le test élémentaire FX_ANEG_01 ne sera sélectionné que si la déclaration PICS indique qu'une valeur zéro peut être prise en charge pour l'attribut A.

Les questions relatives à la déclaration PICS et aux informations PIXIT utilisées dans les expressions de sélection de test sont déclarées en tant que paramètres de suite de tests.

Déclarations de paramètres de suite de tests			
Nom de paramètre	**Type**	**Réf. PICS/PIXIT**	**Commentaires**
TSP_FX	BOOLEAN	PICS question FX1	Q: caractéristique X prise en charge?
TSP_FXA_N	BOOLEAN	PICS question FX2	Q: nég. de caractéristique X prise en charge?
TSP_FXA_NINT	BOOLEAN	PICS question FX3	Q: nég. nécessaire à l'implémentation IUT?
TSP_FXA_MINVAL	INTEGER	PIXIT question FXVAL	Q: valeur VAL = 0 utilisée par l'implémentation IUT?

Ces expressions de sélection de test sont déclarées en tant qu'expressions booléennes, comme défini en 11.5.

Définitions des expressions de sélection de test élémentaire		
Nom de l'expression	**Expression de sélection**	**Commentaires**
SELEXP_100	TSP_FX	Caractéristique X prise en charge
SELEXP_101	TSP_FXA_N	Négociation de caractéristique X
SELEXP_102	TSP_FXA_NINIT	Demande de négociation de caractéristique X
SELEXP_103	TSP_FXA_VAL=0	Acceptation de caractéristique X, VAL = 0

I.8 Exemple de test élémentaire présenté en notation TTCN.MP

Pour l'échantillon de test élémentaire donné ci-dessous:

Comportement dynamique de test élémentaire					
Nom du test élémentaire : PACKET/P4/PROPER/T_02					
Groupe : T_7_02					
Objectif : vérifier que l'implémentation IUT accuse réception d'un code de cause de libération 05 à l'état p4.					
Comportement par défaut :					
Commentaires :					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
0		+R1_PREAMBLE(SVC)			
1		+P4D1_PREAMBLE			
2		!CLEAR START TD	CLR_0(LC)		cause de libération = 5
3	L1	?CLEARC CANCEL TD	CLRC_0(LC)	(PASS)	
4		+R1_POSTAMBLE			
5		?CLEARC CANCEL TD	CLR_L0(LC)	(PASS)	
6		+R1_POSTAMBLE			
7		?RESTART [RST_ON_ERR] CANCEL TD	STRT_DTEA	(PASS)	
8		!RESTARTC	STRTC		
9		+R1_POSTAMBLE			
10		+DIC_UNEXPECTED			
11		->L1			
12		+RSRT_UNEXPECTED			
13		?TIMEOUT TD		FAIL	
14		?OTHERWISE CANCEL TD		FAIL	

La description TTCN.MP correspondante est la suivante:

$BeginTestCase
  $TestCaseId T_7_02
  $TestGroupRef PACKET/P4/PROPER/T_02
  $TestPurpose /* Vérifier que l'implémentation sous test acquitte une annulation de code 05 dans l'état p4 */
  $DefaultsRef
  $BehaviourDescription
    $BehaviourLine
      $Label
      $Line [0] +R1_PREAMBLE(SVC)
      $Cref
      $Verdict
    $End_BehaviourLine
    $BehaviourLine
      $Label
      $Line [1] +P4D1_PREAMBLE
      $Cref
      $Verdict
    $End_BehaviourLine
    $BehaviourLine
      $Label
      $Line [2] !CLEAR START TD
      $Cref CLR_0(LC)
      $Verdict
      $Comment /* cause de libération = 5 */

$End_BehaviourLine
$BehaviourLine
$Label L1
$Line [3] ?CLEARC CANCEL TD
$Cref CLRC_0(LC)
$Verdict (PASS)
$End_BehaviourLine
$BehaviourLine
$Label
$Line [4] +R1_POSTAMBLE
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] ?CLEAR CANCEL TD
$Cref CLR_L0(LC)
$Verdict (PASS)
$End_BehaviourLine
$BehaviourLine
$Label
$Line [4] +R1_POSTAMBLE
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] ?RESTART [RST_ON_ERR] CANCEL TD
$Cref STRT_DTEA
$Verdict (PASS)
$End_BehaviourLine
$BehaviourLine
$Label
$Line [4] !RESTARTC
$Cref STRTC
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [5] +R1_POSTAMBLE
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] +D1C_UNEXPECTED
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [4] -> L1
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] +RSRT_UNEXPECTED
$Cref
$Verdict
$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] ?TIMEOUT TD
$Cref
$Verdict FAIL

$End_BehaviourLine
$BehaviourLine
$Label
$Line [3] ?OTHERWISE CANCEL TD
$Cref
$Verdict FAIL
$End_BehaviourLine
$End_BehaviourDescription
$End_TestCase

La présentation adoptée ici a pour seul but de faciliter la lisibilité.

I.9 Utilisation de références à des composantes pour l'affectation de valeurs de champ dans des contraintes

Lorsqu'un certain nombre de valeurs de champ dans une unité PDU reçue doivent être affectées aux champs dans plusieurs unités PDU d'envoi subséquentes, la table de comportement dynamique peut devenir encombrée par des déclarations d'affectation longues utilisant la notation ponctuée.

La notation TTCN permet des affectations de valeurs de champ d'unité PDU dans les tables de contraintes en utilisant des références à des composantes associées à un paramètre formel. Des primitives ASP ou des unités PDU reçues dans la table de comportement dynamique peuvent être affectées à une variable et transmises par la suite en tant que paramètre effectif dans la référence aux contraintes à un paramètre formel de la table de contraintes. Cette dernière table spécifie alors les affectations de champ requises en utilisant le paramètre formel et ses composantes. Les tables qui suivent illustrent ces principes.

La Figure I.1 illustre les affectations de champ possibles dans la spécification de comportement sans utiliser les références à des composantes.

Comportement dynamique d'un test élémentaire					
Nom du test élémentaire : TTCN_EXAMPLES/STYLE1					
Groupe : ST_EX1					
Objectif : illustrer l'utilisation de références à des composantes dans la description de comportement					
Comportement par défaut :					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1 2		?InASP(v:=InASP.userdata) !OutASP (OutASP.userdata.OutPDU.FieldA:=v.Field2; OutASP.userdata.OutPDU.FieldC:=v.Field3)	Cin1 Cout1		

Figure I.1/X.292 – Les déclarations d'affectation longues encombrent la description de comportement

La Figure I.2 illustre la simplification de la spécification de comportement résultant de l'utilisation de références à des composantes dans des contraintes.

Par souci de simplicité, nous avons omis les définitions de tous les types requis de primitives ASP et d'unités PDU.

Les types InASP et OutASP de primitives ASP sont constitués du champ à un seul paramètre userdata, de types InPDU et OutPDU respectivement. Le type InPDU contient les trois champs Field1, Field2 et Field3, qui sont tous du type IA5String.

Le type OutPDU contient les trois champs FieldA, FieldB et FieldC, qui sont aussi du type IA5String.

v doit être déclaré en tant que variable de test élémentaire d'un type d'unité PDU.

Comportement dynamique d'un test élémentaire					
Nom du test élémentaire : TTCN_EXAMPLES/STYLE1 **Référence** : ST_EX1 **Objectif** : illustrer l'utilisation de références à des composantes dans la description de comportement **Comportement par défaut** :					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1 2		?InASP(v:=InASP.userdata) !OutASP	Cin1 Cout2(v)		

Figure I.2/X.292 – Les déclarations d'affectation longues sont supprimées de la description de comportement

Les tables qui suivent contiennent les déclarations de contrainte de primitive ASP et d'unité PDU requises:

Déclaration de contrainte de primitive ASP		
Nom de contrainte : Cout1 **Type de primitive ASP** : OutASP **Chemin de dérivation** : **Commentaires** :		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
Userdata	CoutPDU1	

Déclaration de contrainte de primitive ASP		
Nom de contrainte : Cout2(p:**PDU**) **Type de primitive ASP** : OutASP **Chemin de dérivation** : **Commentaires** :		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
Userdata	CoutPDU2(p)	

Déclaration de contrainte de primitive ASP		
Nom de contrainte : Cin1 **Type de primitive ASP** : InASP **Chemin de dérivation** : **Commentaires** :		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
Userdata	CinPDU	

Déclaration de contrainte d'unité PDU		
Nom de contrainte : CoutPDU1		
Type d'unité PDU : OutPDU		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
FieldA	'A'	
FieldB	'B'	
FieldC	'C'	

Déclaration de contrainte d'unité PDU		
Nom de contrainte : CoutPDU2(p:PDU)		
Type d'unité PDU : OutPDU		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
FieldA	p.Field2	
FieldB	'B'	
FieldC	p.Field3	

Déclaration de contrainte d'unité PDU		
Nom de contrainte : CinPDU		
Type d'unité PDU : InPDU		
Chemin de dérivation :		
Commentaires :		
Nom de champ	**Valeur de champ**	**Commentaires**
Field1	*	
Field2	*	
Field3	*	

I.10 Test multiparti

La Figure I.3 illustre une configuration de composante de test pour un contexte de test multiparti type. Un seul testeur supérieur est représenté, étant donné que la communication entre plusieurs testeurs supérieurs ou avec la fonction de commande de testeur supérieur (UTCF, *upper tester control function*) est applicable seulement aux contextes où on utilise exclusivement la méthode de test locale.

Dans l'exemple illustré à la Figure I.3, par souci de simplicité, chaque testeur inférieur est spécifié par une seule composante PTC et la fonction LTCF est spécifiée par la composante MTC. Une autre composante PTC est utilisée pour spécifier le testeur supérieur. Des points de coordination sont utilisés entre les composantes PTC du testeur inférieur et la composante MTC.

Il s'agit d'une utilisation directe de la concomitance en vue d'assurer la conformité aux spécifications multiparti, mais cela ne signifie pas nécessairement qu'il doit exister une relation univoque entre les testeurs inférieurs et les composantes PTC, ou entre la fonction LTCF et la composante MTC, ou entre le testeur supérieur et la composante PTC.

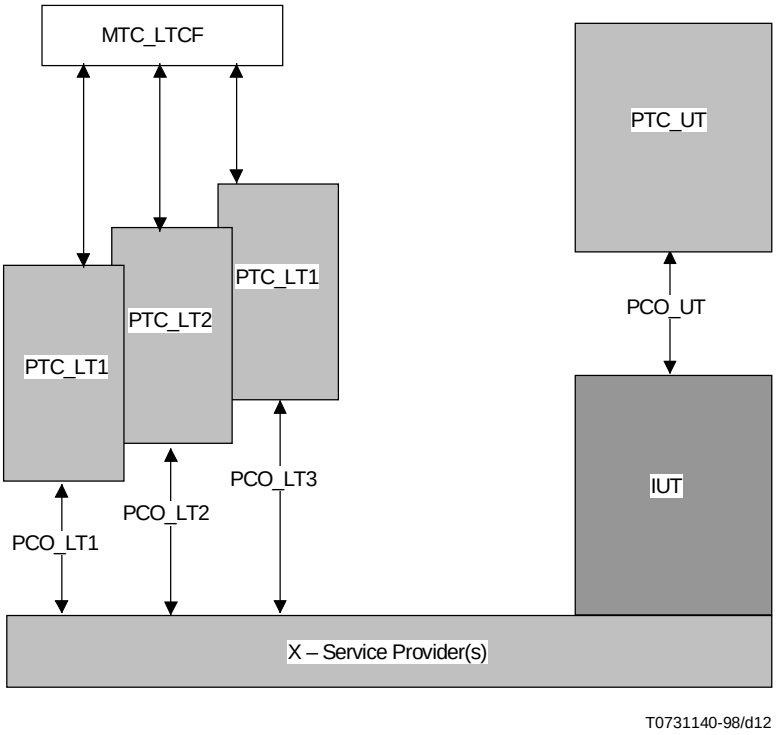


Figure I.3/X.292 – Exemple de configuration de composante de test pour test multiparti avec un seul testeur supérieur

I.11 Multiplexage/démultiplexage

Il existe deux façons d'utiliser la notation TTCN concomitante dans des tests élémentaires faisant appel au multiplexage/démultiplexage. Elles sont illustrées à la Figure I.4. La première façon, représentée à la Figure I.4 a), spécifie le multiplexage et le démultiplexage explicitement dans la composante de test MTC1, les composantes PTC1 et PTC2 traitant chacune le comportement d'une des deux connexions multiplexées. On obtient ainsi la souplesse maximale quant à la façon dont le comportement de multiplexage et de démultiplexage est spécifié, y compris les possibilités de comportement non valide. Cependant, l'inconvénient de cette méthode est que le multiplexeur/démultiplexeur relativement complexe doit être spécifié même si l'objectif du test porte seulement sur le comportement à chacune des deux connexions. L'autre méthode possible consiste à utiliser un point PCO distinct pour chaque flux d'événements distinct et un paramètre de suite de tests (MuxValue) associé à chacun de ces points PCO qui seront multiplexés et démultiplexés dans le fournisseur de services sous-jacent, plutôt que dans le testeur inférieur. La configuration illustrée à la Figure I.4 b) peut alors être utilisée. Comme le multiplexage/démultiplexage est exécuté par le fournisseur de services, il y a deux points PCO dans cette configuration, correspondant aux deux points CP de l'autre configuration, mais on leur attribue une valeur MuxValue commune, MuxA, pour indiquer que dans le fournisseur de services ils doivent être multiplexés. Pour que les choses restent simples, une des deux composantes de test est définie en tant que composante MTC, bien qu'une composante MTC distincte non connectée à un point PCO pourrait être utilisée à la place si on le préférait ainsi.

I.12 Eclatement et recombinaison

Pour spécifier des tests élémentaires comportant l'éclatement et la recombinaison, la seule façon de procéder consiste à spécifier explicitement le comportement d'éclatement et de recombinaison dans le test élémentaire. La concomitance peut être utilisée pour séparer le comportement d'éclatement et de recombinaison en une composante de test, MTC1 à la Figure I.5, à partir du comportement de protocole placé au-dessus de cette fonction, en utilisant une deuxième composante de test, PTC1 à la Figure I.5.

I.13 Tests élémentaires multiprotocolaires

Des tests élémentaires multiprotocolaires, y compris ceux qui utilisent les variantes imbriquées des méthodes de test, peuvent utiliser la notation TTCN concomitante pour séparer le comportement associé à chaque protocole en une composante différente, comme le montre la Figure I.6, en fournissant un exemple de configuration pour une session de test imbriquée dans le service FTAM.

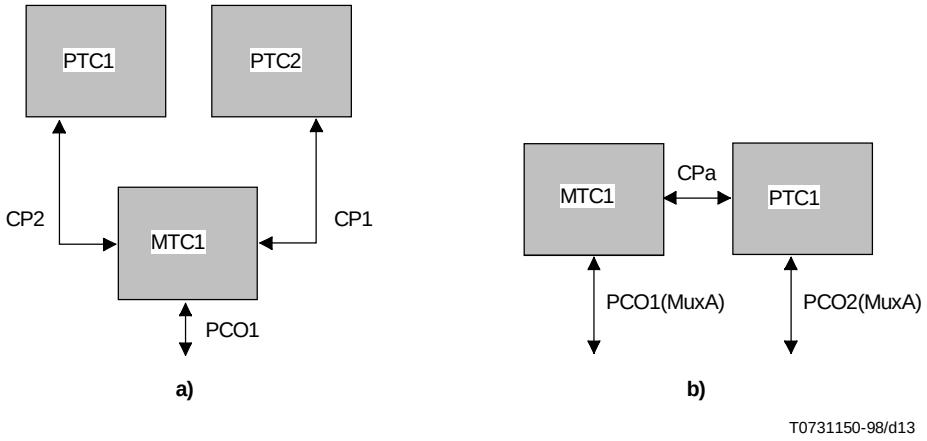


Figure I.4/X.292 – Configurations possibles de tests élémentaires faisant appel au multiplexage/démultiplexage

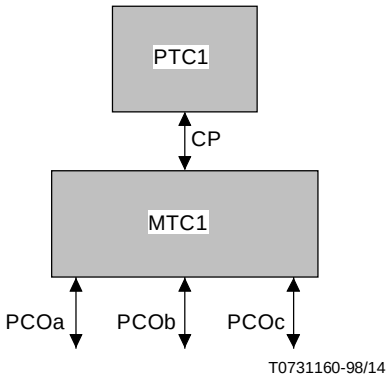


Figure I.5/X.292 – Configuration possible de tests élémentaires faisant appel à l'éclatement et à la recombinaison

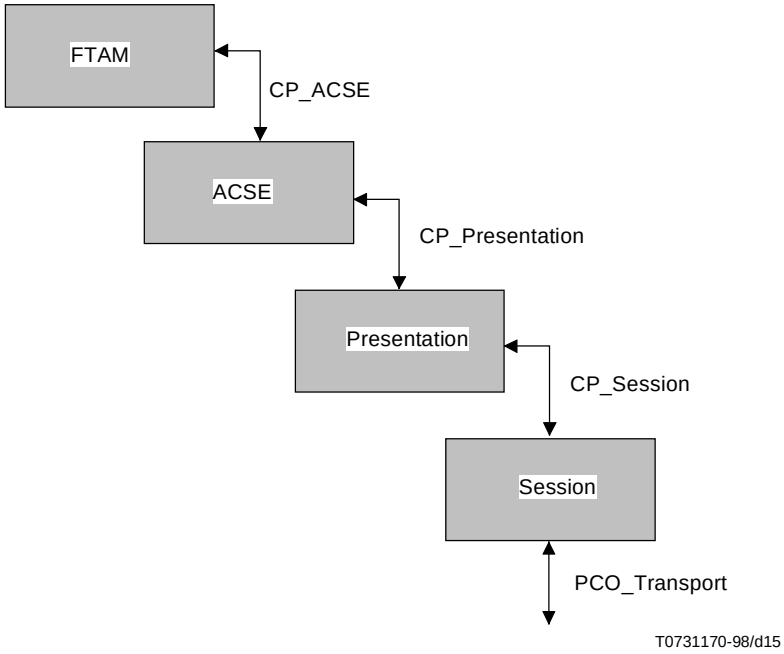
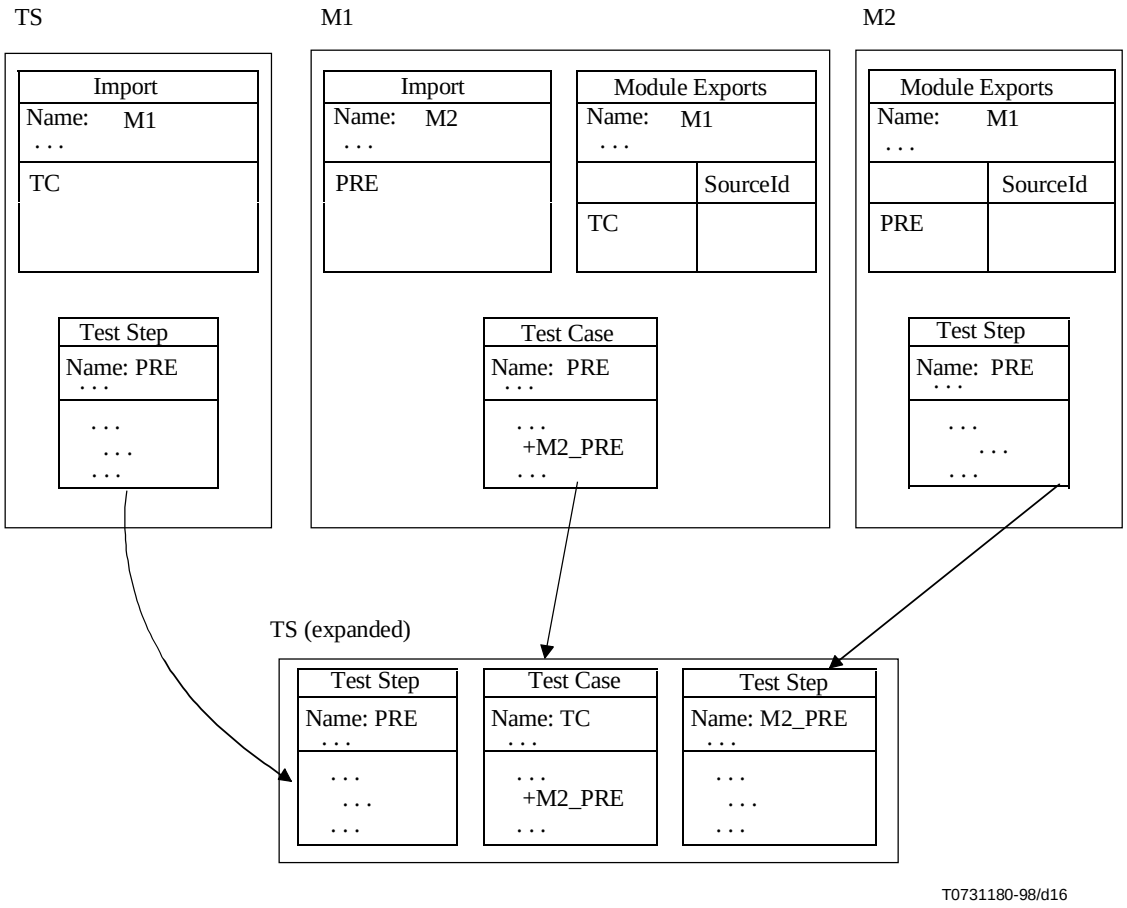


Figure I.6/X.292 – Configuration possible de test multiprotocolaire – Session imbriquée dans le service FTAM

I.14 Exemple de notation TTCN modulaire



Le module de test PRE (défini dans le module M2) est implicitement importé de M1 dans TS.

I.15 Exemple de déclarations CREATE et DONE

NOTE – Un exemple supplémentaire sera inclus dans la seconde édition publiée, afin de clarifier l'utilisation des déclarations CREATE et DONE, particulièrement en ce qui a trait à la transmission implicite de résultats et de verdicts préliminaires, fournissant des explications relatives à la sémantique en montrant le même test élémentaire spécifié au moyen de la transmission explicite de variables à l'aide de messages CM et de points CP.

Aperçu général de la suite de tests:

Structure de suite de tests		
Nom de la suite : Done		
Réf. aux normes :		
Réf. de déclaration PICS :		
Réf. d'informations PIXIT :		
Méthode(s) de test :		
Commentaires : cette suite de tests montre une interprétation possible des déclarations CREATE et DONE, ainsi que de la transmission de verdict et de variable. L'interprétation est faite du point de vue de la communication au moyen de messages CM et de points CP existants.		
Référence de groupe d'un test	**Réf. de sélection**	**Objectif de groupe de tests**
Commentaires détaillés:		

Index des tests élémentaires				
Référence de groupe d'un test	**Identificateur des tests élémentaires**	**Réf. de sélection**	**Description**	**n° de page**
	TC1 TC1_EXPANDED			241 242
Commentaires détaillés:				

Partie déclarative:

Définition de type en notation ASN.1	
Nom du type	**:** TestStop
Commentaires	**:**
Définition de type	
IA5String	
Commentaires détaillés:	

Définition de type en notation ASN.1	
Nom du type	**:** VariableList
Commentaires	**:** Devrait réellement être une représentation de variables et de leurs valeurs.
Définition de type	
NULL	
Commentaires détaillés:	

Déclarations des variables de test élémentaire			
Nom de la variable	**Type**	**Valeur**	**Commentaires**
Verdict	R_Type	fail	
Commentaires détaillés:			

Déclaration des points PCO			
Nom du point PCO	**Type de point PCO**	**Rôle**	**Commentaires**
PCO1	XSAP	LT	
Commentaires détaillés:			

Déclaration des points CP	
Nom du point CP	**Commentaires**
CP1	
Commentaires détaillés:	

Déclaration des composantes de test				
Nom de la composante	**Rôle de la composante**	**Nombre de points PCO**	**Nombre de points CP**	**Commentaires**
PTC1	PTC	1	1	
Master	MTC	0	1	
Commentaires détaillés:				

Déclaration de configuration des composantes de test			
Nom de configuration : Conf1			
Commentaires :			
Composantes utilisées	**Points PCO utilisés**	**Points CP utilisés**	**Commentaires**
Master		CP1	
PTC1	PCO1	CP1	
Commentaires détaillés:			

Définition de type de primitive ASP		
Nom de primitive ASP : ASP1		
Type de point PCO : *XSAP*		
Commentaires :		
Nom de paramètre	**Type de paramètre**	**Commentaires**
Commentaires détaillés:		

Définition de type de primitive ASP		
Nom de primitive ASP : ASP2		
Type de point PCO : *XSAP*		
Commentaires :		
Nom de paramètre	**Type de paramètre**	**Commentaires**
Commentaires détaillés:		

Définition de type de message CM		
Nom de message CM : CM1		
Commentaires :		
Nom de paramètre	**Type de paramètre**	**Commentaires**
Commentaires détaillés:		

Définition de type de message CM		
Nom de message CM : CREATE_CM **Commentaires** :		
Nom de paramètre	**Type de paramètre**	**Commentaires**
StepName Variables	TestStep VariableList	
Commentaires détaillés:		

Définition de type de message CM		
Nom de message CM : DONE_CM **Commentaires** :		
Nom de paramètre	**Type de paramètre**	**Commentaires**
Verdict Variables	R_Type VariableList	
Commentaires détaillés:		

Parties contraintes:

Déclaration des contraintes de message CM		
Nom de contrainte : Create(Step:TestStep) **Type de message CM** : CREATE_CM **Chemin de dérivation** : **Commentaires** :		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
StepName Variables	Step NULL	
Commentaires détaillés:		

Déclaration des contraintes de message CM		
Nom de contrainte : Done **Type de message CM** : DONE_CM **Chemin de dérivation** : **Commentaires** :		
Nom de paramètre	**Valeur de paramètre**	**Commentaires**
Verdict Variables	? NULL	
Commentaires détaillés:		

Partie dynamique:

Comportement dynamique d'un test élémentaire					
Nom du test élémentaire : TC1					
Groupe :					
Objectif :					
Configuration :					
Comportement par défaut :					
Commentaires : notation TTCN concomitante de style normal					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1		CREATE(PTC1: TS1)			
2		CP1!CM1			
3		?DONE(PTC1)		R	
		TS1			
4		CP1?CM1			
5		PCO1!ASP1			
6		PCO1?ASP2		(P)	
Commentaires détaillés:					

Comportement dynamique d'un test élémentaire					
Nom du test élémentaire : TC1_EXPANDED					
Groupe :					
Objectif :					
Configuration :					
Comportement par défaut :					
Commentaires : il s'agit du même test élémentaire que TC_1, sauf que les déclarations CREATE et DONE sont explicitement mises en œuvre par l'intermédiaire de messages CM spéciaux transmis sur un point CP. La seule restriction en notation TTCN est que les contraintes ne peuvent pas accepter le nom d'un module de test comme paramètre, ce qui est émulé dans le présent exemple par une procédure string_matching simple. Un "enum" construit à partir de tous les noms de modules de test serait plus élégant. Chaque composante PTC nécessiterait une procédure PTC_DISPATCHER comme celle du présent exemple.					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1		CREATE_CM	Create("TS1_EXPANDED")		Correspond à CREATE (PTC1:TS1)
2		CP1!CM1			
3		CP1?DONE_CM(Verdict := DONE_CM.Verdict)	Done		
4		[Verdict=pass]		P	
5		[Verdict=fail]		F	
6		[Verdict=inconc]		I	
		PTC1_DISPATCHER			
		CP1?CREATE_CM	Create("TS1_EXPANDED")		
		+TS1			
		CP1!DONE_CM(DONE_CM.Verdict:=R)	Done		
		TS1			
		CP1?CM1			
		PCO1!ASP1			
		PCO1?ASP2		(P)	
Commentaires détaillés:					

Appendice II

Guide stylistique

II.1 Introduction

Le présent appendice présente certaines règles de style qu'il est recommandé de suivre en notation TTCN. Son but est d'assurer une cohérence de base entre styles TTCN utilisés par différents développeurs de suite de tests.

II.2 Structure du test élémentaire

Afin de mieux analyser les résultats d'un test et de déterminer facilement si l'objectif du test est atteint, il est souhaitable d'observer les points suivants relatifs à la manière de structurer les tests élémentaires:

a) le développeur d'une suite de tests doit clairement identifier les sous-arbres de préambule et d'épilogue;

b) l'épilogue et le préambule doivent être spécifiés comme un rattachement d'un arbre de test simple (arbre comportemental local du test élémentaire ou tiré de la bibliothèque des modules de tests) dans l'arbre comportemental principal de test élémentaire. Ces arbres de test peuvent se voir eux-mêmes rattacher d'autres sous-arbres;

c) une fois que les sous-arbres de préambule et d'épilogue sont identifiés dans l'arbre comportemental principal de test élémentaire, les autres événements subsistant de cet arbre peuvent être considérés comme se rapportant au corps de test (c'est-à-dire qu'il s'agit d'événements se rapportant à l'objet du test).

Ce procédé permet de délimiter facilement le préambule, le corps et l'épilogue du test élémentaire. Des étiquettes peuvent être utilisées pour indiquer le début et la fin du corps de test dans le journal de conformité.

Comportement dynamique de test élémentaire					
Nom du test élémentaire : TTCN_EXAMPLES/STYLE1					
Groupe : ST_EX1					
Objectif : illustrer l'identification du préambule et de l'épilogue.					
Configuration :					
Comportement par défaut :					
Commentaires :					
n°	**Etiquette**	**Description de comportement**	**Réf. aux contraintes**	**Verdict**	**Commentaires**
1		+Preamble			
2		!A	A1		Lié à l'objectif
3	Body	?B	B1		Lié à l'objectif
4	CinBody	?C	C1	(PASS)	Lié à l'objectif
5		+postamble_1			
6	DinBody	?D	D1	(PASS)	Lié à l'objectif
7		+postamble_2			
8		?E	E1	INCONC	Lié à l'objectif
9		?OTHERWISE		FAIL	

Figure II.1/X.292 – Identification du préambule et de l'épilogue

Etant donné que les verdicts finals mettent fin à l'exécution du test élémentaire, un développeur de suite de tests ne peut affecter un verdict final dans le corps de test s'il est nécessaire de passer à l'épilogue. Il est toutefois souhaitable de rendre le verdict au niveau du point de test élémentaire où le résultat est effectivement atteint et de ne pas noyer ce verdict dans l'épilogue. Il est donc recommandé d'énoncer les résultats préliminaires dans la colonne verdict lorsqu'un résultat est atteint mais qu'un épilogue doit encore être exécuté. Dans la définition de l'épilogue, un développeur de suite de tests peut utiliser la variable R de résultat en tant que verdict affecté au niveau des feuilles de l'arbre comportemental pour signaler que si aucune erreur n'a été rencontrée dans l'épilogue, le verdit est défini dans le corps du test.

II.3 Utilisation de la notation TTCN dans différentes méthodes de test abstraites

II.3.1 Introduction

Le présent sous-paragraphe lie la notation TTCN aux méthodes de test abstraites définies dans la Recommandation X.291. Il présente la syntaxe TTCN utilisée pour exprimer l'occurrence d'événements aux points PCO, ainsi que les références aux contraintes pour les différentes méthodes de test abstraites considérées.

On suppose que les définitions de type primitive ASP définissent le type du paramètre UserData (données d'usager) en tant qu'unité PDU. Il est alors possible d'utiliser le chaînage des contraintes (c'est-à-dire de renvoyer à une contrainte dans le cas d'une primitive ASP qui comporte une unité PDU dans le paramètre UserData) pour faire référence à une contrainte de primitive ASP comportant une contrainte d'unité PDU en tant que paramètre effectif.

II.3.2 Notation TTCN et méthode de test monocouche locale (LS)

Evénements TTCN possibles:

Description de comportement	*Référence aux contraintes*
LT! N_ASP	N_ASPconstraint (N_PDUconstraint)
LT? N_ASP	N_ASPconstraint (N_PDUconstraint)
UT! T_ASP	T_ASPconstraint
UT? T_ASP	T_ASPconstraint

II.3.3 Notation TTCN et méthode de test monocouche répartie (DS)

Evénements TTCN possibles:

Description de comportement	*Référence aux contraintes*
LT! N_ASP	N_ASPconstraint (T_PDUconstraint)
LT? N_ASP	N_ASPconstraint (T_PDUconstraint)
UT! T_ASP	T_ASPconstraint
UT? T_ASP	T_ASPconstraint

II.3.4 Notation TTCN et méthode de test monocouche coordonnée (CS)

Evénements TTCN possibles:

Description de comportement	*Référence aux contraintes*
LT! N_ASP	N_ASPconstraint (T_PDUconstraint)
LT? N_ASP	N_ASPconstraint (T_PDUconstraint)

En échangeant des unités TM_PDU entre le testeur LT et l'implémentation de protocole de gestion de test (TM) dans l'implémentation sous test, via la liaison utilisée pour le test. On notera que dans ce cas la définition d'unité PDU doit avoir déclaré son champ UserData comme étant de type PDU.

LT! N_ASP	N_ASPconstraint (T_PDUconstraint (TM_PDUconstraint))
LT? N_ASP	N_ASPconstraint (T_PDUconstraint (TM_PDUconstraint))

II.3.5 Notation TTCN et méthode de test monocouche répartie (RS)

Evénements TTCN possibles:

Description de comportement	*Référence aux contraintes*
LT! N_ASP	N_ASPconstraint (T_PDUconstraint)
LT? N_ASP	N_ASPconstraint (T_PDUconstraint)

Etant donné qu'il n'y a pas de testeur UT ni de protocole TMP, l'événement IMPLICIT SEND est utilisé pour décrire les événements envoi côté implémentation sous test de la liaison.

<IUT! N_ASP>	N_ASPconstraint (T_PDUconstraint)
<IUT! T_PDU>	T_PDUconstraint

II.4 Utilisation des comportements par défaut

Question style, un rédacteur de suite de tests doit éviter les situations où l'essai d'une proposition conditionnelle de comportement par défaut est la spécification normale du comportement *prévu* de l'implémentation sous test. Ce serait par exemple le cas si un module de test représente le comportement du testeur LT ou UT et de l'implémentation sous test (IUT) quand les événements valides de test sont envoyés, tandis que les réponses de l'implémentation IUT à des événements de tests non valides ou inopportuns envoyés par le testeur LT ou UT sont spécifiées dans des comportements par défaut implicitement rattachés à ce module de test, lorsqu'il y a appel par d'autres tests élémentaires. Ces comportements par défaut devront alors prendre en charge le verdict Pass (succès).

Cette pratique n'est pas recommandée quand le rattachement d'un arbre par défaut est laissé sans spécification et revêt quelque incertitude. Il faut plutôt utiliser des arbres explicitement rattachés ou l'arbre principal lui-même.

II.5 Limitation du temps d'exécution d'un test élémentaire

Dans les versions précédentes de la notation TTCN, une déclaration ELAPSE (temps écoulé) a été définie, permettant au développeur de test élémentaire de limiter la durée d'un test élémentaire si, par exemple, le traitement d'une image instantanée se poursuit indéfiniment ou si une récursion non contrôlée de rattachement à un arbre se produit.

La déclaration ELAPSE ne fait plus partie de la notation TTCN, le problème qu'elle était censée résoudre étant considéré comme sortant du cadre de la spécification des suites de tests.

Pour limiter le temps d'exécution d'un test élémentaire, il est maintenant recommandé aux développeurs de tests d'élaborer des mécanismes locaux de limitation en même temps que les autres dispositifs de test. Lorsqu'une limite dans le temps pour la réalisation d'un événement doit être fixée, on pourra utiliser des temporisations explicites avec l'événement TIMEOUT (fin de temporisation).

II.6 Types structurés

a) Dans les versions préDIS (projet de norme internationale) de la notation TTCN, des champs et valeurs génériques ont été définis comme caractéristiques permettant de regrouper plusieurs champs ou valeurs dans une table de contraintes, ou de réutiliser ce groupe dans plusieurs tables de contraintes de même contenu;

b) dans la présente version, le groupement des paramètres de primitive ASP et des champs d'unité PDU (ex-types de données) est introduit en premier dans la partie déclarative afin que cette partie soit complète et que la cohérence soit maintenue entre les notations ASN.1 et TTCN. Le 11.2.3.3 donne une définition des tables de définition de type structuré. Une fois qu'un type structuré est déclaré, il peut être utilisé par une ou plusieurs définitions de type primitive ASP ou unité PDU. La table de définition de primitive ASP et d'unité PDU peut donc être de type "plat" (aucun groupe ou un groupe introduit par un appel de macroinstruction) ou structuré (au moyen de spécifications de structures s'appliquant aux paramètres nommés de primitive ASP ou aux champs nommés d'unité PDU);

c) dans la partie contraintes, des valeurs doivent être affectées aux éléments de structure dans les tables de contraintes de type structuré. Les noms de ces contraintes peuvent être utilisés comme valeurs dans les tables de contraintes de base de primitive ASP ou d'unité PDU.

   Les tables des contraintes de primitive ASP et d'unité PDU peuvent donc aussi être:

   – de type plat, c'est-à-dire qu'elles affectent des valeurs à chaque paramètre ou champ et ne renvoient aux tables de contraintes de structure que par appel de macroinstruction;

   – de type structuré, c'est-à-dire qu'elles remplacent les valeurs des groupes déclarés de paramètres ou de champs par des noms de contraintes de groupe;

d) si la primitive ASP ou l'unité PDU déclarée est structurée au moyen de certains paramètres de primitive ASP ou certains champs d'unité PDU spécifiés par référence à des éléments structurés, les contraintes doivent avoir alors la même structure.

   Quelle que soit la forme utilisée, les contraintes de primitive ASP ou d'unité PDU peuvent aussi être:

   – modifiées;

   – paramétrées au moyen d'un paramètre à lier à une valeur de champ ou de paramètre ou à une contrainte de type structuré;

e) les tables de contraintes de type structuré remplacent les tables de champs génériques des versions précédentes de la notation TTCN;

f) le concept de valeurs génériques est supprimé;

g) on trouvera des exemples dans l'Appendice I.

II.7 Abréviations

Dans les versions précédentes de la notation TTCN, il était permis de déclarer, dans une table spécifique, les abréviations à utiliser dans les colonnes comportement des tests élémentaires et des modules de test. Cette fonctionnalité s'est avérée prêter à confusion et a été limitée à l'abréviation des seuls noms de primitives ASP et d'unités PDU, lorsque ces noms apparaissent sur les lignes événements. Cette fonctionnalité a reçu le nom de fonction Alias (pseudonyme).

II.8 Descriptions de tests

Les descriptions informelles de comportement donnant plus de détails que les objets de test, mais moins que les spécifications TTCN des tests élémentaires peuvent, si on le désire, être incluses dans une suite de tests abstraite (ATS) normalisée.

Ces descriptions de tests peuvent utiliser le langage naturel, les diagrammes chronologiques ou toute autre notation et être situées dans la partie commentaires des tables, dans une annexe informative ou dans les deux.

Les spécifications TTCN des tests élémentaires ont toujours la priorité sur les descriptions informelles de test.

II.9 Affectations relatives aux événements SEND

La notation TTCN permet de remplacer des valeurs de contrainte par d'autres avant un événement SEND, dans une déclaration d'affectation sur la ligne événement. Cela signifie que les données à envoyer sont d'abord construites à partir de la définition des contraintes et que les affectations sont ensuite exécutées.

Cette caractéristique doit être utilisée avec prudence car la détermination de la valeur effectivement envoyée risque d'être confuse pour le lecteur de la suite de tests. On estime notamment qu'il est mal venu d'utiliser la même contrainte pour l'envoi et la réception.

II.10 Points PCO multiservice

La spécification précise d'un point PCO couvrant plus d'un point d'accès au service (SAP) est donnée par l'ensemble des primitives ASP et des unités PDU qui peuvent y apparaître.

EXEMPLE II.1 – Un point PCO pour le service FTAM:

Déclarations de point PCO			
Nom du point PCO	**Type de point PCO**	**Rôle**	**Commentaires**
L	A_P_SAPs	LT	Point PCO par lequel on peut observer toutes les primitives ASP d'élément ACSE et toutes les primitives ASP de présentation, sauf P-CONNECT, P-RELEASE et P-ABORT.

Le point PCO "L" est de type A_P_SAP qui est capable d'observer toutes les primitives ASP de présentation et d'éléments de service de contrôle d'association (ACSE), à l'exclusion de P-CONNECT, P-RELEASE et P-ABORT. La colonne type montre que les points SAP "A" et "P" appartiennent à l'ensemble qui doit être observé par le point PCO, chaque point SAP étant séparé par un souligné ("___"). La colonne commentaires décrit exactement ce qui peut être vu par le point PCO.

Cette méthode peut s'étendre à de nombreux points SAP, qui seront tous séparés les uns des autres par un soulignement.

Appendice III

Index

III.1 Introduction

Le présent appendice est l'index alphabétique des termes et acronymes utilisés dans la présente Recommandation. Pour chaque terme ou acronyme, l'index donne un ensemble de références renvoyant aux numéros de paragraphe, de figure et de tableau du corps principal ou des annexes et appendices de la présente Recommandation. La signification de chaque référence est indiquée comme suit:

a) en **gras**, les définitions des termes et acronymes;

b) en *italique*, les principaux emplois des termes et acronymes;

c) en caractères normaux, les autres emplois.

III.2 Index

C

D

E

F

G

H

I

J

O

P

S

T

U

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, de télégraphie, de télécopie, circuits téléphoniques et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	**Réseaux pour données et communication entre systèmes ouverts**
Série Y	Infrastructure mondiale de l'information
Série Z	Langages de programmation