الاتحـــــــــاد الدولـــــــــي للاتصـــــــــالات

ITU-T Z.140

(2006/03)

قطـــاع تقييس الاتصـــالات
في الاتحاد الدولي للاتصالات

السلسلة Z: اللغات والجوانب العامة للبرمجيات في أنظمة الاتصالات

تقنيات الوصف الشكلي (FDT) – الاختبار وترميز ضبط الاختبار (TTCN)

الإصدار 3 من الاختبار وترميز ضبط الاختبار (TTCN-3): لغة النواة

التوصيـــة ITU-T Z.140

الاتحاد الدولي للاتصالات

توصيات السلسلة Z الصادرة عن قطاع تقييس الاتصالات

اللغات والجوانب العامة للبرمجيات في أنظمة الاتصالات

تقنـــيات الوصـــف الشـــكلي (FDT)	
لغة المواصفة والوصف (SDL)	Z.109–Z.100
تطبيق تقنيات الوصف الشكلي	Z.119–Z.110
مخطط تعاقب الرسائل (MSC)	Z.129–Z.120
لغة تعريف الغرض الموسّعة (eODL)	Z.139–Z.130
الاختبار وترميز ضبط الاختبار (TTCN)	**Z.149–Z.140**
ترميز متطلبات المستعملين (URN)	Z.159–Z.150
لغـــات البرمجـــة	
CHILL: لغة المستوى الرفيع لدى قطاع تقييس الاتصالات	Z.209–Z.200
لغـــة الإنســـان–الآلـــة	
مبادئ عامة	Z.309–Z.300
قواعد النَظْم الأساسية وإجراءات التحاور	Z.319–Z.310
لغة الإنسان–الآلة (MML) الموسّعة من أجل مطاريف العرض المرئي	Z.329–Z.320
مواصفة السطح البيني الإنسان–الآلة	Z.349–Z.330
السطوح البينية الإنسان–الآلة الموجهة للمعطيات	Z.359–Z.350
السطوح البينية الإنسان–الآلة من أجل إدارة شبكات الاتصالات	Z.379–Z.360
الجـــودة	
جودة برمجيات الاتصالات	Z.409–Z.400
مظاهر الجودة للتوصيات المرتبطة بالبروتوكولات	Z.459–Z.450
الطـــرائق	
طرائق للتثبت من الصلاحية وللاختبار	Z.519–Z.500
البرمجيـــات الوســـيطة	
بيئة المعالجة الموزعة	Z.609–Z.600

لمزيد من التفاصيل، يرجى الرجوع إلى قائمة التوصيات الصادرة عن قطاع تقييس الاتصالات.

التوصيـــة ITU-T Z.140

الإصدار 3 من الاختبار وترميز ضبط الاختبار (TTCN-3): لغة النواة

الملخص

تحدد هذه التوصية TTCN-3 (*الاختبار وترميز التحكم في الاختبار 3*) المقصود لمواصفة متواليات الاختبارات التي تكون مستقلة عن المنصات وطرائق الاختبار وطبقات البروتوكول والبروتوكولات. ويمكن استخدام TTCN-3 لمواصفة جميع أنماط اختبارات نظام تفاعلي عبر منافذ اتصالات متنوعة. ومن بين مجالات التطبيق اختبارات بروتوكولات (بما في ذلك بروتوكولات اتصالات متنقلة والإنترنت) واختبارات خدمات (بما في ذلك خدمات إضافية) واختبارات وحدة واختبارات المنصات القائمة على معمارية وسيط مطالب لأغراض مشتركة (CORBA) واختبارات السطوح البينية لبرمجة التطبيق (APIs). ومواصفة متواليات الاختبارات لبروتوكولات الطبقة المادية هي خارج مدى هذه التوصية.

يمكن التعبير عن لغة النواة لـ TTCN-3 بأنساق تقديم متنوعة. وبينما تُعرّف هذه التوصية لغة النواة، تُعرّف التوصية ITU-T Z.141 *النسق الجدولي لـ TTCN* (TFT) وتُعرّف التوصية ITU-T Z.142 *النسق البياني لـ TTCN* (GFT) ومواصفة هذه الأنساق هي خارج مدى هذه التوصية. وتخدم لغة النواة ثلاثة أغراض:

1) باعتبارها لغة نص قائم على نص معمم؛

2) باعتبارها نسق مبادلة معياري لمتواليات اختبارات TTCN بين أدوات TTCN؛

3) باعتبارها أساس قواعد تركيب (كلما كان له علاقة وأساس تركيبي) لأنساق تقديم مختلفة.

ويمكن استخدام لغة النواة مستقلة عن أنساق التقديم. ومع ذلك، لا يمكن استخدام النسق الجدولي ولا النسق البياني دون لغة النواة. ويتم استخدام وتنفيذ أنساق التقديم هذه على أساس لغة النواة.

المصدر

وافقت لجنة الدراسات 17 (2005-2008) التابعة لقطاع تقييس الاتصالات بتاريخ 16 مارس 2006 على التوصية ITU-T Z 140 بموجب الإجراء الوارد في التوصية ITU-T A.8.

تمهيـــد

الاتحاد الدولي للاتصالات وكالة متخصصة للأمم المتحدة في ميدان الاتصالات وتكنولوجيات المعلومات والاتصالات (ICT). وقطاع تقييس الاتصالات (ITU-T) هو هيئة دائمة في الاتحاد الدولي للاتصالات. وهو مسؤول عن دراسة المسائل التقنية والمسائل المتعلقة بالتشغيل والتعريفة، وإصدار التوصيات بشأنها بغرض تقييس الاتصالات على الصعيد العالمي.

وتحدد الجمعية العالمية لتقييس الاتصالات (WTSA) التي تجتمع مرة كل أربع سنوات المواضيع التي يجب أن تدرسها لجان الدراسات التابعة لقطاع تقييس الاتصالات وأن تُصدر توصيات بشأنها.

وتتم الموافقة على هذه التوصيات وفقاً للإجراء الموضح في القرار رقم 1 الصادر عن الجمعية العالمية لتقييس الاتصالات.

وفي بعض مجالات تكنولوجيا المعلومات التي تقع ضمن اختصاص قطاع تقييس الاتصالات، تعد المعايير اللازمة على أساس التعاون مع المنظمة الدولية للتوحيد القياسي (ISO) واللجنة الكهرتقنية الدولية (IEC).

ملاحظـــة

تستخدم كلمة "الإدارة" في هذه التوصية لتدل بصورة موجزة سواء على إدارة اتصالات أو على وكالة تشغيل معترف بها.

والتقيد بهذه التوصية اختياري. غير أنها قد تضم بعض الأحكام الإلزامية (بهدف تأمين قابلية التشغيل البيني والتطبيق مثلاً). ويعتبر التقيّد بهذه التوصية حاصلاً عندما يتم التقيّد بجميع هذه الأحكام الإلزامية. ويستخدم فعل "يجب" وصيغ ملزمة أخرى مثل فعل "ينبغي" وصيغها النافية للتعبير عن متطلبات معينة، ولا يعني استعمال هذه الصيغ أن التقيّد بهذه التوصية إلزامي.

حقوق الملكية الفكرية

يسترعي الاتحاد الانتباه إلى أن تطبيق هذه التوصية أو تنفيذها قد يستلزم استعمال حق من حقوق الملكية الفكرية. ولا يتخذ الاتحاد أي موقف من القرائن المتعلقة بحقوق الملكية الفكرية أو صلاحيتها أو نطاق تطبيقها سواء طالب بها عضو من أعضاء الاتحاد أو طرف آخر لا تشمله عملية إعداد التوصيات.

وعند الموافقة على هذه التوصية، لم يكن الاتحاد قد تلقى إخطاراً بملكية فكرية تحميها براءات الاختراع يمكن المطالبة بها لتنفيذ هذه التوصية. ومع ذلك، ونظراً إلى أن هذه المعلومات قد لا تكون هي الأحدث، يوصى المسؤولون عن تنفيذ هذه التوصية بالاطلاع على قاعدة المعطيات الخاصة ببراءات الاختراع في مكتب تقييس الاتصالات (TSB) في الموقع http://www.itu.int/ITU-T/ipr/.

المحتـــويات

		الصفحة
1	مجال التطبيق	1
2	المراجع	1
3	التعاريف والمختصرات	2
	1.3 التعاريف	2
	2.3 المختصرات	4
4	مقدمة	4
	0.4 عـــام	4
	1.4 لغة النواة وأنساق التقديم	5
	2.4 الإجماع على المواصفة	5
	3.4 المطابقة	5
5	عناصر لغة أساسية	6
	0.5 عـــام	6
	1.5 ترتيب عناصر لغة	7
	2.5 معلميّة	7
	3.5 قواعد منظورية	9
	4.5 معرفات وكلمات مفتاحية	11
6	أنماط وقيم	12
	0.6 عـــام	12
	1.6 أنماط وقيم أساسية	12
	2.6 تنميط فرعي لأنماط أساسية	15
	3.6 أنماط وقيم مبنية	17
	4.6 النمط anytype	23
	5.6 المصفوفات	24
	6.6 أنماط تكرارية	25
	7.6 مواءمة نمط	25
7	وحدات	29
	0.7 عام	29
	1.7 تسمية وحدات	29
	2.7 معلمات وحدة	30
	3.7 جزء تعاريف وحدة	30
	4.7 جزء التحكم في الوحدة	31
	5.7 الاستيراد من وحدات	32
8	تشكيلات اختبار	38
	0.8 عام	38
	1.8 نموذج اتصالات منفذ	39
	2.8 تقييدات على توصيلات	40
	3.8 السطح البيني لنظام اختبار مجرد	41
	4.8 تعريف أنماط منفذ اتصالات	42
	5.8 تعريف أنماط مكون	43
	6.8 عنونة كيانات داخل SUT	45
	7.8 مراجع مكون	45
	8.8 تعريف السطح البيني لنظام اختبار	46
9	إعلان عن ثوابت	47
10	إعلان عن متغيرات	47
	0.10 عام	47
	1.10 متغيرات قيمة	47
	2.10 متغيرات مقاس	48

الصفحة

11 الإعلان عن مؤقتات 48
0.11 عـــام 48
1.11 مؤقتات كمعلمات 48
12 إعلان رسائل 49
13 إعلان توقيعات إجراء 49
0.13 عـــام 49
1.13 توقيعات لاتصالات Blocking وNon-blocking 49
2.13 معلمات لتوقيعات إجراء 49
3.13 القيمة التي تعيد إجراءات بعيدة 50
4.13 تحديد استثناءات 50
14 إعلان مقاسات 50
0.14 عـــام 50
1.14 إعلان مقاسات رسالة 51
2.14 إعلان مقاسات توقيع 52
3.14 آليات مواءمة مقاس 53
4.14 معلمية مقاسات 57
5.14 فـــارغ 57
6.14 مقاسات معدلة 57
7.14 تغيير مجالات مقاس 59
8.14 عملية مواءمة 59
9.14 عملية Valueof 59
15 المشغلون 59
0.15 عـــام 59
1.15 مشغلون حسابيون 61
2.15 مشغلو سلسلة 61
3.15 مشغلون اتصاليون 61
4.15 مشغلون منطقيون 63
5.15 مشغلو بتات 63
6.15 مشغلو زحزحة 64
7.15 مشغلو دوران 64
16 وظائف وaltsteps 65
1.16 وظائف 65
2.16 Altsteps 68
3.16 وظائف وaltsteps لأنماط مختلفة لمكون 70
17 اختبارات مجردة 70
0.17 عـــام 70
1.17 معلمية اختبارات مجردة 71
18 نظرة شاملة على بيانات وعمليات برنامج 71
19 تعبيرات وعناصر أساسية لبرنامج 74
0.19 عـــام 74
1.19 تعبيرات 74
2.19 تخصيصات 74
3.19 بيان Log 74
4.19 بيان Label 76
5.19 بيان Goto 76
6.19 بيان If-else 77
7.19 بيان For 78
8.19 بيان While 78
9.19 بيان Do-while 78

الصفحة

10.19 بيان تنفيذ Stop 78
11.19 بيان Select Case 79
20 بيانات سلوكية لبرنامج 80
0.20 عـــام 80
1.20 سلوك بديل 80
2.20 بيان Repeat 84
3.20 سلوك مشذر 84
4.20 بيان Return 85
21 مناولة بالتغيب 86
0.21 عـــام 86
1.21 آلية التغيُّب 86
2.21 مراجع تغيب 87
3.21 عملية Activate 87
4.21 عملية Deactivate 88
22 عمليات تشكيل 88
0.22 عـــام 88
1.22 عملية Create 89
2.22 عمليات Connect وMap 90
3.22 عمليات Disconnect وUnmap 91
4.22 عمليات MTC وSystem وSelf 92
5.22 عملية Start لمكون اختبار 92
6.22 عملية Stop لسلوك اختبار 93
7.22 عملية Running 94
8.22 عملية Done 94
9.22 عملية `kill` لمكون اختبار 95
10.22 عملية Alive 95
11.22 عملية killed 96
12.22 استخدام مصفوفات مكونات 96
13.22 موجز استخدام any وall مع مكونات 97
23 عمليات اتصالات 97
0.23 عـــام 97
1.23 نسق عام لعمليات اتصالات 98
2.23 اتصالات قائمة على رسالة 100
3.23 اتصالات قائمة على أساس إجراء 103
4.23 عملية Check 112
5.23 التحكم في منافذ اتصالات 113
6.23 استخدام any and all مع منافذ 114
24 عمليات مؤقت 114
0.24 عـــام 114
1.24 عملية Start timer 115
2.24 عملية Stop timer 115
3.24 عملية Read timer 116
4.24 عملية Running timer 116
5.24 عملية Timeout 116
6.24 موجز استخدام any وall مع مؤقتات 116
25 عملية Test verdict 117
0.25 عـــام 117
1.25 حكم اختبار مجرد 117
2.25 قيم حكم وقواعد الكتابة الفوقية 117

الصفحة

26 أعمال خارجية 118
27 جزء تحكم وحدة 118
0.27 عـــام 118
1.27 تنفيذ اختبارات مجردة 119
2.27 إنهاء اختبارات مجردة 119
3.27 التحكم في تنفيذ اختبارات مجردة 119
4.27 اختيار اختبارات مجردة 119
5.27 استخدام مؤقتات في التحكم 120
28 تحديد نعوت 121
0.28 عـــام 121
1.28 عرض نعوت 121
2.28 تشفير قيم 121
3.28 نعوت تمديد 123
4.28 منظور لنعوت 123
5.28 قواعد الكتابة الفوقية لنعوت 124
6.28 تغيير نعوت لعناصر لغة مستوردة 125
الملحق A – BNF وعلم الدلالات السكوني 126
1.A TTCN-3 BNF 126
الملحق B – مواءمة قيم واصلة 144
1.B آليات مواءمة مقاس 144
الملحق C – وظائف TTCN-3 المعرّفة مسبقاً 153
0.C إجراءات عامة لمناولة استثناء 153
1.C تحويل صحيح إلى سمة 153
2.C تحويل سمة إلى صحيح 153
3.C تحويل صحيح إلى سمة عالمية 153
4.C تحويل سمة عالمية إلى صحيح 153
5.C تحويل سلسلة ثنائية إلى صحيح 153
6.C تحويل سلسلة ستة عشرية إلى صحيح 153
7.C تحويل سلسلة أثمونات إلى صحيح 154
8.C تحويل سلسلة سمات إلى صحيح 154
9.C تحويل صحيح إلى سلسلة ثنائية 154
10.C تحويل صحيح إلى سلسلة ستة عشرية 154
11.C تحويل صحيح إلى سلسلة أثمونات 154
12.C تحويل صحيح إلى سلسلة سمات 155
13.C طول نمط سلسلة 155
14.C عدد عناصر في قيمة مبنية 155
15.C وظيفة IsPresent 156
16.C وظيفة IsChosen 156
17.C وظيفة Regexp 156
18.C تحويل سلسلة ثنائية إلى سلسلة سمات 157
19.C تحويل سلسلة ستة عشرية إلى سلسلة سمات 157
20.C تحويل سلسلة أثمونات إلى سلسلة سمات 157
21.C تحويل سلسلة سمات إلى سلسلة أثمونات 157
22.C تحويل سلسلة ثنائية إلى سلسلة ستة عشرية 157
23.C تحويل سلسلة ستة عشرية إلى سلسلة أثمونات 158
24.C تحويل سلسلة ثنائية إلى سلسلة أثمونات 158
25.C تحويل سلسلة ستة عشرية إلى سلسلة ثنائية 158
26.C تحويل سلسلة أثمونات إلى سلسلة ستة عشرية 158
27.C تحويل سلسلة أثمونات إلى سلسلة ثنائية 159
28.C تحويل صحيح إلى طليق 159
29.C تحويل طليق إلى صحيح 159

الصفحة

30.C وظيفة مولد عدد عشوائي 159
31.C وظيفة سلسلة فرعية 159
32.C عدد العناصر في نمط مبني 160
33.C تحويل سلسلة سمات إلى طليق 160
34.C وظيفة Replace 160
35.C تحويل سلسلة أثمونات إلى سلسلة سمات 161
36.C تحويل سلسلة سمات إلى سلسلة أثمونات 161
الملحق D (للعلم) – فارغ 162
الملحق E (للعلم) – مكتبة أنماط مفيدة 163
1.E تحديدات 163
2.E أنماط TTCN-3 مفيدة 163
الملحق F (للعلم) – عمليات على أشياء نشيطة TTCN-3 166
1.F عـــام 166
2.F مكونات اختبار 167
3.F المؤقتات 170
4.F منافذ 171
الملحق G (للعلم) – خاصيات لغة لا يُنصح بها 174
1.G تعريف أسلوب زمرة لمعلمات وحدة 174
2.G استيراد متكرر 174
3.G استخدام `all` في تعاريف نمط منفذ 174
بيبليوغرافيا 175

الإصدار 3 من الاختبار وترميز ضبط الاختبار (TTCN-3): لغة النواة

1 مجال التطبيق

تحدد هذه التوصية TTCN-3 (الاختبار وترميز التحكم في الاختبار 3) المقصود لمواصفة متواليات الاختبارات التي تكون مستقلة عن المنصات وطرائق الاختبار وطبقات البروتوكول والبروتوكولات. ويمكن استخدام TTCN-3 لمواصفة جميع أنماط اختبارات نظام تفاعلي عبر منافذ اتصالات متنوعة. ومن بين مجالات التطبيق اختبارات بروتوكولات (بما في ذلك بروتوكولات اتصالات متنقلة والإنترنت) واختبارات خدمات (بما في ذلك خدمات إضافية) واختبارات وحدة واختبارات المنصات القائمة على معمارية وسيط مطالب لأغراض مشتركة (CORBA) واختبارات السطوح البينية لبرمجة التطبيق (APIs). ومواصفة متواليات الاختبارات لبروتوكولات الطبقة المادية هي خارج مدى هذه التوصية.

إن القصد من استخدام TTCN-3 هو لمواصفة متواليات اختبارات مستقلة عن طرق الاختبار والطبقات والبروتوكولات. وتُعّرف أنساق التقديم المختلفة لـ TTCN-3 مثل نسق تقديم جدولي (التوصية ITU-T Z.141 [1]) ونسق تقديم بياني (التوصية ITU-T Z.142 [2]). ومواصفة هذه الأنساق هي خارج مدى هذه التوصية.

وبينما تصميم TTCN-3 قد أخذ في الاعتبار التنفيذ النهائي لمترجمي ومُصرفي TTCN-3 في عين الاعتبار، فإن وسائل تحقيق متواليات الاختبارات القابلة للتنفيذ (ETS) من متواليات اختبارات مجردة (ATS) هي خارج مدى هذه التوصية.

2 المراجع

تتضمن التوصيات التالية لقطاع تقييس الاتصالات وغيرها من المراجع أحكاماً تشكل من خلال الإشارة إليها في هذا النص جزءاً لا يتجزأ من هذه التوصية. وقد كانت جميع الطبعات المذكورة سارية الصلاحية في وقت النشر. ولما كانت جميع التوصيات والمراجع الأخرى تخضع إلى المراجعة، يرجى من جميع المستعملين لهذه التوصية السعي إلى تطبيق أحدث طبعة للتوصيات والمراجع الأخرى الواردة أدناه. وتُنشر بانتظام قائمة توصيات قطاع تقييس الاتصالات السارية الصلاحية. والإشارة إلى وثيقة ما في هذه التوصية لا يضفي على الوثيقة في حد ذاتها صفة التوصية.

[1] التوصية ITU-T Z.141 (2006)، *الاختبار وترميز ضبط الاختبار، الإصدار الثالث (TTCN-3)، نسق التقديم الجدولي.*

[2] التوصية ITU-T Z.142 (2006)، *الاختبار وترميز التحكم في الاختبار، الصياغة 3 (TTCN-3) – نسق تقديم بياني (GFT).*

[3] التوصية ITU-T Z.143 (2006)، *الإصدار 3 من الاختبار وترميز ضبط الاختبار: الدلالة التشغيلية (TTCN-3).*

[4] التوصية ITU-T Z.144 (2006)، *النسخة 3 من الاختبار وترميز ضبط الاختبار (TTCN-3): السطح بيني لوقت التسيير (TRI).*

[5] التوصية ITU-T Z.145 (2006)، *الاختبار وترميز التحكم في الاختبار الصياغة 3 (TTCN-3): السطح البيني للتحكم (TCI).*

[6] التوصية ITU-T Z.146 (2006)، *الاختبار وترميز وضبط الاختبار، الصيغة 3 (TTCN-3): استخدام الترميز ASN-1 مع الترميز TTCN-3.*

[7] ITU-T Recommendation X.290 (1995), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – General concepts*.

ISO/IEC 9646-1:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts*.

[8] ITU-T Recommendation X.292 (2002), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – The Tree and Tabular Combined Notation (TTCN)*.

ISO/IEC 9646-3:1998, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN)*.

[9] ITU-T Recommendation T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange*.

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*.

[10] ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*.

[11] ISO/IEC 6429:1992, *Information technology – Control functions for coded character sets*.

3 التعاريف والمختصرات

1.3 التعاريف

لأغراض هذه التوصية، تنطبق المصطلحات والتعاريف الواردة في التوصيتين ITU-T X.290[7] وITU-T X.290[8] وفي التوصيات التالية:

1.1.3 معلّمة فعلية: قيمة أو تعبير أو مقاس أو مرجع اسم (مُعرف) يمرر على أنه معلمة لكيان منفذ (وظيفة واختبار مجرد وaltstep وما إلى ذلك) كما عرف في مكان التنفيذ.

ملاحظة – يمرر عدد وترتيب ونمط جميع المعلمات التي تعتبر تنفيذاً وحيداً مع قائمة المعلمات الرسمية كما عرفت في الكيان المنفذ.

2.1.3 أنماط أساسية: مجموعة من أنماط TTCN-3 معرفة مسبقاً واردة في 0.1.6 و1.1.6.

ملاحظة – يشار إلى الأنماط الأساسية بواسطة أسمائها.

3.1.3 نمط متلائم: إن TTCN-3 ليس منمطاً بقوة ولكن اللغة تتطلب ملاءمة نمط.

ملاحظة – إن المتغيرات والثوابت والمقاسات وما إلى ذلك لها أنماط متلائمة إذا تم تلبية الشروط في 7.6.

4.1.3 منفذ اتصالات: آلية مجردة لتيسير الاتصالات بين مكونات اختبار.

ملاحظة – يوضع نموذج منفذ اتصالات باعتباره صف انتظار FIFO في الاتجاه المستقبِل. ويمكن أن تكون المنافذ قائمة على رسالة أو إجراء أو كلاهما.

5.1.3 أنماط معطيات: اسم شائع لأنماط أساسية بسيطة ولأنماط سلسلة بسيطة وأنماط مبنية ونمط معطيات خاص **anytype** وجميع الأنماط المعرفة للمستعمل القائمة عليها (انظر الجدول 3).

6.1.3 أنماط معرفة (أنماط TTCN-3 المعرفة): مجموعة لجميع أنماط TTCN-3 المُعرّفة مسبقاً (الأنماط الأساسية وجميع الأنماط المبنية ونمط **anytype** والعنوان والمنفذ وأنماط مكونات ونمط بالغيب) وجميع الأنماط المعرّفة للمستعمِل المعلَن عنها سواء في الوحدة أو مستوردة من وحدات TTCN-3 أخرى.

7.1.3 معلمية دينامية: نوع من المعلمية، تعتمد فيه المعلمات الفعلية على أحداث التنفيذ؛ مثلاً، تكون قيمة المعلمة الفعلية هي القيمة المستقبَلة خلال التنفيذ أو تعتمد على قيمة مستقبَلة بواسطة علاقة منطقية.

8.1.3 الاستثناء: في حالات الاتصالات القائمة على إجراء، يثار الاستثناء (إذا عُرّف) من قبل الكيان المجيب إذا لم يستطع الإجابة على نداء إجراء عن بُعد مع استجابة متوقعة عادية.

9.1.3 معلمة رسمية: اسم منمط أو مرجع مقاس منمط (مُعرّف) لم يتم استبانته وقت تعريف كيان (وظيفة واختبار مجرد وaltstep وما إلى ذلك) ولكن في وقت تنفيذه.

ملاحظة – تمرر قيم فعلية أو مقاسات (أو أسماؤها) لكي تستخدم في موضع المعلمات الرسمية من موضع تنفيذ الكيان (انظر أيضاً تعريف معلمة فعلية).

10.1.3 قابلية للرؤية عامة: نعت لكيان (معلمة وحدة وثابت ومقاس وما إلى ذلك) يمكن الإشارة إلى مُعرفه في أي مكان في الوحدة حيث يُعرف على أنه يشمل جميع الوظائف والاختبارات المجردة وaltstep المُعرفة في نفس الوحدة وجزء التحكم لتلك الوحدة.

11.1.3 بيان مطابقة تنفيذ (ICS): انظر التوصية ITU-T X.290 [7].

12.1.3 معلومات إضافية لتنفيذ اختبارات (IXIT): انظر التوصية ITU-T X.290 [7].

13.1.3 تنفيذ تحت الاختبار: انظر التوصية ITU-T X.290 [7].

14.1.3 أنماط معروفة: مجموعة من جميع أنماط TTCN-3 المعرفة مسبقاً وأنماط معرفة في وحدة TTCN-3 وأنماط مستوردة إلى تلك الوحدة من وحدات TTCN-3 أخرى أو من وحدات غير TTCN-3.

15.1.3 الجانب الأيسر (لتخصيص): مُعرف متغير لقيمة أو مقاس أو اسم مجال لقيمة نمط مبني أو متغير لمقاس (بما في ذلك دليل صفيف إن وجد) يوجد على يسار رمز تخصيص (:=).

ملاحظة – إن ثابتاً أو معلمة محددة أو مؤقتاً أو اسم مجال نمط مبنياً أو رأسية مقاس (بما في ذلك نمط مقاس واسم ومعلمة رسمية) توجد على يسار رمز تخصيص (:=) في إعلانات و/أو تعاريف مقاس معدل هي خارج مدى هذا التعريف باعتبارها ليست جزءاً من تخصيص.

16.1.3 قابلية للرؤية محلية: نعت كيان (ثابت ومتغير وما إلى ذلك) حيث يمكن الإشارة إلى معرفه فقط في الوظيفة أو اختبار مجرد أو **altstep** حيث يُعرف.

17.1.3 مكوّن الاختبار الرئيسي (MTC): انظر التوصية ITU-T X.292 [8].

18.1.3 تمرير معلمة حسب القيمة: طريقة تمرير معلمات حيث المتغيرات تُقيّم قبل دخول الكيان المعلمي.

ملاحظة – تمرر متغيرات القيم فقط وتتغير إلى متغيرات في الكيان المطلوب الذي ليس له تأثير على المتغيرات الفعلية كما يراها الطالب.

19.1.3 تمرير معلمة حسب المرجع: طريقة تمرير معلمات حيث المتغيرات لا تُقيّم قبل دخول الوظيفة وaltstep وما إلى ذلك ويمرر مرجع إلى متغير بواسطة إجراء مناداة (وظيفة وaltstep وما إلى ذلك) إلى الإجراء المطلوب.

ملاحظة – يكون لجميع التغييرات في المتغيرات في الإجراء المطلوب تأثير على المتغيرات الفعلية كما يراها الطالب.

20.1.3 مكون اختبار متوازي (PTC): انظر التوصية [8] ITU-T X.292.

21.1.3 الجانب الأيمن (التخصيص): تعبير أو مرجع مقاس أو مُعرف معلمة توقيع توجد على يمين رمز تخصيص (:=).

ملاحظة – إن تعبير ومراجع مقاسات توجد على يمين رمز تخصيص (:=) في ثابت أو معلمة وحدة أو مؤقت أو مقاس أو إعلانات مقاس معدل هي خارج مدى هذا التعريف باعتبارها ليست جزء من تخصيص.

22.1.3 نمط جذر: نمط أساسي أو نمط مبني أو نمط معطيات خاص أو نمط تشكيل خاص أو نمط بالتغيب خاص يمكن لنمط TTCN-3 المعرف للمستعمل تتبعه.

23.1.3 معلمية سكونية: نوع من معلمية تكون فيه المعلمات الفعلية مستقلة عن أحداث التنفيذ؛ مثلاً، معروف عند وقت التصريف أو، في حالة معلمات وحدة، معروف عند بداية تنفيذ متوالية اختبار (مثلاً، معروفة من مواصفة متوالية اختبار، هنا تُعد تعاريف مستوردة مستمرة أو أن نظام اختبار على وعي بقيمته قبل وقت التنفيذ).

ملاحظة – إن جميع الأنماط معروفة عند وقت التصريف، أي، أنها موثقة سكونياً.

24.1.3 تنميط قوي: تنفيذ صارم لملاءمة نمط بواسطة تكافؤ اسم نمط دون استثناءات.

25.1.3 نظام تحت الاختبار (SUT): انظر التوصية [7] ITU-T X.290.

26.1.3 مقاس: إن مقاسات TTCN-3 هي بنيات معطيات محددة للاختبار؛ مستخدمة إما لإرسال مجموعة من قيم مميزة أو التأكد من أن مجموعة القيم المستقبَلة تتواءم مع مواصفة المقاس.

27.1.3 سلوك اختبار: (أو سلوك) اختبار مجرد أو وظيفة بدأت على مكون اختبار عند تنفيذ بيان مكون (أصفر) وتطلب جميع الوظائف و**execute** أو **start** تكرارياً.

ملاحظة – خلال تنفيذ اختبار مجرد يكون لكل مكون اختبار سلوكه الخاص ومن ثم يمكن تنفيذ سلوكيات اختبار عديدة متلازمة في نظام الاختبار (أي، يمكن اعتبار اختبار مجرد تجميع لسلوكيات اختبار).

28.1.3 اختبار مجرد: انظر التوصية [7] ITU-T X.290.

29.1.3 خطأ اختبار مجرد: انظر التوصية [7] ITU-T X.290.

30.1.3 متوالية اختبار: مجموعة من وحدات TTCN-3 تحتوي على مجموعة كاملة معرفة لاختبارات مجردة، تكمل اختيارياً جزء تحكم TTCN-3 واحد أو أكثر.

31.1.3 نظام اختبار: انظر التوصية [7] ITU-T X.290.

32.1.3 سطح بيني لنظام اختبار: مكون اختبار يوفر تقابل منافذ متاحة في نظام اختبار TTCN-3 (مجرد) يوفرها SUT.

33.1.3 مواءمة نمط: خاصية لغة تسمح باستخدام قيم أو تعبيرات أو مقاسات لنمط معين كقيم فعلية لنمط آخر (مثل، عند التخصيصات، كمعلمات فعلية عند طلب وظيفة ومراجعة مقاس وما إلى ذلك أو كقيمة عودة لوظيفة).

ملاحظة – يتعين أن يكون كلا النمطين والقيمة الحالية لقيمة أو تعبير أو مقاس متوائمة مع النمط الآخر.

34.1.3 معلمية قيمة: المقدرة على تمرير قيمة أو مقاس كمعلمة فعلية في شيء معلمية.

ملاحظة – تكمل معلمة القيمة الفعلية هذه مواصفة ذلك الشيء .

35.1.3 نمط معرف للمستعمِل: نمط يعرف بواسطة تنميط فرعي لنمط أساسي أو الإعلان عن نمط مبني.

ملاحظة – يشار إلى الأنماط المعرفة للمستعملين حسب مُعرفاتها (أسمائها).

36.1.3 ترميز قيمة: ترميز يتصاحب مُعرف مع قيمة ما أو مدى من نمط معين.

ملاحظة – يمكن أن تكون القيم ثوابت أو متغيرات.

2.3 المختصرات

تستخدم هذه التوصية المختصرات التالية:

API السطح البيني لبرمجة تطبيق *(Application Programming Interface)*

ATS متوالية اختبار مجرد *(Abstract Test Suite)*

BMP مستوى أساسي متعدد اللغات *(Basic Multilingual Plane)*

BNF شكل باكوس – ناور *(Backus-Naur Form)*

CORBA معمارية وسيط مطالب لأغراض مشتركة *(Common Object Request Broker Architecture)*

ETS متوالية اختبار قابلة للتطبيق *(Executable Test Suite)*

FIFO الأول في الدخول، الأول في الخروج *(First In First Out)*

ICS بيان مطابقة تنفيذ *(Implementation Conformance Statement)*

IRV صياغة دولية لمرجع *(International Reference Version)*

IUT تنفيذ تحت الاختبار *(Implementation Under Test)*

IXIT معلومات إضافية لتنفيذ اختبار *(Implementation eXtra Information for Testing)*

MTC مكون اختبار رئيسي *(Main Test Component)*

PTC مكون اختبار متوازي *(Parallel Test Component)*

SUT نظام تحت الاختبار *(System Under Test)*

TSI سطح بيني لنظام اختبار *(Test System Interface)*

4 مقدمة

0.4 عـــام

إن TTCN-3 هي لغة مرنة قوية قابلة للتطبيق على مواصفة لجميع أنماط اختبارات نظم تفاعلية عبر تنوع من الأسطح البينية للاتصالات. والمناطق المعيارية للتطبيق هي اختبارات البروتوكولات (بما في ذلك البروتوكولات المتنقلة والإنترنت) واختبارات الخدمات (بما في ذلك خدمات إضافية) واختبارات الوحدات واختبارات المنصات القائمة على CORBA واختبارات API وما إلى ذلك. ولا يقتصر TTCN-3 على اختبارات المطابقة، ويمكن أن تستخدم لأنواع كثيرة أخرى من الاختبارات بما في ذلك القابلية للتشغيل البيني والقوة والارتداد والنظام واختبارات التكامل.

يشمل TTCN-3 الخاصيات الضرورية التالية:

- المقدرة على تحديد تشكيلات اختبارات متلازمة دينامية؛
- عمليات اتصالات قائمة على إجراء وعلى رسالة؛
- المَقدِرة على تحديد تشفير معلومات ونعوت أخرى (بما في ذلك قابلية تمديدها إلى المستعمِل)؛
- المَقدِرة على تحديد معطيات ومقاسات توقيعات مع آليات مواءمة قوية؛
- معلمية قيمة؛
- تخصيص ومناولة أحكام الاختبارات؛
- معلمية متوالية اختبار وآليات اختيار اختبارات مجردة؛
- الاستخدام المجمع لـ TTCN-3 مع لغات أخرى؛
- قواعد تركيب مُعرفة بعناية ونسق تبادل وعلم دلالات سكوني؛
- أنساق تقديم مختلفة (مثل، أنساق تقديم جدولي وبياني)؛
- خوارزمية تنفيذ دقيقة (علم دلالات تشغيلي).

1.4 لغة النواة وأنساق التقديم

تنفصل مواصفة TTCN-3 إلى عدة أجزاء. والجزء الأول، المُعرف في هذه التوصية، هو لغة النواة. والجزء الثاني، المُعرّف في التوصية ITU-T Z.141 [1] هو نسق تقديم جدولي. والجزء الثالث، المعرَّف في التوصية ITU-T Z.142 [2]، هو نسق التقديم البياني. والجزء الرابع، التوصية ITU-T Z.143 [3] يحتوى على علم الدلالات التشغيلي للغة. والجزء الخامس، التوصية ITU-T Z.144 [4]، يعرّف السطح البيني لتنفيذ TTCN-3، والجزء السادس، ITU-T Z.145 [5]، يعرّف السطح البيني لتحكم TTCN-3 والجزء السابع، التوصية ITU-T Z.146 [6]، يُحدد استخدام تعاريف ASN.1 مع TTCN-3.

وتخدم لغة النواة ثلاثة أغراض:

أ) كلغة نص قائمة على نص معمم في حد ذاتها؛

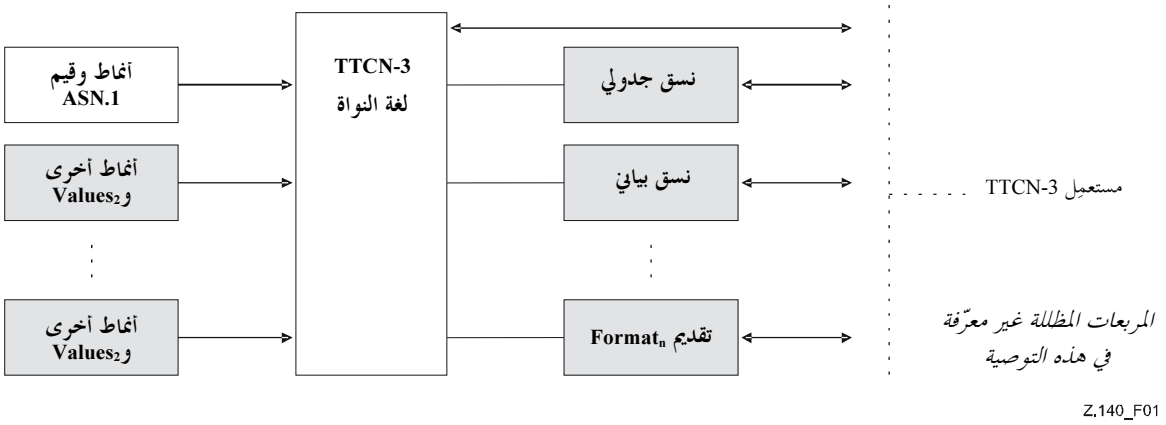
ب) كنسق تبادل معياري لمتواليات اختبارات TTCN-3 بين أدوات TTCN-3؛

ج) كأساس علم الدلالات (حينما يكون له علاقة، أساس تركيبي) لأنساق تقديم مختلفة.

يمكن استخدام لغة النواة مستقلة عن أنساق التقديم. ومع ذلك، لا يمكن استخدام نسق جدولي أو نسق بياني دون لغة النواة. ويجري استخدام وتنفيذ أنساق التقديم هذه على أساس لغة النواة.

إن النسق الجدولي والنسق البياني هما الأولان في مجموعة متوقعة من أنساق تقديم مختلفة. وقد تكون الأنساق الأخرى هذه أنساق تقديم معيارية أو قد تكون أنساق تقديم ملكية عرفها مستعملو TTCN-3 أنفسهم. ولا تعرف الأنساق الإضافية هذه في هذه التوصية.

يمكن استخدام TTCN-3 خيارياً مع ترميزات نمط وقيمة أخرى حيث تستخدم تعاريف في لغات أخرى كبديل لقواعد تركيب نمط معطيات وقيمة. وتحدد أجزاء أخرى من هذا المعيار استخدام بعض لغات أخرى مع TTCN-3. ولا يقتصر دعم لغات أخرى على المحددة في سلسلة ITU-T Z.140 من التوصيات ولكن تدعم لغات يعرف لها استخدام مجمع لـ TTCN-3، وتطبق القواعد الواردة في هذه التوصية.



الشكل Z.140/1 – رؤية المستعمل للغة النواة وأنساق التقديم المختلفة

تُعرّف لغة النواة بواسطة قواعد تركيب كاملة (انظر الملحق A) وعلم دلالات تشغيلي (التوصية ITU-T Z.143 [3]). وتحتوي على علم دلالات أدنى (وارد في متن هذه التوصية وفي الملحق A) الذي لا يقيد استخدام اللغة نتيجة لبعض التطبيق الفرعي لميدان أو منهجية.

2.4 الإجماع على المواصفة

تحدد اللغة بناء على قواعد التركيب وعلم الدلالات على أساس الوصف النصي في متن هذه التوصية (الأقسام من 5 إلى 28) وبطريقة رسمية في الملحق A. وفي كل حالة، عندما لا يكون الوصف النصي غير شامل، يستكمله الوصف الرسمي. وإذا كان الوصفان النصي والرسمي متناقضين، يكون للأخير الأسبقية.

3.4 المطابقة

بالنسبة لتنفيذ يدعي التطابق مع صياغة اللغة هذه، تنفذ جميع الخاصيات المحددة في هذه التوصية باتساق مع المتطلبات الواردة في هذه التوصية وفي التوصية ITU-T Z.143 [3].

5 عناصر لغة أساسية

0.5 عـــام

إن وحدة المستوى العلوي لـ TTCN-3 هي وحدة. ولا يمكن بناء وحدة في وحدات فرعية. ويمكن لوحدة أن تستورد تعاريف من وحدات أخرى. ويمكن أن يكون لوحدات معلمات وحدة تسمح بمعلمية متوالية اختبار.

تتألف وحدة من جزء التعاريف وجزء التحكم. ويعرف جزء التعاريف لوحدة مكونات اختبار ومنافذ اتصالات وأنماط معطيات وثوابت ومقاسات معطيات اختبار ووظائف وتوقيعات لنداءات إجراءات عند منافذ واختبارات مجردة وما إلى ذلك.

يطلب جزء التحكم لوحدة اختبارات مجردة ويتحكم في تنفيذها. ويمكن أن يعلن جزء التحكم أيضاً عن متغيرات (محلية) وما إلى ذلك. ويمكن استخدام بيانات برنامج **if-else** و**do-while** لتحديد اختيار وتنفيذ أمر اختبارات مجردة فردية. ولا يدعم مفهوم المتغيرات العالمية في TTCN-3.

لدى TTCN-3 عدد من أنماط معطيات أساسية معرفة مسبقاً، وكذلك أنماط مبنية مثل سجلات ومجموعات واتحادات وأنماط معدودة ومصفوفات.

ويوفر نوع خاص من بنية معطيات تسمى مقاس معلمية وآليات مواءمة لتحديد معطيات اختبار تُرسَل أو تُستقبَل عبر منافذ الاختبار. وتوفر العمليات على هذه المنافذ مقدرات اتصالات قائمة على رسالة وإجراء. ويمكن أن تستخدم نداءات إجراء لتنفيذ اختبارات لا تقوم على أساس رسالة.

يجري التعبير عن سلوك اختبار دينامي كاختبارات مجردة. وتشمل بيانات برنامج TTCN-3 آليات وصف سلوك قوي مثل استقبال بديل لاتصالات وإحداث مُؤقت وتشذير وسلوك بالتغيب. ويدعم أيضاً تخصيص حكم اختبار وآليات تسجيل.

وأخيراً، يمكن أن يخصص لعناصر لغة TTCN-3 نعوتاً مثل تشفير معلومات ونعوت عرض. ومن الممكن أيضاً تحديد نعوت معرفة للمستعمل (غير معيارية).

الجدول Z.140/1 – نظرة شاملة على عناصر لغة TTCN-3

مكوِّن اللغة	كلمة مفتاحية متصاحبة	محدد في تعاريف وحدة	محدد في تحكم وحدة	محدد في وظائف /اختبارات altsteps مجردة	محدد في نمط مكون اختبار
تعريف وحدة TTCN-3	**module**				
استيراد تعاريف من وحدة أخرى	**import**	نعم			
تجميع تعاريف	**group**	نعم			
تعاريف نمط معطيات	**type**	نعم			
تعاريف منفذ اتصالات	**port**	نعم			
تعاريف مكون اختبار	**component**	نعم			
تعاريف توقيع	**signature**	نعم			
تعاريف وظيفة خارجية/ثابت	**external**	نعم			
تعاريف ثابت	**const**	نعم	نعم	نعم	نعم
تعاريف مقاس معطيات/توقيع	**template**	نعم	نعم	نعم	نعم
تعاريف وظيفة	**function**	نعم			
تعاريف Altstep	**altstep**	نعم			
تعاريف اختبار مجرد	**testcase**	نعم			
إعلانات متغير قيمة	**var**		نعم	نعم	نعم
إعلانات متغير مقاس	**var template**		نعم	نعم	نعم
إعلانات مؤقت	**timer**		نعم	نعم	نعم
ملاحظة – تستخدم أفكار "تعريف" و"إعلان" لمتغيرات وثوابت وأنماط وعناصر لغة أخرى بطريقة تبادلية في جميع أنحاء التوصية هذه. والتمييز بين الفكرتين مفيد فقط لأغراض التنفيذ، كما هي الحالة في لغات البرمجة مثل C و++C وعلى مستوى TTCN-3، تكون الأفكار متساوية في المعنى.					

1.5 ترتيب عناصر لغة

بشكل عام، إن الترتيب الذي يتم للإعلانات هو اعتباطي. وفي داخل فدرة لبيانات وإعلانات، مثل جسم وظيفة أو فرع لبيان **if-else**، تتم جميع الإعلانات (إن وجدت) في بداية الفدرة فقط.

مثـــال:

```
// This is a legal mixing of TTCN-3 declarations
 :
var MyVarType MyVar2 := 3;
const integer MyConst:= 1;
if (x > 10)
{
  var integer MyVar1:= 1;
  :
  MyVar1:= MyVar1 + 10;
  :
}

 :
```

يمكن أن تتم الإعلانات في جزء تعاريف وحدة بأي ترتيب. ومع ذلك، في داخل جزء التحكم لوحدة وتعاريف اختبار مجرد ووظائف altstepsتتطلب جميعها إعلانات يجب أن تتوفر مسبقاً. ويعني هذا بشكل خاص، أن المتغيرات المحلية والمؤقتات المحلية والثوابت المحلية لن تستخدم أبداً قبل إعلانها. والاستثناءات الوحيدة لهذه القاعدة هي الوسوم. ويمكن استخدام مراجع أمامي لوسم في بيانات **goto** قبل حدوث الوسم (انظر 5.19).

2.5 معلميّة

0.2.5 معلميّة سكونية وديناميّة

تدعم TTCN-3 معلمية *value* طبقاً للحدود التالية:

أ) تكون عناصر لغة لا يمكن معلمتها هي: **const** و**var** و**timer** و**control** و**group** و**import**؛

ب) يسمح عنصر اللغة **module** بمعلمية قيمة *سكونية* لدعم معلمات متوالية اختبار، أي، قد يكون أو لا يكون من الممكن استبيان المعلمية هذه عند وقت التصريف ولكن تُستبان عند بدء التنفيذ (أي، *سكوني* عند التنفيذ). ويعني ذلك، أن عند التنفيذ، تكون قيم معلمة وحدة قابلة للرؤية عامة ولكن غير قابلة للتبادل؛

ج) جميع تعاريف **type** المعرفة للمستعمل (بما في ذلك تعاريف نمط مبني مثل record، set) ونمط address لتشكيل خاص يدعم معلمية قيمة *سكونية*، أي، تُستبان هذه المعلمية عند وقت التصريف؛

د) تدعم عناصر اللغة **template** و**signature** و**testcase** و**altstep** و**function** معلمية قيمة *دينامية* (أي، تستبان هذه المعلمية عند وقت التصريف).

يرد في الجدول 2 موجز لأي عناصر لغة يمكن معلمتها والتي يمكن تمريرها باعتبارها معلمات.

الجدول Z.140/2 – نظرة شاملة على عناصر لغة TTCN-3 قابلة للمعلمية

كلمة مفتاحية	معلمية قيمة	أنماط قيم يسمح بظهورها في قوائم معلمات رسمية/فعلية
module	Static at start of run-time	*Values of:* all basic types, all user-defined types and **address** type.
type (الملاحظة 1)	Static at compile-time	*Values of:* all basic types, all user-defined types and **address** type.
template	Dynamic at run-time	*Values of:* all basic types, all user-defined types, **address** type and **template**.
function	Dynamic at run-time	*Values of:* all basic types, all user-defined types, **address** type, **component** type, **port** type, **default**, **template** and **timer**.
altstep	Dynamic at run-time	*Values of:* all basic types, all user-defined types, **address** type, **component** type, **port** type, **default**, **template** and **timer**.
testcase	Dynamic at run-time	*Values of:* all basic types and of all user-defined types, **address** type and **template**.
signature	Dynamic at run-time	*Values of:* all basic types, all user-defined types and **address** type and **component** type.
الملاحظة 1 – إن **record of**، **set of**، **enumerated**، **port**، **component** و**sub-type definitions** لا تسمح بالمعلمية. **الملاحظة 2** – ترد في الأقسام ذات العلاقة في هذه التوصية أمثلة لقواعد التركيب واستخدام محدد للمعلمية مع عناصر لغة مختلفة.		

1.2.5 معلمة تمرر حسب المرجع وحسب القيمة

0.1.2.5 عـــام

تمرر، بالتغيب، جميع المعلمات الفعلية لأنماط أساسية وأنماط سلسلة أساسية وأنماط مبنية معرفة للمستعمل ونمط عنوان ونمط مكون حسب القيمة. ويدل هذا خيارياً بواسطة الكلمة المفتاحية **in**. ولتمرير معلمات من الأنماط المذكورة حسب المرجع، تستخدم الكلمات المفتاحية **out** أو **inout**.

تمر دائماً المؤقتات والمنافذ حسب المرجع. وتعرف معلمات مؤقت حسب الكلمة المفتاحية **timer**. وتعرف معلمات منفذ حسب نمط منفذها. ويمكن استخدام الكلمة المفتاحية **inout** خيارياً لتدل على المرور حسب المرجع.

1.1.2.5 المعلمات التي تمر حسب المرجع

إن المعلمات التي تمر حسب المرجع لها الحدود التالية:

أ) قد تحتوي قوائم معلمات رسمية فقط لـ altsteps وظائف وتوقيعات واختبارات مجردة على معلمات تمرر حسب المرجع؛

ملاحظة – توجد قيود أخرى على كيفية استخدام معلمات تمرر حسب المرجع في توقيعات (انظر القسم 23) وaltsteps (انظر 1.2.16 و1.3.21).

ب) تكون المعلمات الفعلية إما متغيرات قيمة أو مقاس أو منافذ أو مؤقتات.

مثـــال:

```
function MyFunction(inout boolean MyReferenceParameter){ … };
// MyReferenceParameter is passed by reference. The actual parameter can be read and set
// from within the function

function MyFunction(out template boolean MyReferenceParameter){ … };
// MyReferenceParameter is passed by reference. The actual parameter can only be set
// from within the function
```

2.1.2.5 المعلمات التي تمر حسب القيمة

إن المعلمات الفعلية التي تمر حسب القيمة تكون متغيرات وكذلك ثوابت ومقاسات وما إلى ذلك.

```
function MyFunction(in template MyTemplateType MyValueParameter){ … };
// MyValueParameter is passed by value, the in keyword is optional
```

2.2.5 قوائم المعلمات الرسمية والفعلية

إن عدد العناصر والترتيب الذي تظهر به في قائمة معلمة فعلية يكون هو نفس عدد العناصر وترتيبها التي تظهر به في قائمة معلمة رسمية متوافقة. وفضلاً عن ذلك، يكون نمط كل معلمة فعلية متوائم مع نمط كل معلمة رسمية متوافقة.

مثـــال:

```
// A function definition with a formal parameter list
function MyFunction(integer FormalPar1, boolean FormalPar2, bitstring FormalPar3) { … }

// A function call with an actual parameter list
MyFunction(123, true,'1100'B);
```

3.2.5 قائمة فارغة لمعلمة رسمية

إذا كانت قائمة معلمة رسمية لعناصر لغة TTCN-3 **function** أو **testcase** أو **signature** أو **altstep** أو **external function** فارغة، فإن الأقواس الفارغة تتضمن في الإعلان وفي تنفيذ ذلك العنصر. وفي جميع الحالات الأخرى تلغى الأقواس الفارغة.

مثـــال:

```
// A function definition with an empty parameter list shall be written as
function MyFunction(){ … }

// A record definition with an empty parameter list shall be written as
type record MyRecord { … }
```

4.2.5 قوائم معلمات متداخلة

يكون لجميع الكيانات المعلمية المحددة على أنها معلمة فعلية معلماتها الخاصة تمت استبيانها في قائمة المعلمة الفعلية على المستوى العلوي.

مثـــال:

```
// Given the message definition
type record MyMessageType
```

```
{
  integer field1,
  charstring  field2,
  boolean  field3
}

// A message template might be
template MyMessageType MyTemplate(integer MyValue) :=
{
  field1 := MyValue,
  field2 := pattern "abc*xyz",
  field3 := true
}

// A test case parameterized with a template might be
testcase TC001(template MyMessageType RxMsg) runs on PTC1 system TS1 {
:
MyPCO.receive(RxMsg);
}

// When the test case is called in the control part and the parameterized     template is
// used as an actual parameter, the actual parameters for template must be    provided
control
{ :
  execute( TC001(MyTemplate(7)));
  :
}
```

5.2.5 معلمات رسمية لنمط ومقاس

1.5.2.5 معلمية مع مقاسات ونعوت متوائمة

لتمكين مرور مقاسات أو رموز متوائمة باعتبارها معلمات فعلية تضاف كلمة مفتاحية **template** إضافية قبل مجال النمط المتوافق مع معلمة رسمية. ويجعل هذا المعلمة نمط مقاس، وبالفعل يمدد المعلمات المسموح بها لنمط متصاحب ليشمل مجموعة ملائمة لنعوت (انظر الملحق B) وكذلك المجموعة العادية من القيم. ولا تطلب مجالات معلمة مقاس حسب المرجع.

2.5.2.5 عناصر لغة تستخدم معلمات نمط ومقاس

يمكن فقط لـ **function** و**testcase** و**altstep** و**template** و**function** أن يكون لها معلمات رسمية لنمط ومقاس.

مثـــال:

```
function MyBehaviour(template MyMsgType MyFormalParameter)
runs on MyComponentType
{   :
  pco1.receive(MyFormalParameter);
    :
}
```

3.5 قواعد منظورية

0.3.5 عـــام

يوفر TTCN-3 سبع وحدات أساسية لمنظور:

أ) جزء تعاريف وحدة؛

ب) جزء تحكم لوحدة؛

ج) أنماط مكون؛

د) وظائف؛

هـ) altsteps؛

و) اختبارات مجردة؛

ز) "فدرات لبيانات وإعلانات" في بيانات مركبة.

الملاحظة 1 – يرد في 1.3.7 قواعد منظورية إضافية لرزم.

الملاحظة 2 – يرد في 7.19 قواعد منظورية إضافية لعدّادات عروات **for**.

تتألف كل وحدة منظور من إعلانات (خيارية). ووحدات المنظور: جزء التحكم لوحدة ووظائف واختبارات مجردة وaltsteps و"فدرات لبيانات وإعلانات" في بيانات مركبة يمكن أن تحدد – إضافياً – شكل سلوك بواسطة استخدام بيانات وعمليات برنامج TTCN-3 (انظر القسم 18).

إن التعاريف التي تتم في جزء تعاريف وحدة، ولكن خارج وحدات منظورية أخرى، هي منظورة بشكل عام، أي، يمكن أن تستخدم في أي مكان في الوحدة، بما في ذلك جميع وظائف واختبارات مجردة وaltsteps المعرفة في الوحدة وجزء التحكم. وتكون المعرفات المستوردة من وحدات أخرى قابلة للرؤية عامة أيضاً في جميع أنحاء الوحدة المستوردة.

ويكون للتعاريف التي تتم في جزء التحكم للوحدة قابلة للرؤية محلية، أي، يمكن أن تستخدم في جزء التحكم فقط.

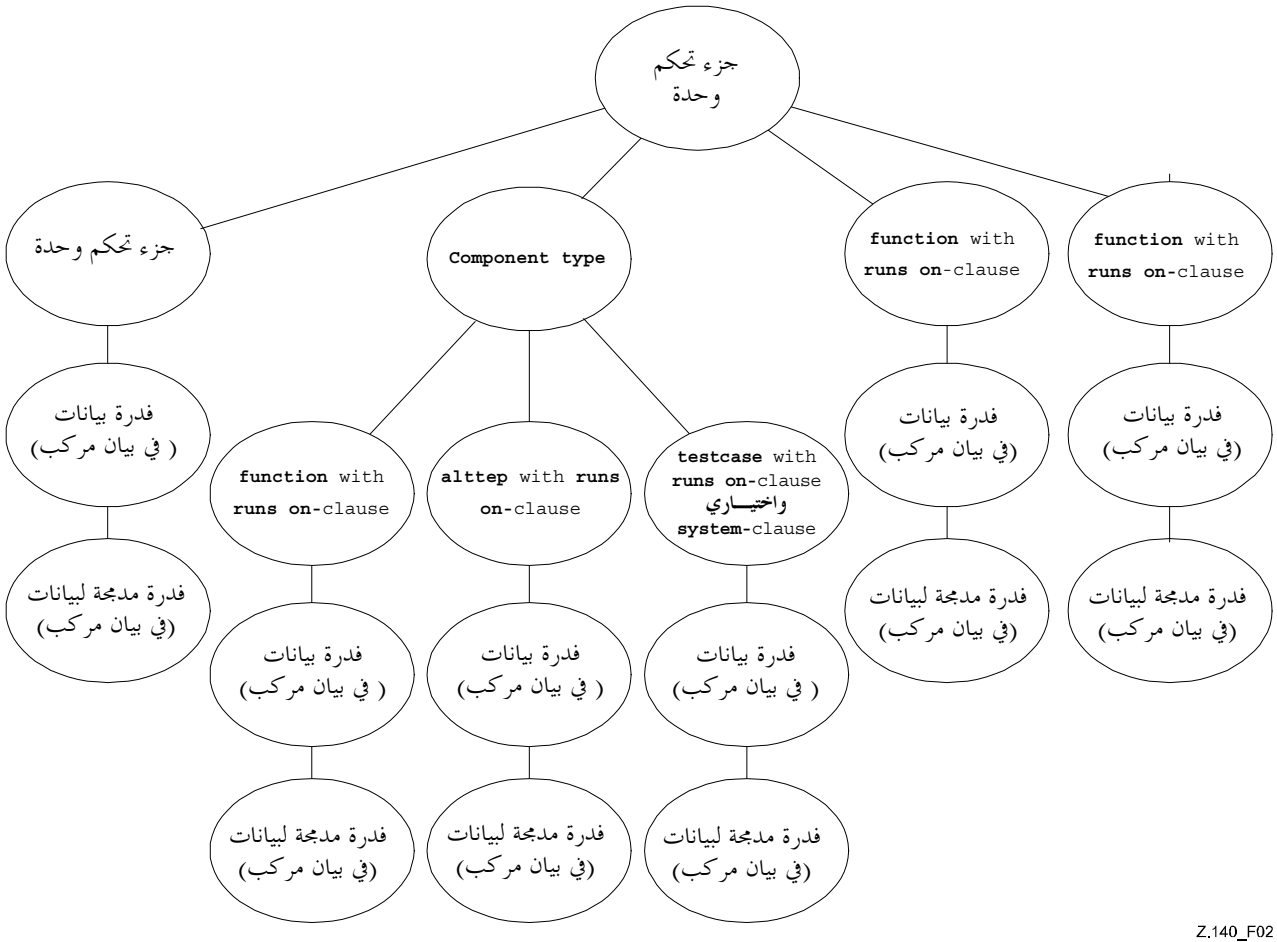
ويمكن استخدام التعاريف التي تتم في نمط مكون اختبار فقط في وظائف واختبارات مجردة وaltsteps التي تشير إلى نمط مكون أو نمط مكون اختبار متسق (انظر 3.16) بواسطة **runs on**-clause.

إن اختبارات مجردة وaltsteps ووظائف هي وحدات منظورية فردية دون أي علاقة تراتبية بينها، أي، تكون الإعلانات التي تتم عند بداية جسمها لها قابلية لرؤية محلية، وتستخدم فقط في اختبار مجرد أو altstep أو وظيفة معينة (مثل، لا يكون إعلان في اختبار مجرد قابل للرؤية في وظيفة طلبها اختبار مجرد أو في altstep استخدم بواسطة الاختبار المجرد).

تشمل البيانات المركبة مثل بيانات **if-else-** أو **while-** أو **do-while-** أو **alt-** "بيانات وإعلانات فدرة". ويمكن أن تستخدم في جزء التحكم من وحدة أو اختبارات مجردة أو altsteps أو وظائف، أو قد تكون مدمجة في بيانات مركبة أخرى، مثل بيان **if-else-** الذي يستخدم في **while**-loop.

يكون "لفدرات بيانات وإعلانات" لبيانات مركبة وبيانات مركبة مدمجة لها علاقة تراتبية لكل من وحدة منظورية، بما في ذلك "فدرة بيانات وإعلانات" معينة، ولأي "فدرة بيانات وإعلانات مدمجة". ويكون لإعلانات تتم في "فدرة بيانات وإعلانات" قابلية لرؤية محلية.

ويرد في الشكل 2 تراتبية وحدات منظورية. وتكون إعلانات وحدة منظورية على مستوى تراتبي أعلى قابلة للرؤية في جميع الوحدات عند مستويات أقل في نفس فرع التراتب. ولا تكون إعلانات وحدة منظورية على مستوى أقل من التراتبية قابلة للرؤية للوحدات على مستوى تراتبي أعلى.



الشكل Z.140/2 – تراتبية وحدات منظورية

مثـــال:

```
module MyModule
{ :
  const integer MyConst := 0; // MyConst is visible to MyBehaviourA and       MyBehaviourB
  :
  function MyBehaviourA()
  {   :
      const integer A := 1;   // The constant A is only visible to    MyBehaviourA
      :
  }

  function MyBehaviourB()
  {   :
      const integer B := 1;   // The constant B is only visible to MyBehaviourB
      :

  }
}
```

1.3.5 منظور معلمات رسمية

يقتصر منظور المعلمات الرسمية في عنصر لغة معلمية (مثلاً، في نداء وظيفة) على التعريف الذي تظهر فيه المعلمات، وعلى المستويات الدنيا للمنظور في نفس التراتب المنظوري. أي، تتبع قواعد المنظور العادية (انظر 0.3.5).

2.3.5 تفرُّد المعرِّفات

يتطلب TTCN-3 تفرد المعرفات، أي، تكون جميع المعرفات في نفس تراتب المنظور مميزة. ويعني هذا أن إعلاناً في مستوى متدني لمنظور لا يعيد استخدام نفس المعرف باعتباره إعلاناً في مستوى أعلى لمنظور في نفس فرع تراتبية المنظور. أما معرفات مجالات لأنماط مبنية وقيم وزمرات تعدد ليس من الضروري أن تكون متفردة بشكل عام؛ ومع ذلك، في حالة قيم تعدد، يُعاد استخدام المعرفات فقط لقيم تعدد في أنماط تعدد أخرى. وتنطبق أيضاً قواعد تفرد المعرف على معرفات معلمات رسمية.

مثـــال:

```
module MyModule
{ :
  const integer A := 1;
  :
  function MyBehaviourA()
  {   :
      const integer A := 1; // Is NOT allowed
      :
      if(…)
      {    :
       const boolean A := true; // Is NOT allowed
       :
      }
  }
}

// The following IS allowed as the constants are not declared in the same scope hierarchy
// (assuming there is no declaration of A in module header)
function MyBehaviourA()
{ :
  const integer A := 1;
  :
}

function MyBehaviourB()
{ :
  const integer A := 1;
  :
}
```

4.5 معرفات وكلمات مفتاحية

إن معرفات TTCN-3 حساسة للحروف اللاتينية الكبيرة والصغيرة. وتُكتب جميع الكلمات المفتاحية لـ TTCN-3 بحروف صغيرة (انظر الملحق A). ولا تستخدم الكلمات المفتاحية لـ TTCN-3 كمعرفات لأشياء TTCN-3 ولا كمعرفات لأشياء مستوردة من وحدات لغات أخرى.

6 أنماط وقيم

0.6 عـــام

يدعم TTCN-3 عدداً من أنماط أساسية معرفة مسبقاً. وتشمل عادة الأنماط الأساسية هذه أنماطاً متصاحبة مع لغة برمجة مثل **integer**، **boolean** وأنماط سلسلة وكذلك بعض أنماط TTCN-3 محددة مثل **verdicttype**. ويمكن بناء أنماط مبنية مثل أنماط **record** وأنماط **set** و**enumerated** وأنماط من الأنماط الأساسية هذه.

يعرف نمط معطيات خاصة **anytype** كاتحاد لجميع أنماط المعطيات المعروفة ونمط العنوان في وحدة.

ويمكن استخدام أنماط خاصة متصاحبة مع تشكيل اختبار مثل **address** و**port** و**component** لتعريف معمارية نظام الاختبار (انظر القسم 22).

يمكن استخدام النمط الخاص **default** لمناولة بالتغيب (انظر القسم 21).

يوجز الجدول 3 أنماط TTCN-3.

الجدول Z.140/3 – نظرة شاملة على أنماط TTCN-3

صنف النمط	الكلمة المفتاحية	نمط فرعي
أنماط بسيطة أساسية	**integer**	مدى، قائمة
	float	مدى، قائمة
	boolean	قائمة
	objid	قائمة
	verdicttype	قائمة
أنماط سلسلة أساسية	**bitstring**	قائمة، مدى
	hexstring	قائمة، مدى
	octetstring	قائمة، مدى
	charstring	مدى، قائمة، طول، مخطط
	universal charstring	مدى، قائمة، طول، مخطط
أنماط مبنية	**record**	قائمة (انظر الملاحظة)
	record of	قائمة (انظر الملاحظة)، طول
	set	قائمة (انظر الملاحظة)
	set of	قائمة (انظر الملاحظة)، طول
	enumerated	قائمة (انظر الملاحظة)
	union	قائمة (انظر الملاحظة)
أنماط خاصة لمعطيات	**anytype**	قائمة (انظر الملاحظة)
أنماط تشكيل خاصة	**address**	
	port	
	component	
أنماط بالتغيب خاصة	**default**	
ملاحظة – قائمة subtyping لهذه الأنماط ممكنة عند تعريف نمط مقيد جديد من نمط رئيسي موجود فعلاً ولكن ليس مباشر عند إعلان النمط الأساسي الأول.		

1.6 أنماط وقيم أساسية

0.1.6 أنماط وقيم أساسية خاصة

يدعم TTCN-3 الأنماط الأساسية التالية:

أ) **integer**: نمط مع قيم مميزة تكون موجبة وسالبة لأعداد كاملة، بما في ذلك صفر.

يدل على قيم نمط صحيح رقم واحد أو أكثر؛ ولا يكون الرقم الأول صفراً ما لم تكن القيمة صفراً؛ ويمثل القيمة صفر، صفر وحيد.

ب) **float**: نمط لوصف أعداد نقطة طليقة.

بشكل عام، يمكن تعريف أعداد نقطة طليقة كما يلي: *<mantissa>* × *<base>*$^{<exponent>}$، حيث *<mantissa>* يكون صحيحاً موجباً أو سالباً، *<base>* صحيح موجب (في أغلب الحالات 2 أو 10 أو 16) و*<exponent>* صحيح موجب أو سالب.

في TTCN-3، يقتصر ترميز قيمة عدد نقطة طليقة على قاعدة ذات قيمة 10. ويمكن التعبير عن قيم نقطة طليقة باستخدام شكلين من ترميز القيمة:

- ترميز عشري مع نقطة في تتابُع عدد، مثل، 1,23 (الذي يمثل 123 × 10^{-2}) أو 2,783 (أي 2783×10^{-3}) أو –123,456789 (الذي يمثل $-123\ 456\ 789 \times 10^{-6}$)؛
- أو بعددين يفصلهما E حيث العدد الأول يحدد mantissa ويحدد الثاني الأس، مثلاً، 12.3E4 (الذي يمثل 123×10^{3}) أو –12,3E-4 (الذي يمثل -123×10^{-5}).

ملاحظة – على عكس التعريف العام لقيم طليقة، يسمح mantissa لترميز قيمة TTCN-3، بجانب الأعداد الصحيحة، بأعداد عشرية كذلك.

ج) **boolean**: نمط يتألف من قيمتين مميزتين.

قيم نمط بولاني يدل عليها **true** و**false**.

د) فارغ.

هـ) **verdicttype**: نمط للاستخدام مع أحكام اختبار تتألف من 5 قيم مميزة. ويدل على قيم **verdicttype** بواسطة **pass** و**fail** و**inconc** و**none** و**error**.

1.1.6 أنماط وقيم سلسلة أساسية

تدعم TTCN-3 أنماط السلسلة الأساسية التالية:

الملاحظة 1 – يشير مصطلح سلسلة أو نمط سلسلة في TTCN-3 إلى **bitstring** و**hexstring** و**octetstring** و**charstring** و**universal charstring**.

أ) **bitstring**: نمط تكون قيمه المميزة مرتبة في تتابع صفر أو واحد أو بتات أكثر.

تكون دلالة قيم نمط **bitstring** عدداً اعتباطياً (من الممكن صفر) لأرقام بتة: 1 0 يسبقه علامة اقتباس وحيدة (') ويتبعه زوج من سمة 'B.

المثـــال 1: `'01101'B`

ب) **hexstring**: نمط تكون قيمه المميزة تتابعات مرتبة لصفر أو واحد أو أرقام ستة عشر، يتوافق كل منها مع تتابع مرتب لأربع بتات.

تكون دلالة قيم نمط **hexstring** عدداً اعتباطياً (من الممكن صفر) لأرقام ستة عشرية (يمكن استخدام الحروف الصغيرة والكبيرة بالتساوي كأرقام ستة):

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

تسبقها علامة اقتباس وحيدة (') ويتبعه زوج من سمة 'H؛ ويستخدم كل رقم ستة عشر ليدل على قيمة شبه أثمون باستخدام تمثيل ستة عشري.

المثـــال 2:
```
'AB01D'H
'ab01d'H
'Ab01D'H
```

ج) **octetstring**: نمط تكون قيمه المميزة تتابعات مرتبة لصفر أو عدد سالب موجب لأرقام ستة عشر (يتوافق كل زوج من الأرقام مع تتابع مرتب لثماني بتات).

تكون دلالة قيم نمط **octetstring** عدداً اعتباطياً، ولكن حتى، لعدد (من الممكن صفر) لأرقام ستة عشرية (يمكن استخدام الحروف الصغيرة والكبيرة بالتساوي كأرقام ستة):

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

تسبقها علامة اقتباس وحيدة (') ويتبعه زوج من سمة 'o؛ ويستخدم كل رقم ستة عشري ليدل على قيمة شبه أثمون باستخدام تمثيل ستة عشري.

المثـــال 3:
```
'FF96'O
'ff96'O
'Ff96'O
```

د) **charstring**: أنماط تكون قيمها المميزة صفر أو واحد أو سمات أكثر لصيغة [9] ITU-T T.50 تمتثل لصيغة مرجعية دولية (IRV) كما ورد في البند 2.8 من التوصية [9] ITU-T T.50.

الملاحظة 2 – إن صيغة IRV للألفبائية المرجعية الدولية (الألفبائية المرجعية IA5 – No.5 السابقة) الواردة في التوصية [9] ITU-T T.50 تكافئ صيغة IRV لـ ISO/IEC 646.

يدل على نمط قيم **charstring** عدد اعتباطي (من الممكن صفر) لسمات من مجموعة سمات ذات علاقة، تسبقها وتتبعها علامة اقتباس مزدوجة (") أو تحسب باستخدام وظيفة تحويل معرفة مسبقاً int2char مع قيمة صحيح موجبة لتشفيرها كمتغير (انظر 1.C).

الملاحظة 3 – تتمكن وظيفة تحويل معرفة مسبقاً من إعادة قيم طول سمة وحيدة فقط.

في الحالات حيث من الضروري تعريف سلاسل تحتوي على علامة اقتباس مزدوجة (") تمثل السمة بواسطة زوج من علامة اقتباس في نفس السطر دون وجود مسافات.

المثـــال 4: `""abcd""` represents the literal string `"abcd"`

هـ) يدل نمط سلسلة سمة يسبقه كلمة مفتاحية **universal** على أنماط قيم مميزة هي صفر أو واحد أو سمات أكثر من ISO/IEC 10646 [10].

يمكن الدلالة على قيم **universal charstring** بعدد اعتباطي (من الممكن صفر) لسمات من مجموعة سمات ذات علاقة، يسبقها ويتبعها علامة اقتباس مزدوجة (")، تحسب باستخدام وظيفة تحويل معرفة مسبقاً (انظر 3.C) مع قيمة صحيح موجبة لتشفيرها كمتغير أو بواسطة "رباعي".

الملاحظة 4: تتمكن وظيفة تحويل معرفة مسبقاً من إعادة قيم طول سمة وحيدة فقط.

في الحالات حيث من الضروري تعريف سلاسل تحتوي على علامة اقتباس مزدوجة (") تمثل السمة بواسطة زوج من علامة اقتباس في نفس السطر دون وجود مسافات.

يتمكن "الرباعي" من الدلالة على سمة وحيدة والدلالة على السمة بواسطة قيم عشرية لزمرتها ومستواها والصف والخلية طبقاً لـ [10] ISO/IEC 10646 تسبقها الكلمة المفتاحية **char** الواردة في زوج الأقواس وتفصلها فصلات (مثال، **char** (0, 0, 1, 113) لتدل على السمة المجرية "ű"). وفي الحالات حيث من الضروري أن تدل السمة على علامة اقتباس (") مزدوجة في سلسلة مخصصة طبقا لأول طريقة (في علامات اقتباس)، تمثل السمة بواسطة زوج من علامة اقتباس في نفس السطر دون مسافات. ويمكن خلط الطريقتين في ترميز وحيد لقيمة سلسلة باستخدام مشغل التسلسل.

المثـــال 5: التخصيص: تمثل"سمة بريل" و(char (0, 0, 40, 48 سلسلة حرفية: سمة بريل ⣿ التي تشبه هذا.

الملاحظة 5 – يمكن الدلالة على سمات تحكم باستخدام وظيفة تحويل معرفة مسبقا أو شكل رباعي.

بالتغيب **universal charstring**، يتوافق مع تمثيل مشفر UCS-4 من المحدد في [10] ISO/IEC 10646.

الملاحظة 6 – إن UCS-4 هو نسق تشفير يمثل أي سمة UCS على مجال طوال ثابت طوله 32 بتة.

يمكن تحييد التشفير بالتغيب هذا باستخدام نعوت متغير معرفة (انظر 3.2.28). وتعرف أنماط سلسلة السمة المفيدة التالية utf8string وbmpstring وutf16string وiso8859string باستخدام هذه النعوت في الملحق E.

2.1.6 نفاذ عناصر فردية لسلسلة

يمكن نفاذ عناصر فردية لنمط سلسلة باستخدام قواعد تركيب مثل صفيف. ويمكن فقط نفاذ عناصر وحيدة لسلسة.

وترد في الجدول 4 وحدات طول عناصر نمط سلسلة مختلفة.

تبدأ الفهرسة بقيمة صفر (0).

مثـــال:

```
// Given
MyBitString := '11110111'B;
// Then doing
MyBitString[4] := '1'B;
// Results in the bitstring '11111111'B
```

2.6 تنميط فرعي لأنماط أساسية

0.2.6 عـــام

يمكن الدلالة على أنماط معرّفة لمستعمِل بواسطة الكلمة المفتاحية **type**. ومع أنماط معرفة لمستعمل من الممكن خلق أنماط فرعية (مثل قوائم وأمدية وتقييدات طول) على أنماط أساسية وأنماط مبنية و**anytype** طبقاً للجدول 3.

1.2.6 قوائم القيم

يسمح TTCN-3 بمواصفة لقائمة قيم مميزة لأنماط أساسية وأنماط مبنية و**anytype** كما ورد في الجدول 3. وتكون القيم في القائمة نمط جذر وتكون مجموعة فرعية حقيقية لقيم عرفها نمط الجذر. ويقيد النمط الفرعي الذي تعرفه هذه القائمة القيم المسموح بها لنمط فرعي لتلك القيم في القائمة.

مثـــال:

```
type bitstring MyListOfBitStrings ('01'B, '10'B, '11'B);
type float pi (3.1415926);
type charstring MyStringList ("abcd", "rgy", "xyz");
type universal charstring SpecialLetters (char(0, 0, 1, 111), char(0, 0, 1,   112), char(0, 0,
1, 113));
```

2.2.6 الأمدية

0.2.2.6 عـــام

يسمح TTCN-3 بمواصفة تقييدات مدى لأنماط **float integer** و**charstring** و**universal charstring** و**float** (أو مشتقات لهذه الأنماط). وبالنسبة لـ **integer** و**float**، يقيد النمط الفرعي المعرف بواسطة مدى القيم المسموح بها للنمط الفرعي لقيم في المدى الذي يشمل الحدود الدنيا والحدود العليا. وفي حالة أنماط **charstring** و**universal charstring**، يقيد المدى القيم المسموح بها لكل سمة منفصلة في السلاسل. وتقيم الحدود صلاحية مواضع السمات طبقاً لجدول (جداول) مجموعة السمات المشفرة للنمط (مثلاً، لا يكون موضع معين فارغاً). والمواضع الفارغة بين الحدود الدنيا والعليا لا تعتبر قيم صالحة لمدى محدد.

المثـــال 1:

```
type integer MyIntegerRange (0 .. 255);
type float piRange (3.14 .. 3142E-3);
```

المثـــال 2:

```
type charstring MyCharString ("a" .. "z");
// Defines a string type of any length with each character within the   specified range
type universal charstring MyUCharString1 ("a" .. "z");
// Defines a string type of any length with each character within the range   from a to z
// (character codes from 97 to 122), like "abxyz";
// strings containing any other character (including control characters),     like
// "abc2" are disallowed.
type universal charstring MyUCharString2 (char(0, 0, 1, 111) .. char(0, 0,    1, 113));
// Defines a string type of any length with each character within the range   specified using
// the quadruple notation
```

1.2.2.6 أمدية لا نهائية

لتحديد صحيح لا نهائي أو مدى طليق، يمكن استخدام الكلمة المفتاحية **infinity** بدلاً من قيمة تدل على عدم وجود حدود دنيا أو عليا. وتكون الحدود العليا أكبر من أو مماثلة للحدود الدنيا.

مثـــال:

```
type integer MyIntegerRange (-infinity .. -1); // All negative integer numbers
```

ملاحظة – تعتمد "القيمة" اللانهائية على التنفيذ. وقد يؤدي استخدام هذه الخاصية إلى مشاكل في التحميل.

2.2.2.6 خلط القوائم والأمدية

فارغ.

ملاحظة – يحل هذا القسم محل القسم 5.2.6.

3.2.6 التقييدات على طول سلسلة

يسمح TTCN-3 بمواصفة تقييدات الطول على أنماط سلسلة. وتقوم حدود الطول على أساس وحدات مختلفة تعتمد على نمط السلسلة التي تستخدمها. وفي جميع الحالات، تقيم هذه الحدود قيم **integer** غير سالبة (أو قيم مشتقة من **integer**).

مثـــال:

```
type bitstring MyByte length(8);              // Exactly length 8
type bitstring MyByte length(8 .. 8);      // Exactly    length 8
type bitstring MyNibbleToByte length(4 .. 8);   //Minimum length 4,  maximum length 8
```

ويحدد الجدول 4 وحدات الطول لأنماط سلسلة مختلفة.

الجدول Z.140/4 – وحدات الطول المستخدمة في مجال مواصفات الطول

النمط	وحدات الطول
bitstring	بتات
hexstring	أرقام ستة عشري
octetstring	أثمونات
character strings	سمات

بالنسبة للحد الأعلى، يمكن استخدام الكلمة المفتاحية **infinity** لتدل على عدم وجود حد للطول. ويكون الحد الأعلى أكبر من، أو مساوٍ للحد الأدنى.

4.2.6 تنميط فرعي لتخطيط أنماط سلسلة سمات

يسمح TTCN-3 باستخدام أنماط السمات المحددة في 5.1.B لتقييد قيم مسموح بها لأنماط **charstring** و**universal charstring**. ويستخدم نمط التقييد الكلمة المفتاحية **pattern** التي يتبعها تخطيط سمة. وتكون جميع القيم التي يدل عليها التخطيط مجموعة فرعية حقيقية لنمط يجرى تنميطه فرعياً.

ملاحظة – يمكن النظر إلى التنميط الفرعي لتخطيط على أنه شكل خاص لقائمة تقييدات، حيث أعضاء القائمة غير معرفين بواسطة تحديد سلاسل سمات محددة ولكن عبر آلية توليد عناصر القائمة.

مثـــال:

```
type charstring MyString (pattern "abc*xyz");
  // all permitted values of MyString have prefix abc and postfix xyz

type universal charstring MyUString (pattern "*\r\n")
  // all permitted values of MyUString are terminated by CR/LF

type charstring MyString2 (pattern "abc?\q{0,0,1,113}");
  // causes an error because the character denoted by the quadruple  {0,0,1,113} is not a
  // legal character of the TTCN-3 charstring type

type MyString MyString3 (pattern "d*xyz");
  // causes an error because the type MyString does not contain a value starting with the
  // character d
```

5.2.6 خلط آليات تنميط فرعي

1.5.2.6 خلط تخطيطات وقوائم وأمدية

في **integer** و**float** (أو مشتقات من هذه الأنماط) يسمح لتعاريف نمط فرعي خلط قوائم وأمدية. ولا يكون تراكب التقييدات المختلفة خطأ.

المثـــال 1:

```
type integer MyIntegerRange (1, 2, 3, 10 .. 20, 99, 100);
```

وفي **charstring** و**universal charstring** لا يسمح لتعاريف نمط فرعي خلط تقييدات تخطيط أو قائمة أو مدى.

المثـــال 2:

```
type charstring MyCharStr0 ('gr', 'xyz');
  // contains character strings gr and xyz;

type charstring MyCharStr1 ('a'..'z');
  // contains character strings of arbitrary length containing characters a   to z.

type charstring MyCharStr2 (pattern '[a-z]#(3,9)');
// contains character strings of length form 3 to 9 characters containing     characters a to
z
```

2.5.2.6 استخدام تقييد الطول مع تقييدات أخرى

في **bitstring** و**hexstring** و**octetstring** يمكن خلط تعاريف نمط فرعي لتقييدات قوائم وطول في نفس تعريف النمط الفرعي.

في **charstring** و**universal charstring** يسمح لتعاريف نمط فرعي بإضافة تقييد طول إلى تقييدات تحتوي على قائمة أو مدى أو نمط فرعي لتخطيط في نفس تعريف النمط الفرعي.

وعند الخلط مع تقييدات أخرى، يكون تقييد الطول آخر عنصر لتعريف نمط فرعي. ويفعل قيد الطول بالاشتراك مع آليات تنميط فرعي أخرى (أي تتألف مجموعة القيم من مجموعة مشتركة من مجموعات القيم عرّفتها القائمة أو المدى أو تنميط فرعي لتخطيط وقيد الطول).

مثـــال:

```
type charstring MyCharStr5 ('gr', 'xyz') length (1..9);
  // contains the character strings gr and xyz;

type charstring MyCharStr6 ( 'a'..'z') length (3..9);
  // contains character strings of length from 3 to 9 characters and containing characters
  // a to z

type charstring MyCharStr7 (pattern '[a-z]#(3,9)') length (1..9);
  // contains character strings of length form 3 to 9 characters containing   characters a to
z

type charstring MyCharStr8 (pattern '[a-z]#(3,9)') length (1..8);
  // contains character strings of length form 3 to 8 characters containing   characters a to
z

type charstring MyCharStr9 (pattern '[a-z]#(1,8)') length (1..9);
  // contains any character strings of length form 1 to 8 characters containing    characters
  // a to z

type charstring MyCharStr10 ('gr', 'xyz') length (4);
  // contains no value (empty type).
```

3.6 أنماط وقيم مبنية

0.3.6 عـــام

تستخدم أيضاً الكلمة المفتاحية **type** لتحديد أنماط محددة مبنية مثل أنماط **record** وأنماط **record of** وأنماط **set** وأنماط **set of** وأنماط **enumerated** وأنماط **union**.

ويمكن الحصول على قيم هذه الأنماط باستخدام ترميز تخصيص واضح أو ترميز مختزل لقائمة قيم.

المثـــال 1:

```
const MyRecordType MyRecordValue:=                                  //assignment notation
{
  field1 := '11001'B,
  field2 := true,
  field3 := "A string"
}

// Or
const MyRecordType MyRecordValue:= {'11001'B, true, "A string"}  //value list notation
```

وعند تحديد قيم جزئية (أي، ضبط القيمة لمجموعة فرعية فقط لمجالات متغير مبني) باستخدام تمييز تخصيص فقط، يجب أن تحدد المجالات التي تخصص للقيم. وتترك المجالات التي لا تذكر ضمنياً. ومن الممكن أيضاً ترك مجالات دون تحديد واضح باستخدام الرمز "-". وباستخدام قائمة القيم، يكون ترميز جميع المجالات في البنية إما مع قيمة أو الرمز "-" غير المستخدم أو الكلمة المفتاحية **omit**.

المثـــال 2:

```
var MyRecordType MyVariable:=                                  //assignment notation
{
  field1 := '11001'B,
  // field2 implicitly unspecified
  field3 := "A string"
}
// Or
var MyRecordType MyVariable:=                                  //assignment notation
{
  field1 := '11001'B,
  field2 := -, // field2 explicitly unspecified
```

```
  field3 := "A string"
}
// Or
var MyRecordType MyVariable:= {'11001'B, -, "A string"}  //value list notation
```

لا يُسمح بخلط ترميزين لقيمتين في نفس السياق (مباشرة).

المثـــال 3:

```
// This is disallowed
const MyRecordType MyRecordValue:= {MyIntegerValue, field2 := true, "A string"}
```

في كل من ترميز التخصيص وترميز قائمة قيم، تحذف المجالات الخيارية باستخدام قيمة **omit** واضحة للمجال ذي العلاقة. ولا تستخدم الكلمة المفتاحية **omit** للمجالات الإلزامية. وعند إعادة تخصيص قيمة مدمثة سابقة، يسبب استخدام الرمز غير المستخدم، أو تخطي مجال في ترميز تخصيص، أن تظل المجالات ذات العلاقة دون تغيير.

المثـــال 4:

```
var MyRecordType MyVariable :=
{
  field1 := '111'B,
  field2 := false,
  field3 := -
}

MyVariable := { '10111'B, -, - };
// after this, MyVariable contains { '10111'B, false /* unchanged */,   <undefined> }

MyVariable :=
{
  field2 := true
}
// after this, MyVariable contains { '10111'B, true, <undefined> }

MyVariable :=
{
  field1 := -,
  field2 := false,
  field3 := -
}
// after this, MyVariable contains { '10111'B, false, <undefined> }
```

1.3.6 أنماط وقيم سجل

0.1.3.6 عـــام

يدعم TTCN-3 أنماطاً مرتبة مبنية معروفة على أنها **record**. ويمكن أن تكون عناصر نمط **record** أي أنماط أساسية أو أنماط معطيات معرفة لمستعمل (مثل السجلات أو المجموعات أو المصفوفات الأخرى). وتكون قيم **record** متوائمة مع أنماط مجالات **record**. وتكون معرفات العنصر محلية لـ **record** وتكون وحيدة في **record** (ولكن لا يتعين أن تكون وحيدة بشكل عام). ولا يحتوي الثابت الذي هو نمط **record** على متغيرات أو معلمات وحدة كقيم مجال، سواء مباشرة أو غير مباشرة.

المثـــال 1:

```
type record MyRecordType
{
  integer      field1,
  MyOtherRecordType    field2 optional,
  charstring       field3
}

type record MyOtherRecordType
{
  bitstring    field1,
  boolean   field2
 }
```

يمكن تعريف السجلات دون مجالات (أي، كسجلات فارغة).

المثـــال 2:

```
type record MyEmptyRecord {}
```

تُخصّص قيمة **record** على أساس عنصر فردي. ويكون ترتيب قيم مجال في ترميز قائمة قيم هو نفس ترتيب المجالات في تعريف النمط ذي العلاقة.

المثـــال 3:

```
var integer MyIntegerValue := 1;

const MyOtherRecordType MyOtherRecordValue:=
{
  field1 := '11001'B,
  field2 := true
}

var MyRecordType MyRecordValue :=
{
  field1 := MyIntegerValue,
  field2 := MyOtherRecordValue,
  field3 := "A string"
}
```

نفس القيمة المحددة لقائمة قيم.

المثـــال 4:

```
MyRecordValue:= {MyIntegerValue, {'11001'B, true}, "A string"};
```

1.1.3.6 مجالات مرجعية لنمط سجل

تحدد عناصر **record** بواسطة ترميز النقط *TypeOrValueId.ElementId*، حيث يقوم *TypeOrValueId* باستبيان اسم النمط أو المتغير المبني. ويقوم ElementId باستبيان اسم مجال في نمط مبني.

مثـــال:

```
MyVar1 := MyRecord1.myElement1;
// If a record is nested within another type then the reference may look like this
MyVar2 := MyRecord1.myElement1.myElement2;
```

2.1.3.6 عناصر خيارية في سجل

تحدد العناصر الخيارية في **record** باستخدام الكلمة المفتاحية **optional**.

المثـــال 1:

```
type record MyMessageType
{
  FieldType1   field1,
  FieldType2   field2  optional,
   :
  FieldTypeN   fieldN
}
```

تحذف المجالات الخيارية باستخدام رمز حذف.

المثـــال 2:

```
MyRecordValue:= {MyIntegerValue, omit , "A string"};

// Note that this is not the same as writing,
// MyRecordValue:= {MyIntegerValue, -, "A string"};
// which would mean the value of field2 is unchanged
```

3.1.3.6 تعاريف أنماط متداخلة لأنماط مجالات

يدعم TTCN-3 تعريف أنماط لمجالات سجل متداخلة في تعريف **record**. ويكون كل من تعريف أنماط مبنية جديدة (**record** و**set** و**enumerated** و**set of** و**record of**) ومواصفة تقييدات نمطاً فرعياً ممكناً.

مثـــال:

```
// record type with nested structured type definitions
type record MyNestedRecordType
{
  record
  {
      integer nestedField1,
      float   nestedField2
} outerField1,
enumerated {
      nestedEnum1,
      nestedEnum2
  } outerField2,
  record of boolean outerField3
}

// record type with nested sub-type definitions
```

```
type record MyRecordTypeWithSubtypedFields
{
  integer    field1 (1 .. 100),
  charstring field2 length ( 2 .. 255 )
}
```

2.3.6 مجموعة أنماط وقيم

0.2.3.6 عـــام

يدعم TTCN-3 الأنماط المبنية غير المرتبة المعروفة بـ **set**. وتكون مجموعة أنماط وقيم مماثلة لسجلات باستثناء أن ترتيب مجالات **set** غير مهم.

مثـــال:

```
type set MySetType
{
  integer     field1,
  charstring     field2
}
```

تكون معرفات المجال محلية للمجموعة وتكون وحيدة في المجموعة (ولكن لا يتعين أن تكون وحيدة بشكل عام).

لا يستخدم ترميز قائمة قيم لضبط قيم لقيم أنماط **set**.

1.2.3.6 مجالات مرجعية لمجموعة أنماط

تحدد عناصر **set** بواسطة ترميز بالنقط (انظر 1.1.3.6).

مثـــال:

```
MyVar3 := MySet1.myElement1;
// If a set is nested in another type then the reference may look like this
MyVar4 := MyRecord1.myElement1.myElement2;
// Note, that the set type, of which the field with the identifier 'myElement2' is
referenced,
// is embedded in a record type
```

2.2.3.6 العناصر الخيارية في مجموعة

تحدد العناصر الخيارية في **set** باستخدام الكلمة المفتاحية **optional**.

3.2.3.6 تعريف نمط متداخل لأنماط مجالات

يدعم TTCN-3 تعريف أنماط لمجالات مجموعة متداخلة في تعريف **set**، مماثلة لآلية أنماط سجل الواردة في 3.1.3.6.

3.3.6 سجلات ومجموعات أنماط وحيدة

0.3.3.6 عـــام

يدعم TTCN-3 مواصفة السجلات والمجموعات التي تكون لعناصرها جميعاً نفس النمط. ويدل على هذه استخدام الكلمة المفتاحية **of**. ولا يوجد لهذه السجلات والمجموعات معرفات عنصر ويمكن اعتبارها مماثلة لصفيف مرتب وصفيف غير مرتب على التوالي.

تستخدم الكلمة المفتاحية **length** لتقييد أطوال **record of** و**set of**.

المثـــال 1:

```
type record length(10) of integer MyRecordOfType; // is a record of exactly 10 integers

type record length(0..10) of integer MyRecordOfType; // is a record of a maximum of 10 integers

type record length(10..infinity) of integer MyRecordOfType; // record of at least 10 integers

type set of boolean MySetOfType; // is an unlimited set of boolean values

type record length(0..10) of charstring StringArray length(12);
// is a record of a maximum of 10 strings each with exactly 12 characters
```

يكون ترميز القيمة لـ **record of** و**set of** ترميز قائمة قيمة أو ترميز مفهرس لعنصر فردي (نفس ترميز قيمة لمصفوفات، انظر 5.6). وهناك استثناء واحد من القاعدة العامة هذه: في حالة تعريف مقاسات معدلة، عندما يكون ترميز تخصيص مسموح أيضاً باستخدامه (انظر 0.6.14).

عندما يستخدم ترميز قائمة قيم، تخصص القيمة الأولى في القائمة للعنصر الأول، وتخصص قيمة الثانية للقائمة للعنصر الثاني وما إلى ذلك. ولا يسمح بتخصيص فارغ (مثلاً، فصلتان، تتبع الثانية الأولى مباشرة أو مسافة بيضاء بينهما). والعناصر التي تترك من التخصيص يجري تخطيها أو حذفها من القائمة بوضوح.

يمكن استخدام ترميزات قيم مفهرسة على كل من الجانب الأيمن والجانب الأيسر من التخصيص. ويكون دليل العنصر الأول صفراً ولا تتجاوز قيمة الدليل الحد المفروض على التنميط الفرعي للطول. وإذا كانت قيمة العنصر التي يدل عليها الدليل على الجانب الأيمن من التخصيص غير معرّفة، يسبب هذا خطأ لعلم الدلالات أو التنفيذ. وإذا أشار مشغل الدليل على الجانب الأيسر من التخصيص إلى عنصر غير موجود، تخصص القيمة على الجانب الأيمن لعنصر، وتخلق جميع العناصر مع دليل أصغر من الدليل الفعلي ودون قيمة مخصصة مع قيمة غير معرفة. ويسمح بعناصر غير معرفة في حالات عابرة فقط (بينما تظل القيمة غير مرئية). ويسبب إرسال قيمة **record of** مع عناصر غير معرفة خطأ اختبار مجرد دينامي.

المثـــال 2:

```
// Given
type record of integer MyRecordOf;
var integer MyVar;
var MyRecordOf MyRecordVar := { 0, 1, 2, 3, 4 };

MyVar := MyRecordVar[0]; // the first element of the "record of" value (integer 0)
                         // is assigned to MyVar

// Indexed values are permitted on the left-hand side of assignments as well:
MyRecordVar[1] := MyVar; // MyVar is assigned to the second element
                         // value of MyRecordVar is  { 0, 0, 2, 3, 4 }

// The assignment
MyRecordVar := { 0, 1, -, 2, omit };
// will change the value of MyRecordVar to{ 0, 1, 2 <unchanged>, 2};
// Note, that the 3rd element would be undefined if it had no previous assigned value.

// The assignment
MyRecordVar[6] := 6;

// will change the value of MyRecordVar to{ 0, 1, 2 , 2, <undefined>,   <undefined>, 6 };
   // Note the 5th and 6th elements (with indexs 4 and 5) had no assigned value before this
// last assignment and are therefore undefined.
```

ملاحظة – يجعل هذا من الممكن نسخ قيم **record of** عنصراً بعنصر في عروة. فمثلاً، تعكس الوظيفة أدناه عناصر قيمة **record of**:

```
function reverse(in MyRecord src) return MyRecord
  {
  var MyRecord dest;
  var integer I;
  for(I := 0; I < sizeof(src); I:= I + 1) {
      dest[sizeof(src) - 1 - I] := src[I];
  }
  return dest;
  }
```

ينتج عن عناصر **record of** و**set of** المدمجة بنية معطيات مماثلة لمصفوفات متعددة الأبعاد (انظر 5.6).

المثـــال 3:

```
// Given
      type record of integer MyBasicRecordOfType;
      type record of MyBasicRecordOfType  MyRecordOfType;

      // Then, the variable myRecordOfArray will have similar attributes to a two-dimensional array:
      var MyRecordOfType myRecordOfArray;
      // and reference to a particular element would look like this
      // (value of the second element of the third 'MyBasicRecordOfType' construct)
      myRecordOfArray [2][1] := 1;
```

1.3.3.6 تعاريف أنماط متداخلة

يدعم TTCN-3 تعريف نمط مجمع متداخل مع تعريف **record of** أو **set of**. ويكون تعريفاً لأنماط مبنية جديدة (**record** و**set** و**enumerated** و**record of** و**set of**) ومواصفة تقييدات نمط فرعي ممكنة.

مثـــال:

```
type record of enumerated { red, green, blue } ColorList;
type record length (10) of record length (10) of integer Matrix;
type set of record { charstring id, charstring val } GenericParameters;
```

4.3.6 أنماط وقيم معددة

يدعم TTCN-3 أنماط **enumerated**. وتستخدم الأنماط المعددة لوضع نماذج لأنماط تأخذ فقط مجموعة من قيم مسماة مميزة. وتسمى مثل هذه القيم المميزة تعديدات. ويكون لكل تعديد مُعرف. وتستخدم العمليات على أنماط معددة فقط لهذه المعرفات وتقتصر على التخصيص والتكافؤ والمشغلين المنظمين. وتكون معرفات التعديد وحيدة في نمط معدد (ولكن ليست وحيدة بشكل عام) وبالتالي تكون مرئية في سياق نمط ما فقط. ويعاد استخدام معرفات التعديد فقط في تعاريف أخرى لنمط مبني ولا تستخدم لمعرفات رؤية محلية أو عامة عند نفس المستوى أو مستوى أدنى لنفس فرع تراتبية المنظور (انظر تراتبية المنظور في 0.3.5).

المثـــال 1:

```
type enumerated MyFirstEnumType {
  Monday, Tuesday, Wednesday, Thursday, Friday
};

type integer Monday;
// This definition is illegal, as the name of the type has local or global visibility

type enumerated MySecondEnumType {
  Saturday, Sunday, Monday
};
// This definition is legal as it reuses the Monday enumeration identifier within
// a different enumerated type

type record MyRecordType {
  integer     Monday
};
// This definition is legal as it reuses the Monday enumeration identifier within
// a distinct structured type as identifier of a given field of this type

type record MyNewRecordType {
  MyFirstEnumType firstField,
  integer     secondField
};

var MyNewRecordType newRecordValue := { Monday, 0 }
// MyFirstEnumType is implicitly referenced via the firstField element of MyNewRecordType

const integer Monday := 7
// This definition is illegal as it reuses the Monday enumeration identifier for a
  // different TTCN-3 object within the same scope unit
```

يمكن أن يكون لكل تعديد خياريٌّ قيمة صحيحة مخصصة، تعرّف بعد اسم التعديد في قوسين. ويكون كل عدد صحيح مخصص مميزاً في نمط وحيد **enumerated**. وبالنسبة لكل تعديد دون قيمة صحيح مخصص، يرتبط النظام التتابعي لعدد صحيح في ترتيب نصي لتعديدات، ابتداء من الجانب الأيسر، وبداية بصفر، حسب الخطوة 1 وتخطي أي عدد مشغول في أي تعديدات مع قيمة مخصصة يدوياً. وتستخدم هذه القيم فقط للسماح باستخدام مشغلين ترابطيين.

الملاحظة 1 – يمكن استخدام قيمة الصحيح أيضاً بواسطة النظام لتشفير/فك تشفير قيم معددة. ومع ذلك، يكون هذا خارج مدى هذه التوصية (باستثناء أن TTCN-3 يسمح بتصاحب نعوت تشفير لبنود TTCN-3).

وبالنسبة لأي استطباق أو مرجع قيمة لنمط **enumerated**، يكون نمط ما مرجعياً ضمنياً أو صراحة.

الملاحظة 2 – إذا كان نمط معدد هو عنصر لنمط مبني معرف لمستعمل، يكون النمط المعدد مرجعياً ضمنياً عبر عنصر ما (أي، بواسطة معرف العنصر أو موضع القيمة في ترميز قائمة قيم) عند تخصيص قيمة واستطباق وما إلى ذلك.

المثـــال 2:

```
   // Valid instantiations of MyFirstEnumType and MySecondEnumType would be
var MyFirstEnumType Today := Tuesday;
var MySecondEnumType Tomorrow := Monday;

// But the following statement is illegal because the two enumeration types are not compatible
Today := Tomorrow
```

5.3.6 اتحادات

0.5.3.6 عـــام

يدعم TTCN-3 نمط **union**. إن نمط **union** هو تجميع مجالات، يُعرَف كل واحد بواسطة مُعرَف. ويكون مجالاً واحداً من مجالات محددة غير موجود في قيمة اتحاد فعلي. وأنماط الاتحاد مفيدة لنموذج بنية يمكنها أن تأخذ عدداً نهائياً واحداً لأنماط معروفة.

مثـــال:

```
type union MyUnionType
{
  integer  number,
  charstring     string
};

// A valid instantiation of MyUnionType would be
var MyUnionType age, oneYearOlder;
var integer ageInMonths;

age.number := 34;      // value notation by referencing the field. Note, that this
                       // notation makes the given field to be the chosen one
oneYearOlder := {number := age.number+1};

ageInMonths := age.number * 12;
```

إن ترميز قائمة قيم لضبط قيم لا تستخدم لقيم أنماط **union**.

1.5.3.6 مجالات مرجعية لنمط اتحاد

تكون مجالات نمط **union** مرجعية بواسطة ترميز بالنقط (انظر 1.1.3.6).

مثـــال:

```
MyVar5 := MyUnion1.myChoice1;
// If a union type is nested in another type then the reference may look like this
MyVar6 := MyRecord1.myElement1.myChoice2;
// Note, that the union type, of which the field with the identifier 'myChoice2' is referenced,
// is embedded in a record type
```

2.5.3.6 قابلية الاختيار والاتحاد

لا يسمح بالمجالات الخيارية لنمط **union**، الذي يعني أن الكلمة المفتاحية **optional** لا تستخدم مع أنماط **union**.

3.5.3.6 تعريف نمط متداخل لأنماط مجالات

يدعم TTCN-3 تعريف أنماط مجالات اتحاد متداخلة في تعريف اتحاد، مماثلة لآلية أنماط التسجيل الواردة في 3.1.3.6.

4.6 النمط anytype

يعرف النمط الخاص **anytype** كاختزال لاتحاد جميع أنماط المعطيات المعروفة ونمط عنوان في وحدة TTCN-3. ويرد تعريف مصطلح "أنماط معروفة" في 1.3، أي، يتألف **anytype** من جميع أنماط المعطيات المعروفة ولكن ليس أنماط **component** و**port** و**default**. ويتضمن نمط **address** إذا كان معرفاً صراحة في تلك الوحدة.

تكون أسماء المجالات لـ **anytype** وحيدة التعريف بواسطة أسماء أنماط متوافقة.

الملاحظة 1 – نتيجة لهذا المتطلب، لا يمكن الوصول إلى أنماط مستوردة مع أسماء متناقضة (سواء مع معرف لتعريف في وحدة مستوردة أو مع معرف مستورد من وحدة ثالثة) عبر **anytype** لوحدة مستوردة.

مثـــال:

```
// A valid usage of anytype would be
var anytype MyVarOne, MyVarTwo;
var integer MyVarThree;

MyVarOne.integer := 34;
MyVarTwo := {integer := MyVarOne.integer + 1};

MyVarThree := MyVarOne.integer * 12;
```

يعرف **anytype** محلياً لكل وحدة (مثل أنماط معرفة مسبقاً) ولا يمكن استيرادها بواسطة وحدة أخرى. ومع ذلك، يمكن استيراد نمط معرف لمستعمل لنمط **anytype** بواسطة وحدة أخرى. وتأثير هذا هو أن جميع أنماط تلك الوحدة مستوردة.

الملاحظة 2 – "يحتوي" نمط معرف لمستعمل **anytype** على جميع الأنماط المستوردة في الوحدة حيث يجري الإعلان عنها. ويمكن أن يسبب استيراد نمط معرف لمستعمل كهذا في وحدة تأثيرات جانبية ومن ثم ينبغي الحذر في مثل هذه الحالات.

5.6 المصفوفات

من الشائع في كثير من لغات البرمجة، لا تعتبر المصفوفات على أنها أنماط في TTCN-3. وبدلاً من ذلك، يمكن تحديدها عند نقطة إعلان متغير. ويمكن الإعلان عن مصفوفات كبُعد وحيد أو متعدد. وتحدد أبعاد صفيف باستخدام تعبيرات ثابتة، تقيم قيم **integer** موجبة.

المثـــال 1:

```
var integer MyArray1[3];    // Instantiates an integer array of 3 elements with the index 0 to 2
var integer MyArray2[2][3]; // Instantiates a two-dimensional integer array of 2 × 3 elements with
                            // indexes from (0,0) to  (1,2)
```

تنفذ عناصر صفيف بواسطة ترميز دليل ([ ])، الذي يجب أن يحدد دليل صالح في مدى الصفيف. ويمكن نفاذ عناصر فردية لمصفوفات ذات أبعاد متعددة بواسطة استخدام متكرر لترميز دليل. ويسبب نفاذ عناصر خارج مدى صفيف وقت تصريف أو خطأ اختبار مجرد.

المثـــال 2:

```
MyArray1[1] := 5;
MyArray2[1][2] := 12;

MyArray1[4] := 12;    // ERROR: index must be between 0 and 2
MyArray2[3][2] := 15; // ERROR: first index must be 0 or 1
```

ويمكن أيضاً تحديد أبعاد صفيف باستخدام أمدية. وفي هذه الحالات، تعرّف القيم الدنيا والعليا لمدى قيم الدليل الدنيا والعليا.

المثـــال 3:

```
var integer MyArray3[1 .. 5]; // Instantiates an integer array of 5 elements
           // with the index  1 to 5
MyArray3[1] := 10; // Lowest index
MyArray3[5] := 50; // Highest index

var integer MyArray4[1 .. 5][2 .. 3 ];  // Instantiates a two-dimensional integer array of
                                        // 5 × 2 elements with indexes from (1,2) to (5,3)
```

تكون قيم عناصر صفيف متوائمة مع إعلان متغير متوافق. ويمكن تخصيص قيم بشكل فردي بواسطة ترميز قائمة قيم أو ترميز مفهرس أو أكثر من واحد أو الجميع مرة واحدة بواسطة ترميز قائمة قيم. وعندما يستخدم ترميز قائمة قيم، تخصص القيمة الأولى للعنصر الأول للصفيف (العنصر مع الدليل 0)، والقيمة الثانية للعنصر الثاني وما إلى ذلك. والعناصر التي تترك من التخصيص يجري تخطيها أو تحذف صراحة في القائمة. ولتخصيص قيم لمصفوفات ذات أبعاد متعددة، يُستبان كل بعد مخصص لمجموعة قيم مغلقة بين أقواس معقوفة. وعند تحديد قيم مصفوفات ذات أبعاد متعددة، يتناظر البعد الأقصى المتروك مع البنية الخارجية للقيمة، والبعد الأقصى الأيمن مع البنية القصوى الداخلية. واستخدام صفيف يقطع المصفوفات ذات أبعاد متعددة، أي، عندما يكون عدد الأدلة لقيمة صفيف أقل من عدد الأبعاد في تعريف صفيف متناظر، يسمح به. وتتناظر أدلة قطع صفيف مع أبعاد تعريف صفيف من اليسار إلى اليمين (أي، يتناظر الدليل الأول للقطعة مع البعد الأول للتعريف). وتتوافق أدلة القطعة مع أبعاد تعريف صفيف متعلق.

المثـــال 4:

```
MyArray1[0]:= 10;
MyArray1[1]:= 20;
MyArray1[3]:= 30;

// or using a value list
MyArray1:= {10, 20, -, 30};

MyArray4:= {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
  // The array value is completely defined

var integer MyArray5[2][3][4] :=
{
  {
    {1, 2, 3, 4}, // assigns a value to MyArray5 slice [0][0]
    {5, 6, 7, 8}, // assigns a value to MyArray5 slice [0][1]
    {9, 10, 11, 12} // assigns a value to MyArray5 slice [0][2]
  }, // end assignments to MyArray5 slice [0]
  {
    {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}
  } // assigns a value to MyArray5 slice [1]
```

```
};

MyArray4[2] := {20, 20};
   // yields {{1, 2}, {3, 4}, {20, 20}, {7, 8}, {9, 10}};
MyArray5[1] := { {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};
   // yields {{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}},
   //         {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}};

MyArray5[0][2] := {3, 3, 3, 3};
   // yields {{{1, 2, 3, 4}, {5, 6, 7, 8}, {3, 3, 3, 3}},
   //         {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}};

var integer MyArrayInvalid[2][2];
MyArrayInvalid := { 1, 2, 3, 4 }
   // invalid as the dimension of the value notation does not corresponds to the dimensions
   // of the definition
MyArrayInvalid[2] := { 1, 2 }
      // invalid as the index of the slice should be 0 or 1
```

ملاحظة – والطريقة البديلة لاستخدام بنيات معطيات ذات أبعاد متعددة هي عبر استخدام أنماط **record** أو of **record** أو **set** أو **set of**.

المثـــال 5:

```
// Given
type record MyRecordType
{
  integer   field1,
  MyOtherStruct  field2,
  charstring     field3
}
// An array of MyRecordType could be
var MyRecordType myRecordArray[10];
// A reference to a particular element would look like this
myRecordArray[1].field1 := 1;
```

6.6 أنماط تكرارية

كلما كانت قابلة للتطبيق، يمكن أن تكون تعاريف نمط TTCN-3 تكرارية. ومع ذلك، يضمن المستعمل أن جميع أنماط التكرار قابلة للاستبانة ولا يحدث تكرار لا نهائي.

7.6 مواءمة نمط

0.7.6 عـــام

بشكل عام، يتطلب TTCN-3 مواءمة نمط لقيم عند التخصيص والاستطباق والمقارنة.

ولأغراض هذا القسم، تسمى القيمة الفعلية التي تخصص وتمرر كمعلمة وما إلى ذلك قيمة "b". ويسمى نمط القيمة "b" نمط "B". ويسمى نمط المعلمة الرسمية، الذي يحصل على القيمة الفعلية لقيمة "b" نمط "A".

1.7.6 مواءمة نمط لأنماط غير مبنية

بالنسبة لمتغيرات وثوابت ومقاسات وما إلى ذلك لأنماط أساسية بسيطة وأنماط سلسلة ثنائية وسلسلة ستة عشرية وسلسلة أثمونات، تكون قيمة "b" متوائمة مع نمط "A" إذا استبان نمط "B" نفس نمط جذر نمط "A" (مثل **integer**) ولا تخل بالتنميط الفرعي (مثل، أمدية وتقييدات طول) للنمط "A".

مثـــال:

```
// Given
type integer MyInteger(1 .. 10);
:
var integer x;
var MyInteger y;

// Then
y := 5; // is a valid assignment

x := y;
// is a valid assignment, because y has the same root type as x and no subtyping is violated

x := 20; // is a valid assignment
y := x;
```

```
// is NOT a valid assignment, because the value of x is out of the range of MyInteger

x := 5; // is a valid assignment
y := x;
// is a valid assignment, because the value of x is now within the range of MyInteger

//Given
type charstring MyChar length (1);
type charstring MySingleChar length (1);
var MyChar myCharacter;
var charstring myCharString;
var MySingleChar mySingleCharString := "B";

//Then
myCharString := mySingleCharString;
//is a valid assignment as charstring restricted to length 1 is compatible with charstring.
myCharacter := mySingleCharString;
//is a valid assignment as two single-character-length charstrings are compatible.

//Given
myCharString := "abcd";

//Then
myCharacter := myCharString[1];
//is valid as the r.h.s. notation addresses a single element from the string

//Given
var charstring myCharacterArray [5] := {"A", "B", "C", "D", "E"}

//Then
myCharString := myCharacterArray[1];
  //is valid and assigns the value "B" to myCharString;
```

بالنسبة للمتغيرات والثوابت والمقاسات وما إلى ذلك لنمط **charstring**، تكون قيمة "b" متوائمة مع نمط "A" **universal charstring** إلا إذا تخلت عن أي مواصفة تقييد نمط (مدى أو قائمة أو طول) لنمط "A".

بالنسبة للمتغيرات والثوابت والمقاسات وما إلى ذلك لنمط **universal charstring**، تكون قيمة "b" متوائمة مع نمط "A" **charstring** إذا كانت جميع السمات المستخدمة في قيمة "b" لها سماتها متناظرة (أي، نفس التحكم أو السمة البيانية المستخدمة لنفس شفرة السمة) في النمط **charstring** ولا تخل بأي مواصفة تقييد نمط (مدى أو قائمة أو طول) لنمط "A".

2.7.6 مواءمة نمط لأنماط مبنية

0.2.7.6 عـــام

في حالة أنماط مبنية (باستثناء نمط **enumerated**)، تكون قيمة "b" لنمط "B" متوائمة مع نمط "A"، إذا كانت بنيات قيمة فعلية نمط "B" ولنمط "A" متوائمة، وفي هذه الحالة يسمح بالتخصيصات والاستطباقات والمقارنات.

1.2.7.6 مواءمة نمط لأنماط معددة

لا تكون الأنماط المعددة متوائمة أبداً مع أنماط أساسية أو مبنية أخرى (أي، بالنسبة لأنماط معددة، يطلب تنميط قوي).

2.2.7.6 مواءمة نمط لأنماط record أو record of

بالنسبة لأنماط **record**، تكون بنيات قيمة فعلية متوائمة إذا: كان العدد والجانب الخياري للمجالات في الترتيب النصي للتعريف مماثلاً؛ وإن أنماط كل مجال متوائمة؛ وتكون قيمة كل مجال موجود لقيمة "b" متوائمة مع نمط مجاله المتناظر في النمط "A". وتخصص قيمة كل مجال في قيمة "b" لمجال متناظر في قيمة نمط "A".

المثـــال 1:

```
// Given
type record AType {
      integer     a(0..10)   optional,
      integer     b(0..10)   optional,
      boolean        c
}
type record BType {
      integer        a       optional,
      integer     b(0..10)   optional,
      boolean        c
}

type record CType {     // type with different field names
      integer        d  optional,
```

```
      integer        e  optional,
      boolean        f

}

type record DType {     // type with field c optional
      integer        a  optional,
      integer        b  optional,
      boolean        c  optional
}

type record EType {     // type with an extra field d
      integer        a  optional,
      integer        b  optional,
      boolean        c,
        float        d  optional
}

      var AType MyVarA := { -, 1, true};
      var BType MyVarB := { omit, 2, true};
      var CType MyVarC := { 3, omit,  true};
      var DType MyVarD := { 4, 4, true};
      var EType MyVarE := { 5, 5, true, omit};

      // Then

      MyVarA := MyVarB;  // is a valid assignment,
                         // value of MyVarA is ( a := <undefined>, b:= 2, c:= true)
      MyVarC := MyVarB;  // is a valid assignment
                         // value of MyVarC is ( d := <undefined>, e:= 2, f:= true)
      MyVarA := MyVarD;  // is NOT a valid assignment because the optionality of fields does not
                         // match
      MyVarA := MyVarE;  // is NOT a valid assignment because the number of fields does not match

      MyVarC := { d:= 20 };// actual value of MyVarC is { d:=20, e:=2,f:= true }
      MyVarA := MyVarC   // is NOT a valid assignment because field 'd' of MyVarC violates subtyping
                // of field 'a' of AType
```

بالنسبة لأنماط ومصفوفات **record of**، تكون بنيات القيمة الفعلية متوائمة إذا كانت أنماط مكوناتها متوائمة وقيمة "b" لنمط "B" لا تخل بأي تنميط فرعي لطول نمط **record of** أو بُعد صفيف لنمط "A". وتخصص قيم عناصر القيمة "b" تتابعياً مع تطابق نمط "A"، بما في ذلك عناصر غير معرفة.

تكون أنماط **record of** ومصفوفات بعد واحد متوائمة مع أنماط **record** إذا كانت بنيات قيمها الفعلية متوائمة وعدد عناصر قيمة "b" لنمط "B" أو بُعد صفيف "b" هو نفسه عدد عناصر نمط **record** "A". وقابلية الاختيار لمجالات نمط **record of** ليس لها أهمية عند تحديد المواءمة، أي، لا تؤثر على حساب المجالات (وتعني أن المجالات الخيارية تُتضمن دائماً في العدِّ). وتخصيص قيم العنصر لنمط **record** أو صفيف لتطابق مع نمط **record** يكون في ترتيب نصي متناظر مع تعريف نمط **record**، بما في ذلك عناصر غير معرّفة. وإذا خصصت قيمة غير معرفة لعنصر خياري لـ **record**، يسبب هذا حذف العنصر الخياري. ومحاولة تخصيص عنصر مع قيمة غير معرفة لعنصر إجباري لـ **record** يسبب خطأ.

ملاحظة – إذا لم يكن لنمط **record of** تقييد طول أو تقييد طول يتجاوز عدد عناصر نمط **record** المقارن ودليل أي عنصر معرف لقيمة **record** أقل أو مساو لعدد عناصر نمط **record** ناقصاً واحد، فإن متطلب المواءمة يتم الإيفاء به دائماً.

ويمكن أيضاً تخصيص قيم نمط **record** لتطابق نمط **record of** أو صفيف بعد وحيد إذا لم يتم الإخلال بتقييد طول نمط **record of** أو كان بُعد الصفيف أكثر من أو مساوٍ لعدد عناصر نمط **record**. وتخصص العناصر الخيارية الغائبة في قيمة **record** كعناصر مع قيم غير معرفة.

المثـــال 2:

```
      // Given
      type record HType {
        integer a,
        integer b optional,
        integer c
      }

      type record of integer IType

      var HType MyVarH := { 1, omit, 2};
      var IType MyVarI;
      var integer MyArrayVar[2];

      // Then
```

```
MyArrayVar := MyVarH;
// is a valid assignment as type of MyArrayVar and HType are compatible

MyVarI := MyVarH;
// is a valid assignment as the types are compatible and no subtyping is violated

MyVarI := { 3, 4 };
MyVarH := MyVarI;
// is NOT a valid assignment as the mandatory field 'c' of Htype receives no value
```

3.2.7.6 مواءمة نمط لأنماط set وset of

إن أنماط **set** هي نمط متوائم فقط مع أنماط **set** أخرى وأنماط **set of**. وبالنسبة لأنماط **set** وأنماط **set of** تنطبق نفس قواعد المواءمة كما لأنماط **record** و**record of**.

الملاحظة 1 – ويعني هذا أن، بالرغم من ترتيب عناصر عند إرسال واستقبال يكون غير معروف، عند تحديد مواءمة نمط لأنماط **set**، يكون الترتيب النصي لمجالات في تعريف النمط حاسمة.

الملاحظة 2 – وفي قيم **set**، يمكن أن يكون ترتيب مجالات اعتباطياً؛ ومع ذلك، لا يؤثر هذا على مواءمة نمط حيث أسماء المجالات غير المبهمة تعرف أي مجالات لنمط **set** ذات علاقة تتناظر مع أي مجالات قيمة **set**.

مثـــال:

```
// Given
type set FType {
  integer aoptional,
  integer boptional,
  boolean c
}

type set GType {
  integer d   optional,
  integer e   optional,
  boolean f
}

var FType MyVarF := { a:=1, c:=true };
var GType MyVarG := { f:=true, d:=7};

// Then

MyVarF := MyVarG; // is a valid assignment as types FType and GType are compatible

MyVarF := MyVarA; // is NOT a valid assignment as MyVarA is a record type
```

4.2.7.6 المواءمة بين بنيات فرعية

إن القواعد المعرفة في هذا القسم لمواءمة أنماط مبنية هي أيضاً صالحة لبنية فرعية لهذه الأنماط.

مثـــال:

```
// Given
type record JType {
      HType H,
      integer  b optional,
      integer  c
}

var JType MyVarJ

// If considering the declarations above, then


MyVarJ.H := MyVarH;
// is a valid assignment as the type of field H of JType and HType are compatible

MyVarI := MyVarJ.H;
// is a valid assignment as IType and the type of field H of JType are compatible
```

3.7.6 مواءمة نمط لأنماط مكون

إن مواءمة نمط لأنماط مكون يتعين النظر إليها في حالتين مختلفتين.

1) مواءمة قيمة مرجع مكون مع نمط مكون (مثل، عند تمرير مرجع مكون باعتباره معلمة فعلية لوظيفة أو altstep أو عند تخصيص قيمة مرجع مكون لمتغير نمط مكون مختلف): يكون مرجع مكون "b" لنمط مكون "B" متوائم مع نمط مكون "A" إذا كانت جميع تعاريف "A" تعاريف مماثلة في "B".

2) مواءمة **Runs on**: يمكن طلب وظيفة أو altsteps يشير إلى نمط مكون "A" في **Runs on** أو بدء تطابق مكون لنمط "B" إذا كانت جميع تعاريف "A" مماثلة للتعاريف في "B".

تُحدد هوية التعاريف في "A" مع تعاريف "B" على أساس القواعد التالية:

- لمطابقات منفذ، يكون كل من النمط والمعرف متماثلين.
- لمطابقات مؤقت، تكون المعرفات مماثلة، ويكون لها مدد أولية مماثلة أو لا توجد مدة أولية.
- لمطابقات متغير وتعاريف ثابتة، تكون المعرفات والأنماط وقيم التدميث مماثلة (في حالة متغيرات، يعني هذا أن أيهما غائب في كل من التعريفين أو نفسه).
- ولتعاريف مقاس محلي، تكون المعرفات والأنماط وقوائم المعلمات الرسمية وقيم مقاس مخصص أو مجال مقاس مماثلة.

4.7.6 مواءمة نمط لعمليات اتصالات

إن عمليات الاتصالات (انظر القسم 23) **send** و**receive** و**trigger** و**call** و**getcall** و**reply** و**getreplay** و**raise** هي استثناءات للقاعدة الأضعف لمواءمة نمط وتتطلب تنميطاً أقوى. ويجب على أنماط قيم أو مقاسات مستخدمة مباشرة كمعلمات لهذه العمليات أن تُعرّف بوضوح في تعريف نمط منفذ متصاحب. وينطبق التنميط القوي أيضاً على تخزين قيمة مستقبلة أو عنوان أو مرجع مكون خلال عملية **receive** أو **trigger**.

5.7.6 تحويل نمط

إذا كان من الضروري تحويل قيم نمط واحد إلى قيم نمط آخر، حيث الأنماط غير مشتقة من نفس نمط الجذر، فإن أي واحد من وظائف تحويل معرفة مسبقاً الواردة في الملحق C أو وظيفة معرفة لمستعمل تستخدم.

مثـــال:

```
// To convert an integer value to a hexstring value use the predefined function int2hex
MyHstring := int2hex(123, 4);
```

7 وحدات

0.7 عام

إن قِدرات البناء الرئيسية لـ TTCN-3 هي الوحدات. فمثلاً، يمكن أن تعرف وحدة متوالية اختبار قابل للتنفيذ بالكامل أو مجرد مكتبة. وتتألف وحدة من جزء تعاريف (خياري) وجزء تحكم وحدة (خياري).

ملاحظة – إن مصطلح "متوالية اختبار" مرادف لوحدة TTCN-3 تامة تحتوي على اختبارات مجردة وجزء تحكم.

1.7 تسمية وحدات

إن أسماء الوحدات هي شكل معرف TTCN-3. وبالإضافة إلى ذلك، يمكن أن تحمل مواصفة وحدة نعتاً خيارياً تعرفه الكلمة المفتاحية **language** التي تعرف نسخة لغة TTCN-3 التي تحدد فيها الوحدة. وحالياً، تدعم سلسلة اللغات التالية: "TTCN-3:2001" لمواصفة وحدة تمتثل لطبعة 2001 TTCN-3، و"TTCN-3:2003" لطبعة 2003، و"TTCN-3:2005" لطبعة 2006.

ملاحظة – إن معرِّف الوحدة هو اسم النص غير الرسمي للوحدة.

مثـــال:

```
module SIPTestSuite language "TTCN-3:2003"
{ … }
```

2.7 معلمات وحدة

0.2.7 عـــام

تُعرف قائمة معلمات الوحدة مجموعة قيم تورِّدها بيئة الاختبار عند التنفيذ. وخلال تنفيذ الاختبار تعامل هذه القيم على أنها ثوابت. ويعلن عن معلمات وحدة بواسطة تحديد النمط وتحديد معرفاتها بعد الكلمة المفتاحية **modulepar**. ولا تكون معلمات وحدة نمط منفذ أو نمط بالتغيب أو نمط مكون. وتكون معلمة وحدة عنواناً نمطاً فقط إذا كان نمط العنوان معرّفاً بوضوح في وحدة متصاحبة. ويعلن عن معلمات وحدة في جزء تعريف الوحدة فقط. ويسمح بأكثر من حدث إعلان معلمات وحدة، ولكن يعلن عن كل معلمة مرة واحدة فقط (أي، لا يُسمح بإعادة تعريف معلمة الوحدة).

مثـــال:

```
module MyModulewithParameters
{
  modulepar integer TS_Par0, TS_Par1;
  modulepar boolean TS_Par2;
  modulepar hexstring TS_Par3;
}
```

1.2.7 قيم بالتغيب لمعلمات وحدة

يُسمح بتحديد قيم بالتغيب لمعلمات وحدة. ويتم هذا بواسطة تخصيص في قائمة معلمات الوحدة. وقد تكون قيمة بالتغيب قيمة حرفية، ويمكن أن تخصص في مكان إعلان المعلمة. وإذا لم يوفر نظام الاختبار قيمة تنفيذ فعلية لمعلمة ما، تستخدم القيمة بالتغيب خلال تنفيذ الاختبار؛ وإلا، تستخدم القيمة الفعلية التي وفرها نظام الاختبار.

مثـــال:

```
module MyModuleDefaultParameter
{
  modulepar integer TS_Par0 := 0, TS_Par1;
  modulepar boolean TS_Par2 := true;
  :
}
```

3.7 جزء تعاريف وحدة

0.3.7 عـــام

يحدد جزء تعاريف الوحدة تعاريف المستوى العلوي للوحدة، ويمكن أن يستورد معرفات من وحدات أخرى. ويرد في 3.5 قواعد منظور لإعلانات تمت في جزء تعاريف الوحدة والإعلانات المستوردة. ويرد في الجدول 1 عناصر اللغة التي يمكن أن تعرف في وحدة TTCN-3. ويمكن استيراد تعاريف الوحدة بواسطة وحدات أخرى.

مثـــال:

```
module MyModule
{ // This module contains definitions only
  :
  const integer MyConstant := 1;
  type record MyMessageType { … }
  :
  function TestStep(){ … }
  :
}
```

تتم إعلانات عناصر لغة دينامية مثل **var** أو **timer** فقط في جزء التحكم أو اختبارات مجردة أو وظائف أو altsteps أو أنماط مكون.

ملاحظة – لا يدعم TTCN-3 إعلان متغيرات في جزء تعاريف الوحدة. ويعني هذا أنه لا يمكن تعريف متغيرات عامة في TTCN-3. ومع ذلك، يمكن استخدام متغيرات معرفة في مكون اختبار بواسطة جميع الاختبارات المجردة والوظائف وما إلى ذلك، المنفذة على ذلك المكون وتوفر المتغيرات المعرفة في جزء التحكم القدرة على الاحتفاظ بقيمها مستقلة عن تنفيذ اختبار مجرد.

1.3.7 زمرات تعاريف

في جزء تعاريف الوحدة، يمكن تجميع التعاريف في زمرات مسماة. ويمكن تحديد زمرة إعلانات حيثما يسمح بإعلان وحيد. ويمكن أن تكون الزمرات متداخلة، أي، قد تحتوي الزمرات على زمرات أخرى. ويسمح هذا لمواصف متوالية الاختبار ببنية، من بين أشياء أخرى، وتجميع معطيات اختبار أو وصف وظائف سلوك اختبار.

ويتم التجميع لمساعدة القابلية على القراءة وإضافة بنية منطقية للوحدة إذا طلبت. والزمرات والزمرات المتداخلة ليس لها منظور إلّا في سياق معرفات زمرة ونعوت مقدمة إلى زمرة بواسطة بيان **with** المتصاحب. ويعني هذا:

- ليست هناك ضرورة لتكون معرفات زمرة عبر كل الوحدة وحيدة. ومع ذلك، تكون جميع معرفات زمرة لزمرات فرعية لزمرة واحدة وحيدة. وإذا لزم الأمر، يستخدم ترميز بالنقط لتعريف الزمرات الفرعية في تراتبية الزمرة على نحو وحيد، مثل، لاستيراد زمرة فرعية محددة.
- يرد في 4.28 القواعد السائدة للنعوت.

مثـــال:

```
module MyModule {
   :
   // A collection of definitions
   group MyGroup {
       const integer MyConst:= 1;
        :
       type record MyMessageType { … };
       group MyGroup1 { // Sub-group with definitions
        type record AnotherMessageType { … };
        const boolean MyBoolean := false
       }
   }

   // A group of altsteps
   group MyStepLibrary {
       group MyGroup1 { // Sub-group with the same name as the sub-group with  definitions
            altstep MyStep11() { … }
            altstep MyStep12() { … }
             :
            altstep MyStep1n() { … }
       }
       group MyGroup2 {
            altstep MyStep21() { … }
            altstep MyStep22() { … }
            :
            altstep MyStep2n() { … }
        }
       }
       :
   }

   // An import statement that imports MyGroup1 within MyStepLibrary
   import from MyModule {
   group MyStepLibrary.MyGroup1
   }
```

4.7 جزء التحكم في الوحدة

يمكن أن يحتوي جزء التحكم في الوحدة على تعاريف محلية ويصف ترتيب التنفيذ (من الممكن تكرارياً) لاختبارات مجردة فعلية. ويعرف اختبار مجرد في جزء تعاريف الوحدة ويطلب في جزء التحكم.

مثـــال:

```
module MyTestSuite
   {   // This module contains definitions …
       :
       const integer MyConstant := 1;
       type record MyMessageType { … }
       template MyMessageType MyMessage := { … }
       :
       function MyFunction1() { … }
       function MyFunction2() { … }
       :
       testcase MyTestcase1() runs on MyMTCType { … }
       testcase MyTestcase2() runs on MyMTCType { … }
```

```
        :
        // … and a control part so it is executable
        control
        {
            var boolean MyVariable; // local control variable
            :
            execute( MyTestCase1()); // sequential execution of test cases
            execute( MyTestCase2());
            :
        }
    }
```

5.7 الاستيراد من وحدات

0.5.7 عـــام

من الممكن إعادة استخدام تعاريف محددة في وحدات مختلفة باستخدام بيان **import**. وليس لـ TTCN-3 بنية تصدير واضحة؛ ومن ثم، بالتغيب، يمكن استيراد جميع تعاريف وحدة في جزء تعاريف الوحدة. ويمكن استخدام بيان **import** في أي مكان في جزء تعاريف الوحدة. ولا يستخدم في جزء التحكم.

وإذا كان لتعريف مستورد نعوت (معرفة بواسطة بيان **with**)، فإن النعوت تكون مستوردة أيضاً. ويرد في 6.28 شرح لآلية تغيير نعوت لتعاريف مستوردة.

ملاحظة – إذا كان للوحدة نعوت شاملة، تتصاحب مع تعاريف دون هذه النعوت.

مثـــال:

```
module MyModuleA
{       // This module contains definitions and imported definitions
        :
        const integer MyConstant := 1;
        import from MyModuleB all; // Scope of the imported definitions is global to MyModuleA
        import from MyModuleC {
            type MyType1, MyType2;
            template all
        }
        type record MyMessageType { … }
        :
        function MyBehaviourC()
        {
            const integer MyConstant := 2;
            // import cannot be used here
             :
        }
        :
        control
        {   // import cannot be used here
            :
        }
       }
```

1.5.7 بنية تعاريف قابلة للاستيراد

يدعم TTCN-3 استيراد التعاريف التالية: معلمات وحدة وأنماط معرفة لمستعمل وتوقيعات وثوابت وثوابت خارجية ومقاسات معطيات ومقاسات توقيع ووظائف ووظائف خارجية وaltsteps واختبارات مجردة. ويكون لكل تعريف *اسم* (يعرف مُعرف التعريف، مثل، اسم وظيفة)، *ومواصفة* (مثل، مواصفة نمط أو توقيع وظيفة) وفي حالة وظائف altsteps واختبارات متصاحبة مع *وصف سلوك*.

مثـــال:

	Name	Specification	Behaviour description

function	MyFunction	(**inout** MyType1 MyPar) **return** MyType2
		runs on MyCompType
			{
			const MyType3 MyConst := …;
			: // further behaviour
			}

	Specification	Name	Specification

Type	record	MyRecordType	{
			field1 MyType4,
			field2 integer
			}

	Specification	Name	Specification

template	MyType5	MyTemplate	:= {
			field1 := 1,
			field2 := MyConst, // MyConst is a module constant
			field3 := ModulePar // ModulePar is module parameter
			}

إن وصف السلوك ليس له تأثير على آلية الاستيراد بسبب أن داخلها يعتبر غير مرئي للمستورد عندما تستورد الوظائف أو altsteps أو الاختبارات المجردة المتناظرة. ومن ثمن لا تعتبر في الوصف التالي.

يحتوي جزء المواصفة لتعريف قابل للاستيراد على *تعاريف محلية* (مثل، أسماء مجالات لتعاريف نمط مبني أو قيم لأنماط متعددة) *وتعاريف مرجعية* (مثل، مراجع لتعاريف نمط أو مقاسات أو ثوابت أو معلمات وحدة). وبالنسبة للأمثلة أعلاه، يعني هذا:

	الاسم	تعاريف محلية	تعاريف مرجعية
function	MyFunction	MyPar	MyType1, MyType2, MyCompType
type	MyRecordType	field1, field2	MyType4, integer
template	MyTemplate		MyType5, field1, field2, field3, MyConst, ModulePar

الملاحظة 1 – يشير عمود التعاريف المحلية إلى معرفات معرفة جيداً في تعريف قابل للاستيراد. ويمكن أيضاً اعتبار قيم مخصصة لمجالات فردية لتعاريف قابلة للاستيراد، مثلاً، في تعاريف مقاس، كتعاريف محلية، ولكنها غير مهمة لشرح آلية الاستيراد.

الملاحظة 2 – إن التعاريف المرجعية field1 وfield2 وfield3 لمقاس MyTemplate هي أسماء مجالات لـ MyType5، أي، يمكن الرجوع إليها عبر MyType5.

إن التعاريف المرجعية هي أيضاً تعاريف قابلة للاستيراد، أي، يمكن بناء مصدر تعريف مرجعي مرة ثانية في اسم وجزء مواصفة، ويحتوي أيضاً جزء المواصفة على تعاريف محلية ومرجعية. وبمعنى آخر، يمكن بناء تعريف قابل للاستيراد على نحو متكرر من تعاريف أخرى قابلة للاستيراد.

تتعلق آلية استيراد TTCN-3 بالتعاريف المحلية والمرجعية المستخدمة في جزء المواصفة لتعاريف قابلة للاستيراد. ولهذا، يحدد الجدول 5 التعاريف المحلية والمرجعية الممكنة لتعاريف قابلة للاستيراد.

الجدول Z.140/5 – تعاريف محلية ومرجعية ممكنة لتعاريف قابلة للاستيراد

تعريف قابل للاستيراد	تعريف محلية ممكنة	تعاريف مرجعية ممكنة
معلمة وحدة		نمط معلمة وحدة
نمط معرف لمستعمل (للجميع)	أسماء معلمة	نمط معلمة
• نمط معدد	قيم محسوسة	
• نمط مبني	أسماء مجال	أنماط مجال
• نمط منفذ		أنماط رسالة، توقيعات
• نمط مكون	أسماء ثابت وأسماء متغير وأسماء مؤقت وأسماء منفذ	أنماط ثابت وأنماط متغير وأنماط منفذ
توقيع	أسماء معلمة	أنماط معلمة ونمط عودة وأنماط استثناءات
ثابت		نمط ثابت
ثابت خارجي		نمط ثابت
مقاس معطيات	أسماء معلمة	نمط مقاس وأنماط معلمة وثوابت ومعلمات وحدة ووظائف
مقاس توقيع		تعريف توقيع وثوابت ووظائف معلمات وحدة
وظيفة	أسماء معلمة	أنماط معلمة ونمط عودة ونمط مكون (**runs on**-clause)
وظيفة خارجية	أسماء معلمة	أنماط معلمة ونمط عودة
Altstep	أسماء معلمة	أنماط معلمة ونمط مكون (**runs on**-clause)
اختبار مجرد	أسماء معلمة	أنماط معلمة وأنماط مكون (**runs on** و**system**-clause)

تُميّز آلية استيراد TTCN-3 بين *معرّف تعريف مرجعي* و*المعلومات الضرورية لاستخدام تعريف مرجعي* في التعريف المستورد. ومن أجل الاستخدام، لا يطلب معرّفاً تعريفاً مرجعياً، ولهذا، لا يُستورد أوتوماتياً.

2.5.7 قواعد بشأن استخدام الاستيراد

عند استخدام الاستيراد، تنطبق القواعد التالية:

أ) يمكن فقط استيراد تعاريف مستوى علوي في الوحدة. ولا تستورد التعاريف التي تحدث عند منظور أدنى (مثل، ثوابت محلية معرفة في وظيفة).

ب) يسمح فقط بالاستيراد المباشر من مصدر وحدة تعريف (أي الوحدة، حيث يوجد التعريف الفعلي لمعرف مرجعي في بيان **import**).

ج) يستورد تعريف مع اسمه وجميع التعاريف المحلية.

الملاحظة 1 – يكون لتعريف محلي، مثل، اسم مجال لنمط تسجيل معرف لمستعمل، معني فقط في سياق التعاريف التي يُعرّف فيه، مثلاً، يمكن فقط لاسم مجال لنمط تسجيل أن يستخدم للنفاذ إلى مجال نمط تسجيل وليس خارج هذا السياق.

د) يستورد تعريف مع جميع معلومات التعاريف المرجعية الضرورية لاستخدام تعريف مرجعي.

الملاحظة 2 – إن بيانات الاستيراد هي انتقالية، مثلاً، إذا استوردت وحدة A تعريفاً من الوحدة B التي تستخدم مرجع نمط معرف في الوحدة C، تستورد المعلومات المتناظرة الضرورية لاستخدام ذلك النمط أوتوماتياً إلى الوحدة A.

هـ) إن معرفات تعاريف مرجعية لا تستورد أوتوماتياً.

الملاحظة 3 – إذا رغب في استخدام تعاريف مرجعية في وحدة مستوردة، تستورد من مصدر الوحدة بوضوح.

و) عند استيراد وظيفة، أو altstep أو اختبار مجرد، تظل مواصفات السلوك المتناظرة وجميع التعاريف المستخدمة داخل مواصفات السلوك غير مرئية للوحدة المستوردة.

ز) يجري حظر الواردات الدورية.

مثـــال:

```
module ModuleONE {

  modulepar integer ModPar1, ModPar2 := 7

  type record RecordType_T1 {
     integer  Field1_T1,
     boolean  Field2_T1
  }
```

```
   type record RecordType_T2 {
      RecordType_T1    Field1_T2,   // Use of RecordType_T1
      RecordType_T1    Field2_T2,
      integer      Field3_T2
   }

   const integer MyConst := 13;

   template RecordType_T2 Template_T2 (RecordType_T1 TempPar_T2):= { //  parameterized template
      Field1_T2 := TempPar_T2,      // Reference to template parameter
      Field2_T2 := {MyConst, true}, // Reference to module constant
      Field3_T2 := ModPar1          // Reference to a module parameter
   }

} // end module ModuleONE


module ModuleTWO {

   import from ModuleONE {
      template Template_T2
   }

   // Only the names Template_T2 and TempPar_T2 will be visible in ModuleTWO. Please note, that
   // the identifier TempPar_T2 can only be used in the context of Template_T2, e.g. when
   // providing an actual parameter value. All information
   // necessary for the usage of Template_T2, e.g. for type checking purposes, are imported
   // for the referenced definitions RecordType_T2, RecordType_T1, Field1_T2, Field2_T2,
   // Field3_T3, MyConst and ModPar1, but their identifiers are not visible in ModuleTWO.
   // This means, e.g. it is not possible to use the constant MyConst or to declare a
   // variable of type RecordType_T1 or RecordType_T2 in ModuleTWO without explicitly importing
   // these types

   import from ModuleONE {
      modulepar ModPar2
   }

   // The module parameter ModPar2 of ModuleONE is imported from ModuleONE and
   // can be used like an integer constant

    } // end module ModuleTWO


module ModuleTHREE {

   import from ModuleONE all; // imports all definitions from ModuleONE

   type port MyPortType {
      inout RecordType_T2
   }

   type component MyCompType {
      var integer MyComponentVar := ModPar2; // Reference to a module parameter of ModuleONE
      port MyPortType MyPort
   }

   function MyFunction () return integer {
      return MyConst    // Returns a module constant defined in ModuleONE
   }

   testcase MyTestCase (out RecordType_T2 MyPar) runs on MyCompType {

      var integer MyTCVar := ModPar2; // Reference to a module parameter of ModuleONE

      MyPort.send(Template_T2); // Sending a template defined in ModuleONE
      MyPort.receive(RecordType_T2 : ?) -> value MyPar;  // The received value is assigned
                                                         // to the out parameter MyPar.

}     // end testcase MyTestCase

} // end ModuleTHREE


module ModuleFOUR {

   import from ModuleTHREE {
      testcase MyTestCase
   }
```

```
  // Only the names MyTestCase and MyPar will be visible and usable in ModuleFOUR.
  // Type information for RecordType_T2 is imported via ModuleTHREE from ModuleONE and
  // type information for MyCompType is imported from ModuleTHREE. All definitions
  // used in the behaviour part of MyTestCase remain hidden for the user of ModuleFOUR.

    } // end ModuleFOUR
```

3.5.7 فارغ

4.5.7 استيراد تعاريف وحيدة

يمكن استيراد تعاريف وحيدة.

مثـــال:

```
import from MyModuleA {
  type MyType1    // imports one type definition from MyModuleA
}

import from MyModuleB {
  type MyType2, Mytype3, MyType4;  // imports three types
  template MyTemplate1;            // imports one template
  const MyConst1, MyConst2         // imports two constants
 }
```

5.5.7 استيراد جميع تعاريف وحدة

يمكن استيراد جميع تعاريف جزء تعاريف وحدة باستخدام الكلمة المفتاحية **all** بجانب اسم الوحدة. وإذا تم استيراد جميع تعاريف وحدة باستخدام الكلمة المفتاحية **all**، لا يستخدم أي شكل من الاستيراد (استيراد تعاريف وحيدة واستيراد نفس النوع وما إلى ذلك) لنفس بيان **import**.

المثـــال 1:

```
import from MyModule all;
```

وإذا كانت هناك رغبة في عدم استيراد بعض الإعلانات، ترد أنواعها ومعرفاتها في قائمة الاستثناء في زوج من الأقواس المعقوفة بعد الكلمة المفتاحية **except**.

المثـــال 2:

```
import from MyModule all except {
  type MyType3, MyType5
      // excludes type declarations MyType3 and MyType5 from the import statement
      // but imports all other declarations of MyModule
}
```

يسمح باستخدام الكلمة المفتاحية **all** في قائمة الاستثناء؛ ويستثني هذا جميع الإعلانات لنفس النوع من بيان الاستيراد.

المثـــال 3:

```
import from MyModule all except {
  type MyType3, MyType5;   // excludes the two types from the import statement
  template all // excludes all templates declared in MyModule from the import
statement
 }
```

6.5.7 زمرات استيراد

يمكن استيراد زمرات تعاريف.

المثـــال 1:

```
import from MyModule {
  group MyGroup
 }
```

يكون تأثير استيراد زمرة مماثلاً لبيان **import** الذي يورد جميع التعاريف القابلة للاستيراد (بما في ذلك الزمرات الفرعية) لهذه الزمرة.

تُستخدم زمرات TTCN-3 فقط لأغراض البناء وهي ليست وحدات منظور. ولهذا، يُسمح باستيراد زمرات فرعية (أي، زمرة تُعرّف في زمرة أخرى) مباشرة، أي دون زمرات تدمج فيها زمرة فرعية. وإذا كان اسم زمرة فرعية ينبغي استيرادها مماثلة لاسم زمرة فرعية أخرى في نفس الوحدة (انظر 1.3.7)، يستخدم ترميز بالنقط لتعريف الزمرة الفرعية التي تستورد بشكل وحيد.

وإذا كانت هناك رغبة في عدم استيراد بعض تعاريف زمرة، ترد أنواعها ومعرفاتها في قائمة الاستثناء في زوج من الأقواس المعقوفة بعد الكلمة المفتاحية **except**.

المثـــال 2:

```
import from MyModule {
  group MyGroup except {
      type MyType3, MyType5
       // excludes type definitions MyType3 and MyType5 from the import statement
       // but imports all other definitions of MyGroup
  }
 }
```

يسمح باستخدام الكلمة المفتاحية **all** في قائمة الاستثناء؛ ويستثني هذا جميع الإعلانات لنفس النوع من بيان الاستيراد.

المثـــال 3:

```
import from MyModule {
  group MyGroup except {
      type MyType3, MyType5;  // excludes the two types from the import statement and
      template all // excludes all templates defined in MyGroup from the import statement
  }
 }
```

7.5.7 تعاريف استيراد نفس النوع

يمكن استخدام الكلمة المفتاحية **all** لاستيراد جميع تعاريف من نفس النوع لوحدة. وتعرف الكلمة المفتاحية **all** المستخدمة مع الكلمة المفتاحية **constant** جميع الثوابت، وكذلك جميع الثوابت الخارجية المعلن عنها في جزء تعاريف الوحدة التي يشير إليها بيان الاستيراد. وبالمثل، تعرف الكلمة المفتاحية **all** المستخدمة مع الكلمة المفتاحية **function** جميع الوظائف وجميع الوظائف الخارجية في الوحدة التي يدل عليها بيان الاستيراد.

المثـــال 1:

```
import from MyModule {
  type all;           // imports all types of MyModule
  template all        // imports all templates of MyModule
}
```

في بعض أنواع الإعلانات التي يراد استثناؤها من بيان استيراد ما، ترد معرفاتها بعد الكلمة المفتاحية **except**.

المثـــال 2:

```
import from MyModule {
  type all except MyType3, MyType5; // imports all types except MyType3 and MyType5
  template all                      // imports all templates defined in Mymodule
}
```

8.5.7 مناولة تناقضات اسم بشأن الاستيراد

يكون لجميع وحدات TTCN-3 مسافة اسمها الخاص، حيث تكون جميع التعاريف معرفة بشكل وحيد. ويمكن أن تحدث تناقضات اسم نتيجة للاستيراد، مثلاً، الاستيراد من وحدات مختلفة. وتستبان تناقضات اسم بواسطة وضع سابقة لتعريف مستورد (الذي يسبب تناقض اسم) بواسطة مُعرف الوحدة الذي تم استيراده. ويفصل بين السابقة والمعرف نقطة.

وفي الحالات التي لا يوجد فيها غموض، ليست هناك حاجة لسابقة (ولكن يمكن) عندما تستخدم تعاريف مستوردة، وعندما يكون التعريف مرجعياً في نفس الوحدة حيث تكون معرفة، يمكن استخدام معرف الوحدة للوحدة (الوحدة الحالية) لوضع سابقة لمعرف التعريف.

مثـــال:

```
module MyModuleA  {
   :
  type bitstring MyTypeA;
  import from SomeModuleC {
      type MyTypeA,     // Where MyTypeA is of type character string
           MyTypeB      // Where MyTypeB is of type character string
  }
   :
  control {
       :
      var  SomeModuleC.MyTypeA MyVar1 := "Test String"; // Prefix must be used
      var MyTypeA MyVar2 := '10110011'B; // This is the original MyTypeA
       :
      var MyTypeB MyVar3 := "Test String"; // Prefix need not be used …
      var SomeModuleC.MyTypeB MyVar3 := "Test String"; // … but it can be if wished
       :
  }
}
```

ملاحظة – يفترض دائماً أن التعاريف، مع نفس الاسم المعرف في وحدات مختلفة، أن تكون مختلفة، حتى إذا كانت التعاريف الفعلية في الوحدات المختلفة مماثلة. فمثلاً، يؤدى نمط معرف محلياً، حتى مع نفس الاسم، إلى نمطين مختلفين متاحين في الوحدة.

9.5.7 مناولة مراجع متعددة لنفس التعريف

يمكن أن يؤدى استخدام **import** في تعاريف وحيدة، وزمرات تعاريف، وتعاريف لنفس النوع وما إلى ذلك، إلى حالات حيث يشار إلى نفس التعريف لأكثر من مرة. وتُستبان مثل هذه الحالات بواسطة النظام والتعاريف وتستورد مرة واحدة فقط.

ملاحظة – إن الآليات لاستبانة هذا الغموض، مثل الإفراط في الكتابة وإرسال تحذيرات إلى المستعمِل، هي خارج مدى هذه التوصية وينبغي أن توفرها أدوات TTCN-3.

إن جميع بيانات وتعاريف **import** في بيانات الاستيراد تعامل مستقلة واحدة بعد الأخرى حسب ترتيب ظهورها. ومن المهم الإشارة إلى أن بيان **except** لا يستثني التعاريف الواردة من المستوردة بشكل عام؛ ويمكن النظر إلى جميع بيانات تعاريف مستوردة على أنها ترميز مختزل للقائمة المتكافئة لمعرفات تعاريف وحيدة. ويستثني بيان **except** التعاريف من القائمة الوحيدة هذه فقط.

مثـــال:

```
import from MyModule {
   type all except MyType3; // imports all types of MyModule  except MyType3
   type MyType3             // imports MyType3 explicitly
}
```

10.5.7 تعاريف استيراد من غير وحدات TTCN-3

في الحالات حيث التعاريف مستوردة من مصادر أخرى غير وحدات TTCN-3، تستخدم مواصفة لغة تدل على لغة (قد تكون مع رقم صيغة) المصدر (مثل، وحدة أو مجموعة أو مكتبة أو حتى ملف) تستورد منه التعاريف. وتتألف من الكلمة المفتاحية **language** وإعلان نصي لاحق للغة دالة عليه. إن استخدام مواصفة لغة خياري عند الاستيراد من وحدة TTCN-3 لنفس الطبعة مثل وحدة الاستيراد. وعند اكتشاف عدم المواءمة بين تعرف اللغة (بما في ذلك التعرف الضمني بواسطة حذف مواصفة اللغة) وقواعد تركيب الوحدة من تعاريف مستوردة، توفر أدواته جهوداً معقولة لاستبانة النـــزاع.

إن معرفات لغة TTCN-3 التالية تعرف:

- 'TTCN-3:2001' – يستخدم مع وحدات تمتثل لصيغة 2001 لهذه التوصية (انظر الببليوغرافيا).
- 'TTCN-3:2003' – يستخدم مع وحدات تمتثل لصيغة 2003 لهذه التوصية (انظر الببليوغرافيا).
- 'TTCN-3:2005' – يستخدم مع وحدات تمتثل لهذه التوصية.

مثـــال:

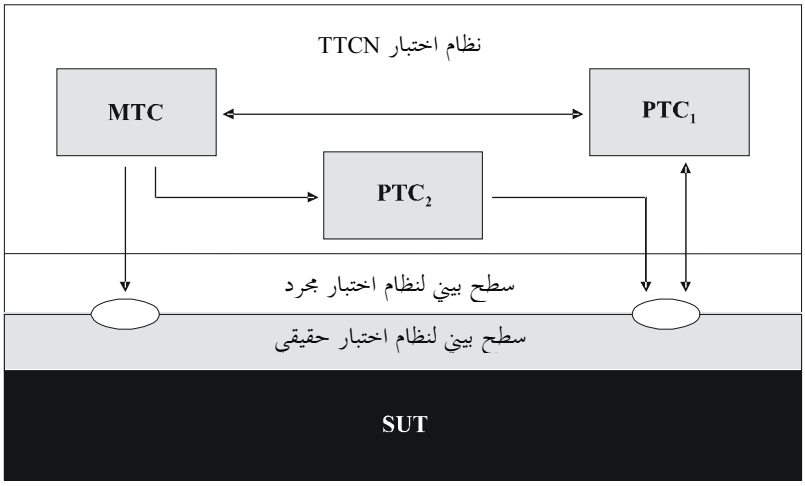
```
import from MyModule language "TTCN-3:2003" {
  type MyType
 }
```

ملاحظة – تصمم آلية الاستيراد للسماح بإعادة استخدام تعاريف من TTCN-3 آخر أو وحدات لغة أخرى. وقواعد تعاريف الاستيراد من مواصفات مكتوبة بلغات أخرى، مثل، مجموعات DSL، يمكن أن تتبَع قواعد أو قد تكون معرفة على نحو منفصل.

8 تشكيلات اختبار

0.8 عام

يسمح TTCN-3 بمواصفة (دينامية) لتشكيلات اختبار متلازمة (أو تشكيلات باختصار). ويتألف تشكيل من مجموعة من مكونات اختبار موصّلة داخلياً مع منافذ اتصالات معرفة جيداً ونظام اختبار واضح يعرف حدود نظام الاختبار.



الشكل Z.140/3 – نظرة مفهومية لتشكيل اختبار TTCN-3 نمطي

في كل تشكيل، يوجد مكون اختبار رئيسي (MTC) واحد (وواحد فقط). وتسمى مكونات اختبار ليست MTC مكونات اختبار متوازية أو PTC. ويخلق النظام MTC أوتوماتياً عند بدء تنفيذ كل اختبار. وينفذ السلوك المعرف في جسم اختبار مجرد على هذا المكون. وخلال تنفيذ اختبار مجرد، يمكن خلق مكونات أخرى بواسطة الاستخدام الواضح لعملية **create**.

وينتهي تنفيذ اختبار مجرد عند انتهاء MTC. وتعالج جميع MTC الأخرى بالتساوي؛ أي، لا توجد تراتبية واضحة للعلاقة فيما بينها وانتهاء PTC وحيد لا ينهي مكونات أخرى أو MTC. وعندما ينتهي MTC، يتعين أن يوقف نظام الاختبار جميع PTC في لحظة انتهاء تنفيذ الاختبار المجرد.

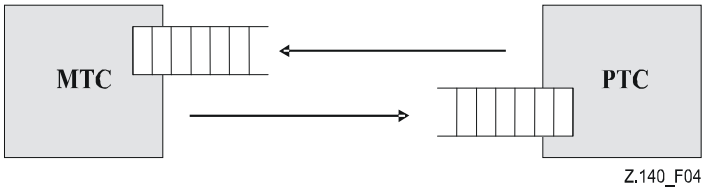
تتحقق الاتصالات بين مكونات اختبار وبين المكونات والسطح البيني لنظام اختبار عبر منافذ اتصالات (انظر 1.8).

تعرف أنماط مكون اختبار وأنماط منفذ، التي تدل عليها الكلمات المفتاحية **component** و**port** في جزء تعاريف الوحدة. ويتحقق التشكيل الفعلي للمكونات والتوصيلات بينها بواسطة أداء سطح بيني لنظام اختبار بواسطة عملية **create** و**connect** (انظر 2.22).

1.8 نموذج اتصالات منفذ

توصل مكونات اختبار عبر منافذ، أي، توصيلات فيما بين مكونات وبين مكون والسطح البيني لنظام اختبار موجه نحو المنفذ. ويكون نموذج كل منفذ باعتباره صف انتظار FIFO لا نهائي يخزن الرسائل الواصلة أو نداءات إجراء حتى تعالج بواسطة مكون يمتلك ذلك المنفذ.

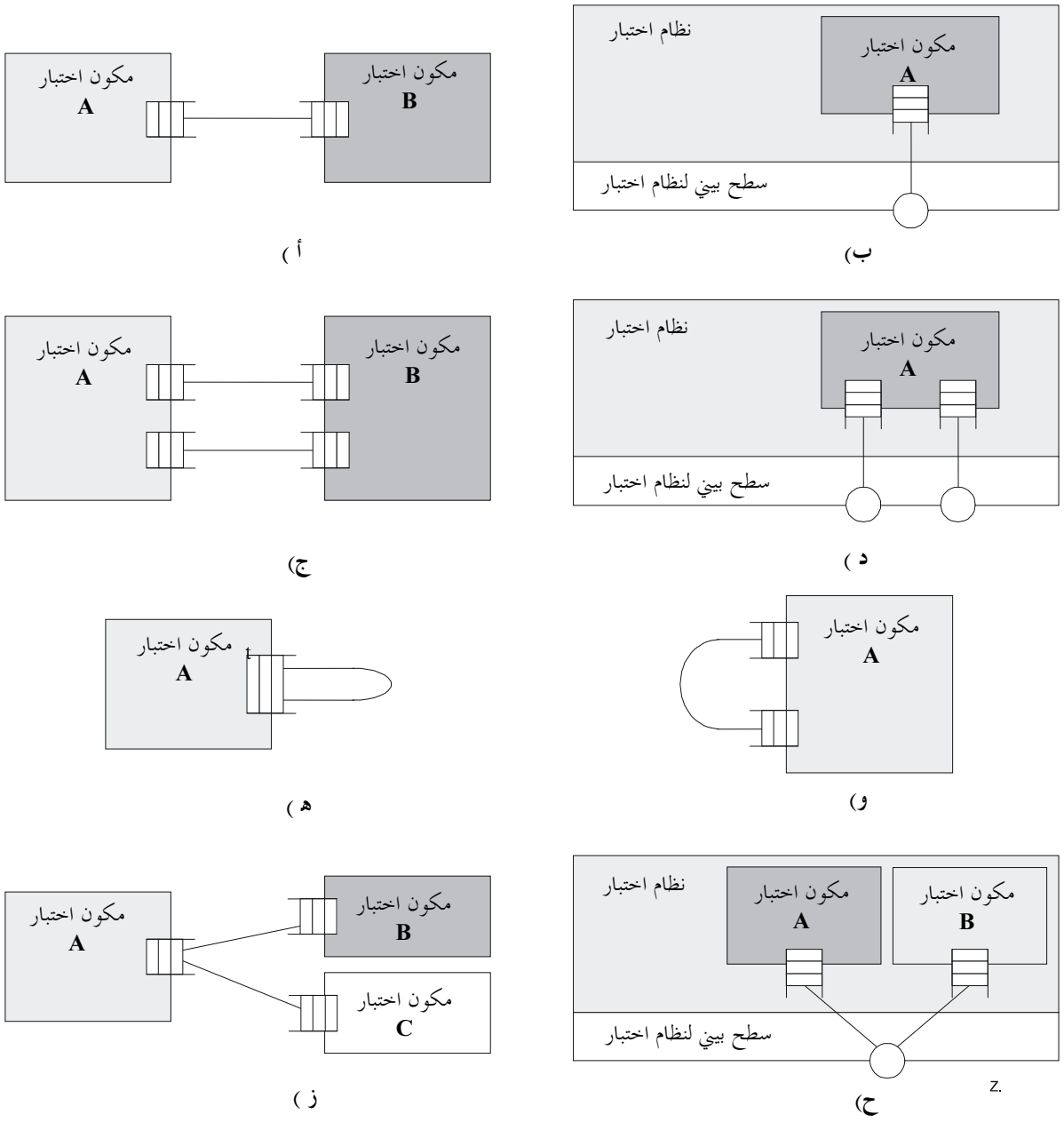
ملاحظة – بينما تكون منافذ TTCN-3 لا نهائية من ناحية المبدأ، يمكن أن تكون في نظام اختبار حقيقي فياضة. وينبغي معالجة هذا كخطأ اختبار مجرد (انظر 1.2.25).



الشكل Z.140/4 – نموذج منفذ اتصالات TTCN-3

2.8 تقييدات على توصيلات

إن توصيلات TTCN-3 هي توصيلات من منفذ إلى منفذ ومن منفذ إلى سطح بيني لنظام اختبار (انظر الشكل 5). ولا توجد تقييدات على عدد التوصيلات التي يحتفظ بها مكون. ويسمح بواحد إلى كثيرين أيضاً (مثلاً، انظر الشكل 5 (ز) أو الشكل 5 (ح)).

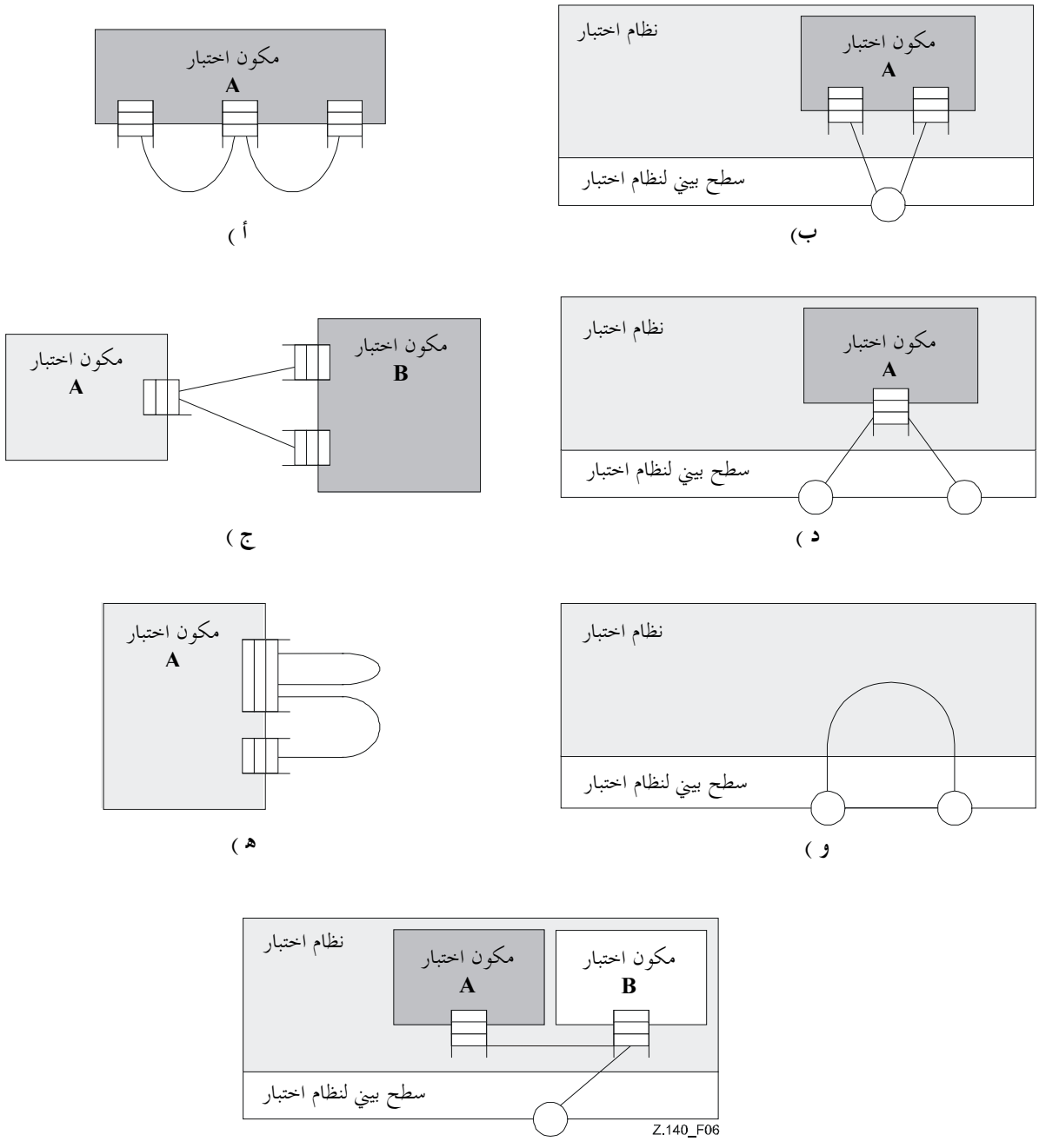


الشكل Z.140/5 – التوصيلات المسموح بها

لا يسمح بالتوصيلات التالية:

- لا يسمح بتوصيل منفذ يمتلكه مكون A بمنفذين أو أكثر يمتلكها نفس المكون (الشكلان 6 أ) و6 هـ)).
- لا يوصل منفذ يمتلكه مكون A مع منفذين أو أكثر لمكون B (انظر الشكل 6(ج)).
- يمكن أن يكون لمنفذ يمتلكه المكون A توصيل من واحد إلى واحد فقط مع السطح البيني لنظام اختبار. ويعني هذا، عدم السماح للتوصيلات المبينة في الشكلين 6(ب) و6(د).
- لا يسمح بتوصيلات في السطح البيني لنظام اختبار (انظر الشكل 6 و)).
- إن المنفذ الذي يوصل لا يتقابل والمنفذ المتقابل لا يوصل (انظر الشكل 6 ز)).

نظراً لأن TTCN-3 يسمح بتشكيلات وعناوين دينامية، لا يمكن دائماً التأكد من التقييدات على التوصيلات باعتبارها وقت تصريف. ويتم التأكد عند التنفيذ ويؤدي إلى خطأ اختبار مجرد عند العطل.



الشكل Z.140/6 – التوصيلات غير المسموح بها

3.8 السطح البيني لنظام اختبار مجرد

يستخدم TTCN-3 لتنفيذ اختبار. ويكون الشيء الذي يختبر معروفاً باعتباره اختبار تحت التنفيذ أو IUT. ويمكن أن يوفر IUT سطوح بينية مباشرة لاختبار أو يمكن أن يكون جزءاً من نظام، وفي هذه الحالة يكون الشيء المختبر معروفاً باعتباره نظام تحت اختبار أو SUT. وفي الحالات الدنيا، يكون IUT وSUT متكافئين. وفي هذه التوصية يستخدم مصطلح SUT بطريقة عامة ليعني سواء SUT أو IUT.

وفي بيئة اختبار حقيقي، تحتاج اختبارات مجردة إلى الاتصال مع SUT. ومع ذلك، فإن مواصفة توصيل مادي حقيقي هو خارج مدى TTCN-3. وبدلاً من ذلك، يتصاحب سطح بيني لاختبار معرف جيداً (ولكن مجرد) مع كل اختبار مجرد. ويكون تعريف سطح بيني لنظام اختبار مماثل لتعريف مكون، أي، أنه قائمة لجميع منافذ الاتصالات الممكنة التي يوصل من خلالها الاختبار المجرد بـ SUT.

يعرف السطح البيني لنظام اختبار سكونياً العدد ونمط توصيلات المنفذ بـ SUT خلال تنفيذ اختبار. ومع ذلك، تكون التوصيلات بين السطح البيني لنظام اختبار ومكونات اختبار TTCN-3 دينامية في طابعها ويمكن تعديلها خلال تنفيذ الاختبار باستخدام عمليتي **map** و**unmap** (انظر 2.22 و3.22).

4.8 تعريف أنماط منفذ اتصالات

0.4.8 عام

تُيسِّر المنافذ الاتصالات بين مكونات اختبار وبين مكونات اختبار وسطح بيني لنظام اختبار.

يدعم TTCN-3 المنافذ القائمة على رسالة والقائمة على إجراء. ويُعرَّف كل منفذ على أنه قائم على رسالة أو قائم على إجراء (أو كلاهما في نفس الوقت كما ورد في 1.4.8). وتُعرَّف المنافذ القائمة على رسالة بواسطة الكلمة المفتاحية **message** والمنافذ القائمة على إجراء بالكلمة المفتاحية **procedure** في تعريف نمط منفذ متصاحب.

إن المنافذ هي ثنائية الاتجاه. ويحدد الاتجاهات الكلمات المفتاحية **in** (للاتجاه الداخل) و**out** (للاتجاه الخارج) و**inout** (لكلا الاتجاهين). ويكون لكل نمط منفذ قائمة واحدة أو أكثر تشير إلى التجميع المسموح به لأنماط (رسالة) و/أو إجراءات مع اتجاه الاتصالات المسموح بها.

عندما يعرف توقيع (انظر أيضاً القسم 13) في الاتجاه "out" لمنفذ قائم على إجراء، تكون أنماط جميع معلماته **inout** و**out** ونمط عودته وأنماط استثنائه أوتوماتياً جزء من الاتجاه "in" لهذا المنفذ. وعندما يعرف توقيع (انظر أيضاً القسم 13) في الاتجاه "in" لمنفذ قائم على إجراء، تكون أنماط جميع معلماته **inout** و**out** ونمط عودته وأنماط استثنائه أوتوماتياً جزء من الاتجاه "out" لهذا المنفذ.

مثـــال:

```
// Message-based port which allows types MsgType1 and MsgType2 to be received at, MsgType3 to be
// sent via and any integer value to be send and received over the port
type port MyMessagePortType message
{
    in   MsgType1, MsgType2;
    out  MsgType3;
    inout    integer
}

// Procedure-based port which allows the remote call of the procedures Proc1, Proc2 and Proc3.
// Note that Proc1, Proc2 and Proc3 are defined as signatures
type port MyProcedurePortType procedure
{
    out  Proc1, Proc2, Proc3
}
```

ملاحظة – يستخدم المصطلح "رسالة" ليعني كلا الرسالتين كما عُرِّفنا بواسطة مقاسات وقيم فعلية ناتجة من تعبيرات. ومن ثم، تكون القائمة التي يمكن أن تستخدم على منفذ قائم على رسالة هي ببساطة قائمة أسماء نمط.

1.4.8 منافذ مختلطة

من الممكن تعريف منفذ على أنه يسمح لكلا النوعين من الاتصالات. ويدل على هذا الكلمة المفتاحية **mixed**. ويعني هذا أن القوائم لمنافذ مختلطة تكون أيضاً مختلطة وتشمل توقيعات وأنماط. ولا يجري فصل في التعريف.

```
// Mixed port, defining a message-based and a procedure-based port with the same name. The in,
// out and inout lists are also mixed: MsgType1, MsgType2, MsgType3 and integer refer to the
// message-based part of the mixed port and Proc1, Proc2, Proc3, Proc4 and Proc5 refer to the
// procedure-based port.
type port MyMixedPortType mixed
{
  in  MsgType1, MsgType2, Proc1, Proc2;
  out MsgType3, Proc3, Proc4;
  inout   integer, Proc5;
}
```

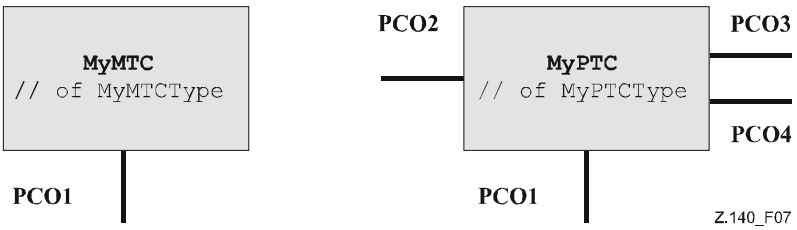
يعرف منفذ مختلط في TTCN-3 كترميز مختزل لمنفذين، أي، منفذ قائم على رسالة ومنفذ قائم على إجراء مع نفس الاسم. وعند وقت التنفيذ، يكون التمييز بين منفذين هو بواسطة عمليات اتصالات.

تؤدي العمليات المستخدمة للتحكم في منافذ (انظر 5.23)، أي، **start** و**stop** و**clear** العملية على كل من صفوف انتظار (بترتيب اعتباطي) إذا طلبت مع معرف منفذ مختلط.

5.8 تعريف أنماط مكون

0.5.8 عـــام

يعرف نمط **component** أي منافذ متصاحبة مع مكون. وتتم هذه التعاريف في جزء تعاريف الوحدة. وتكون أسماء منفذ في تعريف مكون محلية لذلك المكون؛ أي، يمكن أن يكون لمكون آخر منافذ مع نفس الأسماء. ويكون لمنافذ نفس المكون أسماء وحيدة. ولا يعني تعريف مكون بمفرده أن هناك أي توصيل بين المكونات عبر هذه المنافذ.



الشكل Z.140/7 - المكونات النمطية

مثـــال:

```
type component MyMTCType
{
  port MyMessagePortType   PCO1
}

type component MyPTCType
{
  port MyMessagePortType     PCO1, PCO4;
  port MyProcedurePortType PCO2;
  port MyAllMesssagesPortTypePCO3
}
```

1.5.8 إعلان متغيرات وثوابت ومؤقتات محلية في مكون

من الممكن إعلان ثوابت ومتغيرات ومؤقتات محلية في مكون معين.

مثـــال:

```
type component MyMTCType
{
  var integer MyLocalInteger;
  timer MyLocalTimer;
  port MyMessagePortType   PCO1
}
```

تكون هذه الإعلانات مرئية لجميع الاختبارات المجردة وaltsteps التي تنفذ على المكون. ويرد هذا بوضوح باستخدام الكلمة المفتاحية **runs on** (انظر القسم 16).

تتصاحب متغيرات ومؤقتات مكون مع مطابقة مكون وتتبع قواعد المنظور المعرفة في 3.5. ومن ثم، يكون لكل مطابقة جديدة لمكون مجموعتها من المتغيرات والمؤقتات المحددة في تعريف المكون (بما في ذلك أي قيمة أولية، إذا ذكرت).

ملاحظة - عندما تستخدم كأسطح بينية لنظام اختبار (انظر 8.8)، لا يمكن أن تستفيد المكونات من أي ثوابت ومتغيرات ومؤقتات معلنة في المكون.

2.5.8 تعريف مكونات مع مصفوفات منافذ

من الممكن تعريف مصفوفات منافذ في تعاريف نمط مكون (انظر أيضاً 21.22).

```
type component My3pcoCompType
      {
        port MyMessageInterfaceType PCO[3]
        port MyProcedureInterfaceType PCOm[3][3]
        // Defines a component type which has an array of 3 message ports and a two-dimensional
        // array of 9 procedure ports.
       }
```

3.5.8 تمديد أنماط مكون

من الممكن تعريف أنماط مكون كتمديد لأنماط مكون أخرى، باستخدام الكلمة المفتاحية **extends**.

المثـــال 1:

```
type component MyExtendedMTCType extends MyMTCType
{
  var float MyLocalFloat;
  timer MyOtherLocalTimer;
  port MyMessagePortType PCO2;
}
```

وفي مثل هذا التعريف، يشار إلى تعريف نمط جديد بالنمط المُمدد ويشار إلى تعريف النمط التالي للكلمة المفتاحية **extends** بنمط أساسي.

وتأثير هذا التعريف هو أن النمط الممدد يحتوي ضمنياً أيضاً على جميع التعاريف من النمط الرئيسي. ولهذا، فإن التعريف أعلاه هو مكافئ للكتابة (ومن ثم يسمى *تعريف نمط فعال*):

المثـــال 2:

```
// effectively, the definition from Example 1 is equivalent to this one:
type component MyExtendedMTCType
{
  /* the definitions from MyMTCType */
  var integer MyLocalInteger;
  timer MyLocalTimer;
  port MyMessagePortType PCO1

  /* the additional definitions */
  var float MyLocalFloat;
  timer MyOtherLocalTimer;
  port MyMessagePortType PCO2;
}
```

يسمح بتمديد أنماط مكون معرفة بواسطة تمديد، طالما لا تخلق سلسلة دورية من التعريف.

المثـــال 3:

```
type component MTCTypeA extends MTCTypeB { /* … */ };
type component MTCTypeB extends MTCTypeC { /* … */ };
type component MTCTypeC extends MTCTypeA { /* … */ };  // ERROR - cyclic extension
type component MTCTypeD extends MTCTypeD { /* … */ };  // ERROR - cyclic extension
```

عند تعريف أنماط مكون بواسطة تمديد، لا يوجد تناقض اسم بين التعاريف التي تؤخذ من النمط الرئيسي والتعاريف التي تضاف إلى نمط ممدد، أي لا يوجد معرف منفذ أو متغير أو ثابت أو مؤقت أو مقاس معلن عنهما في النمط الرئيسي (بطريقة مباشرة أو غير مباشرة بواسطة تمديد) والنمط الممدد.

المثـــال 4:

```
type component MyExtendedMTCType extends MyMTCType
{
  var integer MyLocalInteger; // ERROR - already defined in MyMTCType (see example 2)
  var float MyLocalTimer;     // ERROR - timer with that name exists in MyMTCType
  port MyOtherMessagePortType PCO1; // ERROR - port with that name exists in MyMTCType
}

type component MyBaseComponent { timer MyLocalTimer };
type component MyInterimComponent extends MyBaseComponent { timer MyOtherTimer };
type component MyExtendedComponent extends MyInterimComponent
{
  timer MyLocalTimer; // ERROR - already defined in MyInterimComponent via extension
}
```

من المسموح وجود نمط مكون واحد يمتد إلى أنماط رئيسية عديدة في تعريف واحد، يتعين أن تحدد باعتبارها قائمة منفصلة بفصله لأنماط في التعريف.

المثـــال 5:

```
type component MyCompA extends MyCompB, MyCompC, MyCompD {
  /* additional definitions for MyCompA */
}
```

ويمكن أيضاً تعريف أي نمط رئيسي بواسطة تمديد.

يجري الحصول على تعريف نمط مكون فعال لنمط ممدد كتجميع لجميع تعاريف الثابت والمتغير والمؤقت والمنفذ والمقاس بواسطة الأنماط الرئيسية (محددة بشكل متكرر إذا كان نمط رئيسي معرف أيضاً بواسطة تمديد) والتعاريف معلن عنها في نمط ممدد مباشرة. ويكون تعريف نمط مكون فعال خال من تناقض اسم. ولتلبية هذا الشرط، في مجموعة أنماط رئيسية في تعريف نمط ممدد، يكون لجميع التعاريف أسماء وحيدة وأسماء تختلف عن أي أسماء تعاريف معلن عنها مباشرة في النمط الممدد.

الملاحظة 1 – لا يعتبر إعلان مختلف، ومن ثم لا يسبب خطأ، إذا ساهم نفس التعريف في النمط الممدد بواسطة أنماط رئيسية مختلفة (عبر مسيرات تمديد مختلفة).

المثـــال 6:

```
type component MyCompB { timer T };
type component MyCompC { var integer T };
type component MyCompD extends MyCompB, MyCompC {}
  // ERROR - name clash between MyCompB and MyCompC

// MyCompB is defined above
type component MyCompE extends MyCompB {
  var integer MyVar1 := 10;
}

type component MyCompF extends MyCompB {
  var float MyVar2 := 1.0;
}

type component MyCompG extends MyCompB, MyCompE, MyCompF {
  // No name clash.
  // All three parent types of MyCompG have a timer T, either directly or via extension of
  // MyCompB; as all these stem (directly or via extension) from timer T declared in MyCompB,
  // which make this form of collision legal.
  /* additional definitions here */
}
```

يُعرِّف علم دلالات أنماط مكون مع تمديدات بواسطة استبدال كل تعريف نمط مكون بواسطة تعريف نمط مكون فعال حسب خطوة ما قبل معالجتها قبل استخدامه.

الملاحظة 2 – لمواءمة نمط مكون، يعني هذا أن مرجع مكون c لنمط CT1، الذي يمدد CT2 متوائم مع CT2، والاختبارات المجردة والوظائف وaltsteps المحددة CT2 في clauses **runs on** يمكن تنفيذها على c (انظر 3.7.6).

6.8 عنونة كيانات داخل SUT

يمكن أن يتألف SUT من كيانات عديدة يتعين عنونتها بشكل منفرد. ونمط معطيات عنوان هو نمط للاستخدام مع عمليات منفذ لعنونة كيانات SUT. وعندما يستخدم مع **to** و**from** و**sender**، يستخدم نمط معطيات العنوان فقط في عمليات استقبال وإرسال منافذ متقابلة مع سطح بيني لنظام اختبار. ويستبان تمثيل معطيات فعلية لـ **address** سواء بواسطة تعريف نمط واضح في متوالية اختبار أو خارجياً بواسطة نظام اختبار (أي يترك نمط **address** لنمط مفتوح في مواصفة TTCN-3). ويسمح هذا لاختبارات مجردة بتحديدها مستقلة عن أي آلية عنوان حقيقي محدد لـ SUT.

وتولد عناوين SUT الواضحة فقط داخل وحدة TTCN-3 إذا كان النمط معرفاً داخل الوحدة. وإذا لم يكن النمط معرفاً داخل الوحدة، تمرر عناوين SUT الواضحة فقط في معلمات أو تستقبل في مجالات رسالة أو كمعلمات لنداءات إجراء بعيدة.

وبالإضافة إلى ذلك، تتاح القيمة الخاصة **null** لتدل على عنوان غير معرف، مثلاً، لتدميث متغيرات لنمط عنوان.

مثـــال:

```
// Associates the type integer to the open type address
type integer address;
 :
// new address variable initialized with null
var address MySUTentity := null;
 :
// receiving an address value and assigning it to variable MySUTentity
PCO.receive(address:*) -> value MySUTentity;
 :
// usage of the received address for sending template MyResult
PCO.send(MyResult) to MySUTentity;
 :
// usage of the received address for receiving a confirmation template
PCO.receive(MyConfirmation) from MySUTentity;
```

7.8 مراجع مكون

إن مراجع مكون هي مراجع وحيدة لمكونات اختبار خلقت خلال تنفيذ اختبار مجرد. ويخلق مرجع المكون الوحيد هذا نظام اختبار عند وقت خلق مكون، أي، يكون مرجع مكون هو نتيجة عملية **create** (انظر 1.22). وبالإضافة إلى ذلك، تعاد مراجع مكون بواسطة العمليات المعرفة مسبقاً **system** (يعيد مرجع مكون لتعريف منافذ سطح بيني نظام الاختبار) و**mtc** (يعيد مرجع مكون MTC) و**self** (يعيد مرجع مكون لمكون يطلب فيه **self**).

تستخدم مراجع مكون في تشكيل عمليات **connect**، **map** و**start** (انظر القسم 22) لإقامة تشكيل اختبار وفي أجزاء **from** و**to** و**sender** لعمليات اتصالات لمنافذ موصلة بمكونات اختبار أخرى غير نظام اختبار **interface** لأغراض العنونة (انظر القسم 23 والشكل 5).

وبالإضافة إلى ذلك، تتاح القيمة الخاصة **null** لتدل على مرجع مكوّن غير معرف، مثلاً، لتدميث متغيرات لمناولة مراجع مكون.

يُستبان تمثيل معطيات فعلية لمراجع مكون خارجياً بواسطة نظام اختبار. ويسمح هذا لاختبارات مجردة أن تحدد بشكل مستقل عن أي بيئة وقت تنفيذ TTCN-3 حقيقي؛ وبمعنى آخر، لا يقصر TTCN-3 التنفيذ على نظام الاختبار فيما يتعلق بمناولة وتعرف مكونات اختبار.

ملاحظة – يشمل مرجع مكون معلومات نمط مكون. ويعني هذا، مثلاً، أن متغير لمناولة مراجع مكون يجب أن تستخدم اسم نمط مكون متناظر في إعلانه.

مثـــال:

```
// A component type definition
type component MyCompType {
  port PortTypeOne PCO1;
  port PortTypeTwo PCO2
}

// Declaring one variable for the handling of references to components of type MyCompType
// and creating a component of this type
var MyCompType MyCompInst := MyCompType.create;

// Usage of component references in configuration operations
// always referring to the component created above
connect(self:MyPCO1, MyCompInst:PCO1);
map(MyCompInst:PCO2, system:ExtPCO1);
MyCompInst.start(MyBehavior(self)); // self is passed as a parameter to MyBehavior

// Usage of component references in from- and to- clauses
MyPCO1.receive from MyCompInst;
 :
MyPCO2.receive(integer:?) -> sender MyCompInst;
 :
MyPCO1.receive(MyTemplate) from MyCompInst;
 :
MPCO2.send(integer:5) to MyCompInst;

// The following example explains the case of a one-to-many connection at a Port PCO1
// where values of type M1 can be received from several components of the different types
// CompType1, CompType2 and CompType3 and where the sender has to be retrieved.
// In this case the following scheme may be used:
 :
var M1 MyMessage, MyResult;
var MyCompType1 MyInst1 := null;
var MyCompType2 MyInst2 := null;
var MyCompType3 MyInst3 := null;
 :
alt {
   [] PCO1.receive(M1:?) from MyInst1 -> value MyMessage sender MyInst1 {}
   [] PCO1.receive(M1:?) from MyInst2 -> value MyMessage sender MyInst2 {}
   [] PCO1.receive(M1:?) from MyInst3 -> value MyMessage sender MyInst3 {}
}
 :
MyResult := MyMessageHandling(MyMessage); // some result is retrieved from a function
 :
if (MyInst1 != null) {PCO1.send(MyResult) to MyInst1};
if (MyInst2 != null) {PCO1.send(MyResult) to MyInst2};
if (MyInst3 != null) {PCO1.send(MyResult) to MyInst3};
 :
```

8.8 تعريف السطح البيني لنظام اختبار

يستخدم تعريف نمط مكون لتعريف السطح البيني لنظام اختبار بسبب أن، مفهومياً، تعاريف نمط مكون وتعاريف سطح بيني لنظام اختبار لها نفس الشكل (كلاهما تجميع لمنافذ تعرف نقاط توصيل ممكنة).

ملاحظة – لن يكون لمتغيرات ومؤقتات وثوابت معلن عنها في أنماط مكون، تستخدم كسطوح بينية لنظام اختبار، أي تأثير.

```
type component MyISDNTestSystemInterface
      {
        port MyBchannelInterfaceType     B1;
        port MyBchannelInterfaceType     B2;
        port MyDchannelInterfaceType     D1
      }
```

عامةً، يتصاحب مرجع نمط مكون يعرف السطح البيني لنظام اختبار مع كل اختبار مجرد باستخدام أكثر من مكون اختبار واحد. تستطبق أوتوماتياً منافذ السطح البيني لنظام اختبار بواسطة نظام مع MTC عندما يبدأ تنفيذ الاختبار المجرد.

وتكون العملية التي تنفذ مرجع مكون لسطح بيني لنظام اختبار هي **system**. وتستخدم هذه لتناول منافذ نظام اختبار.

مثـــال:

```
map(MyMTCComponent:Port2, system:PCO1);
```

وفي حالة كون MTC المكون الوحيد الذي يُستطبق خلال تنفيذ اختبار، لا يحتاج سطح بيني لنظام اختبار أن يتصاحب مع اختبار مجرد. وفي هذه الحالة، يعرف بوضوح تعريف نمط مكون متصاحب مع MTC السطح البيني لنظام اختبار متناظر.

9 إعلان عن ثوابت

يمكن الإعلان عن ثوابت واستخدامها في جزء تعاريف وحدة وتعاريف نمط مكون وجزء تحكم الوحدة واختبارات مجردة ووظائف وaltsteps. ويدل على تعاريف ثابت الكلمة المفتاحية **const**. ولا تكون الثوابت نمط منفذ. وتخصص قيمة الثابت عند نقطة الإعلان.

ملاحظة – تكون القيمة الوحيدة التي يمكن تخصيصها لثابت بالتغيب وأنماط مكون هي القيمة الخاصة **null**.

المثـــال 1:

```
const integer MyConst1 := 1;
const boolean MyConst2 := true, MyConst3 := false;
```

ويمكن أن يتم تخصيص القيمة لثابت في الوحدة أو يمكن أن تتم خارجياً. والحالة الأخيرة هي إعلان ثابت خارجي تدل عليه الكلمة المفتاحية **external**.

المثـــال 2:

```
external const integer MyExternalConst; // external constant declaration
```

يمكن أن يكون لثابت خارجي نمط اعتباطي، باستثناء نمط منفذ أو نمط بالتغيب أو نمط مكون، ويتعين أن يكون النمط معروفاً في الوحدة، أي يكون نمط جذر أو نمط معرف لمستعمل معرف في الوحدة أو مستورد من وحدة أخرى. وتقابل النمط مع تمثيل خارجي لثابت خارجي وآلية كيفية تمرير ثابت خارجي في الوحدة هما خارج مدى هذه التوصية.

10 إعلان عن متغيرات

0.10 عام

يمكن أن تكون المتغيرات أنماطاً بسيطة أساسية وأنماط سلسلة أساسية وأنماطاً مبنية وأنماط معطيات خاصة (بما في ذلك أنماط فرعية مشتقة من هذه الأنماط) وكذلك أنماط عنوان أو مكون أو بالتغيب.

ملاحظة – يمكن الإعلان عن متغيرات نمط مبني ونمط مكون على أساس أنماط معرفة لمستعمل فقط.

يمكن الإعلان عن متغيرات واستخدامها في جزء تحكم وحدة واختبارات مجردة ووظائف وaltsteps. وبالإضافة إلى ذلك، يمكن الإعلان عن متغيرات في تعريف نمط مكون. ويمكن أن تستخدم هذه المتغيرات في اختبارات مجردة وaltsteps ووظائف تنفذ على نمط مكون ما. ولا يعلن عن متغيرات أو تستخدم في جزء تعاريف وحدة (أي، متغيرات عالمية لا تدعم في TTCN-3).

يسبب استخدام متغيرات غير مدمثة أو غير كاملة التدميث في أماكن أخرى غير الجانب الأيسر من تخصيصات أو معلمات فعلية تمرر إلى معلمات رسمية خارجية خطأ.

1.10 متغيرات قيمة

يعلن عن متغير قيمة بواسطة الكلمة المفتاحية **var** يتبعه معرف نمط ومعرف متغير. ويمكن تخصيص قيمة أولية عند الإعلان. وتخزن متغيرات قيمة قيم فقط، ويمكن أن تستخدم في الجانب الأيمن وكذلك الجانب الأيسر من تخصيصات وفي تعبير تتبعها الكلمة المفتاحية **return** في أجسام وظائف مع شرط إعادة في رأسيّتها، ويمكن أن تمرر إلى كل من قيمة ونمط–مقاس معلمات رسمية.

مثـــال:

```
var integer MyVar0;
var integer MyVar1 := 1;
var boolean MyVar2 := true, MyVar3 := false;
```

2.10 متغيرات مقاس

يعلَن عن متغيرات مقاس بواسطة الكلمة المفتاحية **var template** يتبعها معرف نمط ومعرف متغير. ويمكن تخصيص محتوى أولي عند الإعلان. وعند الإفراط في القيم، يمكن أن تخزن متغيرات مقاس أيضاً آليات مواءمة (انظر 3.14). ويمكن أن تستخدم على الجانب الأيمن وكذلك على الجانب الأيسر من تخصيصات، تتبعها الكلمة المفتاحية **return** في أجسام وظائف تعرف قيمة عودة نمط–مقاس في رأسيتها، ويمكن أن تمرر كمعلمات فعلية لمعلمات رسمية لنمط–مقاس. وعندما تستخدم على الجانب الأيمن من تخصيصات، لا تتأثر بمشغلي TTCN-3 (انظر القسم 15) ويكون المتغير على الجانب الأيسر متغير مقاس أيضاً. ويسمح أيضاً بتخصيص مطابق مقاس لمتغير مقاس أو مجال متغير مقاس.

ملاحظة – تكون متغيرات مقاس، المشابهة لمقاسات عالمية ومحلية، محددة بالكامل لتستخدم في إرسال واستقبال عمليات.

مثـــال:

```
template MyRecord MyTempl ( template boolean par_bool ) :=
  { field1 := par_bool, field2 := * }
:
function Myfunc () return template MyRecord {
  var template integer MyVarTemp1 := ?;
  var template MyRecord MyVarTemp2 := { field1 := true, field2 := * },
               MyVarTemp3 := { field1 := ?, field2 := MyVarTemp1 };
  MyVarTemp2 := MyTempl (?);
:
  return MyVarTemp2
}
```

بينما من غير المسموح أن تطبق عمليات TTCN-3 مباشرة على متغيرات مقاس، يسمح باستخدام ترميز بالنقط وترميز دليل للتفتيش على مجالات متغير مقاس وتعديلها. وترد في 1.3.14 قواعد تطبق عندما تحاول الترميزات هذه الوصول إلى مجالات خارج آليات مواءمة.

11 الإعلان عن مؤقتات

0.11 عـــام

يمكن الإعلان عن المؤقتات واستخدامها في جزء تحكم الوحدة واختبارات مجردة ووظائف وaltsteps. وبالإضافة إلى ذلك، يمكن الإعلان عن مؤقتات في تعريف نمط مكون. ويمكن استخدام هذه المؤقتات في اختبارات مجردة ووظائف وaltsteps التي تنفذ على نمط مكون ما. ويمكن أن يكون لإعلان مؤقت قيمة مدة خيارية بالتغيب مخصصة له. ويبدأ المؤقت مع هذه القيمة إذا لم تحدد قيمة أخرى. وتكون هذه القيمة قيمة **float** غير سالبة (أي، أكبر من أو مساوية لـ 0.0) حيث وحدة القاعدة هي ثواني.

المثـــال 1:

```
timer MyTimer1 := 5E-3;// declaration of the timer MyTimer1 with the default value of 5 ms

timer MyTimer2;   // declaration of MyTimer2 without a default timer value i.e. a value has
                  //  to be assigned when the timer is started
```

وبالإضافة إلى مطابقات مؤقت وحيد، يمكن أيضاً الإعلان عن مصفوفات مؤقت. وتخصص المدد بالتغيب لعناصر صفيف مؤقت باستخدام صفيف قيمة. ويستخدم تخصيص مدد بالتغيب ترميز قيمة صفيف كما حُدد في 5.6. وإذا كانت هناك رغبة في تخطي تخصيص مدة بالتغيب لبعض العناصر لصفيف مؤقت، يعلن عنه صراحة باستخدام الرمز ("-").

المثـــال 2:

```
timer t_Mytimer1[5] := { 1.0, 2.0, 3.0, 4.0, 5.0 }
      // all elements of the timer array get a default duration.

timer t_Mytimer2[5] := { 1.0, -, 3.0, 4.0, 5.0 }
  // the second timer (t_Mytimer2[1]) is left without a default duration.
```

1.11 مؤقتات كمعلمات

يمكن أن تمرر مؤقتات فقط بواسطة مرجع إلى وظائف وaltsteps. وتعرف المؤقتات التي تمرر إلى وظيفة أو altstep داخل تعريف السلوك للوظيفة أو altsteps.

يمكن استخدام مؤقتات تمرر كمعلمات بواسطة مرجع مثل أي مؤقت آخر، أي لا تحتاج إلى إعلان. ويمكن أيضاً تمرير مؤقت بادئ إلى وظيفة أو altstep. ويستمر المؤقت في التنفيذ، أي لا يكون متوقفاً ضمنياً. وبالتالي، يمكن مناولة أحداث إمهال داخل وظيفة أو altstep يمرر خلالها المؤقت.

مثـــال:

```
// Function definition with a timer in the formal parameter list
function MyBehaviour (timer MyTimer)
{  :
   MyTimer.start;
   :
  }
```

12 إعلان رسائل

إن أحد العناصر الرئيسية لـ TTCN-3 هو القدرة على إرسال واستقبال رسائل معقدة عبر منافذ اتصالات معرفة بواسطة تشكيل اختبار. ويمكن أن تكون هذه الرسائل هي المعنية بوضوح باختبار SUT أو مع التنسيق الداخلي ورسائل تحكم محددة لتشكيل اختبار معين.

ملاحظة – في TTCN-2، تكون هذه الرسائل بدائية خدمة مجردة (ASP) ووحدة معطيات البروتوكول (PDUs) ورسائل تنسيق. ولغة النواة لـ TTCN-3 هي تنوعية، بمعنى أنها لا تقوم بتمييزات تركيبية أو دلالية من هذا النوع.

13 إعلان توقيعات إجراء

0.13 عـــام

هناك حاجة إلى توقيعات إجراء (أو توقيعات فقط) لاتصالات قائمة على إجراء. ويمكن أن تستخدم اتصالات قائمة على إجراء لاتصالات في نظام اختبار، أي فيما بين مكونات اختبار، أو لاتصالات بين نظام اختبار وSUT. وفي الحالة الأخيرة، يكون الإجراء إما أن ينفذ في SUT (أي يؤدي نظام الاختبار النداء) أو ينفذ في نظام الاختبار (أي، يؤدي SUT النداء). ولجميع الإجراءات المستخدمة، أي أن الإجراءات المستخدمة لاتصالات فيما بين مكونات اختبار وإجراءات مطلوبة من SUT وإجراءات مطلوبة من نظام الاختبار وإجراء مكتمل **signature** تُعرّف في وحدة TTCN-3.

1.13 توقيعات لاتصالات Blocking وNon-blocking

يدعم TTCN-3 *blocking* و*none-blocking* لاتصالات قائمة على إجراء. وتستخدم تعاريف توقيعات اتصالات non-blocking الكلمة المفتاحية **noblock**، ويكون لها معلمات **in** فقط (انظر 2.13) ولا يكون لها قيمة عودة (انظر 3.13)، ولكن يمكن أن تثير استثناءات (انظر 4.13). وبالتغيب، تفترض تعاريف توقيع دون الكلمة المفتاحية **noblock** أن تستخدم لسد اتصالات قائمة على إجراء.

مثـــال:

```
signature MyRemoteProcOne ();       // MyRemoteProcOne will be used for blocking
                                    // procedure-based communication. It has neither
                                    // parameters nor a return value.

signature MyRemoteProcTwo () noblock;   // MyRemoteProcTwo will be used for non blocking
                                        // procedure-based communication. It has neither
                                        // parameters nor a return value.
```

2.13 معلمات لتوقيعات إجراء

يمكن أن يكون لتعاريف توقيع معلمات. وفي تعريف **signature** يمكن أن تشمل قائمة معلمات مُعرفات معلمة وأنماط معلمة واتجاهها، أي **in** أو **out** أو **inout**. ويدل اتجاه **inout** و**out** على أن هذه المعلمات تستخدم لاسترداد معلومات من إجراء بعيد. ولاحظ أن توجيه معلمات يُري على أنه الطرف *المطلوب* بدلاً من الطرف *الطالب*.

مثـــال:

```
signature MyRemoteProcThree (in integer Par1, out float Par2, inout integer Par3);
// MyRemoteProcThree will be used for blocking procedure-based communication. The procedure
// has three parameters: Par1 an in parameter of type integer, Par2 an out parameter of
// type float and Par3 an inout parameter of type integer.
```

3.13 القيمة التي تعيد إجراءات بعيدة

يمكن أن يعيد إجراء بعيد قيمة بعد انتهائها. ويحدد نمط قيمة العودة بواسطة قسم **return** في تعريف توقيع متناظر.

مثـــال:

```
signature MyRemoteProcFour (in integer Par1) return integer;
// MyRemoteProcFour will be used for blocking procedure-based communication. The procedure
// has the in parameter Par1 of type integer and returns a value of type integer after its
// termination
```

4.13 تحديد استثناءات

يمكن أن تثار استثناءات بواسطة إجراءات بعيدة تمثل في TTCN-3 كقيم لنمط محدد. ولهذا يمكن استخدام مقاسات وآليات مواءمة لتحديد أو التأكد من قيم عودة لإجراءات بعيدة.

ملاحظة – إن تحويل استثناءات ولدها أو أرسلت إلى SUT في نمط TTCN-3 متناظر أو تمثيل SUT هو أداة–نظام محدد ولهذا، يكون خارج مدى هذه التوصية.

تُعرّف الاستثناءات في شكل قائمة استثناءات واردة في تعريف **signature**. وتُعرّف هذه القائمة جميع الأنماط المختلفة الممكنة المتصاحبة مع مجموعة استثناءات ممكنة (معاني الاستثناءات نفسها تتميز عادة فقط بواسطة قيم محددة لهذه الأنماط).

مثـــال:

```
signature MyRemoteProcFive (inout float Par1) return integer
      exception (ExceptionType1, ExceptionType2);
// MyRemoteProcFive will be used for blocking procedure-based communication. It returns a
// float value in the inout parameter Par1 and an integer value, or may raise exceptions of
// type ExceptionType1 or ExceptionType2

signature MyRemoteProcSix (in integer Par1) noblock
      exception (integer, float);
// MyRemoteProcSix will be used for non-blocking procedure-based communication. In case of
// an unsuccessful termination, MyRemoteProcSix raises exceptions of type integer or float.
```

14 إعلان مقاسات

0.14 عـــام

تستخدم مقاسات إما لإرسال مجموعة من قيم مميزة أو اختبار ما إذا كانت مجموعة القيم المستقبَلة تتواءم مع مواصفة مقاس. ويمكن تعريف مقاسات عالمياً في تعاريف وحدة أو محلياً في اختبار مجرد أو altstep أو وظيفة أو statement أو فدرة بيان أو داخل الخط كمتغيرات لعملية اتصالات أو معلمة فعلية لاختبار مجرد أو وظيفة أو نداء altstep.

توفر مقاسات الإمكانيات التالية:

أ) إنها طريقة لتنظيم وإعادة استخدام معطيات اختبار، بما في ذلك شكل بسيط من التلازم؛

ب) يمكن معلمتها؛

ج) تسمح بآليات مواءمة؛

د) يمكن استخدامها إما مع اتصالات قائمة على رسالة أو قائمة على إجراء.

في قيم مقاس وأمدية ونعوت مواءمة يمكن أن تحدد ثم تستخدم في اتصالات قائمة على رسالة أو قائمة على إجراء. ويمكن تحديد مقاسات لأي نمط TTCN-3 أو توقيع إجراء. وتستخدم مقاسات قائمة على نمط لاتصالات قائمة على رسالة وتستخدم مقاسات توقيع في اتصالات قائمة على إجراء.

يجب أن يحدد إعلان مقاس مجموعة قيم قاعدة أو رموز مواءمة لكل مجال معرف في نمط ملائم أو تعريف توقيع، أي محدد بالكامل. ويحدد إعلان مقاس معدل (انظر 6.16) فقط المجالات التي تغير من مقاس قاعدة، أي يكون مواصفة جزئية. ويستخدم NotUsedSymbol فقط في مقاسات توقيع لمعلمات ليس لها علاقة وفي إعلانات مقاس معدل ومقاسات في الخط معدلة لتدل على عدم وجود تغيير لمجال أو لعنصر محدد.

يوجد عدد من التقييدات على الوظائف المستخدمة في تعبيرات عند تحديد مقاسات أو مجالات مقاس؛ وتحدد هذه في 4.1.16.

1.14 إعلان مقاسات رسالة

0.1.14 عـــام

يمكن تحديد مطابقات رسائل مع قيم فعلية باستخدام مقاسات. ويمكن اعتبار مقاس على أنه مجموعة من التعليمات لبناء رسالة لإرسال أو مواءمة رسالة مستقبَلة.

يمكن تحديد مقاسات لأي نمط TTCN-3 معرف في الجدول 3 باستثناء أنماط **port** و**default**.

مثـــال:

```
// When used in a receiving operation this template will match any integer value
template integer Mytemplate := ?;
// This template will match only the integer values 1, 2 or 3
template integer Mytemplate := (1, 2, 3);
```

1.1.14 مقاسات لإرسال رسائل

إن مقاس يستخدم في عملية **send** يعرّف مجموعة كاملة من قيم مجال تتألف من رسائل ترسل عبر منفذ اختبار. وفي وقت عملية **send**، يعرّف المقاس بالكامل، أي، يتم استبانة جميع المجالات للقيم الفعلية ولا تستخدم آليات مواءمة في مجالات المقاس، بطريقة مباشرة أو غير مباشرة.

ملاحظة – لإرسال مقاسات، يعتبر حذف مجال عملية على أنه ترميز قيمة بدلاً من آلية مواءمة.

مثـــال:

```
// Given the message definition
type record MyMessageType
{
  integer field1 optional,
  charstring  field2,
  boolean field3
}

// a message template could be
template MyMessageType MyTemplate:=
{
  field1 := omit,
  field2 := "My string",
  field3 := true
}

// and a corresponding send operation could be
MyPCO.send(MyTemplate);
```

2.1.14 مقاسات لاستقبال رسائل

إن مقاس يستخدم في عملية **receive** أو **trigger** أو **check** يعرّف مقاس معطيات مقابل رسالة واصلة لتتم مواءمتها. ويمكن استخدام آليات مواءمة، كما عرّفت في الملحق B، في مقاسات مستقبَلة. ولا يحدث إسناد لقيم واصلة لمقاس.

مثـــال:

```
// Given the message definition
type record MyMessageType
{
  integer field1 optional,
  charstring  field2,
  boolean  field3
}

// a message template might be
template MyMessageType MyTemplate:=
{
  field1 := ?,
  field2 := pattern "abc*xyz",
  field3 := true
}

// and a corresponding receive operation could be
MyPCO.receive(MyTemplate);
```

2.14 إعلان مقاسات توقيع

0.2.14 عـــام

إن مطابقات قوائم معلمات إجراء مع قيم فعلية يمكن تحديدها باستخدام مقاسات. وتعرّف مقاسات لأي إجراء بواسطة الإشارة إلى تعريف توقيع متصاحب. ويعرّف مقاس توقيع القيم وآليات المواءمة لمعلمات إجراء فقط، وليس لقيمة عودة. ويتعين تعريف القيم أو آليات المواءمة لعودة في عملية **replay** أو **getreply** (انظر 3.3.23 و4.3.23 على التوالي).

مثـــال:

```
// signature definition for a remote procedure
signature RemoteProc(in integer Par1, out integer Par2, inout integer Par3) return integer;

// example templates associated to defined procedure signature
template RemoteProc Template1:=
{
  Par1 := 1,
  Par2 := 2,
  Par3 := 3
}

template RemoteProc Template2:=
{
  Par1 := 1,
  Par2 := ?,
  Par3 := 3
}

template RemoteProc Template3:=
{
  Par1 := 1,
  Par2 := ?,
  Par3 := ?
}
```

1.2.14 مقاسات لإجراءات تنفيذ

إن مقاساً مستخدماً في عملية **call** أو **replay** يعرّف مجموعة كاملة لقيم مجال لجميع معلمات **in** و**inout**. وفي وقت عملية **call**، يتم استبانة المعلمات في المقاس للقيم الفعلية **in** و**inout**؛ ولا تستخدم آليات مواءمة في هذه المجالات، بطريقة مباشرة أو غير مباشرة. ويجري تجاهل أي مواصفة مقاس لمعلمات **out**؛ ولهذا، يُسمح بتحديد آليات مواءمة لهذه المجالات أو حذفها (انظر الملحق B).

مثـــال:

```
// Given the examples in clause 14.2.0
// Valid invocation since all in and inout parameters have a distinct value
MyPCO.call(RemoteProc:Template1);

// Valid invocation since all in and inout parameters have a distinct value
MyPCO.call(RemoteProc:Template2);

// Invalid invocation because the inout parameter Par3 has a matching attribute not a value
MyPCO.call(RemoteProc:Template3);

// Templates never return values. In the case of Par2 and Par3 the values returned by the
// call operation must be retrieved using an assignment clause at the end of the call statement
```

2.2.14 مقاسات لتنفيذ إجراء قبول

إن مقاساً يستخدم في عملية **getcall** يعرّف مقاس معطيات مقابل رسالة واصلة لتتم مواءمتها. ويمكن استخدام آليات مواءمة، كما عُرّفت في الملحق B، في مقاسات مستقبَلة. ولا يحدث إسناد لقيم واصلة لمقاس. ويتم تجاهل أي معلمات **out** في عملية المواءمة.

مثـــال:

```
// Given the examples in clause 14.2.0
// Valid getcall, it will match if Par1 == 1 and Par3 == 3
MyPCO.getcall(RemoteProc:Template1);

// Valid getcall, it will match if Par1 == 1 and Par3 == 3
MyPCO.getcall(RemoteProc:Template2);

// Valid getcall, it will match on Par1 == 1 and Any value of Par3
MyPCO.getcall(RemoteProc:Template3);
```

3.14 آليات مواءمة مقاس

0.3.14 عـــام

عامةً، تُستخدم آليات مواءمة لتحل محل قيم مجالات مقاس وحيد أو لتحل محل المحتويات الكاملة لمقاس. ويمكن استخدام بعض الآليات في مركب.

يمكن أيضاً استخدام آليات مواءمة في الخط في الأحداث المستقبلة فقط (أي، عمليات **receive** و**trigger** و**getcall** و**getreplay** و**catch**). ويمكن أن تظهر في قيم واضحة.

المثـــال 1:

```
MyPCO.receive(charstring:"abcxyz");
MyPCO.receive (integer:complement(1, 2, 3));
```

ويمكن حذف مُعرف النمط عندما تُعرَّف القيمة غير الغامضة النمط.

المثـــال 2:

```
MyPCO.receive("AAAA"O);
```

ملاحظة – يمكن حذف الأنماط التالية: صحيح وطليق وبولاني وسلسلة ثنائية وسلسلة ستة عشرية وسلسلة أثمونات.

ومع ذلك، يكون النمط لمقاس في الخط في قائمة منافذ يستقبل المقاس عبره. وفي حالة وجود غموض بين النمط الوارد ونمط القيمة الموفرة (مثلاً، من خلال تنميط فرعي)، فإن اسم النمط يُتضمن في البيان المستقبَل.

يجري ترتيب آليات المواءمة في أربع زمرات:

أ) قيم محددة:
- تعبير يقيم قيمة محددة؛
- **(omit)**: تحذف قيمة؛

ب) رموز خاصة يمكن أن تستخدم بدلاً من قيم:
- (...): قائمة قيم؛
- **complement** (...): تكملة قائمة **values**؛
- ?: سمة واحدة لأي قيمة؛
- *: سمة واحدة لأي قيمة أو دون **value** على الإطلاق (أي، قيمة محذوفة)؛
- (*lowerBound..upperBound*): مدى **integer** أو قيم طليق بيِّن وبما في ذلك الحدود الدنيا والعليا؛
- **superset**: على الأقل جميع العناصر الواردة، أي، من الممكن أكثر؛
- **subset**: وعلى الأكثر العناصر الواردة، أي من الممكن أقل؛

ج) رموز خاصة يمكن أن تستخدم داخل قيم؛
- ?: سمة واحدة لأي عنصر وحيد في سلسلة أو صفيف أو **record of** أو **set of**؛
- *: سمة واحدة لأي عدد من عناصر تتابعية في أي سلسلة أو صفيف أو **record of** أو **set of** أو دون عناصر على الإطلاق (أي، عنصر محذوف)؛
- **premutation**: جميع العناصر الواردة ولكن بترتيب اعتباطي (ملاحظة، يسمح بـ **?** و***** كعناصر لقائمة إبدال)؛

د) رموز خاصة تصف نعوتاً لقيم؛
- **length**: تقييدات على طول سلسلة لأنماط سلسلة وعدد عناصر لـ **record of** و**set of** ومصفوفات؛
- **ifpresent**: لمواءمة قيم مجال خياري (إذا لم يكن محذوفاً).

يرد في الجدول 6 آليات المواءمة المدعمة ورموزها المتصاحبة (إن وجدت) ومدى تطبيقها. ويبين العمود الأيمن لهذا الجدول جميع أنماط TTCN-3 التي تنطبق عليها آليات المواءمة هذه. ويوجد في الملحق B وصف كامل لكل آلية مواءمة.

الجدول Z.140/6 – آليات مواءمة TTCN-3

يستخدم مع قيم	القيمة		بدلاً من قيم							داخل قيم			نعوت	
	قيمة محددة	احذف قيمة	قائمة مكملة	قائمة قيم	أي قيمة (؟)	أي قيمة أو لا شيء (*)	مدى	مجموعة ثانوية	مجموعة فرعية	أي عنصر (؟)	أي عنصر أو لا شيء (*)	حسب الإسكات	تقييد الطول	إن وُجد
boolean	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]
integer	نعم	نعم	نعم	نعم	نعم	نعم[أ]	نعم							نعم[ب]
float	نعم	نعم	نعم	نعم	نعم	نعم[أ]	نعم							نعم[ب]
bitstring	نعم	نعم	نعم	نعم	نعم	نعم[أ]				نعم	نعم		نعم	نعم[ب]
octetstring	نعم	نعم	نعم	نعم	نعم	نعم[أ]				نعم	نعم		نعم	نعم[ب]
hexstring	نعم	نعم	نعم	نعم	نعم	نعم[أ]				نعم	نعم		نعم	نعم[ب]
character strings	نعم	نعم	نعم	نعم	نعم	نعم[أ]	نعم			نعم	نعم		نعم	نعم[ب]
record	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]
record of	نعم	نعم	نعم	نعم	نعم	نعم[أ]				نعم	نعم	نعم	نعم	نعم[ب]
array	نعم	نعم	نعم	نعم	نعم	نعم[أ]				نعم	نعم		نعم	نعم[ب]
set	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]
set of	نعم	نعم	نعم	نعم	نعم	نعم		نعم	نعم	نعم	نعم		نعم	نعم[ب]
enumerated	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]
union	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]
anytype	نعم	نعم	نعم	نعم	نعم	نعم[أ]								نعم[ب]

[أ] – عندما يستخدم، ينطبق على مجالات خيارية لتسجيل ومجموعة أنماط فقط (دون تقييد على نمط ذلك المجال).
[ب] – عندما يستخدم، ينطبق على سجل ومجموعة مجالات فقط (دون تقييد على نمط ذلك المجال).

1.3.14 تحديد مرجعية عناصر مقاسات أو مجالات مقاسات

1.1.3.14 تحديد مرجعية عناصر فردية لسلسلة

لا يسمح بتحديد مرجع لعناصر فردية لسلسلة داخل مقاسات أو مجالات مقاسات.

مثـــال:

```
var template charstring t_Char1 := 'MYCHAR';
var template charstring t_Char2;

t_Char2 := t_Char1[1];
  // shall cause an error as referencing individual string elements is not allowed;
```

2.1.3.14 تحديد مراجع مجالات record وset

تسمح كل من مقاسات ومتغيرات مقاسات بتحديد مراجع مجالات فرعية داخل تعريف مقاس باستخدام ترميز بالنقط. ومع ذلك، يمكن أن يكون مجال مرجعي مجال فرعي لمجال مبني تخصص له آلية مواءمة. ويوفر هذا القسم قواعد لهذه الحالات.

- **Omit**, AnyValueOrNone، وقوائم قيم وقوائم مكملة: تحديد مرجع مجال فرعي في مجال مبني يكون فيه مخصص **omit**, AnyValueOrNone(*) لقائمة قيم أو قائمة مكملة، ويسبب خطأً.

المثـــال 1:

```
type record R1 {
```

```
        integer f1 optional,
        R2      f2 optional
}
type record R2 {
        integer g1,
        R2      g2 optional
}

:
var template R1 t_R1 := {
        f1 := 5,
        f2 := omit
}
var template R2 t_R2 := t_R1.f2.g2;
        // causes an error as omit is assigned to t_R1.f2
t_R1. f2 := *
t_R2 := t_R1.f2.g2;
        // causes an error as * is assigned to t_R1.f2

t_R1 := ({f1:=omit, f2:={g1:=0, g2:=omit}},{f1:=5, f2:={g1:=1, g2:={g1:=2, g2:=omit}}});

t_R2 := t_R1.f2;
t_R2 := t_R1.f2.g2;
t_R2 := t_R1.f2.g2.g2;
        // all these assignments cause error as a value list is assigned to t_R1

t_R1 :=
        complement({f1:=omit, f2:={g1:=0, g2:=omit}},{f1:=5, f2:={g1:=1, g2:={g1:=2, g2:=omit}}})

t_R2 := t_R1.f2;
t_R2 := t_R1.f2.g2;
t_R2 := t_R1.f2.g2.g2;
        // all these assignments cause error as a complemented list is assigned to t_R1
```

- (AnyValue: عند تحديد مرجع مجال فرعي في مجال مبني يكون فيه (?) AnyValue مخصص عند الجانب الأيمن للتخصيص، ويعاد (?) AnyValue لمجالات فرعية إلزامية ويعاد AnyvalueOrNone لمجالات فرعية خيارية.

عندما يحدد مرجع مجال فرعي في مجال مبني يخصص له (?) AnyValue، عند الجانب الأيسر للتخصيص، يتعين تمديد المجال المبني بتكرار حتى عمق المجال الفرعي المرجعي. وخلال هذا التمديد، يخصص أي (?) AnyValue لمجالات فرعية إلزامية ويخصص AnyValueOrNone لمجالات فرعية خيارية. وبعد هذا التمديد، تخصص القيمة أو آلية المواءمة على الجانب الأيمن من التخصيص لمجال فرعي مرجعي.

المثـــال 2:

```
t_R1 := {f1:=0, f2:=?}
t_R2 := t_R1.f2.g2;
  // after the assignment t_R2 will be {g1:=?, g2:=*}
t_R1.f2.g2.g2 := ({g1:=1, g2:=omit},{g1:=2, g2:=omit});
  // first the field t_R1.f2 has hypothetically be expanded to {g1:=?,g2:={g1:=?,g2:=*}}
  // thus after the assignment t_R1 will be:
  //   {f1:=0, f2:={g1:=?,g2:={g1:=?,g2:=({g1:=1, g2:=omit},{g1:=2, g2:=omit})}}}
```

- نعت **Ifpresent**: تحديد مرجع مجال فرعي في مجال مبني يكون فيه نعت **ifpresent** مضافاً له، ويسبب خطأ (على الرغم من القيمة وآلية المواءمة التي تذيل بها **ifpresent**).

3.1.3.14 تحديد مراجع عناصر record of وset of

تسمح كل من مقاسات ومتغيرات مقاسات بتحديد مراجع عناصر **record of** أو **set of** لمقاس أو مجال باستخدام ترميز الدليل. ومع ذلك، يمكن تخصيص آلية مواءمة لمقاس أو مجال يحدد فيه مرجع العنصر. ويوفر هذا القسم قواعد لهذه الحالات.

- **omit**, AnyValueOrNone وقوائم قيم وقوائم مكملة ومجموعة فرعية ومجموعة زائدة: تحديد مرجع عنصر في سجل أو مجموعة مجالات يخصص لـ **omit** ويخصص (*) AnyValueOrNone مع أو دون نعت طول أو قائمة قيم أو قائمة مكملة أو مجموعة فرعية أو مجموعة زائدة، يسبب خطأ.

المثـــال 1:

```
type record of integer RoI;
type record of RoI RoRoI;

:
var template RoI t_RoI;
var template RoRoI t_RoRoI;
var template integer t_Int;
:
t_RoRoI := ({},{0},{0,0},{0,0,0});
t_RoI := t_RoRoI[0];
//  shall cause an error as value list is assigned to t_RoRoI;
```

- AnyValue: عند تحديد مرجع عنصر مقاس أو مجال **record of** أو **set of** يخصص له (?) AnyValue (دون نعت طول)، عند الجانب الأيمن للتخصيص، يعاد (?) AnyValue. وإذا أرفق نعت طول بـ (?) AnyValue، لا يخل دليل المرجع بنعت الطول.

عندما يحدد مرجع عنصر في مقاس أو مجال **record of** أو **set of** يخصص له (دون نعت طول)، عند الجانب الأيسر للتخصيص، تخصص القيمة أو آلية المواءمة عند الجانب الأيمن للتخصيص للعنصر المرجعي، ويخصص (?)AnyElement لجميع العناصر قبل العنصر المرجعي (إن وُجد) ويضاف (*)AnyElementsOrNone في النهاية. وعندما يرفق نعت طول إلى نعت (?)AnyValue ينقل إلى مقاس أو مجال جديد بشفافية. ولا يخل الدليل بتقييدات نمط في أي من الحالات أعلاه.

المثـــال 2:

```
type record of integer RoI;
type record of RoI RoRoI;

:
var template RoI t_RoI;
var template RoRoI t_RoRoI;
var template integer t_Int;
:
t_RoI := ?;
t_Int := t_RoI[5];
  // after the assignment t_Int will be AnyValue(?);

t_RoRoI := ?;
t_RoI := t_RoRoI[5];
  // after the assignment t_RoI will be AnyValue(?);
t_Int := t_RoRoI[5].[3];
  // after the assignment t_Int will be AnyValue(?);

t_RoI := ? length (2..5);
t_Int := t_RoI[3];
  // after the assignment t_Int will be AnyValue(?);
t_Int := t_RoI[5];
  // shall cause an error as the referenced index is outside the length attribute
  // (note that index 5 would refer to the 6th element);

t_RoRoI[2] := {0,0};
  // after the assignment t_RoRoI will be {?,?,{0,0},*};
t_RoRoI[4] := {1,1};
  // after the assignment t_RoRoI will be {?,?,{0,0},?,{1,1},*};
t_RoI[0] := -5;
  // after the assignment t_RoI will be {-5,*}length(2..5);
t_RoI := ? length (2..5);
t_RoI[1] := 1;
  // after the assignment t_RoI will be {?,1,*}length(2..5);
t_RoI[3] := ?
  // after the assignment t_RoI will be {?,1,?,?,*}length(2..5);
t_RoI[5] := 5
  // after the assignment t_RoI will be {?,1,?,?,?,5,*}length(2..5); note that t_RoI
  // becomes an empty set but that shall cause no error;
```

- إبدال: عندما يحدد مرجع عنصر في مقاس أو مجال **record of**، يوجد داخل إبدال (على أساس دليله)، يسبب هذا خطأ. وتحدد أدلة عناصر يحميها إبدال على أساس عدد عناصر الإبدال. ويسبب AnyValueOrNone كعنصر إبدال حماية الإبدال لجميع أدلة العنصر **record of**.

المثـــال 3:

```
t_RoI := {permutation(0,1,3,?),2,?}
t_Int := t_RoI[5];
  // after the assignment t_Int will be AnyValue(?)

t_RoI := {permutation(0,1,3,?),2,*}
t_Int := t_RoI[5];
  // after the assignment t_Int will be * (AnyValueOrNone)
t_Int := t_RoI[2];
  // causes error as the third element (with index 2) is inside permutation

t_RoI := {permutation(0,1,3,*),2,?}
t_Int := t_RoI[5];
  // causes error as the permutation contains AnyValueOrNone(*) that is able to
  // cover any record of indexes
```

- النعت **IFpresent**: يشير إلى عنصر داخل حقل **record of** أو **set of** المُرفق به هذا النعت، ويتسبّب في خطأ (بصرف النظر عن القيمة أو آلية المواءمة الملحق بها النعت **Ifpresent**).

4.14 معلمية مقاسات

0.4.14 عـــام

تمكن معلمية مقاسات لكل من عمليات الإرسال والاستقبال. ويمكن أن تشمل المعلمات الفعلية لمقاس قيم ومقاسات ووظائف ورموز مواءمة خاصة. وتتبع قواعد قوائم المعلمات الرسمية والفعلية كما عرفت في 2.5.

مثـــال:

```
// The template
template MyMessageType MyTemplate (integer MyFormalParam):=
{
  field1 := MyFormalParam,
  field2 := pattern "abc*xyz",
  field3 := true
}

// could be used as follows
pco1.send(MyTemplate(123));
```

5.14 فـــارغ

6.14 مقاسات معدلة

0.6.14 عـــام

عادةً، يحدد مقاس مجموعة لقاعدة أو قيم بالتغيب أو رموز موائمة لكل مجال معرف في نمط ملائم أو تعريف توقيع. وفي الحالات حيث هناك حاجة إلى تغييرات صغيرة لتحديد مقاس جديد، من الممكن تحديد مقاس معدل. ويحدد مقاس معدل تعديلات على مجالات معينة للمقاس الأصلي، سواء بطريقة مباشرة أو غير مباشرة.

تدل الكلمة المفتاحية **modifies** على مقاس رئيسي يشتق منه مقاس جديد أو معدل. ويكون المقاس الرئيسي هذا إما مقاس أصلي أو مقاس معدل.

تحدث التعديلات بطريقة موصولة في النهاية بتتبُّع المقاس الأصلي. وإذا حدد مجال مقاس وقيمته المتناظرة أو رمز مواءمة في مقاس معدل، فإن القيمة المحددة أو رمز المواءمة يحل محل المحدد في المقاس الرئيسي. وإذا لم يحدد مجال مقاس وقيمته المتناظرة أو رمز مواءمة في مقاس معدل، فإن القيمة أو رمز المواءمة في المقاس الرئيسي تستخدم. وعندما يتداخل مجال يتعين تعديله مع مجال مقاس هو مجال مبني نفسه، لا يتغير مجال آخر لمجال مبني باستثناء المجال (المجالات) الدالة عليه بوضوح.

ولا يشير مقاس معدل إلى نفسه، سواء بطريقة مباشرة أو غير مباشرة، أي، لا يسمح باشتقاق متكرر.

المثـــال 1:

```
// Given
type record MyRecordType
{
  integer field,
  charstring field2,
```

```
  boolean field3
}
template MyRecordType MyTemplate1 :=
{
  field1 := 123,
  field2 := "A string",
  field3 := true
}
// then writing
template MyRecordType MyTemplate2 modifies MyTemplate1 :=
{
  field1 := omit,  // field1 is optional but present in MyTemplate1
  field2 := "A modified string"
            // field3 is unchanged
}
// is the same as writing
template MyRecordType MyTemplate2 :=
{
  field1 := omit,
  field2 := "A modified string",
  field3 := true
}
```

وعند الرغبة في تغيير قيم فردية لمقاس معدل أو مجال مقاس معدل **record of**، في هذه الحالات فقط، يمكن أيضاً استخدام تخصيص ترميز قيمة، حيث يكون الجانب الأيسر من التخصيص دليل العنصر الذي يتغير.

المثـــال 2:

```
template MyRecordOfType MyBaseTemplate := { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
template MyRecordOfType MyModifTemplate modifies MyBaseTemplate := { [2] := 3, [3] := 2 };
// MyModifTemplate shall match the sequence of values { 0, 1, 3, 2, 4, 5, 6, 7, 8, 9 }
```

1.6.14 معلمية مقاسات معدل

إذا كان لمقاس قاعدة قائمة معلمات رسمية، تنطبق القواعد التالية على جميع المقاسات المعدلة المشتقة من مقاس القاعدة ذلك، سواء كانت مشتقة أم لا في خطوة أو خطوات تعديل عديدة:

أ) لا يحذف مقاس مشتق معلمات معرّفة عند أي خطوة تعديل بين مقاس قاعدة والمقاس المعدل الفعلي؛

ب) يمكن أن يكون لمقاس مشتق معلمات إضافية (مذيّلة) عند الرغبة؛

ج) تتبع قائمة المعلمات الرسمية اسم المقاس لكل مقاس معدل.

مثـــال:

```
// Given
template MyRecordType MyTemplate1(integer MyPar):=
{
  field1 := MyPar,
  field2 := "A string",
  field3 := true
}

// then a modification could be
  template MyRecordType MyTemplate2(integer MyPar) modifies MyTemplate1 :=
{ // field1 is parameterized in Template1 and remains also parameterized in Template2
  field2 := "A modified string",
}
```

2.6.14 مقاسات معدلة في الخط

بالإضافة إلى خلق مقاسات معدلة مسماة بوضوح، يسمح TTCN-3 بتعريف مقاسات معدلة في الخط.

مثـــال:

```
// Given
template MyMessageType Setup :=
{ field1 := 75,
  field2 := "abc",
  field3 := true
}
// Could be used to define an in-line modified template of Setup
pco1.send (modifies Setup := {field1:= 76});
```

7.14 تغيير مجالات مقاس

في عمليات اتصالات (مثل **send** و**receive** و**call** و**getcall** وما إلى ذلك)، يسمح بتغيير مجالات مقاس عبر معلمية أو بواسطة مقاسات مشتقة في الخط فقط. ولا تدوم آثار التغييرات هذه على قيمة مجال مقاس في المقاس اللاحق لحدث اتصالات متناظر.

لا يستخدم ترميز بالنقط *MyTemplated.FieldId* لضبط أو استرداد قيم في أحداث اتصالات. ويستخدم رمز "<-" لهذا الغرض (انظر القسم 23).

8.14 عملية مواءمة

تسمح عملية **match** مواءمة بمقارنة قيمة متغير أو معلمة بمقاس. وتعيد العملية قيمة بولاني. وإذا لم تكن أنماط المقاس والمتغير متوائمة (انظر 7.6)، تعيد العملية false. وإذا كانت الأنماط متوائمة، تدل قيمة العودة للعملية ما إذا كانت قيمة المتغير متطابقة مع المقاس المحدد.

مثـــال:

```
template integer LessThan10 := (-infinity..9);

testcase TC001()
runs on MyMTCType
{
  var integer RxValue;
   :
  PCO1.receive(integer:?) -> value RxValue;

  if( match( RxValue, LessThan10)) { … }
// true if the actual value of Rxvalue is less than 10 and false otherwise
   :
}
```

9.14 عملية Valueof

تسمح عملية **valueof** لقيمة محددة في مقاس أن تخصص لمتغير. ويكون المتغير والمقاس من نمط متوائم (انظر 7.6) ويستبان كل مجال مقاس لقيمة وحيدة.

مثـــال:

```
type record ExampleType
{
  integer field1,
  boolean field2
}

template ExampleType SetupTemplate :=
{
  field1 := 1,
  field2 := true
}

 :
var ExampleType RxValue := valueof(SetupTemplate);
```

15 المشغلون

0.15 عـــام

يدعم TTCN-3 عدداً من المشغلين المعرّفين مسبقاً يمكن استخدامهم في مصطلحات تعبيرات TTCN-3. وينقسم المشغلون إلى سبع فئات:

أ) مشغلون حسابيون؛

ب) مشغلون لسلسلة؛

ج) مشغلون اتصاليون؛

د) مشغلون منطقيون؛

هـ) مشغلو بتات؛

و) مشغلو زحزحة؛

ز) مشغلو دوران.

ويرد في الجدول 7 هؤلاء المشغلين.

الجدول Z.140/7 – قائمة مشغلي TTCN-3

الفئة	المشغل	الرمز أو الكلمة المفتاحية
مشغلون حسابيون	addition	**+**
	subtraction	**-**
	multiplication	*****
	division	**/**
	modulo	**mod**
	remainder	**rem**
مشغلو سلسلة	concatenation	**&**
مشغلون اتصاليون	equal	**==**
	less than	**<**
	greater than	**>**
	not equal	**!=**
	greater than or equal	**>=**
	less than or equal	**<=**
مشغلون منطقيون	logical not	**not**
	logical and	**and**
	logical or	**or**
	logical xor	**xor**
مشغلو بتات	bitwise not	**not4b**
	bitwise and	**and4b**
	bitwise or	**or4b**
	bitwise xor	**xor4b**
مشغلو زحزحة	shift left	**<<**
	shift right	**>>**
مشغلو دوران	rotate left	**<@**
	rotate right	**@>**

ترد أسبقية هؤلاء المشغلين في الجدول 8. وفي أي صف في هذا الجدول، يكون للمشغلين الواردين أسبقية مساوية. وإذا ظهر أكثر من مشغل واحد لأسبقية مساوية في تعبير، تُقيّم العمليات من اليسار إلى اليمين. ويمكن استخدام قاطعتين لجمع متأثرين في تعبيرات، وفي هذه الحالة يكون لتعبير بين قاطعتين الأسبقية الأعلى للتقييم.

الجدول Z.140/8 – أسبقية المشغلين

الأولوية	نمط العملية	المشغل
العليا		**(…)**
	أوحد	**+, -**
	إثنيني	***, /, mod, rem**
	إثنيني	**+, -, &**
	أوحد	**not4b**
	إثنيني	**and4b**
	إثنيني	**xor4b**
	إثنيني	**or4b**
	إثنيني	**<<, >>, <@, @>**
	إثنيني	**<, >, <=, >=**
	إثنيني	**==, !=**
	أوحد	**not**
	إثنيني	**and**
	إثنيني	**xor**
الدنيا	إثنيني	**or**

1.15 مشغلون حسابيون

يمثل المشغلون الحسابيون عمليات الجمع والطرح والضرب والقسمة والمقاس والمتبقي. وتكون متأثرات هؤلاء المشغلين من نمط **integer** (بما في ذلك مشتقات من **integer**) أو **float** (بما في ذلك مشتقات من **float**، باستثناء **mod** و**rem** الذي يستخدم مع أنماط **integer** فقط (بما في ذلك مشتقات **integer**).

مع أنماط **integer**، يكون نمط النتيجة عمليات حسابية **integer**. ومع أنماط طليقة، يكون نمط النتيجة عمليات حسابية **float**.

وفي الحالة التي يستخدم فيها زائد (+) أو ناقص (–) كمشغل أحادي، تنطبق قواعد المتأثرات أيضاً. ونتيجة استخدام مشغل ناقص تكون القيمة سالبة لمتأثر إذا كانت موجبة والعكس بالعكس.

إن نتيجة أداء عملية قسمة (/) على اثنين:

أ) تعطي قيم **integer** جزء **integer** الكامل للقيمة الناتجة من تقسيم أول **integer** بالثانية (أي، يتم تجاهل الكسور)؛

ب) تعطي قيم **float** قيمة **float** الناتجة من تقسيم أول **float** بالثانية (أي، يتم تجاهل الكسور).

يحسب المشغلون **rem** متأثرات نمط **mod** وتكون النتيجة لنمط **integer**. وتحسب عمليات **integer** الباقي المتبقي من تقسيم صحيح x بـ y وy mod x. ولهذا تعرّف فقط لمتأثرات y غير صفر. وبالنسبة لـ x by y موجب، يكون لكل من x and y، x rem y وx mod y نفس النتيجة ولكن لمتغيرات سالبة مختلفة.

ورسمياً، يُعرّف **mod** و**rem** كما يلي:

```
x rem y = x - y * (x/y)
x mod y   = x rem |y|      whenx >= 0
      = 0          when x < 0  and   x   rem |y| = 0
      = |y| + x rem |y| when  x < 0 and x rem |y| < 0
```

يوضح الجدول 9 الفرق بين مشغل **mod** و**rem**:

الجدول Z.140/9 – تأثير مشغل mod وrem

x	–3	–2	–1	0	1	2	3
x mod 3	0	1	2	0	1	2	0
x rem 3	0	–2	–1	0	1	2	0

2.15 مشغلو سلسلة

يؤدي مشغلو سلسلة معرفين مسبقاً تسلسل قيم متوائمة مع أنماط سلسلة. والعملية هي تسلسل بسيط من اليسار إلى اليمين. ولا تتضمن شكلاً حسابياً إضافياً. والنمط الناتج هو نمط جذر لمتأثرات.

مثـــال:

```
'1111'B & '0000'B & '1111'B gives '111100001111'B
```

3.15 مشغلون اتصاليون

يمثل مشغلون اتصاليون معرّفون مسبقاً علاقات المساواة (==)، أقل من (<) وأكبر من (>) وعدم مساواة (=!) وأكبر من أو مساو (=>) وأقل من أو مساو (=<). ويمكن أن تكون المتأثرات للمساواة وعدم المساواة اعتباطية ولكن أنماط متوائمة مع استثناء نمط **enumerated**، تكون فيه المتأثرات مطابقات لنفس النمط. ويكون لجميع المشغلين الاتصاليين متأثرات من نمط **integer** فقط (بما في ذلك مشتقات **integer**) أو **float** (بما في ذلك مشتقات **float**) أو مطابقات لنفس أنماط **enumerated**. ويكون النمط الناتج لهذه العمليات **boolean**.

تكون قيمتا **charstring** أو **universal charstring** مساويتين فقط إذا كان لهما نفس الأطوال والسمات عند جميع المواضع هي نفسها. وبالنسبة لقيم أنماط **bitstring** أو **hexstring** أو **octetstring**، تنطبق نفس المساواة مع استثناء أن العناصر متساوية عند جميع المواضع هي بتات أو أرقام ستة عشرية أو أزواج لأرقام ستة عشرية على التوالي.

تكون قيمتا **record** أو قيم **set** أو قيم **record of** أو قيم **set of** متساوية إذا، وإذا فقط، كانت بنيات قيمتها الفعلية متوائمة (انظر 7.6) وقيم جميع المجالات المتناظرة متساوية. ويمكن أيضاً مقارنة قيم سجل بسجل قيم ومجموعة قيم لمجموعة قيم. وفي هذه الحالات، تنطبق نفس القواعد كما في قيمتي **record** أو **set** المقارنتين.

ملاحظة – يعني "all fields" أن المجالات التشغيلية غير الموجودة في قيمة فعلية من نمط **record** تؤخذ على أنها قيمة غير معرفة. ومثل هذا المجال يمكن أن يكون مساوياً لمجال عملية ناقص (يعتبر أيضاً قيمة غير معرفة) عند مقارنته بقيمة نمط **record** آخر أو بعنصر مع قيمة غير معرفة عند مقارنته بقيمة من نمط **record of**. وينطبق نفس المبدأ عند مقارنة نمطين **set** أو **set** ونمط **set of**.

تكون قيمتان لأنماط **union** متساوية إذا، وإذا فقط، في كلا القيمتين تكون الأنماط المختارة لمجالات متوائمة والقيم الفعلية لمجالات مختارة متساوية.

مثـــال:

```
// Given
type  set  SetA{
      integer    a1   optional,
      integer    a2   optional,
      integer    a3   optional
      };

type  set  SetB{
      integer    b1   optional,
      integer    b2   optional,
      integer    b3   optional
      };


type  set  SetC{
      integer    c1   optional,
      integer    c2   optional,
      };

type  set of integer   SetOf;

type  union UniD  {
      integer    d1,
      integer    d2,
      };

type  union UniE  {
      integer    e1,
      integer    e2,
      };

type  union UniF  {
      integer    f1,
      integer    f2,
      boolean    f3,
      };

// And
const      Set A  conSetA1  := { a1 := 0, a2 := omit, a3 := 2 };
            // Notice that the order of defining values of the fields does    not matter
const      SetB   conSetB1  := { b1 := 0, b3 := 2, b2 := omit };
const      SetB   conSetB2  := { b2 := 0, b3 := 2, b1 := omit };
const      SetC   conSetC1  := { c1 := 0, c2 :=2 };
const      SetOf  conSetOf1 := { 0, omit, 2 };
const      SetOf  conSetOf2 := { 0, 2 };
const      UniD   conUniD1  := { d1:= 0 };
const      UniE   conUniE1  := { e1:= 0 };
const      UniE   conUniE2; := { e2:= 0 };
const      UniF   conUniF1; := { f1:= 0 };

// Then
conSetA1 == conSetB1;
  // returns true
conSetA1 == conSetB2;
  // returns false, because neither a1 nor a2 are equal to their counterparts
  // ( the corresponding element is not omitted )
conSetA1 == conSetC1;
  // returns false, because the effective value structures of SetA and SetC are not
compatible
conSetA1 == conSetOf1;
  // returns true
conSetA1 == conSetOf2;
  // returns false, as the counterpart of the omitted a2 is 2,
  // but the counterpart of a3 is undefined
conSetC1 == conSetOf2;
  // returns true
conUniD1 == conUniE1;
  // returns true
conUniD1 == conUniE2;
```

```
  // returns false, as the chosen field e2 is not the counterpart of the field d1 of UniD1
conUniD1 == conUniF1;
  // returns false, as the effective value structures of UniD1 and UniF are not compatible
```

4.15 مشغلون منطقيون

يؤدي مشغلو **boolean** المعرّفين مسبقاً عمليات نفي و**and** منطقية و**or** منطقية و**xor** منطقية. وتكون متأثراتها من نمط **boolean**. ويكون نمط النتيجة لعمليات منطقية **boolean**.

يكون **not** منطقي مشغل أوحد يعيد القيمة true إذا كان متأثره قيمة **false** ويعيد قيمة **false** إذا كان المتأثر قيمة **true**.

يعيد **and** المنطقي القيمة **true** إذا كان كل منهما متأثره **true**؛ وإلا، يعيد قيمة **false**.

يعيد **or** المنطقي قيمة **true** إذا كان واحد على الأقل من المتأثرات هو **true**؛ ويعيد قيمة **true** فقط إذا كان كل من المتأثرين **false**.

يعيد **xor** المنطقي قيمة **true** إذا كان واحد من متأثراته هو **true**؛ ويعيد قيمة **false** فقط إذا كان كل من المتأثرين **false** أو كان كلاهما **true**.

يستخدم تقييم دوائر قصيرة لتعبير بولاني، أي، يتوقف تقييم متأثرات لمشغلين منطقيين بمجرد معرفة النتيجة: في حالة مشغل **and**، إذا قُيِّم المتغير على اليسار **false**، فإن المتغير على اليمين لا يُقيَّم ويُقيم التعبير بكامله **false**. وفي حالة مشغل **or**، إذا قُيم المتغير على اليسار **true**، فإن المتغير على اليمين لا يقيم ويقيم التعبير بكامله **true**.

5.15 مشغلو بتات

يؤدي مشغلو بتات معرفين مسبقاً عمليات **not** لبتات و**and** لبتات و**or** لبتات. ويعرف هؤلاء المشغلون على أنهم **xor** على التوالي.

ملاحظة – تقرأ "not for bit" و"and for bit " وما إلى ذلك.

وتكون متأثراتهم من نمط **bitstring** أو **hexstring** أو **octetststring**. وفي حالة **and4b** و**or4b** و**not4b**، تكون المتأثرات أنماطاً متوائمة. ويكون نمط النتيجة لمشغلي بتات نمط جذر لمتأثرات.

ويعكس مشغل أوحد **not4b** بتات قيم البتات الفردية لمتأثره. ولكل بتة في متأثر، تضبط بته 1 على صفر وتضبط بتة صفر على 1. أي:

```
not4b  '1'B  gives  '0'B
not4b  '0'B  gives  '1'B
```

المثـــال 1:

```
not4b  '1010'B  gives  '0101'B
not4b  '1A5'H  gives 'E5A'H
not4b  '01A5'O gives 'FE5A'O
```

يقبل مشغل **and4b** لبتات متأثرين لطول متساوي. ولكل موضع بتة متناظرة، تكون قيمة النتيجة 1 إذا ضُبطت البتتان على 1؛ وإلاّ، تكون قيمة البتة الناتجة صفراً. أي:

```
'1'B and4b '1'B gives  '1'B
'1'B and4b '0'B gives  '0'B
'0'B and4b '1'B gives  '0'B
'0'B and4b '0'B gives  '0'B
```

المثـــال 2:

```
'1001'B and4b '0101'B gives '0001'B
'B'H and4b '5'H  gives  '1'H
'FB'O and4b '15'O  gives  '11'O
```

يقبل مشغل **or5b** لبتات متأثرين لطول متساوي. ولكل موضع بتة متناظرة، تكون قيمة النتيجة صِفراً إذا ضُبطت البتتان على صفر؛ وإلا، تكون قيمة البتة الناتجة 1. أي:

```
'1'B or4b '1'B gives  '1'B
'1'B or4b '0'B gives  '1'B
'0'B or4b '1'B gives  '1'B
'0'B or4b '0'B gives  '0'B
```

المثـــال 3:

```
'1001'B or4b '0101'B gives '1101'B
'9'H or4b '5'H gives  'D'H
'A9'O or4b 'F5'O gives  'FD'O
```

يقبل مشغل **xor4b** لبتات متأثرين لطول متساوي. ولكل موضع بتة متناظرة، تكون قيمة النتيجة صفراً إذا ضبطت البتتان على صفر أو إذا ضبطت البتتان على 1؛ وإلا، تكون قيمة البتة الناتجة 1. أي:

```
'1'B xor4b '1'B gives  '0'B
'0'B xor4b '0'B gives  '0'B
'0'B xor4b '1'B gives  '1'B
'1'B xor4b '0'B gives  '1'B
```

6.15 مشغلو زحزحة

يؤدي مشغلو زحزحة معرفين مسبقاً عمليات زحزحة على اليسار (**<<**). وزحزحة على اليمين (**<<**). ويكون المتأثر على اليسار من نمط **bitstring** أو **hexstring** أو **octetstring**. ويكون المتأثر على اليمين من نمط **integer**. ويكون نمط النتيجة لهؤلاء المشغلين نفسه كما لمتأثر اليسار.

يسلك مشغلو الزحزحة بشكل مختلف على أساس نمط المتأثر على اليسار. وإذا كان نمط المتأثر على اليسار:

أ) **bitstring**، تكون وحدة الزحزحة المطبقة بتة 1؛

ب) **hexstring**، تكون وحدة الزحزحة المطبقة رقم ستة عشري 1؛

ج) **octetstring**، تكون وحدة الزحزحة المطبقة أثمون 1.

يقبل مشغل زحزحة إلى اليسار (**<<**) متأثرين. ويزحزح المتأثر على اليسار بواسطة عدد وحدات الزحزحة إلى اليسار كما حدد بواسطة المتأثر على اليمين. ويتم تجاهل وحدات الزحزحة الزائدة (بتات أو أرقام ستة عشرية أو أثمونات). ولكل وحدة زحزحة مزحزحة إلى اليسار، يُدرج صفر أو '0'B أو '0'H أو '00'O (المحدد طبقاً لنمط متأثر على اليسار) من الجانب الأيمن لمتأثر على اليسار.

المثـــال 1:

```
'111001'B  << 2 gives  '100100'B
'12345'H  << 2 gives  '34500'H
'1122334455'O  << (1+1)  gives  '3344550000'O
```

يقبل مشغل زحزحة إلى اليمين (**>>**) متأثرين. ويزحزح المتأثر على اليسار بواسطة عدد وحدات الزحزحة إلى اليمين كما حدد بواسطة المتأثر على اليمين. ويتم تجاهل وحدات الزحزحة الزائدة (بتات أو أرقام ستة عشرية أو أثمونات). ولكل وحدة زحزحة مزحزحة إلى اليمين، يدرج صفر أو '0'B أو '0'H أو '00'O المحدد طبقا لنمط متأثر على اليسار) من الجانب الأيسر لمتأثر على اليسار.

المثـــال 2:

```
'111001'B  >> 2 gives  '001110'B
'12345'H  >> 2 gives  '00123'H
'1122334455'O  >> (1+1)  gives  '0000112233'O
```

7.15 مشغلو دوران

يؤدي مشغلو دوران معرفين مسبقاً مشغلي دوران على اليسار (**<@**) ودوران على اليمين (**@>**). ويكون متأثرها على اليسار من نمط **bitstring** أو **hexstring** أو **octetstring** أو **charstring** أو **universal charstring**. ويكون متأثرها على اليمين من نمط **integer**. ويكون نمط النتيجة لهؤلاء المشغلين نفسه كما لمتأثر على اليسار:

يسلك مشغلو دوران على نحو مختلف على أساس نمط متأثرهم على اليسار. وإذا كان نمط المتأثر على اليسار هو:

أ) **bitstring**، تكون وحدة الدوران المطبقة بتة 1؛

ب) **hexstring**، تكون وحدة الدوران المطبقة رقم ستة عشري 1؛

ج) **octetstring**، تكون وحدة الدوران المطبقة أثمون 1؛

د) **charstring** أو **universal charstring** تكون وحدة الدوران المطبقة سمة واحدة.

يقبل مشغل دوران على اليسار (**<@**) متأثرين. ويقوم بدوران متأثر على اليسار بواسطة عدد من وحدات زحزحة إلى اليسار كما حدد المتأثر على اليمين. ويعاد إدراج وحدات الزحزحة الزائدة (بتات أو أرقام ستة عشرية أو أثمونات أو سمات) في المتأثر على اليسار من جانبه الأيمن.

المثـــال 1:

```
'101001'B  <@ 2 gives  '100110'B
'12345'H   <@ 2 gives  '34512'H
'1122334455'O  <@ (1+2)  gives  '4455112233'O
"abcdefg" <@ 3 gives "defgabc"
```

يقبل مشغل دوران على اليمين **(<@)** متأثرين. ويقوم بدوران متأثر على اليسار بواسطة عدد من وحدات زحزحة إلى اليمين كما حدد المتأثر على اليمين. ويعاد إدراج وحدات الزحزحة الزائدة (بتات أو أرقام ستة عشرية أو أثمونات أو سمات) في المتأثر على اليسار من جانبه الأيسر.

المثـــال 2:

```
'100001'B  @> 2 gives  '011000'B
'12345'H   @> 2 gives  '45123'H
'1122334455'O  @> (1+2)  gives  '3344551122'O
"abcdefg" @> 3 gives "efgabcd"
```

16 وظائف وaltsteps

في TTCN-3، تستخدم وظائف وaltsteps لتحديد وبناء سلوك اختبار وتعرف سلوك بالتغيب وبناء حساب في وحدة وما إلى ذلك كما ورد في الأقسام التالية.

1.16 وظائف

0.1.16 عـــام

تستخدم وظائف في TTCN-3 للتعبير عن سلوك اختبار أو تنظيم تنفيذ اختبار أو بناء حساب في وحدة، مثلاً، لحساب قيمة وحيدة وتدميث مجموعة متغيرات أو التأكد من بعض الشروط. ويمكن أن تعيد وظائف قيمة أو مقاس. ويدل على عودة قيمة الكلمة المفتاحية **return** يتبعها مُعرف نمط. ويدل على عودة مقاس الكلمة المفتاحية **return template** يتبعها مُعرف نمط.

إن الكلمة المفتاحية **return**، عندما تستخدم في جسم الوظيفة مع عودة قيمة معرفة في رأسيتها، يتبعها دائماً تعبير يمثل قيمة العودة. ويكون نمط قيمة العودة متوائم مع نمط العودة. والكلمة المفتاحية **return**، عندما تستخدم في جسم الوظيفة مع عودة مقاس معرفة في رأسيتها، يتبعها دائماً تعبير أو مطابق مقاس يمثل مقاس العودة. ويكون نمط مقاس العودة متوائماً مع نمط مقاس العودة.

ويسبب بيان العودة في جسم الوظيفة إنهاء الوظيفة ويعيد قيمة العودة إلى موقع نداء الوظيفة.

المثـــال 1:

```
// Definition of MyFunction which has no parameters
function MyFunction() return integer
{

  return 7;    // returns the integer value 7 when the function terminates
}

// Definition of functions which may return matching symbols or templates
function MyFunction2() return template integer
{
:
  return ?;    // returns the matching mechanism AnyValue
}
function MyFunction3() return template octetstring
{
:
  return "FF??FF"O;    // returns an octetstring with AnyElement inside it
}
```

يمكن تعريف وظيفة في وحدة أو يعلن عنها باعتبارها معرفة خارجياً (أي، **external**). وبالنسبة لوظيفة خارجية فقط، يتعين توفير سطح بيني الوظيفة في وحدة TTCN-3. وتحقيق الوظيفة الخارجية هو خارج مدى هذه التوصية. ولا يسمح للوظائف الخارجية بأن تحتوي على عمليات منفذ. ولا يسمح لوظائف خارجية أن تعيد مقاسات.

```
external function MyFunction4() return integer;// External function without parameters
                                               // which returns an integer value

external function InitTestDevices();    // An external function which only has an
                                        // effect outside the TTCN-3 module
```

وفي وحدة، يمكن تعريف سلوك وظيفة باستخدام بيانات وعمليات برنامج ترد في القسم 18. وإذا استخدمت وظيفة متغيرات وثوابت ومؤقتات ومنافذ معلن عنها في نمط مكون، يجري تحديد مرجع نمط مكون باستخدام الكلمات المفتاحية **runs on** في رأسية الوظيفة. والاستثناء الوحيد لهذه القاعدة هو عندما تمرر معلومات المكون الواسع الضرورية في الوظيفة كمعلمات.

المثـــال 2:

```
function MyFunction3() runs on MyPTCType {
                            // MyFunction3 doesn't return a value, but
  var integer MyVar := 5;   // does make use of the port operation
  PCO1.send(MyVar);         // send and therefore requires a runs on
                            // clause to resolve the port identifiers
}                           // by referencing a component type
```

إن وظيفة دون شرط **runs on** لا تنفذ وظيفة أو altstep أو تنشط altstep بالتغيب مع شرط **runs** محلي.

إن الوظائف التي تبدأ باستخدام عملية مكون اختبار **start** يكون لها دائماً شرط **runs on** (انظر 5.22) وتعتبر أنها تنفذ في مكون قيد البدء، أي، ليس محلياً. ومع ذلك، يمكن تنفيذ عملية مكون اختبار **runs** في وظائف دون شرط **runs on**.

ملاحظة – إن التقييدات المتعلقة بالشرط **runs on** تتعلق فقط بوظائف وaltsteps وليس باختبارات مجردة.

إن الوظائف المستخدمة في جزء تحكم TTCN-3 ليس لها شرط **runs on**. ومع ذلك، يسمح لها بتنفيذ اختبارات مجردة.

1.1.16 معلمية وظائف

يمكن معلمية وظائف. وتتبع قواعد قوائم معلمات رسمية كما عُرفت في 2.5.

مثـــال:

```
function MyFunction2(inout integer MyPar1) {
                          // MyFunction2 doesn't return a value
  MyPar1 := 10 * MyPar1;  // but changes the value of  MyPar1 which
}                         // is passed in by reference
```

2.1.16 تنفيذ وظائف

تنفذ وظيفة بالإشارة إلى اسمها وتوفير قائمة فعلية لمعلمات. وتنفذ الوظائف التي لا تعيد قيماً مباشرة. أما الوظائف التي تعيد قيماً فتنفذ مباشرة أو داخل تعبيرات. وتتبع قواعد قوائم معلمات رسمية كما عرفت في 2.5.

مثـــال:

```
MyVar := MyFunction4();// The value returned by MyFunction4 is assigned to MyVar.
                       // The types of the returned value and MyVar have to be compatible

MyFunction2(MyVar2);   // MyFunction2 doesn't return a value and is called with the
                       // actual parameter MyVar2, which may be passed in by reference

MyVar3 := MyFunction6(4)+ MyFunction7(MyVar3); // Functions used in expressions
```

تنطبق تقييدات خاصة على وظائف مسندة بمكونات اختبار باستخدام عملية مكون اختبار **start**. وترد هذه التقييدات في 5.22.

3.1.16 وظائف معرّفة مسبقاً

يحتوي TTCN-3 على عدد من وظائف (مبنية) معرّفة مسبقاً لا تحتاج إلى الإعلان عنها قبل الاستخدام.

الجدول 10/ Z.140 – قائمة وظائف معرفة مسبقاً لـ TTCN-3

الفئة	الوظيفة	الكلمة المفتاحية
وظائف تحويل	حوّل قيمة **integer** إلى قيمة **charstring**	**int2char**
	حوّل قيمة **integer** إلى قيمة **universalcharstring**	**int2unichar**
	حوّل قيمة **integer** إلى قيمة **bitstring**	**int2bit**
	حوّل قيمة **integer** إلى قيمة **hexstring**	**int2hex**
	حوّل قيمة **integer** إلى قيمة **octetstring**	**int2oct**
	حوّل قيمة **integer** إلى قيمة **charstring**	**int2str**
	حوّل قيمة **integer** إلى قيمة **float**	**int2float**
	حوّل قيمة **float** إلى قيمة **integer**	**float2int**
	حوّل قيمة **charstring** إلى قيمة **integer**	**char2int**
	حوّل قيمة **charstring** إلى قيمة **octetstring**	**char2oct**
	حوّل قيمة **universal charstring** إلى قيمة **integer**	**unichar2int**
	حوّل قيمة **bitstring** إلى قيمة **integer**	**bit2int**
	حوّل قيمة **bitstring** إلى قيمة **hexstring**	**bit2hex**
	حوّل قيمة **bitstring** إلى قيمة **octetstring**	**bit2oct**
	حوّل قيمة **bitstring** إلى قيمة **charstring**	**bit2str**
	حوّل قيمة **hexstring** إلى قيمة **integer**	**hex2int**
	حوّل قيمة **hexstring** إلى قيمة **bitstring**	**hex2bit**
	حوّل قيمة **hexstring** إلى قيمة **octetstring**	**hex2oct**
	حوّل قيمة **hexstring** إلى قيمة **charstring**	**hex2str**
	حوّل قيمة **octetstring** إلى قيمة **integer**	**oct2int**
	حوّل قيمة **octetstring** إلى قيمة **bitstring**	**oct2bit**
	حوّل قيمة **octetstring** إلى قيمة **hexstring**	**oct2hex**
	حوّل قيمة **octetstring** إلى قيمة **charstring**	**oct2str**
	حوّل قيمة **octetstring** إلى قيمة **charstring**	**oct2char**
	حوّل قيمة **charstring** إلى قيمة **octetsting**	**str2int**
	حوّل قيمة **charstring** إلى قيمة **octestring**	**str2oct**
	حوّل قيمة **charstring** إلى قيمة **float**	**str2float**
وظائف طول/حجم	أعد طول قيمة لأي نمط سلسلة	**lengthof**
	أعد عدد العناصر في **record** أو **record of** أو **template** أو **set** أو **set of** أو **array**	**sizeof**
	أعد عدد العناصر في نمط مبني	**sizeoftype**
وظائف وجود/اختيار	حدد إذا كان المجال خيارياً موجوداً في **record** أو **record of** أو **template** أو **set** أو **set of**	**ispresent**
	حدد أي اختيار تم في نمط **union**	**ischosen**
وظائف مناول سلسلة	أعد جزء سلسلة دخل يتواءم مع وصف تخطيط محدد	**regexp**
	أعد الجزء المحدد لسلسلة دخل	**substr**
	استبدل سلسلة فرعية لسلسلة أو أدرج سلسلة دخل في سلسلة	**replace**
وظائف أخرى	وَلّد عدداً طليقاً عشوائياً	**rnd**

عندما تنفذ وظيفة معرّفة مسبقاً:

1) يكون عدد المعلمات الفعلية هو نفسه كعدد المعلمات الرسمية؛
2) تقيم كل معلمة فعلية عنصراً لنمط معلمتها الرسمية المتناظرة؛
3) تكون جميع المتغيرات الظاهرة في قائمة معلمات فعلية مسندة.

يرد في الملحق C الوصف الكامل للوظائف المعرّفة مسبقاً.

4.1.16 تقييدات على وظائف مطلوبة من أماكن محددة

يمكن طلب وظائف تعيد قيماً خلال عملية اتصالات (في مقاس أو مجال مقاس أو مقاسات في الخط) أو خلال تقييم لقطة (في حراسات بولاني لبيانات alt أو altsteps (انظر 1.1.20) وفي تدميث تعاريف محلية altstep (انظر 2.2.16). ولتجنب الآثار الجانبية التي تسبب تغيير حالة المكون أو اللقطة الفعلية ولمنع نتائج مختلفة لتقييمات متلاحقة على لقطة غير متغيرة، لا تستخدم العمليات التالية في وظائف تطلب في الحالات المحددة أعلاه:

- جميع عمليات مكون، أي، **create**، **start** (مكون) و**stop** (مكون) و**kill** و**running** (مكون) (انظر الملاحظات 1 و3 و4 و6).
- جميع عمليات منْفذ، أي، **start** (منْفذ) و**stop** (منْفذ) و**halt** و**clear** و**send** و**receive** و**trigger** و**call** و**getcall** و**reply** و**getreply** و**raise** و**catch** و**check** و**connect** و**map** (انظر الملاحظات 1 و2 و3 و6).
- عملية **action** (انظر الملاحظتين 2 و6).
- جميع عمليات مؤقت، أي **start** (مؤقت) و**stop** (مؤقت) و**running** (مؤقت) و**read** و**timout** (انظر الملاحظتين 4 و6).
- طلب وظائف خارجية (انظر الملاحظتين 4 و6).
- طلب وظيفة معرفة مسبقاً **rnd** (انظر الملاحظتين 4 و6).
- تغيير متغيرات مكون، أي، باستخدام متغيرات مكون على الجانب الأيمن للتخصيص، وفي استطباق معلمات **out** و**inout** (انظر الملاحظتين 4 و6).
- طلب عملية **setverdict** (انظر الملاحظتين 4 و6).
- تنشيط ووقف تنشيط بالتغيب، أي، بيانات activate وdeactivate (انظر الملاحظتين 5 و6).
- طلب وظائف مع معلمات **out** أو **inout** (انظر الملاحظتين 7 و8).

الملاحظة 1 – يمكن أن يسبب تنفيذ عمليات **start** و**stop** و**done** و**killed** و**halt** و**clear** و**receive** و**trigger** و**getcall** و**getreply** و**catch** و**check** تغييرات للقطة الحالية.

الملاحظة 2 – يجري تجنب عمليات **send** و**call** و**reply** و**raise** و**action** لأغراض قابلية القراءة، أي، تكون جميع الاتصالات صريحة ولا تكون أثراً جانبياً لعملية اتصالات أخرى أو تقييم لقطة.

الملاحظة 3 – يجري تجنب عمليات **map** و**unmap** و**connect** و**disconnect** و**create** لأغراض قابلية القراءة، أي، تكون جميع عمليات التشكيل صريحة ولا تكون أثراً جانبياً لعملية اتصالات أخرى أو تقييم لقطة.

الملاحظة 4 – يجري تجنب طلب عمليات خارجية **rnd** و**running** و**alive** و**read** و**setverdict** وكتابة متغيرات مكون بسبب أنها قد تؤدي إلى نتائج مختلفة لتقييمات لاحقة لنفس اللقطة، ومن ثم، مثلاً، مما يجعل اكتشاف سد غير ممكن.

الملاحظة 5 – يجري تجنب عمليات **activate** و**deactivate** بسبب أنها تعدل مجموعة بالتغيب اعتبرت خلال تقييم اللقطة الحالية.

الملاحظة 6 – تنطبق التقييدات، باستثناء الحدود على استخدام معلمية **out** أو **inout** بشكل متكرر، أي، لا يُسمح باستخدامها مباشرة أو عبر سلسلة طويلة اعتباطية لتنفيذ وظائف.

الملاحظة 7 – لا ينطبق التقييد على وظائف مطلوبة مع معلمات **out** أو **inout** بشكل متكرر، أي، تكون وظائف مطلوبة هي في حد ذاتها وظائف طلب مع معلمات **out** أو **inout** قانونية.

الملاحظة 8 – يجري تجنب استخدام معلمات **out** أو **inout** بسبب أنها قد تؤدي إلى نتائج مختلفة لتقييمات لاحقة لنفس اللقطة.

2.16 Altsteps

0.2.16 عـــام

يستخدم altsteps TTCN-3 لتحديد سلوك بالتغيب أو لبناء بدائل لبيان **alt**. إن altsteps هي وحدات منظور مشابهة لوظائف. ويعرف جسم alstep مجموعة تشغيلية لتعاريف محلية ومجموعة بديلة، ما تسمى *top alternatives*، التي تشكل جسم altstep. إن قواعد التركيب لبدائل قمة هي مماثلة لقواعد تركيب بدائل بيانات **alt**.

يمكن تعريف سلوك altstep باستخدام بيانات وعمليات برنامج موجزة في القسم 18. وإذا شمل altstep عمليات منفذ أو استخدامات متغيرات مكون أو ثوابت أو مؤقتات، يحدد مرجع نمط المكون المتصاحب باستخدام الكلمات المفتاحية **runs on** في رأسية altstep. والاستثناء الوحيد لهذه القاعدة إذا كانت جميع المنافذ والمتغيرات والثوابت والمؤقتات المستخدمة في altstep مرّت كمعلمات.

مثـــال:

```
// Given
type component MyComponentType {
  var integer MyIntVar := 0;
  timer MyTimer;
  port MyPortTypeOne PCO1, PCO2;
  port MyPortTypeTwo PCO3;
}

// Altstep definition using PCO1, PCO2, MyIntVar and MyTimer of MyComponentType

altstep AltSet_A(in integer MyPar1) runs on MyComponentType {
   [] PCO1.receive(MyTemplate(MyPar1, MyIntVar) {
      setverdict(inconc);
      }
   [] PCO2.receive {
       repeat
      }
   [] MyTimer.timeout {
      setverdict(fail);
       stop
      }
}
```

يمكن أن ينفذ altsteps وظائف وaltsteps أو altsteps نشيطة بالتغيب. ولا ينفذ altstep دون شرط **runs on** وظيفة أو altstep أو altstep نشيط بالتغيب مع شرط **runs on** محلي.

1.2.16 معلمية altsteps

يمكن معلمية altsteps. إن altstep الذي ينشط بالتغيب يكون له معلمات **in** ومعلمات منفذ ومعلمات مؤقت. وaltstep الذي ينفذ فقط كبديل في بيان أو **alt** كبيان بمفرده في وصف سلوك TTCN-3 يكون له معلمات **in** و**out** و**inout**. وتتبع قواعد قوائم معلمات رسمية كما عرّفت في 2.5.

2.2.16 تعاريف محلية في altsteps

0.2.2.16 عـــام

يمكن أن يعرّف altsteps تعاريفاً محلية لثوابت ومتغيرات ومؤقتات. وتعرف التعاريف المحلية قبل مجموعة بدائل.

مثـــال:

```
altstep AnotherAltStep(in integer MyPar1) runs on MyComponentType {
  var integer MyLocalVar := MyFunction();       // local variable
  const float MyFloat := 3.41;              // local constant
  [] PCO1.receive(MyTemplate(MyPar1, MyLocalVar) {
     setverdict(inconc);
     }
  [] PCO2.receive {
     repeat
     }
}
```

تقييدات لتدميث تعاريف محلية في altsteps 1.2.2.16

يمكن أن يكون لتدميث تعاريف محلية بواسطة طلب وظائف قيمة مطلوبة آثار جانبية. ولتجنب الآثار الجانبية التي تسبب عدم اتساق بين لقطة محلية وحالة مكون، ولمنع نتائج مختلفة لتقييمات لاحقة في لقطة غير متغيرة، تنطبق التقييدات الواردة في 4.1.16 على تدميث تعاريف محلية.

3.2.16 تنفيذ altsteps

يتعلق تنفيذ altstep دائماً ببيان **alt**. ويتم التنفيذ إما ضمنياً بواسطة آلية بالتغيب (انظر القسم 21) أو صراحة بواسطة نداء مباشر في بيان **alt** (انظر 6.1.20). ولا يسبب تنفيذ altstep لقطة جديدة ويتم تقييم بدائل قمة لـ altstep باستخدام اللقطة الفعلية لبيان **alt** الذي طلب منه altstep.

ملاحظة – تؤخذ لقطة جديدة في altstep، في بديل قمة مختار، إذا كان بيان **alt** جديداً محدداً وأدخل.

بالنسبة لتنفيذ ضمني لـ altstep بواسطة آلية بالتغيب، ينشط altstep بالتغيب بواسطة بيان **activate** قبل وصول مكان التنفيذ.

المثـــال 1:

```
 :
var default MyDefVarTwo := activate(MySecondAltStep()); // Activation of an altstep as default
 :
```

يبدو نداء واضح لبيان altstep كنداء **alt** وظيفة لبديل.

المثـــال 2:

```
 :
alt {
   [] PCO3.receive {
        …
      }
   [] AnotherAltStep(); // explicit call of altstep AnotherAltStep as an alternative
                        // of an alt statement
   [] MyTimer.timeout {}
}
```

عندما يطلب altstep صراحة في بيان **alt**، يكون البديل التالي الذي يتم التأكد منه البديل الأول لـ **altstep**. ويتم التأكد من بدائل **altstep** وتنفذ بنفس الطريقة كبدائل بيان **alt** (انظر 1.20) باستثناء عدم أخذ لقطة جديدة عند دخول **altstep**. ويسبب الانتهاء غير الناجح لـ altstep (أي، تم التأكد من جميع بدائل قمة **altstep** ولم يوجد فرع مواءمة) تقييم البديل التالي أو تنفيذ آلية بالتغيب (إذا كان النداء الواضح هو آخر بديل لبيان **alt**). ويمكن أن يسبب انتهاءً ناجحاً، وإما انتهاء مكون اختبار، أي، ينتهي **altstep** مع بيان **stop** أو لقطة جديدة وإعادة تقييم بيان **alt**، أي، ينتهي **altstep** مع **repeat** (انظر 2.20) أو استمرار مباشر بعد بيان **alt**، أي، ينتهي بديل قمة مختار لـ altstep دون **repeat** أو **stop** واضح.

يمكن أيضاً طلب **altstep** كبيان بمفرده في وصف سلوك TTCN-3. وفي هذه الحالة، يمكن تفسير نداء **altstep** كاختزال لبيان **alt** مع بديل واحد فقط يصف النداء الواضح لـ **altstep**.

المثـــال 3:

```
// The statement
AnotherAltStep(); // AnotherAltStep is assumed to be a correctly defined altstep

//is a shorthand for

alt {
   [] AnotherAltStep();
}
```

3.16 وظائف وaltsteps لأنماط مختلفة لمكون

انظر 3.7.6.

17 اختبارات مجردة

0.17 عـــام

إن اختبارات مجردة هي نوع خاص لوظيفة. وفي جزء تحكم وحدة، يستخدم بيان **execute** لبدء اختبارات مجردة (انظر 1.27). وتكون نتيجة اختبار مجرد دائماً قيمة لنمط **verdicttype**. ويحتوي كل اختبار مجرد على MTC واحد وواحد فقط، يحدد مرجع النمط في رأسية تعريف الاختبار المجرد. ويكون السلوك المعرف في جسم الاختبار المجرد هو سلوك MTC.

عندما ينفذ اختبار مجرد، يخلق MTC، ومنافذ MTC ويستطبق السطح البيني لنظام الاختبار، ويبدأ السلوك المحدد في تعريف الاختبار المجرد على MTC. وتتم جميع هذه الأعمال ضمنياً، أي، دون عمليات **create** و**start** واضحة.

لتوفير معلومات تسمح بحدوث هذه العمليات ضمنياً، يكون لرأسية اختبار مجرد جزأين:

أ) جزء سطح بيني (إلزامي): تدل عليه الكلمة المفتاحية **runs on** التي تشير إلى نمط المكون المطلوب لـ MTC وتجعل اسم المنفذ المتصاحب مرئياً في سلوك MTC؛

ب) جزء نظام الاختبار (خياري): تدل عليه الكلمة المفتاحية **system** التي تشير إلى نمط المكون الذي يعرف منافذ مطلوبة لسطح بيني الاختبار المجرد. ويحذف جزء نظام الاختبار فقط إذا، خلال تنفيذ الاختبار، استنطق MTC. وفي هذه الحالة، يعرف نمط MTC منافذ سطح بيني نظام الاختبار ضمنياً.

مثـــال:

```
testcase MyTestCaseOne()
runs on MyMtcType1      // defines the type of the MTC
system MyTestSystemType // makes the port names of the TSI visible to the MTC
{
  :   // The behaviour defined here executes on the mtc when the test case invoked
}

// or, a test case where only the MTC is instantiated
testcase MyTestCaseTwo() runs on MyMtcType2
{
  :   // The behaviour defined here executes on the mtc when the test case invoked
}
```

1.17 معلمية اختبارات مجردة

يمكن معلمية اختبارات مجردة. وتتبع قواعد قوائم معلمات رسمية كما عرفت في 2.5.

18 نظرة شاملة على بيانات وعمليات برنامج

إن عناصر برنامج أساسية لاختبارات مجردة ووظائف وaltsteps وجزء تحكم وحدات TTCN-3 هي تعبيرات وبيانات أساسية لبرنامج مثل تخصيصات وبنيات عروة وما إلى ذلك، وبيانات سلوك مثل سلوك تتابعي وسلوك بديل وتشذير وبالتغيب وما إلى ذلك وعمليات مثل **send**، **receive**، **create**، وما إلى ذلك.

ويمكن أن تكون بياناته إما بيانات وحيدة (لا تشمل بيانات أخرى لبرنامج) أو بيانات مركبة (يمكن أن تشمل بيانات أخرى وفدرات بيانات وإعلانات).

تنفذ البيانات بترتيب ظهورها، أي، تتابعياً، كما يوضح الشكل 8.

الشكل Z.140/8 – توضيح سلوك تتابعي

يتم فصل البيانات الفردية في التتابع بواسطة المحدد ";".

مثـــال:

```
MyPort.send(Mymessage); MyTimer.start; log("Done!");
```

يمكن أن توجد مواصفة فِدرة فارغة لبيانات وإعلانات، أي، { } في بيانات مركبة، مثل، فرع في بيان **alt** ومتضمنة عدم اتخاذ إجراءات.

الجدول Z.140/11 – نظرة شاملة على تعبيرات وبيانات وإعلانات TTCN-3

البيان	الكلمة المفتاحية أو الرمز المصاحب	يمكن أن يستخدم في تحكم وحدة	يمكن أن يستخدم في وظائف واختبارات مجردة وaltsteps	يمكن أن يستخدم في وظائف طلبت من مقاسات وبولاني وحراسات أو من تدميث altstep لتعاريف محلية
تعبيرات	(…)	نعم	نعم	نعم
بيانات أساسية لبرنامج				
تخصيصات	:=	نعم	نعم	نعم (انظر الملاحظة 3)
تسجيل	log	نعم	نعم	نعم
Label and Goto	label / goto	نعم	نعم	نعم
If-else	if (…) {…} else {…}	نعم	نعم	نعم
لعروة	for (…) {…}	نعم	نعم	نعم
While loop	while (…) {…}	نعم	نعم	نعم
Do while loop	do {…} while (…)	نعم	نعم	نعم
أوقف التنفيذ	stop	نعم	نعم	
اختار اختبار	select case (…) { case (…) {…} case else {…}}	نعم	نعم	نعم
بيانات سلوكية لبرنامج				
سلوك بديل	alt {…}	نعم (انظر الملاحظة 1)	نعم	
إعادة تقييم سلوك بديل	repeat	نعم (انظر الملاحظة 1)	نعم	
سلوك تشذير	interleave {…}	نعم (انظر الملاحظة 1)	نعم	
تحكم عودة	return		نعم (انظر الملاحظة 4)	نعم
بيانات لمناولة بالتغيب				
نشط بالغيب	activate	نعم (انظر الملاحظة 1)	نعم	
أوقف التنشيط بالغيب	deactivate	نعم (انظر الملاحظة 1)	نعم	
عمليات تشكيل				
أوجد مكون اختبار متوازي	create		نعم	
وصل منفذ مكون بمنفذ مكون	connect		نعم	
فك توصيل منفذي مكون	disconnect		نعم	
تقابل منفذ مع سطح بيني لاختبار	map		نعم	
فك تقابل منفذ من سطح بيني لنظام اختبار	unmap		نعم	
احصل على قيمة مرجع مكون MTC	mtc		نعم	نعم
احصل على قيمة مرجع مكون سطح بيني لنظام اختبار	system		نعم	نعم
احصل على قيمة مرجع مكون	self		نعم	نعم
ابدأ تنفيذ سلوك مكون اختبار	start		نعم	
أوقف تنفيذ سلوك مكون اختبار	stop		نعم	
اسحب مكون اختبار من النظام	kill		نعم	
تأكد من انتهاء سلوك PTC	running		نعم	
تأكد من وجود PTC في نظام الاختبار	alive		نعم	
انتظر انتهاء سلوك PTC	done		نعم	

البيان	الكلمة المفتاحية أو الرمز المصاحب	يمكن أن يستخدم في تحكم وحدة	يمكن أن يستخدم في وظائف واختبارات مجردة وaltsteps	يمكن أن يستخدم في وظائف طلبت من مقاسات وبولاني وحراسات أو من تدميث altstep لتعاريف محلية
انتظر توقف PTC ليخرج	**killed**		نعم	
عمليات اتصالات				
أرسل رسالة	**send**		نعم	
نفذ نداء إجراء	**call**		نعم	
أجب على نداء إجراء من كيان بعيد	**reply**		نعم	
ولد استثناء (لنداء مقبول)	**raise**		نعم	
استقبل رسالة	**receive**		نعم	
ابدأ رسالة	**trigger**		نعم	
اقبل نداء إجراء من كيان بعيد	**getcall**		نعم	
ناول استجابة من نداء سابق	**getreply**		نعم	
احصل على استثناء (من كيان مطلوب)	**catch**		نعم	
تأكد من رسالة (حالية)/نداء مستقبل	**check**		نعم	
حرر صف انتظار منفذ	**clear**		نعم	
حرر صف انتظار وتمكن من الإرسال والاستقبال عند منفذ	**start**		نعم	
أخمد الإرسال وأخمد عمليات الاستقبال لمواءمة عند منفذ	**stop**		نعم	
أخمد الإرسال وأخمد عمليات الاستقبال لمواءمة رسائل/نداءات جديدة	**halt**		نعم	
عمليات مؤقت				
ابدأ مؤقت	**start**	نعم	نعم	
أوقف مؤقت	**stop**	نعم	نعم	
اقرأ وقت الانقضاء	**read**	نعم	نعم	
تأكد إذا كان المؤقت ينفذ	**running**	نعم	نعم	
حدث إمهال	**timeout**	نعم	نعم	
عمليات تحكم				
اضبط حكم محلي	**setverdict**		نعم	
احصل على حكم محلي	**getverdict**		نعم	نعم
إجراءات خارجية				
حاكي إجراء (SUT) خارجياً	**action**	نعم	نعم	
تنفيذ اختبارات مجردة				
نفذ اختبار مجرد	**execute**	نعم	نعم (انظر الملاحظة 2)	

الملاحظة 1 – يمكن أن يستخدم للتحكم في عمليات مؤقت فقط.

الملاحظة 2 – يمكن أن يستخدم فقط في وظائف وaltsteps التي تستخدم في تحكم وحدة.

الملاحظة 3 – لا يسمح بتغيير متغيرات مكون.

الملاحظة 4 – يمكن أن يستخدم في وظائف وaltsteps ولكن ليس في اختبار مجرد.

19 تعبيرات وعناصر أساسية لبرنامج

0.19 عـــام

يمكن أن تستخدم تعبيرات وعناصر أساسية لبرنامج في جزء تحكم وحدة ووظائف وaltsteps واختبارات مجردة لـ TTCN-3.

الجدول 12أ)/Z.140 – نظرة شاملة على بيانات أساسية لبرنامج

عناصر أساسية لبرنامج	
البيان	**الكلمة المفتاحية أو الرمز المتصاحب**
Assignments	`:=`
Logging	`log`
Label and Goto	`label / goto`
If-else	`if (…) { … } else { … }`
For loop	`for (…) { … }`
While loop	`while (…) { … }`
Do while loop	`do { … } while (…)`
Stop execution	`stop`
Select case	`select case (…) { case (…) {…} case else {…} }`

1.19 تعبيرات

0.1.19 عـــام

يسمح TTCN-3 بمواصفة تعبيرات باستخدام مشغلين معرفين في القسم 15. وتبني التعبيرات من تعبيرات (بسيطة) أخرى. ويمكن أن تستخدم التعبيرات وظائف عودة قيمة فقط. وتكون نتيجة تعبير هي قيمة نمط محدد ويكون المشغلون المستخدمون متوائمين مع نمط متأثرات.

مثـــال:

```
(x + y - increment(z))*3;
```

1.1.19 تعبيرات بولانية

يحتوي تعبير **boolean** فقط على قيم **boolean** و/أو مشغلين **boolean** و/أو مشغلين اتصاليين ويقيم قيمة **boolean** إما **true** أو **false**.

مثـــال:

```
((A and B) or (not C) or (j<10));
```

2.19 تخصيصات

يمكن تخصيص قيم لمتغيرات. ويدل على هذا الرمز ":=". وخلال تنفيذ تخصيص، يقيم الجانب الأيمن من التخصيص قيمة متوائمة مع نمط على الجانب الأيسر. ويكون تأثير تخصيص هو ربط المتغير بقيمة التعبير. ولا يحتوي التعبير على متغيرات غير مربوطة. وتحدث جميع التخصيصات بالترتيب الذي تظهر به، أي، معالجة من اليسار إلى اليمين.

مثـــال:

```
MyVariable := (x + y - increment(z))*3;
```

3.19 بيان Log

يوفر بيان **log** وسيلة لكتابة بند تسجيل واحد أو أكثر لجهاز تسجيل متصاحب مع تحكم اختبار أو مكون اختبار يستخدم فيه البيان. وتعرف البنود التي تسجل بواسطة قائمة فاصلة مفصولة في متغير بيان تسجيل. ويمكن أن تكون بنود تسجيل عناصر فردية للغة محددة في الجدول 12b أو تعبيرات تتألف من بنود التسجيل هذه.

يوصي بشدة ألا يكون لتنفيذ بيان تسجيل تأثير على سلوك اختبار. وخاصة، ينبغي على الوظائف المستخدمة في بيان تسجيل سواء صراحة أو ضمنياً ألا تُغير قيم متغير مكون أو منفذ أو حالة مؤقت، ولا ينبغي أن تغير قيمة أي من معلمات **inout** أو **out**.

مثـــال:

```
var integer myVar:= 1;
log("Line 248 in PTC_A: ", myVar, " (actual value of myVar)");
// The string "Line 248 in PTC_A: 1 (actual value of myVar)" is written to some log device
// of the test system
```

الملاحظة 1 – لا ينبغي أن تستخدم الوظائف المستخدمة مباشرة أو بطريقة غير مباشرة بيانات من غير **if...else** و**for** و**while** و**do..while** و**label** و**goto** و**return** و**mtc** و**system** و**self** و**running** و**read** و**getverdict** (PTC أو مؤقت) **read** و**getverdict**.

الملاحظة 2 – يعتبر خارج مدى هذه التوصية تعريف تسجيل معقد وتتبع مقدرات يمكن أن تكون أداة مستقلة.

الجدول Z.140/12b – عناصر لغة TTCN-3 التي يمكن تسجيلها

مستخدم في بيان تسجيل	ما يُسجل	التعليق
معرف معلمة وحدة	قيمة فعلية	
قيمة صرفية	قيمة	يشمل هذا أيضاً نصاً حراً
معرف ثابت معطيات	قيمة فعلية	
معرف ثابت خارجي	قيمة فعلية	
مطابق مؤقت	مؤقت فعلي أو قيم مجال ورموز مواءمة	
معرف متغير نمط معطيات	قيمة فعلية أو "UNITIALIZED"	انظر الملاحظتين 3 و4.
self و**mtc** و**system** أو معرف متغير نمط مكون	قيمة فعلية إذا خصصت اسماً مطابقاً مكوناً أو "UNITIALIZED"	عند تسجيل قيم فعلية انظر الملاحظات من 2 إلى 4. تسجل حالات المكون الفعلي طبقا للملاحظة 5.
عملية تنفيذ (مكون أو مؤقت)	قيمة عودة	**true** أو **false** في حالة مكون أو مؤقت، تتضمن مواصفة مصفوفات وعنصر صفيف.
عملية حية (مكون)	قيمة عودة	**true** أو **false** في حالة مصفوفات، تتضمن مواصفات عنصر صفيف.
مطابق منفذ	حالة فعلية	تسجل حالات منفذ طبقا للملاحظة 6.
معرف متغير نمط بالتغيب	حالة فعلية أو 'UNITIALIZED'	تسجل حالات بالتغيب طبقا للملاحظة 7. انظر أيضاً الملاحظات من 2 إلى 4.
اسم مؤقت	حالة فعلية	تسجل حالات مؤقت طبقا للملاحظة 8.
عملية قراءة	قيمة عودة	انظر 3.24.
وظائف معرفة مسبقاً	قيمة عودة	انظر الملحق C.
مطابق وظيفة	قيمة عودة	يسمح فقط بوظائف مع شرط عودة.
مطابق وظيفة خارجية	قيمة عودة	يسمح فقط بوظائف خارجية مع شرط عودة.
معرف معلمة رسمية	انظر عمود التعليق	يتبع تسجيل معلمات فعلية القواعد المحددة لعناصر لغة تحل محلها. وفي حالة معلمات قيمة لقيمة معلمة فعلية؛ وفي حالة معلمات مقاس–نمط، المقاس الفعلي أو قيمة مجال ورموز مواءمة؛ وفي حالة معلمات نمط مكون، مرجع مكون فعلي وما إلى ذلك؛ تسجل. ولمعلمات مؤقت، يسمح استخدام عملية قراءة ولنمط مكون ومعلمات مؤقت، واستخدام عملية تنفيذ أيضاً.

الجدول Z.140/12b – عناصر لغة TTCN-3 التي يمكن تسجيلها

الملاحظة 1 – تكون القيمة/المقاس الفعلي هي قيمة/مقاس لحظة تنفيذ بيان تسجيل.
الملاحظة 2 – يكون نمط قيمة مسجلة أداة مستقلة.
الملاحظة 3 – في حالة معرفات صفيف دون مواصفة عنصر صفيف، تسجل القيم الفعلية وأسماء مراجع مكون لجميع عناصر صفيف.
الملاحظة 4 – تسجل سلسلة "UNITIALIZED" فقط إذا كان بند التسجيل غير مسند (غير مدمث).
الملاحظة 5 – إن حالات مكون يمكن تسجيلها هي: Inactive وRunning وStopped وKilled (لمزيد من التفاصيل، انظر الملحق F).
الملاحظة 6 – إن حالات منفذ يمكن تسجيلها هي: Stopped وStarted (لمزيد من التفاصيل، انظر الملحق F).
الملاحظة 7 – إن حالات بالتغيب يمكن تسجيلها هي: Deactivated وActivated.
الملاحظة 8 – إن حالات مؤقت يمكن تسجيلها هي: Inactive وRunning وExpired (لمزيد من التفاصيل، انظر الملحق F).

4.19 بيان Label

يسمح بيان **label** بمواصفة لوسوم في اختبارات مجردة ووظائف وaltsteps وجزء تحكم وحدة. ويمكن استخدام بيان **label** بحرية مثل بيانات سلوكية لبرنامج TTCN-3 الأخرى طبقاً لقواعد التركيب المعرفة في الملحق A. ويمكن استخدامه قبل أو بعد بيان TTCN-3 ولكن ليس كأول بيان لبديل أو بديل قمة في بيان **alt** أو بيان **interleave** أو **altstep**. وتكون الوسوم المستخدمة التي تتبع الكلمة المفتاحية **label** وحيدة بين جميع الوسوم المعرفة في نفس اختبار مجرد أو وظيفة أو altstep أو جزء تحكم.

مثـــال:

```
label MyLabel;          // Defines the label MyLabel

// The labels L1, L2 and L3 are defined in the following TTCN-3 code fragment
 :
label L1;              // Definition of label L1
alt{
[] PCO1.receive(MySig1)
   {   label L2;         // Definition of label L2
       PCO1.send(MySig2);
       PCO1.receive(MySig3)
   }
[] PCO2.receive(MySig4)
   {   PCO2.send(MySig5);
       PCO2.send(MySig6);
       label L3;         // Definition of label L3
       PCO2.receive(MySig7);
   }
}
 :
```

5.19 بيان Goto

يمكن استخدام بيان **goto** في وظائف واختبارات مجردة وaltsteps وجزء تحكم وحدة TTCN-3. ويؤدي بيان **goto** قفزة لـ **label** .

يوفر بيان **goto** إمكانية القفز بحرية، أي، إلى الأمام والخلف، في تتابع بيانات والقفز خارج بيان مركب وحيد (مثل، عروة **while**) والقفز عبر مستويات عديدة خارج بيانات مركبة متداخلة (مثل، بدائل متداخلة). ومع ذلك، يقيد استخدام بيان **goto** بواسطة القواعد التالية:

أ) لا يُسمح بالقفز خارج أو في وظائف واختبارات مجردة وaltsteps وجزء تحكم وحدة TTCN-3.

ب) لا يُسمح بالقفز في تتابع بيانات معرفة في بيان مركب (أي، بيان **alt** وعروة **while** وعروة **for** وبيان **if-else** وعروة **do-while** وبيان **interleave**).

ج) لا يُسمح باستخدام بيان **goto** في بيان **interleave**.

مثـــال:

```
// The following TTCN-3 code fragment includes
 :
label L1;          // … the definition of label L1,
MyVar := 2 * MyVar;
if (MyVar < 2000) { goto L1; } // … a jump backward to L1,
MyVar2 := Myfunction(MyVar);
if (MyVar2 > MyVar) { goto L2; }    // … a jump forward to L2,
PCO1.send(MyVar);
PCO1.receive -> value MyVar2;
```

```
label L2;          // … the definition of label L2,
PCO2.send(integer: 21);
alt {
  [] PCO1.receive { }
  [] PCO2.receive(integer: 67) {
      label L3;        // … the definition of label L3,
      PCO2.send(MyVar);
      alt {
        [] PCO1.receive { }
        [] PCO2.receive(integer: 90) {
           PCO2.send(integer: 33);
           PCO2.receive(integer: 13);
           goto L4;// … a jump forward out of two nested alt statements,
           }
        [] PCO2.receive(MyError) {
           goto L3;// … a jump backward out of the current alt statement,
           }
        [] any port.receive {
           goto L2;// … a jump backward out of two nested alt statements,
           }
      }
     }
  [] any port.receive {
  goto L2;     // … and a long jump backward out of an alt statement.
     }
}
label L4;
 :
```

6.19 بيان If-else

يستخدم بيان **if-else**، المعروف أيضاً كبيان شَرطي، ليدل على تفريع في تدفق تحكم نتيجة لتعبيرات **boolean**. وتخطيطياً، يظهر البيان الشرطي كما يلي:

```
if (expression1)
   statementblock1
else
   statementblock2
```

حيث يشير $statementblock_x$ إلى فدرة بيانات.

```
if (date == "1.1.2005") { return ( fail ); }

if (MyVar < 10) {
      MyVar := MyVar * 10;
      log ("MyVar < 10");
}
else {
  MyVar := MyVar/5;
}
```

ويمكن أن يكون مخططاً معقداً أكثر:

```
if (expression1)
  statementblock1
else if (expression2)
  statementblock2
:
else if (expressionn)
  statementblockn
else
  statementblockn+1
```

وفي هذه الحالات، تعتمد قابلية القراءة بثقل على الأنساق، إلا أن الأنساق ليس له معنى تركيبي أو دلالي.

7.19 بيان For

يعرف بيان **for** عروة عداد. وتزداد قيمة متغير دليل أو تنخفض أو تتضاعف بطريقة يتم فيها الوصول بعد عدد معين من عروات التنفيذ إلى معايير إنهاء.

يحتوي بيان **for** على بيانين وتعبير **boolean**. ويكون التخصيص الأول ضرورياً لتدميث متغير دليل (أو عداد) العروة. وينهي تعبير **boolean** العروة ويستخدم التخصيص الثاني لتناول متغير الدليل.

المثـــال 1:

```
for (j:=1; j<=10; j:= j+1) { … }
```

ويعبر عن معيار الإنهاء لعروة التعبير **boolean**. ويجري التأكد منه في بداية كل تكرار عروة جديدة. وإذا قُيّم **true**، يستمر التنفيذ مع فدرة بيانات في بيان **for**؛ وإذا قيم **false**، يستمر التنفيذ مع بيان يتبع مباشرة عروة **for**.

يمكن الإعلان عن متغير دليل عروة **for** قبل استخدامه في بيان أو يمكن الإعلان عنه وتدميثه في رأسية بيان **for**. وإذا أعلن عن متغير الدليل ودمث في رأسية بيان **for**، يقتصر منظور متغير الدليل على جسم العروة، أي، يكون مرئياً فقط لجسم العروة.

المثـــال 2:

```
var integer j;          // Declaration of integer variable j
for (j:=1; j<=10; j:= j+1) { … }    // Usage of variable j as index variable of the for loop


for (var float i:=1.0; i<7.9; i:= i*1.35) { … }// Index variable i is declared and initialized
                                               // in the for loop header. Variable i only is
                                               // visible in the loop body.
```

8.19 بيان While

تنفذ عروة **while** طالما يتم الاحتفاظ بشرط العروة. ويجري التأكد من شرط العروة في بداية كل تكرار عروة جديدة. وإذا لم يتم الاحتفاظ بشرط العروة، فإن العروة تخرج ويستمر التنفيذ مع بيان، يتبع مباشرة عروة **while**.

مثـــال:

```
while (j<10){ … }
```

9.19 بيان Do-while

إن عروة **do-while** مماثلة لعروة **while** مع استثناء أن شرط العروة يتم التأكد منه في نهاية كل تكرار عروة. ويعني هذا أنه عند استخدام عروة **do-while** ينفذ السلوك مرة واحدة على الأقل قبل أن يقيم شرط العروة لأول مرة.

مثـــال:

```
do { … } while (j<10);
```

10.19 بيان تنفيذ Stop

ينهي بيان **stop** التنفيذ بطرق مختلفة يعتمد على السياق الذي يستخدم فيه. وعندما يستخدم في جزء تحكم وحدة أو في وظيفة يستخدمها جزء تحكم وحدة، ينهي تنفيذ جزء تحكم الوحدة. وعندما يستخدم في اختبار مجرد أو altstep أو وظيفة تنفذ في مكون اختبار، ينهي مكون الاختبار المعني.

مثـــال:

```
module MyModule {
    : // Module definitions
   testcase MyTestCase() runs on MyMTCType system MySystemType{
      var MyPTCType ptc:= MyPTCType.create; // PTC creation
      ptc.start(MyFunction());          // start PTC execution
       :       // test case behaviour continued
      stop     // stops the MTC, all PTCs and the whole test case
   }
   function MyFunction() runs on MyPTCType {
        :
      stop  // stops the PTC only, the test case continues
   }
   control {
        :   // test execution
      stop // stops the test campaign
   } // end control
  } // end module
```

ملاحظة – يكون علم دلالات بيان **stop** الذي ينهي مكون اختبار مماثل لعملية مكون توقف **self.stop** (انظر 6.22).

11.19 بيان Select Case

يكون بيان **select case** بديل لاستخدام بيانات **if .. else** عند مقارنة قيمة بقيمة واحدة أو عدة قيم أخرى. ويحتوي البيان على جزء رأسية وصفر أو فروع أكثر. ولا ينفذ أكثر من فرع واحد. وتخطيطياً، يظهر بيان **select case** كما يلي:

```
select (expression)
   {
   case (templateInstance1a, templateInstance1b,…)
      statementblock1
   case (templateInstance2a, templateInstance2b,…)
      statementblock2
   case else
      statementblock3
   }
```

حيث يشير templateInstance إلى مقاس معرف أو في الخط ويشير $statementblock_x$ إلى فدرة بيانات.

ملاحظة – يكافئ الشكل التخطيطي أعلاه الشكل التخطيطي التالي باستخدام بيانات if-else:

```
if (match(expression, templateInstance1a or match(expression, templateInstance1b
or …)
   statementblock1
else if (match(expression, templateInstance2a or match(expression, templateInstance2b or …)
   statementblock2

else
   statementblock3
```

في جزء الرأسية لبيان **select case**، يتوفر تعبير. ويبدأ كل فرع بالكلمة المفتاحية **case** تتبعها قائمة templateInstance (يمكن أن تحتوي قائمة فرع على عنصر وحيد) أو الكلمة المفتاحية **else** (وفرع else) وفدرة بيانات.

تكون جميع templateInstance في جميع فروع قائمة من نمط متوائم مع نمط التعبير في الرأسية. ويختار فرع قائمة وتنفذ فدرة بيانات لفرع مختار فقط، إذا تواءم أي templateInstance مع قيمة التعبير في رأسية البيان. وعند تنفيذ فِدرة بيانات لفرع مختار (أي، لا يقفز خارجاً بواسطة بيان **go to**)، يستمر التنفيذ مع بيان يتبع بيان اختبار مختار.

تنفذ دائماً فِدرة بيانات فرع else إذا لم يختار فرع آخر يسبقه نصاً فرع **else**.

تقيّم أفرع في ترتيبها النصي. وإذا لم يوائم أي من templateInstance-s قيمة التعبير في الرأسية ولا يحتوي البيان على فرع **else**، يستمر التنفيذ دون أي فروع **select case**.

مثـــال:

```
select (MyModulePar)  // where MyModulePar is of charstring type
{
  case ("firstValue")
      {
       log ("The first branch is selected");
      }
  case (MyCharVar, MyCharConst)
      {
       log ("The second branch is selected");
      }
  case else
      {
       log ("The value of the module parameter MyModulePar is selected");
      }
}
```

20 بيانات سلوكية لبرنامج

0.20 عـــام

يمكن استخدام بيانات سلوكية لبرنامج في اختبارات مجردة ووظائف وaltsteps، باستثناء:

أ) بيان **return** يستخدم فقط في وظائف؛

ب) بيان **alt** وبيان **interleave** وبيان **repeat** التي يمكن أن تستخدم أيضاً في تحكم وحدة.

تحدد بيانات سلوكية لبرنامج السلوك الدينامي لمكونات اختبار عبر منافذ اتصالات. ويمكن التعبير عن سلوك اختبار تتابعياً كمجموعة بدائل أو تركيبات من كلاهما. ويسمح مشغل تشذير بمواصفة تتابعات مشذرة أو بدائل.

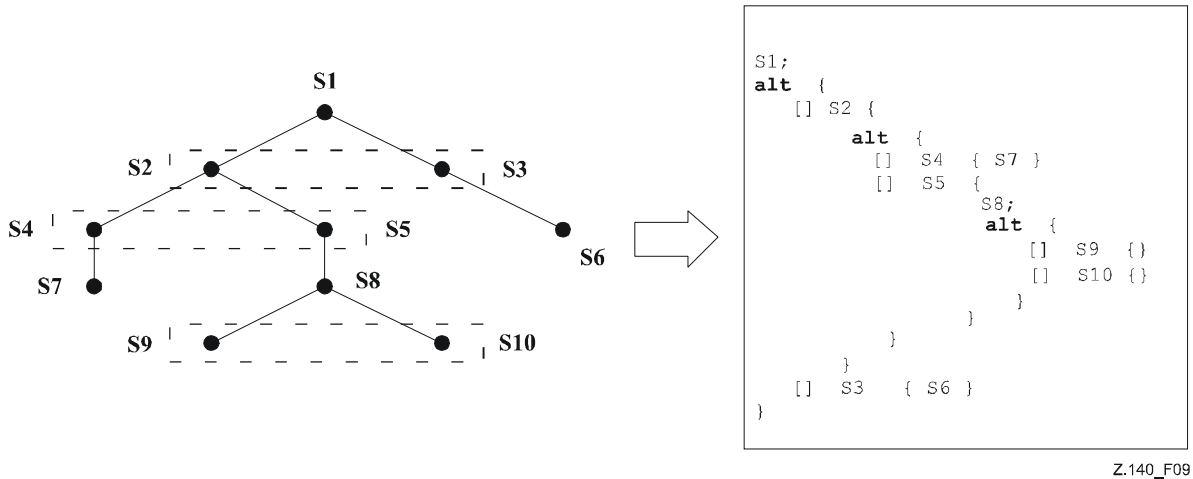
الجدول Z.140/13 – نظرة شاملة على بيانات سلوكية لبرنامج TTCN-3

بيانات سلوكية لبرنامج	
بيان	**الكلمة المفتاحية أو الرموز المتصاحبة**
سلوك بديل	**alt { … }**
إعادة تقييم بيانات alt	**repeat**
سلوك مشذّر	**interleave { … }**
تحكم عودة	**return**

1.20 سلوك بديل

0.1.20 عـــام

إن الشكل المعقد الأكثر لسلوك هو حيث تتابعات بيانات يعبر عنها كمجموعة من بدائل ممكنة في شكل شجرة لمسيرات تنفيذ، كما يبين الشكل 9.



الشكل Z.140/9 – توضيح سلوك بديل

يدل بيان **alt** على تفريع سلوك اختبار نتيجة استقبال ومناولة اتصالات و/أو أحداث مؤقت و/أو إنهاء مكونات اختبار متوازية، أي، متعلق باستخدام عمليات TTCN-3 **receive** و**trigger** و**getcall** و**getreply** و**catch** و**check** و**timeout** و**done** و**killed**. ويدل بيان **alt** على مجموعة أحداث ممكنة تتواءم مقابل لقطة خاصة (انظر 1.1.20).

مثـــال:

```
// Use of nested alternative statements
 :
alt {
  [] L1.receive(DL_REL_CO:*) {
      setverdict(pass);
      TAC.stop;
      TNOAC.start;
      alt {
        [] L1.receive(t_DL_EST_IN) {
            TNOAC.stop;
            setverdict(pass);
           }
        [] TNOAC.timeout {
           L1.send(t_DEL_EST_RQ);
           TAC.start;
           alt {
             [] L1.receive(DL_EST_CO:*) {
            TAC.stop;
            setverdict(pass)
            }
             [] TAC.timeout {
            setverdict(inconc);
                }
             [] L1.receive {
            setverdict(inconc)
                }
           }
       }
        [] L1.receive {
            setverdict(inconc)
           }
      }
     }
  [] TAC.timeout {
      setverdict(inconc)
     }
  [] L1.receive {
      setverdict(inconc)
     }
}
 :
```

1.1.20 تنفيذ سلوك بديل

عند دخول بيان **alt**، تؤخذ لقطة. وتعتبر لقطة أنها حالة جزئية لمكون اختبار تشمل جميع المعلومات الضرورية لتقييم شروط بولانية تحرس فروعاً بديلة وجميع مكونات اختبار معني متوقف وجميع أحداث إمهال معنية ورسائل قمة ونداءات وإجابات واستقبالات في صفوف انتظار منفذ واصل معنية. وأي مكون اختبار ومؤقت تم تحديد مرجعه في بديل واحد على الأقل في بيان **alt**، أو في بديل قمة لـ altstep ينفذ على أنه بديل في بيان **alt** أو منشط بالتغيب يعتبر أن له علاقة. ويرد وصف تفصيلي لعلم دلالات لقطة في علم الدلالات التشغيلي لـ TTCN-3 (التوصية ITU-T Z.143 [3]).

الملاحظة 1 – إن اللقطات هي وسائل مفهومية لوصف سلوك بيان **alt**. والخوارزميات الملموسة لمناولة لقطة يمكن أن نجدها في التوصية ITU-T Z.143 [3].

الملاحظة 2 – يفترض علم دلالات TTCN-3 أن أخذ لقطة هو آنيّ، أي، ليس له مدة. وفي التنفيذ الفعلي، قد يستغرق أخذ لقطة بعض الوقت وقد تحدث شروط تسابق. ومناولة شروط التسابق هذه هي خارج مدى هذه التوصية.

تعالج فروع بديلة في بيان **alt** وبدائل قمة لتنفيذ altsteps وaltsteps التي تنشُط بالتغيب بترتيب ظهورها. وإذا كانت تغيبات عديدة نشطة، يحدد الترتيب العكسي لتنشيطها ترتيب تقييم بدائل القمة في التغيبات. ويتم التوصل إلى فروع بديلة في تغيبات نشطة بواسطة آلية التغيب الواردة في القسم 21.

إن الفروع البديلة الفردية هي إما فروع يمكن حراستها بواسطة تعبير بولاني أو else-branches، أي، الفروع البديلة البادئة مع **[else]**.

تُختار دائماً else-branches وتنفذ عند الوصول إليها (انظر 3.1.20).

إن الفروع التي تحرسها تعبيرات بولانية إما تنفذ (*altstep-branch*) altstep أو تبدأ مع عملية **done** (*done-branch*)، أو عملية **killed** (*killed-branch*) أو عملية **timeout** (*timeout-branch*) أو عملية استقبال (*receiving-branch*)، أي، عملية **receive** أو **trigger** أو **getcall** أو **getreply** أو **catch** أو **check**. ويقوم تقييم حراسات بولانية على أساس اللقطة. وتعتبر حراسة بولانية *fulfilled* إذا لم تعرف حراسة بولانية، أو إذا قُيِّمت حراسة بولانية على أنها **true**. وتعالج الفروع وتنفذ بالطريقة التالية.

يختار *altstep-branch* إذا تم تلبية حراسة بولانية. ويسبب اختيار *altstep-branch* تنفيذ altstep مرجعي، أي، ينفذ altstep ويستمر تقييم اللقطة مع altstep.

يختار *done-branch* إذا تم تلبية حراسة بولانية وإذا كان مكون اختبار محدد في قائمة مكونات stopped للقطة. ويسبب الاختيار تنفيذ فِدرة بيانات تتبُّع عملية **done**. ولا يكون لعملية **done** نفسها تأثيراً إضافياً.

يختار *killed-branch* إذا تم تلبية حراسة بولانية وإذا كان مكون اختبار محدد في قائمة مكونات killed للقطة. ويسبب الاختيار تنفيذ فدرة بيانات تتبع عملية **killed**. ولا يكون لعملية **killed** نفسها تأثير إضافي.

يختار *timeout-branch* إذا تم تلبية حراسة بولانية وإذا كان حدث إمهال في قائمة إمهال للقطة. ويسبب الاختيار تنفيذ عملية **timeout** المحددة، أي، إزالة حدث إمهال من قائمة إمهال، وتنفيذ فدرة بيانات تتبع عملية **timeout**.

يختار *receiving-branch* إذا تم تلبية حراسة بولانية وإذا تم تلبية معايير مواءمة لعملية استقبال بواسطة إحدى الرسائل أو النداءات أو الإجابات أو لاستثناءات في اللقطة. ويسبب الاختيار تنفيذ عملية استقبال، أي، إزالة رسالة مواءمة أو نداء أو إجابة أو استثناء من صف انتظار المنفذ، ويمكن أن يكون تخصيص لمعلومات مستقبَلة لمتغير وتنفيذ فدرة بيانات تتبع عملية استقبال. وفي حالة عملية **trigger**، تُزال أيضاً رسالة قمة لصف انتظار إذا كانت حراسة بولانية تم تلبيتها ولكن لم تتم تلبية معايير المواءمة. وفي هذه الحالة، لا تنفذ فدرة بيانات لبديل ما.

الملاحظة 3 – يصف علم دلالات TTCN-3 تقييم لقطة كسلسلة لأعمال غير مرئية لمكون اختبار. ولا يفترض علم الدلالات أن تقييم لقطة له مدة. وخلال تقييم لقطة، يمكن أن تتوقف مكونات اختبار وتمهل المؤقتات ويمكن أن تدخل رسائل جديدة أو نداءات أو إجابات أو استثناءات صف انتظار منفذ المكوِّن. ومع ذلك، لا تغير هذه الأحداث اللقطة الفعلية ومن ثم لا ينظر فيها لتقييم لقطة.

إذا لم يكن من الممكن اختيار أو تنفيذ أي من الفروع البديلة في بيان **alt** وبدائل قمة في **alt** منفذ وتغييات نشطة، ينفذ بيان **alt** مرة ثانية، أي، تؤخذ لقطة جديدة، ويكرر تقييم الفروع البديلة مع اللقطة الجديدة. ويستمر الإجراء المتكرر حتى يتم اختيار فرع بديل وينفذ أو يتوقف اختبار مجرد بواسطة مكون آخر أو بواسطة نظام اختبار (مثلاً، بسبب أن MTC توقف) أو مع خطأ دينامي.

يتوقف الاختبار المجرد ويدل على خطأ دينامي إذا سُدّ مكون اختبار بالكامل. ويعني هذا عدم اختيار أي من البدائل ولا ينفذ مكون اختبار له علاقة ولا ينفذ مؤقت له علاقة، وتحتوي جميع المنافذ المعنية على رسالة أو نداء أو إجابة أو استثناء واحد على الأقل لا يتواءم.

الملاحظة 4 – إن الإجراء المتكرر بأخذ لقطة كاملة وإعادة تقييم جميع البدائل هو وسيلة مفهومية لوصف علم دلالات بيان **alt**. وتكون الخوارزمية الملموسة التي تنفذ علم الدلالات هذه خارج مدى هذه التوصية.

2.1.20 اختيار/عدم اختيار بديل

إذا لزم الأمر، من الممكن إقرار صلاحية/عدم إقرار صلاحية بديل بواسطة تعبير بولاني موضوع بين أقواس **'[]'** للبديل.

يمكن أن يكون لتقييم بديل حراسة تعبير بولاني تأثيرات جانبية. ولتجنب التأثيرات الجانبية التي تسبب عدم الاتساق بين اللقطة الفعلية وحالة المكوّن، تنطبق نفس التقييدات مثل التقييدات على تدميث تعاريف محلية في altsteps (انظر 1.2.2.16).

تكون الأقواس المربعة المفتوحة والمغلقة **'['']'** موجودة عند بدء كل بديل، حتى إذا كانت فارغة. ولا يساعد هذا قابلية القراءة فحسب، بل أيضاً ضروري لتمييز تركيبي لبديل واحد عن آخر.

مثـــال:

```
// Use of alternative with Boolean expressions (or guard)
 :
alt {
  [x>1] L2.receive {       // Boolean guard/expression
      setverdict(pass);
     }
  [x<=1] L2.receive {      // Boolean guard/expression
      setverdict(inconc);
     }
}
 :
```

3.1.20 فرع Else في بدائل

يمكن تعريف أي فرع في بيان **alt** كفرع else بواسطة تضمين الكلمة المفتاحية **else** بين أقواس الفتح والغلق في بداية البديل. ولا يحتوي فرع else على أي من الأعمال المسموح بها في الفروع التي يحرسها تعبير بولاني (أي، call **altstep** أو **done** أو **killed** أو **timeout** أو عملية استقبال). وتنفذ فدرة بيان فرع else دائماً إذا لم يسبق فرع else بديل آخر نصي.

مثـــال:

```
// Use of alternative with Boolean expressions (or guard) and else branch
 :
alt {
  [x>1] L2.receive {
  setverdict(pass);
     }
  [x<=1] L2.receive {
  setverdict(inconc);
     }
  [else] {     // else branch
  MyErrorHandling();
  setverdict(fail);
  stop;
     }
}
 :
```

ينبغي ملاحظة أن آلية تغيب (انظر القسم 21) تنفذ دائماً في نهاية جميع البدائل. وإذا عرّف فرع **else**، لا تطلب آلية التغيب أبداً، أي، لا تدخل تغيبات نشطة أبداً.

الملاحظة 1 – من الممكن أيضاً استخدام **else** في altsteps.

الملاحظة 2 – يسمح باستخدام بيان **repeat** في فرع **else**.

الملاحظة 3 – يسمح بتعريف أكثر من فرع else واحد في بيان alt أو في altstep، ومع ذلك، ينفذ فرع else الأول دائماً.

4.1.20 فارغ

5.1.20 إعادة تقييم بيانات alt

يمكن تحديد إعادة تقييم بيان **alt** باستخدام بيان **repeat** (انظر 2.20).

مثـــال:

```
alt {
  [] PCO3.receive {
  count := count + 1;
  repeat       // usage of repeat
     }
  [] T1.timeout { }
  [] any port.receive {
  setverdict(fail);
  stop;
     }
}
```

6.1.20 تنفيذ altsteps كبدائل

يسمح TTCN-3 بتنفيذ بدائل في بيانات **alt** (انظر 3.2.16).

مثـــال:

```
 :
alt {
  [] PCO3.receive { }
  [] AnotherAltStep(); // explicit call of altstep AnotherAltStep as alternative
         // of an alt statement
  [] MyTimer.timeout { }
}
  :
```

2.20 بيان Repeat

يسبب بيان **repeat**، عندما يستخدم في فدرة بيانات وإعلانات لبدائل بيانات **alt**، إعادة تقييم بيان **alt**، أي، تؤخذ لقطة جديدة وتقيّم بدائل بيان **alt** بترتيب مواصفتها. وعندما يستخدم في فدرات بيانات وإعلانات استجابة، وأجزاء مناولة استثناء لنداءات إجراء سد، يسبب بيان repeat إعادة تقييم الاستجابة وجزء مناولة استثناء لنداء (انظر 5.1.3.23).

المثـــال 1:

```
// Usage of repeat in an alt statement
alt {
  [] PCO3.receive {
  count := count + 1;
  repeat    // usage of repeat
     }
  [] T1.timeout { }
  [] any port.receive {
   setverdict(fail);
   stop;
     }
}
```

إذا استخدم بيان **repeat** في بديل قمة في تعريف altstep، يسبب لقطة جديدة وإعادة تقييم بيان **alt** من altstep الذي طُلب منه. ويمكن أن يتم نداء altstep ضمنياً بواسطة آلية تغيب (انظر القسم 21) أو صراحة في بيان **alt** (انظر 6.1.20).

المثـــال 2:

```
// Usage of repeat in an altstep
altstep AnotherAltStep() runs on MyComponentType {
  [] PCO1.receive{
      setverdict(inconc);
      repeat       // usage of repeat
     }
  [] PCO2.receive {}

 }
```

3.20 سلوك مشذر

يسمح بيان **interleave** بتحديد حدوث مشذر ومناولة بيانات **done** و**killed** و**timeout** و**receive** و**trigger** و**getcall** و**catch** و**check**.

لا تستخدم بيانات نقل تحكم لنداء مباشر لـ **for**، **while**، **do-while**، **goto**، **activate**، **deactivate**، **stop**، **repeat**، **return** كبدائل ونداءات (مباشرة وغير مباشرة) لتعاريف معرّفة لمستعمل، تشمل عمليات اتصالات، في بيانات **interleave**. وبالإضافة إلى ذلك، لا يسمح بحراسة فروع بيان **interleave** مع تعبيرات بولانية (أي، يكون '[ ]' فارغاً دائماً). ولا يُسمح أيضاً بتحديد فروع **else** في سلوك مشذر.

يمكن دائماً إحلال السلوك المشذر بمجموعة متكافئة لبدائل متداخلة. ويرد في التوصية ITU-T Z.143 [3] إجراءات هذا الإحلال وعلم الدلالات التشغيلي للتشذير.

وتكون قاعدة تقييم بيان تشذير هو ما يلي:

أ) عندما ينفذ بيان استقبال، تنفذ البيانات غير الاستقبال التالية بعد ذلك حتى يتم التوصل إلى بيان الاستقبال التالي أو ينتهي التتابع المشذر؛

ملاحظة – إن بيانات استقبال هي بيانات TTCN-3 يمكن أن تحدث في مجموعات بدائل، أي، **receive**، **check**، **trigger**، **getcall**، **getreply**، **catch**، **done**، **killed**، **timeout**. وتدل بيانات غير استقبال على جميع بيانات نقل أخرى غير التحكم يمكن استخدامها في بيان **interleave**.

ب) ثم يستمر التقييم بواسطة أخذ اللقطة التالية.

ويُعرَّف علم الدلالات التشغيلي للتشذير بالكامل في التوصية ITU-T Z.143 [1].

مثـــال:

```
// The following TTCN-3 code fragment
 :
interleave {
[] PCO1.receive(MySig1)
   {   PCO1.send(MySig2);
       PCO1.receive(MySig3);
   }
[] PCO2.receive(MySig4)
   {   PCO2.send(MySig5);
       PCO2.send(MySig6);
       PCO2.receive(MySig7);
   }
}
:

// is a shorthand for
:
alt {
[] PCO1.receive(MySig1)
   {   PCO1.send(MySig2);
       alt {
       []   PCO1.receive(MySig3)
       {    PCO2.receive(MySig4);
            PCO2.send(MySig5);
            PCO2.send(MySig6);
            PCO2.receive(MySig7)
       }
       []   PCO2.receive(MySig4)
       {    PCO2.send(MySig5);
            PCO2.send(MySig6);
            alt {
            []   PCO1.receive(MySig3) {
                 PCO2.receive(MySig7); }
            []   PCO2.receive(MySig7) {
                 PCO1.receive(MySig3); }
       }
            }
       }
   }
[] PCO2.receive(MySig4)
   {   PCO2.send(MySig5);
       PCO2.send(MySig6);
       alt {
       []   PCO1.receive(MySig1)
        {   PCO1.send(MySig2);
       alt {
       []   PCO1.receive(MySig3)
        {   PCO2.receive(MySig7);
        }
       []   PCO2.receive(MySig7)
        {   PCO1.receive(MySig3);
        }
       }
        }
       []   PCO2.receive(MySig7)
        {   PCO1.receive(MySig1);
            PCO1.send(MySig2);
            PCO1.receive(MySig3);
       }
       }
   }
     }

     :
```

4.20 بيان Return

ينهي بيان **return** تنفيذ وظيفة أو altstep ويعيد التحكم إلى النقطة التي طلبت منها الوظيفة أو altstep. وعندما يستخدم في وظائف، يمكن أن يتصاحب بيان **return** خيارياً مع قيمة عودة.

ملاحظة – إن بيان **return**، عندما يستخدم في altsteps، له نفس التأثير كما لو كان نهاية فدرة بيانات وإعلانات لبديل مختار قد تم التوصل إليه، مثلاً، عندما يطلب altsteps من بيان **alt**، يستمر التنفيذ مع أول بيان يتبع بيان **alt**.

مثـــال:

```
function MyFunction() return boolean {
   :
  if (date == "1.1.2005") {
     return false; // execution stops on the 1.1.2000 and returns the boolean false
  }
   :
  return true; // true is returned
}

function MyBehaviour() return verdicttype {
   :
  if (MyFunction()) {
     setverdict(pass); // use of MyFunction in an if statement
  }
  else {
     setverdict(inconc);
  }
   :
  return getverdict; // explicit return of the verdict
}
```

21 مناولة بالتغيب

0.21 عـــام

يسمح TTCN-3 بتنشيط altsteps (انظر 2.16) كتغيبات. ولكل مكون اختبار، تخزن التغيبات، أي، altsteps منشطة، كقائمة مرتبة. وترد التغيبات بالترتيب العكسي لتنشيطها، أي، يكون آخر تغيب منشط هو أول عنصر في قائمة التغيبات النشطة. وتعمل عمليات TTCN-3 **activate** (انظر 3.21) و**deactivate** (انظر 4.21) في قائمة التغيبات. ويضع **activate** تغيباً جديداً كأول عنصر في القائمة ويزيل **deactivate** تغيباً من القائمة. ويمكن تعريف تغيب في قائمة تغيبات بواسطة مرجع تغيب يولد كنتيجة لعملية **activate** متناظرة.

الجدول Z.140/14 – نظرة شاملة على بيان TTCN-3 لمناولة بالتغيب

بيانات لمناولة بالتغيب	
البيان	**الكلمة المفتاحية أو الرمز المتصاحب**
تنشيط بالتغيب	**Activate**
وقف تنشيط بالتغيب	**deactivate**

1.21 آلية التغيُّب

ينفذ آلية التغيب في نهاية كل بيان **alt**، نتيجة للقطة فعلية، إذا لم يكن من الممكن تنفيذ أي من البدائل. وتنفذ آلية تغيب منفذة أول altstep في قائمة التغيبات، أي، آخر تغيب منشط، وتنتظر نتيجة انتهائه. ويمكن أن يكون الانتهاء ناجحاً أو غير ناجح. ويعني عدم النجاح أنه لا يمكن اختيار أي من بدائل القمة لـ **altstep** (انظر 2.16) التي تعرف سلوك تغيب، ويعني ناجح أن أحد بدائل القمة لتغيب قد تم اختياره ونُفذ.

وفي حالة انتهاء غير ناجح، تنفذ آلية التغيّب التغيّب التالي في القائمة. وإذا انتهى آخر تغيب في القائمة دون نجاح، تعود آلية التغيب إلى المكان في بيان **alt** الذي نُفذت فيه، أي، في نهاية بيان **alt**، وتدل على تنفيذ تغيب غير ناجح. وتجري الدلالة على تنفيذ تغيب غير ناجح إذا كانت قائمة التغيبات فارغة.

ويمكن أن يسبب تنفيذ تغيب غير ناجح لقطة جديدة أو خطأ دينامي إذا سد مكون الاختبار (انظر 1.20).

وفي حالة انتهاء ناجح، يمكن للتغيب إما أن يوقف مكون الاختبار بواسطة بيان **stop** أو يستمر تدفق التحكم الرئيسي لمكون الاختبار مباشرة بعد بيان **alt** الذي طلبت منه آلية التغيب أو يأخذ مكون الاختبار لقطة جديدة ويعيد تقييم بيان . ويتعين تحديد الأخير بواسطة بيان **alt** (انظر 2.20). وإذا انتهي بديل قمة مختار دون بيان **repeat**، يستمر تدفق تحكم مكون الاختبار مباشرة بعد بيان **alt**.

ملاحظة – لا يقيد TTCN-3 تنفيذ آلية تغيب. ويمكن تنفيذه، مثلاً، في شكل عملية تطلب ضمنيا في نهاية كل بيان **alt** أو في شكل خيط منفصل يكون مسؤول فقط عن مناولة بالتغيب. والمتطلب الوحيد هو أن التغيبات تطلب بترتيب عكسي لتنشيطها عندما تنفذ آلية التغيب.

2.21 مراجع تغيب

إن مراجع التغيب هي مراجع وحيدة لتنشيط متغيبات. ويولد مرجع تغيب وحيد مكون اختبار عندما ينشط altstep كمتغيب، أي، مرجع تغيب هو نتيجة عملية **activate** (انظر 3.21).

إن مراجع تغيب لها نمط **default** خاص ومعرف مسبقاً. ويمكن استخدام متغيرات نمط **default** لمناولة تغييات نشطة في مكونات اختبار. وتكون القيمة الخاصة **null** متاحة لتدل على مرجع تغيب غير معرف، مثلاً، لتدميث متغيرات لمناولة مراجع تغيب.

تستخدم مراجع تغيب في عمليات **deactivate** (انظر 4.21) لتعريف التغيب الذي ينشط.

يُستبان تمثيل معطيات فعلية لنمط **default** خارجياً بواسطة نظام الاختبار. ويسمح هذا بتحديد اختبارات مجردة بشكل مستقل عن أي بيئة وقت تنفيذ TTCN-3 حقيقي؛ وبمعنى آخر، لا يقيد TTCN-3 تنفيذ نظام اختبار فيما يتعلق بمناولة وتعرف متغيبات.

مثـــال:

```
// Declaration of a variable for the handling of defaults
// and initialization with the null value
var default MyDefaultVar := null;
 :
// Usage of MyDefaultVar for storing an activated default
MyDefaultVar := activate(MyDefAltStep()); // MyDefAltStep is activated as default
 :
// Usage of MyDefaultVar for the deactivation of default MyDefAltStep
deactivate(MyDefaultVar);
 :
```

3.21 عملية Activate

0.3.21 عام

تستخدم عملية **activate** لتنشيط altsteps كمتغيبات. وتضع عملية **activate** altsteps المرجعي كأول عنصر في قائمة المتغيبات وتعيد مرجع تغيب. ويكون مرجع التغيب مُعرف وحيد للمتغيب ويمكن أن يستخدم في عملية **deactivate** لوقف تنشيط المتغيب.

ويكون تأثير عملية **activate** محلياً لمكون الاختبار الذي طُلب فيه. ويعني هذا أنه لا يمكن لمكون اختبار تنشيط متغيب في مكون اختبار آخر.

المثـــال 1:

```
 :
// Declaration of a variable for the handling of defaults
var default MyDefaultVar := null;
 :
// Declaration of a default reference variable and activation of an altstep as default
var default MyDefVarTwo := activate(MySecondAltStep());
 :
// Activation of altstep MyAltStep as a default
MyDefaultVar := activate(MyAltStep()); // MyAltStep is activated as default
 :
```

يمكن طلب عملية **activate** دون ادخار مرجع تغيب معاد. وهذا الشكل مفيد في اختبارات مجردة لا تتطلب وقف تنشيط واضح لمتغيب منشط، أي، يتم وقف تنشيط متغيب ضمنياً عند انتهاء MTC.

المثـــال 2:

```
// Activation of an altstep as a default, without assignment of default reference
activate(MyCommonDefault());
```

1.3.21 تنشيط altsteps مُعلمة

إن المعلمات الفعلية لـ altstep الُمعلمة (انظر 1.2.16)، التي ينبغي تنشيطها كمتغيب، توفر في بيان **activate** المتناظر. ويعني هذا أن المعلمات الفعلية يتعين أن تسند إلى التغيب في وقت تنشيطها (وليس، مثلاً، وقت تنفيذها بواسطة آلية تغيب). ويُعلن عن جميع مطابقات مؤقت في قائمة معلمات فعلية كمؤقتات محلية لنمط مكون (انظر 1.5.8).

مثـــال:

```
altstep MyAltStep2 ( integer    par_value1, MyType par_value2,
                     MyPortType par_port,   timer  par_timer )
{
 :
}

function MyFunc () runs on MyCompType
{ :
var default MyDefaultVar := null;

MyDefaultVar := activate(MyAltStep2(5, myVar, myCompPort, myCompTimer);
   // MyAltStep2 is activated as default with the actual parameters 5 and
   // the value of myVar. A change of myVar before a call of MyAltStep2 by
   // the default mechanism will not change the actual parameters of the call.
 :
}
```

4.21 عملية Deactivate

تستخدم عملية **deactivate** لوقف تنشيط متغيبات، أي، altsteps منشطة في السابق. وتزيل عملية **deactivate** التغيب المرجعي من قائمة المتغيبات.

ويكون تأثير عملية **deactivate** محلياً لمكون الاختبار الذي طُلب فيه. ويعني هذا أنه لا يمكن لمكون اختبار وقف تنشيط متغيب في مكون اختبار آخر.

توقف عملية **deactivate** تنشيط دون معلمة جميع متغيبات مكون اختبار.

لا يكون لطلب عملية **deactivate** مع قيمة خاصة **null** أي تأثير. إن طلب عملية **deactivate** مع مرجع تغيب غير محدد، مثلاً، مرجع قديم لتغيب تم وقف تنشيطه أو متغير مرجع تغيب غير مدمث، يسبب خطأ في التنفيذ.

مثـــال:

```
 :
var default MyDefaultVar := null;
var default MyDefVarTwo := activate(MySecondAltStep());
var default MyDefVarThree := activate(MyThirdAltStep());
 :
MyDefaultVar := activate(MyAltStep());
 :
deactivate(MyDefaultVar); // deactivates MyAltStep
 :
deactivate; // deactivates all other defaults, i.e., in this case     MySecondAltStep
      // and MyThirdAltStep
 :
```

22 عمليات تشكيل

0.22 عـــام

تستخدم عمليات تشكيل (انظر الجدول 15) لإنشاء والتحكم في مكونات اختبار. وتستخدم هذه العمليات فقط اختبارات مجردة ووظائف وaltsteps TTCN-3 (أي، ليس في جزء تحكم وحدة).

الجدول Z.140/15 – نظرة شاملة على عمليات تشكيل TTCN-3

العملية	الشرح	أمثلة قواعد التركيب
عمليات توصيل		
connect	توصل منفذ مكون اختبار واحد بمنفذ مكون اختبار آخر	**connect**(ptc1:p1, ptc2:p2);
disconnect	تفك توصيل منفذين موصولين أو أكثر	**disconnect**(ptc1:p1, ptc2:p2);
map	تقابل منفذ لمكون اختبار واحد بمنفذ سطح بيني لنظام اختبار	**map**(ptc1:q, **system**:sutPort1);
unmap	تفك تقابل منفذين متقابلين أو أكثر	**unmap**(ptc1:q, **system**:sutPort1);

الجدول Z.140/15 – نظرة شاملة على عمليات تشكيل TTCN-3

عمليات مكون اختبار		

create	خلق مكون اختبار عادي أو حي؛ يتم التمييز بين مكونات اختبار عادية وحية خلال الخلق (يسلك MTC كما لو كان مكون اختبار عادي)	Non-alive test components:
		`var PTCType c := PTCType.create;`
		Alive test components:
		`var PTCType c := PTCType.create alive;`

start	بدء سلوك اختبار على مكون اختبار. لا تؤثر بداية سلوك على حالة متغيرات مكون أو مؤقتات أو منافذ	`c.start(PTCBehaviour());`
stop	وقف سلوك اختبار على مكون اختبار	`c.stop;`
kill	يسبب عدم وجود مكون اختبار	`c.kill;`
alive	يعيد true إذا كان مكون الاختبار قد خلق وعلى استعداد للتنفيذ أو يقوم فعلا بتنفيذ سلوك؛ وإلا يعيد false	`if (c.alive) …`
running	يعيد true طالما مكون الاختبار ينفذ سلوك؛ وإلا يعيد false	`if (c.running) …`
done	يتأكد من أن الوظيفة المنفذة على مكون اختبار قد انتهت	`c.done;`
killed	يتأكد من أن مكون الاختبار قد توقف عن الوجود	`c.killed { … }`
عمليات مراجع		
mtc	يحصل على مرجع لـ MTC	`connect(mtc:p, ptc:p);`
system	يحصل على مرجع لسطح بيني لنظام اختبار	`map(c:p, system:sutPort);`
self	يحصل على مرجع مكون اختبار ينفذ هذه العملية	`self.stop;`

1.22 عملية Create

إن MTC هو مكون الاختبار الوحيد الذي يتم خلقه أوتوماتياً عندما يبدأ اختبار مجرد. وتخلق جميع مكونات اختبار أخرى (PTC) صراحة خلال تنفيذ اختبار بواسطة عملية **create**. ويتم خلق مكون مع مجموعة منافذه الكاملة حيث تكون صفوف انتظار الدخل فارغة مع مجموعة كاملة من الثوابت والمتغيرات والمؤقتات. وفضلاً عن ذلك، إذا عرف منفذ على أنه من النمط **in** أو **inout** يكون في حالة استماع على استعداد لاستقبال حركة عبر التوصيل.

يُعاد ضبط جميع متغيرات ومؤقتات مكون على قيمها الأولية (إن وجدت)، ويعاد ضبط جميع ثوابت مكون على قيمها المخصصة عندما يخلق المكون صراحة أو ضمنياً.

يميز نوعان من PTCs: PTC يمكن أن ينفذ وظيفة سلوك فقط مرة واحدة وPTC يحتفظ به حياً بعد انتهاء وظيفة سلوك ولهذا، يمكن إعادة استخدامه لتنفيذ وظيفة أخرى. ويخلق الأخير باستخدام كلمة مفتاحية إضافية **alive**. ويجب تدمير PTC لنمط حي صراحة باستخدام عملية **kill** (انظر 9.22)، بينما PTC غير حي يدمر ضمنياً بعد إنهاء وظيفة سلوكه. وإنهاء اختبار مجرد، أي، ينهي MTC جميع PTC التي مازالت موجودة، إن وجدت.

نظراً لتدمير جميع مكونات ومنافذ اختبار ضمنياً عند الانتهاء من كل اختبار مجرد، يخلق كل اختبار مجرد بالكامل تشكيل وتوصيلات مكوناته المطلوبة عند تنفيذه.

تعيد عملية **create** مرجع مكون وحيد لمطابقة تم خلقها جديداً. ويخزن المرجع الوحيد لمكون في متغير (انظر 7.8) ويمكن استخدامه لتوصيل مطابقات ولأغراض اتصالات مثل إرسال واستقبال.

وخيارياً، يمكن أن يتصاحب اسم مع مطابقة مكون تم خلقه جديداً. ويكون الاسم قيمة **charstring** وعند تخصيصه يظهر كمتغير لوظيفة **create**. ويتصاحب نظام الاختبار مع أسماء 'MTC' مع MTC و'SYSTEM' مع السطح البيني لنظام الاختبار أوتوماتياً عند الخلق. ولا يطلب أن تكون أسماء مكونات متصاحبة وحيدة.

ملاحظة – يستخدم اسم مطابقة مكون لأغراض التسجيل (انظر 3.19) فقط ولا يستخدم للإشارة إلى مطابقة المكون (يستخدم مرجع المكون لهذا الغرض) وليس له تأثير على المواءمة.

مثـــال:

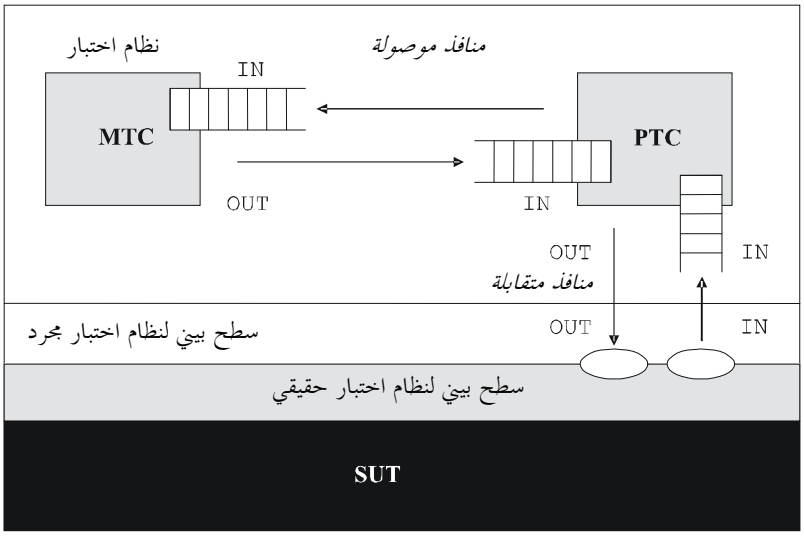
```
// This example declares variables of type MyComponentType, which is used to store the
// references of newly created component instances of type MyComponentType which is the
// result of the create operations. An associated name is allocated to some of the created
// component instances.
 :
var MyComponentType MyNewComponent;
var MyComponentType MyNewestComponent;
var MyComponentType MyAliveComponent;
var MyComponentType MyAnotherAliveComponent;
 :
MyNewComponent := MyComponentType.create;
MyNewestComponent := MyComponentType.create("Newest");
MyAliveComponent := MyComponentType.create alive;
MyAnotherAliveComponent := MyComponentType.create("Another Alive") alive;
 :
```

يمكن خلق مكونات في أي نقطة في تعريف سلوك موفرة مرونة كاملة فيما يتعلق بتشكيلات دينامية (أي، يمكن لأي مكون أن يخلق PTC آخر). وتتبع رؤية مراجع مكون نفس قواعد منظور مثل المتغيرات، ولتحديد مرجع مكونات خارج منظور خلقها، يمرر مرجع مكون كمعلمة أو كمجال في رسالة.

2.22 عمليات Connect وMap

0.2.22 عـــام

يمكن وصل منافذ مكون اختبار بمكونات أخرى أو بمنافذ سطح بيني لنظام اختبار. وفي حالة توصيلات بين مكوني اختبار، تستخدم عملية **connect**. وعند توصيل مكون اختبار بسطح بيني لنظام اختبار، تستخدم عملية **map**. وتوصل عملية **connect** مباشرة منفذ واحد بآخر مع جانب **in** موصل بجانب **out** والعكس بالعكس. ويمكن النظر إلى عملية **map** على الجانب الآخر كترجمة اسم تعرف لكيفية تحديد مرجع تدفقات الاتصالات.



الشكل Z.140/10 – توضيح عمليات connect وmap

مع كل من عمليتي **connect** و**map**، تعرف المنافذ التي توصل بواسطة مراجع مكون لمكونات توصل وأسماء منافذ توصل.

تعرف عملية MTC **mtc**، وتعرف عملية **system** السطح البيني لنظام اختبار وتعرف عملية **self** مكون الاختبار الذي طلب فيه **self** (انظر 4.22). ويمكن استخدام جميع هذه العمليات لتعريف وتوصيل منافذ.

يمكن طلب عمليتي **connect** و**map** من أي تعريف سلوك باستثناء جزء تحكم وحدة. ومع ذلك، قبل طلب أي عملية، تكون المكونات التي توصل قد خلقت ومراجع مكونها معروفة مع أسماء المنافذ ذات العلاقة.

تسمح عمليتا **map** و**connect** بتوصيل منفذ إلى أكثر من منفذ آخر. ولا يسمح بتوصيل منفذ متقابل أو تقابل مع منفذ موصل.

مثـــال:

```
      // It is assumed that the ports Port1, Port2, Port3 and PCO1 are properly defined and declared
      // in the corresponding port type and component type definitions
       :
      var MyComponentType  MyNewPTC;
       :
      MyNewPTC := MyComponentType.create;
       :
       :
      connect(MyNewPTC:Port1, mtc:Port3);
      map(MyNewPTC:Port2, system:PCO1);
       :
       :
// In this example a new component of type MyComponentType is created and its reference stored
      // in variable MyNewPTC. Afterwards in the connect operation, Port1 of this new component
      // is connected with Port3 of the MTC. By means of the map operation, Port2 of the new
component
      // is then connected to port PCO1 of the test system interface
```

1.2.22 التوصيلات والتقابلات المتسقة

لكل من عمليتي **connect** و**map**، يسمح فقط بتوصيلات متسقة.

مع افتراض ما يلي:

أ) منافذ PORT1 وPORT2 هي المنافذ التي توصل؛

ب) تعرف inlist-PORT1 الرسائل أو الإجراءات للاتجاه الداخل لـ PORT1؛

ج) تعرف outlist-PORT1 الرسائل أو الإجراءات للاتجاه الخارج لـ PORT1؛

د) تعرف inlist-PORT2 الرسائل أو الإجراءات للاتجاه الداخل لـ outlist-PORT2؛

هـ) تعرف outlist-PORT2 الرسائل أو الإجراءات للاتجاه الخارج لـ PORT2،

يسمح بعملية **connect** إذا، وإذا فقط:

- outlist-PORT1 ⊆ inlist-PORT2 and outlist-PORT2 ⊆ inlist-PORT1؛

يسمح بعملية **map** (بافتراض أن PORT2 هو منفذ سطح بيني لنظام اختبار) إذا وإذا فقط:

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

وفي جميع الحالات الأخرى، لا يسمح بالعمليات.

ونظراً لأن TTCN-3 يسمح بتشكيلات وعناوين دينامية، لا يمكن القيام بجميع تأكيدات الاتساق هذه سكونياً عند وقت مُصرف. وجميع التأكيدات، التي لا يمكن القيام بها وقت مُصرف، يمكن القيام بها وقت التنفيذ وتؤدي إلى خطأ اختبار مجرد عند العطل.

3.22 عمليات Disconnect وUnmap

إن عمليات **disconnect** وunmap هي عكس عمليات **connect** و**map**. وتؤدي فك التوصيل لمنافذ (موصلة سابقاً) لمكونات اختبار وفك تقابل لمنافذ (متقابلة في السابق) لمكونات اختبار ومنافذ في سطح بيني لنظام اختبار.

يمكن طلب كل من عمليتي **disconnect** و**unmap** من أي مكون إذا كانت مراجع المكون ذات العلاقة مع أسماء المنافذ المعنية معروفة. ويكون لعملية **disconnect** أو **unmap** تأثير إذا كان التوصيل أو التقابل الذي يتعين إزالته قد تم خلقه مسبقاً.

المثـــال 1:

```
       :
       :
      connect(MyNewComponent:Port1, mtc:Port3);
      map(MyNewComponent:Port2, system:PCO1);
       :
       :
      disconnect(MyNewComponent:Port1, mtc:Port3);    // disconnect previously made connection
      unmap(MyNewComponent:Port2, system:PCO1);       // unmap previously made mapping
```

ولتسهيل عمليات **disconnect** و**unmap** المتعلقة بجميع التوصيلات والتقابلات لمكون أو منفذ، يسمح باستخدام عمليات **disconnect** و**unmap** مع متغير واحد فقط. ويحدد المتغير الواحد هذا جانب واحد من التوصيلات التي تفكك أو يفكك تقابلها. ويمكن استخدام الكلمة المفتاحية **all port** لتدل على جميع منافذ مكوّن.

المثـــال 2:

```
 :
disconnect(MyNewComponent:Port1);        // disconnects all connections of Port1, which
                                         // is owned by component MyNewComponent.
unmap(MyNewComponent:all port);          // unmaps all ports of component MyNewComponent
 :
```

إن استخدام عملية **disconnect** أو **unmap** دون أي معلمة يكون شكل مختزل لاستخدام عملية مع معلمة **self**. ويفكك توصيل أو يفكك تقابل جميع منافذ المكون الذي يطلب العملية.

المثـــال 3:

```
 :
disconnect;                    // is a shorthand form for …
disconnect(self:all port);     // which disconnects all ports of the component
                               // that called the operation
 :
unmap;                         // is a shorthand form for …
unmap(self:all port);          // which unmaps all ports of the component
                               // that called e operation
 :
```

تستخدم الكلمة المفتاحية **all component** فقط في مركب مع الكلمة المفتاحية **all port**، أي،**all component:all port**، وتستخدم فقط بواسطة MTC. وفضلاً عن ذلك، يستخدم متغير **all port** كمتغير والمتغير الوحيد لعملية **unmap** أو **all port** ويسمح بتحرير جميع توصيلات وتقابلات تشكيل اختبار.

المثـــال 4:

```
 :
 :
disconnect(all component:all port);        // the MTC disconnects all ports of all
                                           // components in the test configuration.
 :
 :
unmap(all component:all port);             // the MTC unmaps all ports of all
                                           // components in the test configuration.
 :
```

4.22 عمليات MTC وSystem وSelf

يتوفر لمرجع مكون (انظر 7.8) ثلاث عمليات: **mtc** و**system** تعيد مرجع مكون الاختبار الرئيسي والسطح البيني لنظام اختبار على التوالي. ويمكن استخدام عملية **self** لعودة مرجع مكون يطلب فيه.

مثـــال:

```
var MyComponentType MyAddress;
MyAddress := self; // Store the current component reference
```

والعمليات الوحيدة المسموحة لمرجع المكونات هي الإحالة والمساواة وعدم المساواة.

5.22 عملية Start لمكون اختبار

بمجرد خلق PTC وتوصيله، يتعين إسناد السلوك لـ PTC هذا ويبدأ تنفيذ سلوكه. ويتم هذا باستخدام عملية **start** (لأن خلق PTC لا يبدأ تنفيذ سلوك مكون). والسبب في التمييز بين **create** و**start** هو السماح بعمليات توصيل تتم قبل التنفيذ الفعلي لمكون الاختبار.

تربط عملية **start** السلوك المطلوب بمكون الاختبار. ويعرف السلوك هذا مرجع لوظيفة معرفة فعلاً.

يمكن لـ PTC لنمط حي أداء وظائف سلوك بترتيب تتابعي. ويبدأ وظيفة سلوك ثانية على PTC غير حي، أو يبدأ وظيفة على PTC مازال ينفّذ نتائج في خطأ اختبار مجرد. وإذا بدأت وظيفة على PTC حي بعد انتهاء وظيفة سابقة، تستخدم قيم متغير ومؤقتات ومنافذ وحكم محلي نظراً لأنها تركت بعد انتهاء الوظيفة السابقة. وخاصة، إذا بدأ مؤقت في وظيفة سابقة، ينبغي إقرار صلاحية وظيفة لاحقة لمناولة حدث إمهال ممكن.

مثـــال:

```
function MyFirstBehaviour() runs on MyComponentType { … }
function MySecondBehaviour() runs on MyComponentType { … }
 :
var MyComponentType MyNewPTC;
var MyComponentType MyAlivePTC;
 :
MyNewPTC := MyComponentType.create;        // Creation of a new non-alive test component.
MyAlivePTC := MyComponentType.create alive;// Creation of a new alive-type test component
 :
MyNewPTC.start(MyFirstBehaviour());        // Start of the non-alive component.
MyNewPTC.done;                             // Wait for termination
MyNewPTC.start(MySecondBehaviour());       // Test case error
 :
MyAlivePTC.start(MyFirstBehaviour());      // Start of the alive-type component
MyAlivePTC.done;                           // Wait for termination
MyAlivePTC.start(MySecondBehaviour());     // Start of the next function on the same component
  :
```

تنطبق التقييدات التالية على وظيفة منفذة في عملية مكون اختبار **start**:

إذا كان لهذه الوظيفة معلمات، تكون في معلمات **in** فقط، أي، معلمات حسب القيمة.

يكون لهذه الوظيفة تعريف **runs on** يشير إلى نمط مكون متوائم مع مكون تم خلقه جديدا (انظر 3.7.6).

لا تمرر منافذ ومؤقتات إلى هذه الوظيفة.

ملاحظة – بما أن منفذي **in** و**inout** يبدآن الاستماع عندما يخلق المكون، وفي اللحظة عندما يبدآن التنفيذ يمكن أن تكون هناك رسائل في صفوف الانتظار الواصلة للمنافذ التي تنتظر أن تعالج.

6.22 عملية Stop لسلوك اختبار

باستخدام بيان مكون اختبار **stop**، يمكن أن يوقف مكون اختبار تنفيذ سلوك اختباره الذي يجري تنفيذه أو تنفيذ سلوك اختبار ينفذ على مكون اختبار آخر. وإذا لم يوقف مكون سلوكه، ولكن السلوك يجري تنفيذه على مكون اختبار آخر في نظام الاختبار، يتعين على المكون واجب التوقف أن يعرف باستخدام مرجع مكونه. ويمكن لمكون أن يوقف سلوكه باستخدام بيان تنفيذ **stop** (انظر 10.19) أو بواسطة تناوله لعملية **stop**، مثلاً، باستخدام عملية **self**.

المثـــال 1:

الملاحظة 1 – بينما يمكن استخدام عمليات **create** و**start** و**running** و**done** و**killed** لـ PTC(s) فقط، يمكن أن تنطبق عملية **stop** على MTC.

```
var MyComponentType MyComp := MyComponentType.create;// A new test component is created
MyComp.start(CompBehaviour());                       // The new component is started
 :
if (date == "1.1.2005") {
  MyComp.stop;     // The component "MyComp" is stopped
}

 :
if (a < b ) {
   :
  self.stop;      // The test component that is currently executing stops its own behaviour
}
 :
  stop            // The test component stops its own behaviour
```

إن وقف مكون اختبار هو شكل واضح لإنهاء تنفيذ سلوك يجري تنفيذه. وينتهي سلوك مكون اختبار أيضاً بواسطة إكمال تنفيذه عندما يصل إلى نهاية الاختبار المجرد أو الوظيفة التي بدأت على هذا المكون أو بواسطة بيان **return** واضح. ويسمى هذا الإنهاء أيضاً وقف ضمني. ويكون للوقف الضمني نفس تأثيرات الوقف الصريح، أي، يحين الحكم العالمي مع الحكم المحلي لمكون اختبار متوقف (انظر قسم 25).

إذا كان مكون الاختبار المتوقف هو MTC، تحرر موارد جميع PTCs الموجودة، وتزال PTCs من نظام الاختبار وينتهي الاختبار المجرد (انظر 2.27).

إن وقف مكون اختبار لنمط غير حي (بشكل صريح أو ضمني) يدمره، وتحرر جميع الموارد المتصاحبة مع مكون الاختبار.

إن وقف مكون لنمط حي يوقف سلوك تنفيذ جاري فقط، ولكن يستمر المكون في الوجود ويمكن أن ينفذ سلوك جديد (بدأت عليه عملية **start**). ويترك المكون في حالة اتساق بعد توقف سلوكه.

الملاحظة 2 – فمثلاً، إذا توقف سلوك مكون لنمط حي خلال تخصيص قيمة جديدة لمتغير مسند فعلاً، يظل المتغير مسنداً بعد توقف المكون (مع القيمة القديمة أو الجديدة). وبالمثل، إذا توقف المكون خلال إعادة بدء مؤقت ينفذ فعلاً، يترك المؤقت في حالة تنفيذ بعد إنهاء السلوك.

يرد في القسم 25 قواعد إنهاء اختبارات مجردة وحساب حكم الاختبار النهائي.

يمكن أن يستخدم MTC الكلمة المفتاحية **all** فقط لوقف جميع PTCs المنفذة وليس MTC نفسه.

الملاحظة 3 – يمكن أن يوقف PTC تنفيذ اختبار مجرد بواسطة وقف MTC.

المثـــال 1:

```
 :
all component.stop // The MTC stops all PTCs of the test case but not itself.
 :
```

الملاحظة 4 – إن الآلية الملموسة لوقف PTCs هي خارج مدى هذه التوصية.

7.22 عملية Running

تسمح عملية **running** تنفيذ سلوك على مكون اختبار للتأكد من أن تنفيذ السلوك على مكون اختبار آخر قد تم. ويمكن استخدام عملية **running** لـ PTCs فقط. وتعيد عملية التنفيذ **true** إلى PTCs التي بدأت ولكن لم تنته أو تتوقف بعد. وتعيد **false**. وتعتبر عملية **running** تعبير بولاني، ومن ثمن تعيد قيمة **boolean** لتدل على ما إذا كان مكون اختبار محدد (أو جميع مكونات اختبار) قد انتهت. وعلى عكس عملية **done**، يمكن أن تستخدم عملية **running** حرية في تعبير **boolean**.

عندما تستخدم الكلمة المفتاحية **all** مع عملية **running**، تعيد **true** إذا بدأت PTCs ولكن لم تتوقف صراحة بواسطة مكون آخر وتنفذ سلوكها. وإلا تعيد **false**.

عندما تستخدم الكلمة المفتاحية **any** مع عملية **running**، تعيد **true** إذا كان PTC واحد على الأقل ينفذ سلوكه. وإلا يعيد **false**.

مثـــال:

```
if (PTC1.running)             // usage of running in an if statement
{
  // Do something!
}

while (all component.running != true) { // usage of running in a loop condition
  MySpecialFunction()
}
```

8.22 عملية Done

تسمح عملية **done** تنفيذ سلوك على مكون اختبار للتأكد من أن تنفيذ السلوك على مكون اختبار آخر قد تم. ويمكن استخدام عملية **done** لـ PTCs فقط.

تستخدم عملية **done** بنفس طريقة عملية استقبال أو عملية **timeout**. ويعني هذا عدم استخدام تعبير **boolean**، ولكن يمكن استخدامه لتحديد بديل في بيان **alt** أو بيان بمفرده في وصف سلوك. وفي الحالة الأخيرة، تعتبر عملية **done** اختزالاً لبيان **alt** مع بديل واحد، أي، له علم دلالات سد، وبالتالي يوفر إمكانية انتظار سلبي لإنهاء مكونات اختبار.

تنطبق عملية **done** على PTC، وتتواءم فقط إذا كان سلوك PTC قد توقف (ضمنياً أو صراحة) أو قد تم killed. وإلا، تكون المواءمة غير ناجحة.

عندما تستخدم الكلمة المفتاحية **all** مع عملية **done**، تتواءم إذا لم يكن PTC واحد ينفذ سلوكه. وتتواءم إذا لم يخلق PTC.

عندما تستخدم الكلمة المفتاحية **any** مع عملية **done**، تتواءم إذا توقف أو PTC killed واحد على الأقل. وإلا، تكون المواءمة غير ناجحة.

ملاحظة – ينتج أيضاً عن وقف سلوك مكون غير حي إزالة ذلك المكون من نظام الاختبار، بينما توقف مكون من نمط حي يترك المكون حياً في نظام الاختبار. وفي كلا الحالتين، تتواءم عملية **done**.

مثـــال:

```
// Use of done in alternatives
:
alt {
   [] MyPTC.done {
       setverdict(pass)
      }

   [] any port.receive {
       repeat
      }
}
:

var MyComp c := MyComp.create alive;
c.start(MyPTCBehaviour());
:
c.done;
  // matches as soon as the function MyPTCBehaviour (or function/altstep called by it) stops
c.done;
  // matches the end of MyPTCBehaviour (or function/altstep called by it) too
if(c.running) {c.done}
  // done here matches the end of the next behaviour only

// the following done as stand-alone statement:
 :
all component.done;
 :

// has the following meaning:
 :
alt {
   [] all component.done {}
}
 :

// and thus, blocks the execution until all parallel test components have terminated
```

9.22 عملية kill لمكون اختبار

تُوقف عملية **kill** المطبقة على مكون اختبار تنفيذ سلوك يجرى تنفيذه – إن وُجد – وتُحرر جميع الموارد المتصاحبة معها (بما في ذلك جميع توصيلات منفذ لمكون killed) وتزيل المكون من نظام الاختبار. ويمكن تطبيق عملية **kill** على مكون الاختبار الجاري نفسه بواسطة بيان **kill** بسيط أو بواسطة تناوله باستخدام عملية **self** بالتزامن مع عملية **kill**. ويمكن أن تنطبق أيضاً عملية **kill** على مكون اختبار آخر. وفي هذه الحالة، يتم تناول المكون الذي killed باستخدام مرجع مكونه. وإذا طبقت عملية **kill** على MTC، مثلاً، **mtc.kill** ، تنهي الاختبار المجرد.

المثـــال 1:

```
var PTCType MyAliveComp := PTCType.create alive;    // Create an alive-type test component
MyAliveComp.start(MyFirstBehavior());               // The new component is started
MyAliveComp.done;                                   // Wait for termination
MyAliveComp.start(MySecondBehavior());              // Start the component a 2nd time
MyAliveComp.done;                                   // Wait for termination
MyAliveComp.kill;                                   // Free its resources
```

يمكن أن يستخدم MTC الكلمة المفتاحية **all** فقط لوقف وkill جميع PTCs التي تنفذ إلا MTC نفسه.

المثـــال 2:

```
all component.kill;    // The MTC stops all (alive-type and normal) PTCs of the test case first
// and frees their resources.
```

10.22 عملية Alive

إن عملية **alive** هي عملية بولانية تتأكد من أن مكون اختبار قد تم خلقه وعلى استعداد للتنفيذ أو مستعد لتنفيذ وظيفة سلوك. وعند تطبيقها على مكون اختبار عادي، تعيد عملية **alive** إذا كان المكون غير نشط أو ينفذ وظيفة، ويعيد true إذا كان غير ذلك. وعند تطبيقها على مكون اختبار لنمط حي، تعيد العملية false إذا كان المكون غير نشط أو ينفذ أو توقف. وتعيد true إذا كان المكون killed.

يمكن أن تستخدم عملية **alive** مثل عملية **running** على PTCs فقط (انظر 7.22). وخاصة، في مركب مع الكلمة المفتاحية **all** ، تعيد **any** إذا كانت جميع PTCs حية (نمط حي أو عادي).

تعيد عملية **all** المستخدمة في مركب مع الكلمة المفتاحية keyword **any** إذا كان واحد من PTCs حياً.

مثـــال:

```
 :
PTC1.done;                          // Waits for termination of the component
if (PTC1.alive) {                   // If the component is still alive …
  PTC1.start(AnotherFunction());    // … execute another function on it.
}
```

11.22 عملية killed

تسمح عملية **killed** التأكد من أن مكون اختبار مختلف حي أو قد أزيل من نظام الاختبار.

تستخدم عملية **killed** بنفس طريقة عمليات استقبال. ويعني هذا أنها لا تستخدم في تعبيرات **boolean**، ولكن يمكن استخدامها لتحديد بديل في بيان **alt** أو كبيان بمفرده في وصف سلوك. وفي الحالة الأخيرة، تعتبر عملية **done** أنها اختزال لبيان **alt** مع بديل واحد فقط، أي، أن لها علم دلالات سد، وبالتالي توفر مقدرة لانتظار سلبي لإنهاء مكونات اختبار.

ملاحظة – عند التأكد من مكونات اختبار عادية، تتواءم عملية killed إذا أوقفت (ضمنياً أو صراحة) تنفيذ سلوكها أو كانت **killed** صراحة، أي، تكون العملية متكافئة لعملية **done** (انظر 8.22). ومع ذلك، عند التأكد من مكونات اختبار لنمط حي، تتواءم عملية **killed** فقط إذا كان المكون **killed** باستخدام عملية **kill**. وإلا تكون عملية **killed** غير ناجحة.

تستخدم عملية **killed** لـ PTCs فقط.

عندما تستخدم الكلمة المفتاحية **all** مع عملية **killed**، تتواءم إذا توقفت جميع PTCs عن الوجود. وتتواءم أيضاً إذا لم يخلق أي PTC.

عندما تستخدم الكلمة المفتاحية **any** مع عملية **killed**، تتواءم مع PTC واحد على الأقل توقف عن الوجود. وإلا، تكون المواءمة غير ناجحة.

مثـــال:

```
var MyPTCType ptc := MyPTCType.create alive;   // create an alive-type test component
timer T(10.0);                                 // create a timer
T.start;                                       // start the timer
ptc.start(MyTestBehavior());                   // start executing a function on the PTC
alt {
[] ptc.killed {                                // if the PTC was killed during execution …
  T.stop;                                      // … stop the timer and …
  setverdict(inconc);                          // … set the verdict to 'inconclusive'
   }
[] ptc.done {                                  // if the PTC terminated regularly …
  T.stop;                                      // … stop the timer and …
  ptc.start(AnotherFunction());                // … start another function on the PTC
   }
[] T.timeout {                                 // if the timeout occurs before the PTC stopped
  ptc.kill;                                    // … kill the PTC and …
  setverdict(fail);                            // … set the verdict to 'fail'
   }
}
```

12.22 استخدام مصفوفات مكونات

لا تعمل عمليات **create** و**connect** و**start** و**stop** و**kill** مباشرة على مصفوفات لمكونات. وبدلاً من ذلك، يوفر عنصر محدد لصفيف كمعلمة لهذه العمليات. وبالنسبة لمكونات، يتحقق تأثير صفيف باستخدام مراجع مكونات وتخصيص عنصر صفيف ذي علاقة لنتيجة عملية **create**.

مثـــال:

```
// This example shows how to model the effect of creating, connecting and running arrays of
// components using a loop and by storing the created component reference in an array of
// component references.

testcase MyTestCase() runs on MyMtcType system MyTestSystemInterface
{
   :
  var integer i;
  var MyPTCType1  MyPtc[11];
   :
  for (i:= 0; i<=10; i:=i+1)
  {


     MyPtc[i] := MyPTCType1.create;
     connect(self:PtcCoordination, MyPtc[i]:MtcCoordination);
     MyPtc[i].start(MyPtcBehaviour());
  }
   :
}
```

13.22 موجز استخدام any وall مع مكونات

يمكن استخدام الكلمات المفتاحية **any** وall مع عمليات تشكيل كما ورد في الجدول 16.

الجدول Z.140/16 – Any وAll مع مكونات

العملي	مسموح		مثال	تعليق
	any (انظر الملاحظة)	**all** (انظر الملاحظة)		
create				
start				
running	نعم ولكن من MTC فقط	نعم ولكن من MTC فقط	**any component.running;** **all component.running;**	هل هناك أي PTC يؤدى سلوك اختبار؟ هل كل اختبارات PTC تؤدي سلوك اختبار؟
alive	نعم ولكن من MTC فقط	نعم ولكن من MTC فقط	**any component.alive;** **all component.alive;**	هل هناك أي PTC حي؟ هل جميع PTCs حية؟
done	نعم ولكن من MTC فقط	نعم ولكن من MTC فقط	**any component.done;** **all component.done;**	هل هناك أي PTC أنهي التنفيذ؟ هل أنهت جميع PTCs تنفيذها؟
killed	نعم ولكن من MTC فقط	نعم ولكن من MTC فقط	**any component.killed;** **all component.killed;**	هل هناك أي PTC توقف عن الوجود؟ هل توقفت جميع PTCs عن الوجود؟
stop		نعم ولكن من MTC فقط	**all component.stop;**	أوقف السلوك على جميع PTCs.
kill		نعم ولكن من MTC فقط	**all component.kill;**	Kill جميع PTCs، أي، توقفت عن الوجود.
ملاحظة – تشير any وall إلى PTCs فقط، أي، لا ينظر في MTC.				

23 عمليات اتصالات

0.23 عـــام

يدعم TTCN-3 عمليات *message-based* و*procedure-based*. وفضلاً عن ذلك، يسمح TTCN-3 بفحص عنصر قمة في صفوف انتظار منفذ واصلة ويتحكم في النفاذ إلى منافذ بواسطة *controlling operations*.

الجدول Z.140/17 – نظرة شاملة على عمليات اتصالات TTCN-3

عمليات اتصالات			
عمليات اتصالات	**الكلمة المفتاحية**	**يمكن أن تستخدم عند منافذ قائمة على رسالة**	**يمكن أن تستخدم عن منافذ قائمة على إجراء**
اتصالات قائمة على رسالة			
أرسل رسالة	**send**	نعم	
استقبل رسالة	**receive**	نعم	
مقداح على رسالة	**trigger**	نعم	
اتصالات قائمة على إجراء			
نفذ نداء إجراء	**call**		نعم
اقبل نداء إجراء من كيان بعيد	**getcall**		نعم
أجب على نداء إجراء من كيان بعيد	**reply**		نعم
اطلب استثناء (لنداء مقبول)	**raise**		نعم
ناول استجابة من نداء سابق	**getreply**		نعم
احصل على استثناء (من كيان مطلوب)	**catch**		نعم
افحص عنصر قمة لصفوف انتظار منفذ واصل			
تأكد من رسالة/نداء/استثناء/إجابة مستقبلة	**check**	نعم	نعم
عمليات تحكم			
حرر صف انتظار منفذ	**clear**	نعم	نعم
حرر صف انتظار وقرر صلاحية إرسال واستقبال عند منفذ	**start**	نعم	نعم
أجمد إرسال ولا تسمح بعمليات استقبال للمواءمة عند منفذ	**stop**	نعم	نعم
أجمد إرسال ولا تسمح بعمليات استقبال لمواءمة رسائل/نداءات جديدة	**halt**	نعم	نعم

1.23 نسق عام لعمليات اتصالات

0.1.23 عـــام

تستخدم عمليات مثل **send** و**call** لتبادل المعلومات فيما بين مكونات اختبار وبين SUT ومكونات اختبار. ولشرح النسق العام لهذه العمليات، يمكن بناؤها في فئتين:

أ) يرسل مكون اختبار رسالة (عملية **send**) أو إجراء نداءات (عملية **call**) أو إجابات على نداء مقبول (عملية **reply**) أو طلب استثناء (عملية **raise**). وتشير هذه الأعمال جميعها إلى *sending operations*؛

ب) يستقبل مكون رسالة (عملية **receive**) أو ينتظر رسالة (عملية **trigger**) أو يقبل نداء إجراء (عملية **getcall**) أو يستقبل إجابة لإجراء مطلوب في السابق (عملية **getreply**) أو يحصل على استثناء (عملية **catch**). وتشير هذه الأعمال جميعها إلى *receiving operations*.

1.1.23 نسق عام لعمليات إرسال

تتألف عمليات إرسال من جزء *send*، وفي حالة عملية **call** قائمة على إجراء سد، و*response* وجزء *exception handling*.

الجزء send:

- يحدد المنفذ الذي تحدث فيه العملية المحددة؛
- يعرف رسالة أو نداء إجراء لكي يرسل؛
- يعطي جزء عنوان (خياري) يعرف على نحو وحيد شريك واحد أو أكثر لاتصالات ترسل فيها رسالة أو نداء أو إجابة أو استثناء.

يوجد اسم منفذ واسم عملية وقيمة في جميع عمليات إرسال. ويكون جزء العنوان (تدل عليه الكلمة المفتاحية **to**) خيارياً ويحتاج للتحديد فقط في حالات توصيلات من واحد إلى كثيرين حيث:

- تستخدم اتصالات unicast ويعرف كيان مستقبِل واحد صراحة؛
- تستخدم اتصالات multicast ويتعين تعريف مجموعة كيانات مستقبِلة صراحة؛
- تستخدم اتصالات broadcast ويتعين تناول جميع الكيانات الموصولة بمنفذ محدد.

ملاحظة – تستخدم مصطلحات اتصالات "unicast" و"multicast" و"broadcast" فيما يتعلق باتصالات منفذ. ويعني هذا أنه من الممكن تناول واحد أو عديد أو جميع مكونات اختبار موصولة بمنفذ محدد. ويمكن استخدام Unicast وmulticast وbroadcast أيضاً لمنافذ متقابلة. وفي هذه الحالة، يمكن الوصول إلى واحد أو عديد أو جميع كيانات في SUT عبر منفذ متقابل محدد.

المثـــال 1:

جزء Send			**استجابة (خيارية) واستثناء**
منفذ وعملية	**جزء قيمة**	**جزء عنوان (خياري)**	**جزء مناولة**
`MyP1.`**`send`**	`(MyVariable + YourVariable - 2)`	**`to`** `MyPartner;`	

هناك حاجة لمناولة استجابة واستثناء فقط في حالات اتصالات قائمة على إجراء. ويكون جزء مناولة استجابة واستثناء لعملية **call** خيارياً ويطلب في حالات يعيد فيها إجراء مطلوب قيمة أو معلمات **out** أو **inout** تكون هناك حاجة لقيمها في مكون طالب ولحالات حيث الإجراء المطلوب يمكن أن يطلب استثناءات تدعو إليه الحاجة لمناولة مكون طالب.

يستفيد جزء مناولة استجابة واستثناء لعملية نداء من عمليات **getreply** و**catch** لتوفير الوظيفة المطلوبة.

المثـــال 2:

جزء Send			**استجابة (خيارية) وجزء مناولة استثناء**
منفذ وعملية	**جزء قيمة**	**جزء عنوان (خياري)**	

`MyP1.`**`call`**	`(MyProc:{MyVar1})`		`{`
			`  [] MyP1.`**`getreply`**`(MyProc:{MyVar2}) {}`
			`  [] MyP1.`**`catch`**`(MyProc, ExceptionOne) {}`
			`}`

2.1.23 نسق عام لعمليات استقبال

تتألف عملية استقبال من جزء *receive* وجزء *assignment* (خياري).

الجزء receive:

أ) يحدد المنفذ التي تحدث فيه العملية؛

ب) يعرف جزء المواءمة الذي يحدد دخل مقبول يتواءم مع البيان؛

ج) يعطي تعبير عنوان (خياري) يحدد على نحو وحيد شريك اتصالات (في حالة توصيلات من واحد إلى كثيرين).

يوجد اسم منفذ واسم عملية وقيمة في جميع عمليات استقبال. ويكون تعرف شريك اتصالات (تدل عليه الكلمة المفتاحية **from**) خيارياً ويحتاج للتحديد فقط في حالات توصيلات من واحد إلى كثيرين حيث يحتاج الكيان المستقبِل إلى تعريفه صراحة.

إن جزء التخصيص في عملية استقبال هو خياري. وفي منافذ قائمة على رسائل، يستخدم عندما يطلب لتخزين رسائل مستقبَلة. وفي حالة منافذ قائمة على إجراءات، يستخدم لتخزين معلمات **in** و**inout** لنداء مقبول، لتخزين قيمة عودة أو تخزين استثناءات. وبالنسبة لجزء تخصيص، يطلب تنميط قوي؛ مثلاً، يكون لمتغير مستخدم لتخزين رسالة نفس نمط الرسالة الواصلة.

وبالإضافة إلى ذلك، يمكن أيضاً استخدام جزء التخصيص لتخصيص عنوان **sender** لرسالة أو استثناء أو **reply** أو **call** لمتغير. وهذا مفيد لتوصيلات من واحد إلى كثيرين حيث، مثلاً، يمكن استقبال نفس الرسالة أو النداء من مكونات مختلفة، ولكن يجب إرسال الرسالة أو الإجابة أو الاستثناء إلى المكون المرسل الأصلي.

مثـــال:

جزء Receive				جزء تخصيص (خياري)		
منفذ وعملية	جزء مواءمة	تعيير عنوان (خياري)		تخصيص قيمة (خياري)	تخصيص قيمة معلمة (خياري)	تخصيص قيمة مرسل (خياري)
MyP1.getreply	(AProc:{?} value 5)		->		param (V1)	sender APeer

جزء Receive				جزء تخصيص (خياري)		
منفذ وعملية	جزء مواءمة	تعيير عنوان (خياري)		تخصيص قيمة (خياري)	تخصيص قيمة معلمة (خياري)	تخصيص قيمة مرسل (خياري)
MyP2.receive	(MyTemplate(5,7))	from APeer	->	value MyVar		

2.23 اتصالات قائمة على رسالة

0.2.23 عـــام

إن اتصالات قائمة على رسالة هي اتصالات قائمة على تبادل رسائل لا تزامنية. وتكون الاتصالات القائمة على رسالة لا تسد في عملية **send**، كما يوضح الشكل 11، حيث تستمر المعالجة في SENDER مباشرة بعد حدوث عملية **send**. ويسد RECEIVER في عملية **receive** حتى تُعالج الرسالة المستقبَلة.

بالإضافة إلى عملية **receive**، يوفر TTCN-3 عملية **trigger** التي ترشح الرسائل مع معايير مواءمة معينة من تدفق رسائل مستقبَلة على منفذ واصل معيّن. وتُزال الرسائل عند قمة صف انتظار لا يلبي معايير المواءمة من المنفذ دون اتخاذ إجراء إضافي.

الشكل Z.140/11 – توضيح لا تزامن send وreceive

1.2.23 عملية Send

0.1.2.23 عـــام

تستخدم عملية **send** لوضع رسالة على منفذ رسالة خارجة. ويمكن تحديد الرسالة بالرجوع إلى مقاس معرف أو يمكن تعريفها كمقاس في الخط. وعند تعريف رسالة في الخط، يستخدم جزء النمط الخياري إذا كان هناك غموضاً في نمط الرسالة التي تُرسل.

تستخدم عملية **send** فقط على منافذ قائمة على رسائل (أو مختلطة) ويكون نمط المقاس الذي يرسل في قائمة الأنماط الخارجة لتعريف نمط المنفذ.

مثـــال:

```
MyPort.send(MyTemplate(5,MyVar));   // Sends the template MyTemplate with the actual
                                    // parameters 5 and MyVar via MyPort.

MyPort.send(5);                     // Sends the integer value 5 (which is an in-line template)
```

1.1.2.23 إرسال unicast أو multicast أو broadcast

يدعم TTCN-3 اتصالات unicast وmulticast وbroadcast. وتحدد آلية الاتصالات المستخدمة بواسطة شرط **to** خياري في عملية **send**. ويمكن حذف شرط **to** في حالة توصيل من واحد إلى واحد حيث اتصالات unicast تستخدم ويحدد مستقبِل الرسالة الوحيد بواسطة بنية نظام الاختبار. ويكون شرط **to** موجوداً في حالة توصيلات من واحد إلى كثيرين.

تحدد اتصالات Unicast إذا كان شرط **to** يتناول شريك اتصالات واحد فقط. وتستخدم اتصالات Multicast إذا كان شرط **to** يشمل قائمة شركاء اتصالات. ويعرف Broadcast باستخدام شرط **to** مع الكلمة المفتاحية **all component**.

مثـــال:

```
MyPort.send(charstring:"My string") to MyPartner;
          // Sends the string "My string" to a component with a
          // component reference stored in variable MyPartner

MyPCO.send(MyVariable + YourVariable - 2) to MyPartner;
          // Sends the result of the arithmetic expression to MyPartner.

MyPCO2.send(MyTemplate) to (MyPeerOne, MyPeerTwo);
          // Specifies a multicast communication, where the value of
          // MyTemplate is sent to the two component references stored
          // in the variables MyPeerOne and MyPeerTwo.

MyPCO3.send(MyTemplate) to all component;
          // Broadcast communication: the value of Mytemplate is sent to
          // all components which can be addressed via this port. If
          // MyPCO3 is a mapped port, the components may reside inside
          // the SUT.
```

2.2.23 عملية Receive

0.2.2.23 عـــام

تستخدم عملية **receive** لاستقبال رسالة من صف انتظار منفذ رسالة واصلة. ويمكن تحديد الرسالة بواسطة تحديد مرجع مقاس معرف أو يمكن تعريفها كمقاس في الخط. وعند تعريف رسالة في الخط، يكون جزء النمط الخياري موجوداً عندما يكون نمط الرسالة الذي يُستقبل غامضاً. وتستخدم عملية **receive** فقط على منافذ قائمة على رسالة (أو مختلطة) ويتضمن نمط القيمة الذي يُستقبَل في قائمة أنماط واصلة لتعريف نمط منفذ.

تزيل عملية **receive** رسالة القمة من صف انتظار منفذ واصل إذا، وإذا فقط، لبّت رسالة تلك القمة جميع معايير المواءمة المتصاحبة مع عملية **receive**. ولا يحدث ربط للقيم الواصلة على أساس التعبير أو المقاس.

وإذا كانت المواءمة غير ناجحة، لا تزال رسالة القمة من صف انتظار منفذ، أي، إذا استخدمت عملية **receive** كبديل لبيان **alt** ولم تكن ناجحة، يستمر تنفيذ الاختبار المجرد مع البديل التالي لبيان **alt**.

تتعلق معايير المواءمة بنمط وقيمة الرسالة التي تستقبل. ويحدد نمط وقيمة الرسالة التي تستقبل متغير عملية **receive**، أي، يمكن أن يشتق من مقاس معرف أو محدد في الخط. ويستخدم مجال نمط خياري في معايير مواءمة لعملية **receive** لتجنب أي غموض لنمط القيمة الذي يجري استقباله.

الملاحظة 1 – يشارك أيضاً تشفير نعوت في المواءمة بطريقة صريحة، من خلال منع مفكك التشفير من إنتاج قيمة مجردة من الرسالة المستقبَلة المشفرة بطريقة مختلفة عن التي حددتها النعوت.

وفي حالة توصيلات من واحد إلى كثيرين، يمكن قصر عملية **receive** على شريك اتصالات معين. ويتم هذا التقييد باستخدام الكلمة المفتاحية **from**.

المثـــال 1:

```
MyPort.receive(MyTemplate(5, MyVar));   // Matches a message that fulfils the conditions
                                        // defined by template MyTemplate at port MyPort.

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

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

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

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

إذا كانت المواءمة ناجحة، يمكن تخزين القيمة المزالة من صف انتظار منفذ في متغير ويمكن استردادها وتخزينها في متغير. ويتم هذا بواسطة الرمز '<-' والكلمة المفتاحية **value**.

ومن الممكن استرداد وتخزين مرجع مكون أو عنوان مرسل لرسالة. ويتم هذا بواسطة الكلمة المفتاحية **sender**.

الملاحظة 2 – عندما تُستقبل رسالة على منفذ موصول، يُخزن فقط مرجع المكون في الكلمة المفتاحية **sender**، ولكن يخزن نظام الاختبار داخلياً اسم المكون أيضاً، إن وجد (ليستخدم في التسجيل).

المثـــال 2:

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

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

MyPort.receive(MyTemplate:{5, MyVarOne}) -> value MyVarTwo sender MyPeer;
// The received message value is stored in MyVarTwo and the sender address is stored in MyPeer.
```

1.2.2.23 استقبال أي رسالة

تزيل عملية **receive**، مع عدم وجود قائمة متغيرات لنمط وقيمة معايير متوائمة لرسالة، تستقبل الرسالة على القمة لصف انتظار منفذ واصل (إن وجد) إذا تمت تلبية جميع معايير المواءمة الأخرى.

لا تخصص رسالة مستقبَلة من *ReceiveAnyMessage* لمتغير.

مثـــال:

```
MyPort.receive;                          // Rmoves the top value from MyPort.

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

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

2.2.2.23 استقبال على أي منفذ

لاستقبال رسالة **receive** على أي منفذ، تستخدم الكلمات المفتاحية **any port**.

مثـــال:

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

3.2.23 عملية Trigger

0.3.2.23 عـــام

تزيل عملية **trigger** رسالة القمة من صف انتظار منفذ واصل متصاحب. وإذا لبت رسالة تلك القمة معايير المواءمة، تسلك عملية **trigger** نفس طريقة عملية **receive**. وإذا لم تلبِّ رسالة تلك القمة معايير المواءمة، تُزال من صف الانتظار دون أي عمل إضافي. وتستخدم عملية **trigger** فقط على منافذ قائمة على رسالة (أو مختلطة) ويتضمن نمط القيمة الذي يُستقبل في قائمة الأنماط الواصلة لتعريف نمط منفذ.

ملاحظة – إن الملاحظة 1 في 0.2.2.22 صالحة أيضاً لعملية **trigger**.

يمكن أن تُستخدم عملية **trigger** كبيان بمفرده في وصف سلوك. وفي الحالة الأخيرة هذه تعتبر عملية **trigger** أنها اختزال لبيان **alt** مع بديل واحد فقط، أي، يكون لها علم دلالات سد، وبالتالي توفر مقدرة على انتظار الرسالة التالية التي تتواءم مع مقاس محدد أو قيمة على صف الانتظار ذلك.

المثـــال 1:

```
MyPort.trigger(MyType:?);
// Specifies that the operation will trigger on the reception of the first message observed of
// the type MyType with an arbitrary value at port MyPort.
```

تتطلب عملية **trigger** اسم المنفذ ومعايير المواءمة للنمط والقيمة وتقييد **from** خياري (أي، اختيار شريك اتصالات) وتخصيص اختياري لرسالة متوائمة ومكون مرسل لمتغيرات.

المثـــال 2:

```
MyPort.trigger(MyType:?) from MyPartner;
// Triggers on the reception of the first message of type MyType at port MyPort
// received from MyPartner.
```

```
MyPort.trigger(MyType:?) from MyPartner -> value MyRecMessage;
// This example is almost identical to the previous example. In addition, the message which
// triggers i.e., all matching criteria are met, is stored in the variable MyRecMessage.

MyPort.trigger(MyType:?) -> sender MyPartner;
// This example is almost identical to the first example. In addition, the reference of the
// sender component will be retrieved and stored in variable MyPartner.

MyPort.trigger(integer:?) -> value MyVar sender MyPartner;
// Trigger on the reception of an arbitrary integer value which afterwards is stored in
// variable MyVar. The reference of the sender component will be stored in variable MyPartner.
```

1.3.2.23 على أي رسالة Trigger

تبدأ عملية **trigger** ليس لها قائمة متغيرات عند استقبال أي رسالة. ومن ثم، يكون معناها مماثل لمعنى استقبال أي رسالة. ولا تخصص رسالة مستقبَلة من *TriggerOnAnyMessage* لمتغير.

مثـــال:

```
MyPort.trigger;

MyPort.trigger from MyPartner;

MyPort.trigger -> sender MySenderVar;
```

2.3.2.23 Trigger على أي منفذ

لكي **trigger** على رسالة عند أي منفذ، تستخدم الكلمات المفتاحية **any port**.

مثـــال:

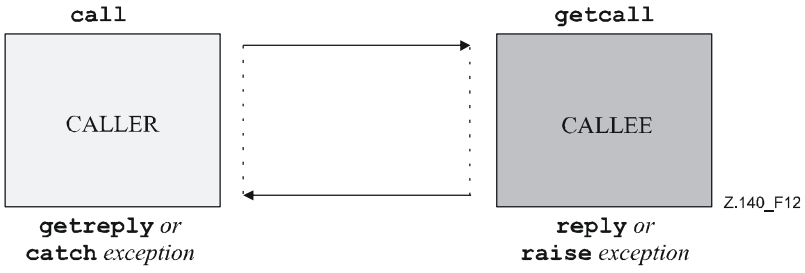
```
any port.trigger
```

3.23 اتصالات قائمة على أساس إجراء

0.3.23 عـــام

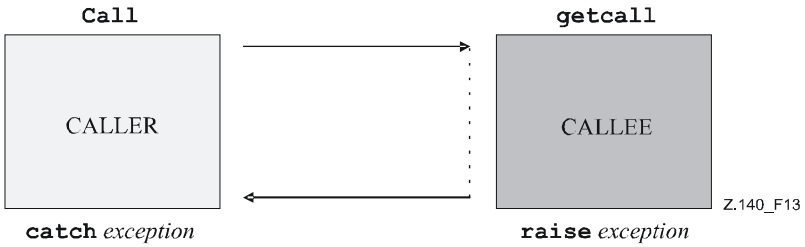
إن مبدأ اتصالات قائمة على أساس إجراء هو طلب إجراءات في كيانات بعيدة. ويدعم TTCN-3 اتصالات قائمة على أساس إجراء *blocking* و*non-blocking* وسد اتصالات قائمة على أساس إجراء هو سد جانب الطالب والمطلوب، بينما اتصالات قائمة على أساس إجراء لإزالة السد هي تسد جانب المطلوب. وتحدد توقيعات الإجراءات المستخدمة لاتصالات قائمة على أساس إجراء لإزالة السد طبقاً للقواعد في القسم 13.

يرد في الشكل 12 تخطيط اتصالات سد اتصالات قائمة على أساس إجراء. ويطلب CALLER إجراء بعيداً في CALLEE باستخدام عملية **call**. ويقبل CALLEE النداء بواسطة عملية **getcall** ويرد إما باستخدام عملية **reply** للإجابة على النداء أو بطلب (عملية **raise**) استثناء. ويتناول CALLER الرد أو الاستثناء باستخدام عمليات **getreply** أو **catch** وفي الشكل 12، يدل على سد CALLER وCALLEE خطوط متقطعة.



الشكل Z.140/12 – توضيح سد اتصالات قائمة على أساس إجراء

يرد في الشكل 13 تخطيط اتصالات لإزالة سد اتصالات قائمة على أساس إجراء. ويطلب CALLER إجراء بعيداً في CALLEE باستخدام عملية **call** ويستمر في تنفيذه، أي، لا ينتظر رداً أو استثناء. ويقبل CALLEE النداء بواسطة عملية **getcall** وينفذ الإجراء المطلوب. وإذا كان التنفيذ ناجحاً، يمكن أن يطلب استثناء ليخطر CALLEE ويمكن أن يتناول CALLER الاستثناء باستخدام عمليات **catch** في بيان **alt**. وفي الشكل 13، يدل سد CALLEE حتى نهاية مناولة النداء وإمكانية طلب استثناء بواسطة خط متقطع.



الشكل Z.140/13 – توضيح إزالة سد اتصالات قائمة على أساس إجراء

1.3.23 عملية Call

0.1.3.23 عـــام

تستخدم عملية **call** لتحديد أن مكون الاختبار يطلب إجراء في SUT أو في مكون اختبار آخر. وتستخدم عملية **call** فقط على منافذ قائمة على إجراء (أو مختلطة). ويشمل تعريف نمط لمنفذ تتم فيه عملية النداء اسم الإجراء في قائمة **out** أو **inout**، أي، يجب أن يسمح طلب هذا الإجراء عند المنفذ هذا.

تكون المعلومات التي ترسل في جزء send لعملية **call** هي توقيع يمكن تعريفه إما في شكل مقاس توقيع أو يعرف في الخط. ويكون لجميع معلمات **in** و**inout** لتوقيع قيمة محددة، أي، لا يسمح باستخدام آليات مواءمة مثل *AnyValue*.

لا تستخدم متغيرات توقيع لعملية **call** لاسترداد أسماء متغير لمعلمات **out** و**inout**. ويتم التخصيص الفعلي لإجراء عودة قيمة وقيم معلمات **call** لمتغيرات صراحة في الاستجابة وجزء مناولة التنفيذ لعملية **getreply** و**catch** بواسطة عمليات **call**. ويسمح هذا باستخدام مقاسات توقيع في عمليات بنفس الطريقة مثل المقاسات التي يمكن أن تستخدم للأنماط.

المثـــال **1**:

```
// Given …
signature MyProc (out integer MyPar1, inout boolean MyPar2);
 :
// a call of MyProc
MyPort.call(MyProc:{ -, MyVar2}) {      // in-line signature template for the call of MyProc
   [] MyPort.getreply(MyProc:{?, ?}) { }
}

// … and another call of MyProc
MyPort.call(MyProcTemplate) {           // using signature template for the call of MyProc
   [] MyPort.getreply(MyProc:{?, ?}) { }
}
```

في حالات توصيلات من واحد إلى كثيرين، يحدد شريك الاتصالات بشكل وحيد. ويتم هذا باستخدام الكلمة المفتاحية **to**.

المثـــال 2:

```
MyPort.call(MyProcTemplate) to MyPeer {     // calling MyProc at MyPeer
   [] MyPort.getreply(MyProc:{?, ?}) { }
}
```

1.1.3.23 مناولة استجابات واستثناءات لـ Call

في حالة اتصالات قائمة على أساس إجراء لإزالة سد (انظر 4.1.3.23)، تتم مناولة استثناءات عمليات **call** باستخدام عمليات **catch** (انظر 6.3.23) كبدائل في بيانات **alt**.

إذا استُخدم خيار **nowait** (انظر 2.1.3.23)، يتم مناولة استجابات أو استثناءات عمليات **call** باستخدام عمليات **getreply** (انظر 4.3.23) و**catch** (انظر 6.3.23) كبدائل في بيانات **alt**.

في حالة اتصالات قائمة على أساس إجراء لسد، تتم مناولة استجابات أو استثناءات لنداء في جزء مناولة استجابة واستثناء لعملية **call** بواسطة عمليات **getreply** (انظر 4.3.23) و**catch** (انظر 6.3.23).

يبدو جزء مناولة استجابة واستثناء لعملية **call** مماثل لجسم بيان **alt**. ويعرف مجموعة بدائل تصف استجابات واستثناءات ممكنة لنداء. ويقوم اختيار البدائل فقط على عمليات **getreply** و**catch** لإجراء مطلوب. وتعالج عمليات **getreply** و**catch** غير المؤهلة الردود فقط والاستثناءات المطلوبة من إجراء مطلوب. ولا يسمح باستخدام فروع **else** وتنفيذ altsteps.

وإذا لزم الأمر، من الممكن إقرار صلاحية/إخماد بديل بواسطة تعبير **boolean** موضوع بين أقواس '[ ]' للبديل.

ينفذ جزء مناولة استجابة واستثناء لعملية نداء مثل بيان **alt** دون أي تغيب نشيط. ويعني هذا أن لقطة متناظرة تشمل جميع المعلومات الضرورية لتقييم حراسات بولانية (خيارية)، يمكن أن تشمل عنصر القمة (إن وُجد) للمنفذ الذي طُلب عبره الإجراء، ويمكن أن يشمل استثناء إمهال مولدة مؤقت (خياري) يشرف على النداء (انظر 2.1.3.23).

يمكن أن يكون لتقييم تعبيرات بولانية تحرس بدائل في جزء مناولة استجابة واستثناء آثار جانبية. ولتجنب الآثار الجانبية غير المتوقعة، تنطبق نفس القواعد لحراسات بولانية في بيانات **alt** (انظر 1.1.20).

مثـــال:

```
// Given
signature MyProc3 (out integer MyPar1, inout boolean MyPar2) return MyResultType
  exception (ExceptionTypeOne, ExceptionTypeTwo);
 :

// Call of MyProc3
MyPort.call(MyProc3:{ -, true }) to MyPartner {

  [] MyPort.getreply(MyProc3:{?, ?}) -> value MyResult param (MyPar1Var,MyPar2Var) { }

  [] MyPort.catch(MyProc3, MyExceptionOne) {
  setverdict(fail);
  stop;
     }
  [] MyPort.catch(MyProc3, ExceptionTypeTwo : ?) {
  setverdict(inconc);
     }
  [MyCondition] MyPort.catch(MyProc3, MyExceptionThree) { }
}
```

2.1.3.23 مناولة استثناءات إمهال لـ Call

يمكن أن تشمل خيارياً عملية **call** إمهالاً. وتُعرّف هذه كقيمة صريحة أو ثابت لنمط **float** وتعرّف طول الوقت بعد أن بدأت عملية **call** بأن تنفيذ **timeout** يتولد بواسطة نظام الاختبار. وإذا لم يكن جزء قيمة إمهال محيّناً في عملية **call**، لا يولد استثناء **timeout**.

المثـــال 1:

```
MyPort.call(MyProc:{5,MyVar}, 20E-3) {

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

  [] MyPort.catch(timeout) {            // timeout exception after 20ms
  setverdict(fail);
  stop;
     }
}
```

إن استخدام الكلمة المفتاحية **nowait** بدلاً من قيمة استثناء إمهال في عملية **call** يسمح طلب إجراء بالاستمرار دون انتظار سواء استجابة أو استثناء طلبه الإجراء المطلوب أو استثناء الإمهال.

المثـــال 2:

```
MyPort.call(MyProc:{5, MyVar}, nowait);  // The calling test component will continue
                  // its execution without waiting for the
                  // termination of MyProc
```

إذا استُخدمت الكلمة المفتاحية **nowait**، يتعين إزالة الاستجابة أو الاستثناء الممكن لإجراء مطلوب من صف انتظار منفذ باستخدام عملية **getreply** أو **catch** في بيان **alt** لاحق.

3.1.3.23 طلب إجراءات سد دون عودة قيمة ومعلمات out ومعلمات inout واستثناءات

قد لا يكون لإجراء سد قيم عودة ولا معلمات **out** و**inout** ويمكن أن يطلب استثناء. ويكون أيضاً لعملية نداء لحالات إجراء استجابة وجزء مناولة استثناء لمناولة سد بطريقة موحدة.

مثـــال:

```
// Given …
signature MyBlockingProc (in integer MyPar1, in boolean MyPar2);
 :
// a call of MyBlockingProc
MyPort.call(MyBlockingProc:{ 7, false }) {
  [] MyPort.getreply( MyBlockingProc:{ -, - } ) { }
}
```

4.1.3.23 طلب إجراءات لإزالة سد

ليس لإجراء إزالة سد معلمات **out** و**inout** ولا قيمة عودة، وتدل خاصية إزالة سد في تعريف توقيع متناظر بواسطة الكلمة المفتاحية **noblock**.

ليس لعملية **call** لإجراء إزالة سد استجابة وجزء مناولة استثناء، ولا يطلب استثناء إمهال ولا تستخدم الكلمة المفتاحية **nowait**.

ويتعين إزالة الاستثناءات الممكنة التي طلبتها إجراءات إزالة سد من صف انتظار منفذ باستخدام عمليات **catch** في بيانات **alt** أو **interleave** لاحقة.

5.1.3.23 نداءات Unicast وmulticast وbroadcast لإجراءات

مثل لعملية **send**، يدعم TTCN-3 أيضاً نداءات unicast وmulticast وbroadcast لإجراءات. ويمكن أن يتم هذا بنفس الطريقة الواردة في 1.1.2.23، أي، يكون متغير شرط **to** لعملية **call**، لنداءات unicast، هو عنوان كيان واحد مستقبل، (أو يمكن حذفه في حالة توصيلات من واحد إلى واحد)؛ ولنداءات multicast، قائمة عناوين لمجموعة مستقبلات؛ ولنداءات broadcast، الكلمة المفتاحية **all component**. وفي حالة توصيلات من واحد إلى واحد، يمكن حذف شرط **to** بسبب أن الكيان المستقبَل معرّف بشكل وحيد بواسطة بنية النظام.

لقد تم شرح مناولة استجابات واستثناءات لسد أو إزالة سد لعمليات unicast **call** في 1.1.3.23 إلى 4.1.3.23. ويمكن أن تسبب عملية multicast أو broadcast **call** استجابات واستثناءات عديدة من شركاء اتصالات مختلفين.

في حالة عملية multicast أو broadcast **call** لإجراء إزالة سد، يمكن مناولة جميع الاستثناءات التي يمكن طلبها من شركاء اتصالات مختلفين في بيانات **catch**، **alt** أو **interleave** لاحقة.

في حالة multicast أو لعملية broadcast **call** لإجراء إزالة سد، يوجد خياران: في الأول، تجري مناولة استجابة أو استثناء واحد فقط في جزء مناولة استجابة واستثناء لعملية **call**. ثم، يمكن مناولة استجابات واستثناءات إضافية في بيانات **alt** أو **interleave** لاحقة. وفي الثاني، تجري مناولة استجابات أو استثناءات عديدة باستخدام تكرار بيانات في واحد أو أكثر من فدرة بيانات وإعلانات لجزء مناولة استجابة واستثناء لعملية النداء: ويسبب تنفيذ تكرار بيان إعادة تقييم جسم النداء.

ملاحظة – في الحالة الثانية، يحتاج المستعمل لمناولة عدد من التكرارات.

المثـــال 1:

```
var boolean first:= true;
MyPort.call(MyProc:{5,MyVar}, 20E-3) to (MyPeerOne, MyPeerTwo) {   // Multicast call of MyProc
  // Handles the response from MyPeerOne
  [first] MyPort.getreply(MyProc:{?, ?}) from MyPeerOne {
  if (first) { first := false; repeat; }
  :
  }
  // Handles the response from MyPeerTwo
  [first] MyPort.getreply(MyProc:{?, ?}) from MyPeerTwo {
  if (first) { first := false; repeat; }
  :
  }
  [] MyPort.catch(timeout) {        // timeout exception after 20ms
  setverdict(fail);
  stop;
   }
}

alt {
  [] MyPort.getreply(MyProc:{?, ?}) {   // Handles all other responses to the broadcast call
  repeat
  }
}
```

في حالة multicast أو broadcast لعملية **call** لإجراء سد، حيث تستخدم الكلمة المفتاحية **nowait**، يتعين مناولة جميع الاستجابات والاستثناءات في بيانات **alt** أو **interleave** لاحقة.

المثـــال 2:

```
MyPort.call(MyProc:{5,MyVar}) to (MyPeer1, MyPeer2) nowait; // Multicast call of MyProc

interleave {
  [] MyPort.getreply(MyProc:{?, ?}) from MyPeer1 { } // Handles the response of MyPeer1
  [] MyPort.getreply(MyProc:{?, ?}) from MyPeer2 { } // Handles the response of MyPeer2
}
```

2.3.23 عملية Getcall

0.2.3.23 عـــام

تستخدم عملية **getcall** لتحديد أن مكون اختبار يقبل نداء من SUT، أو مكون اختبار آخر. وتستخدم عملية **getcall** فقط على منافذ قائمة على إجراء (أو مختلطة) وأن يتضمن توقيع نداء إجراء يتعين قبوله في قائمة الإجراءات الواصلة المسموح بها لتعريف نمط منفذ.

تزيل عملية **getcall** نداء القمة من صف انتظار منفذ واصل إذا، وإذا فقط، لبّت معايير المواءمة المتصاحبة مع عملية **getcall**. وتتعلق معايير المواءمة هذه بتوقيع النداء الذي يُعالج وشريك الاتصالات. ويمكن تحديد معايير المواءمة لتوقيع إما في الخط أو تشتق من مقاس توقيع.

يمكن أن تقتصر عملية **getcall** على شريك اتصالات معين في حالة توصيلات من واحد إلى كثيرين. ويتم هذا التقييد باستخدام الكلمة المفتاحية **from**.

المثـــال 1:

```
MyPort.getcall(MyProc: MyProcTemplate(5, MyVar)); // accepts a call of MyProc at MyPort

MyPort.getcall(MyProc:{5, MyVar}) from MyPeer; // accepts a call of MyProc at MyPort from
MyPeer
```

لا يستخدم متغير توقيع لعملية **getcall** ليمر في أسماء متغير لمعلمات **in** و**inout**. ويتم تخصيص قيم معلمة **in** و**inout** لمتغيرات في جزء التخصيص لعملية **getcall**. ويسمح هذا باستخدام مقاسات توقيع في عمليات **getcall** بنفس الطريقة مثل المقاسات التي تستخدم للأنماط

يتألف جزء التخصيص (الخياري) لعملية **getcall** من تخصيص قيم معلمة **in** و**inout** لمتغيرات واسترداد عنوان المكون الطالب. ولا يستخدم جزء تخصيص القيمة مع عملية getcall. وتستخدم الكلمة المفتاحية **param** لاسترداد قيم معلمة لنداء.

تستخدم الكلمة المفتاحية **sender** عندما يطلب استرداد عنوان المرسل (مثلاً، لتناول **reply** أو استثناء لطرف طالب في تشكيل من واحد إلى كثيرين).

المثـــال 2:

```
MyPort.getcall(MyProc:{?, ?}) from MyPartner -> param (MyPar1Var, MyPar2Var);
// The in or inout parameter values of MyProc are assigned to MyPar1Var and MyPar2Var.

MyPort.getcall(MyProc:{5, MyVar}) -> sender MySenderVar;
// Accepts a call of MyProc at MyPort with the in or inout parameters 5 and MyVar.
// The address of the calling party is retrieved and stored in MySenderVar.

// The following getcall examples show the possibilities to use matching attributes
// and omit optional parts, which may be of no importance for the test specification.

MyPort.getcall(MyProc:{5, MyVar}) -> param(MyVar1, MyVar2) sender MySenderVar;

MyPort.getcall(MyProc:{5, ?}) -> param(MyVar1, MyVar2);

MyPort.getcall(MyProc:{?, MyVar}) -> param( - , MyVar2);
// The value of the first inout parameter is not important or not used

// The following examples shall explain the possibilities to assign in and inout parameter
// values to variables. The following signature is assumed for the procedure to be called:

signature MyProc2(in integer A, integer B, integer C, out integer D, inout integer E);

MyPort.getcall(MyProc2:{?, ?, 3, - , ?}) -> param (MyVarA, MyVarB, - , -, MyVarE);
// The parameters A, B, and E are assigned to the variables MyVarA, MyVarB, and
// MyVarE. The out parameter D needs not be considered.

MyPort.getcall(MyProc2:{?, ?, 3, -, ?}) -> param (MyVarA:= A, MyVarB:= B, MyVarE:= E);
// Alternative notation for the value assignment of in and inout parameter to variables. Note,
// the names in the assignment list refer to the names used in the signature of MyProc2

MyPort.getcall(MyProc2:{1, 2, 3, -, *}) -> param (MyVarE:= E);
// Only the inout parameter value is needed for the further test case execution
```

1.2.3.23 قبول أي نداء

تزيل عملية **getcall** مع عدم وجود قائمة متغيرات لمعايير مواءمة توقيع النداء على قمة صف انتظار منفذ واصل (إن وُجد) إذا تم تلبية جميع معايير المواءمة الأخرى. ولا تخصص معلمات نداءات مقبولة من *AcceptAnyCall* لمتغير.

مثـــال:

```
MyPort.getcall;                          // Removes the top call from MyPort.

MyPort.getcall from MyPartner;           // Removes a call from MyPartner from port MyPort

MyPort.getcall -> sender MySenderVar;    // Removes a call from MyPort and retrieves
                                         // the address calling entity
```

2.2.3.23 Getcall على أي منفذ

يدل **getcall** على أي منفذ الكلمة المفتاحية **any**.

مثـــال:

```
any port.getcall(MyProc)
```

3.3.23 عملية Reply

تستخدم عملية **reply** للرد على نداء مقبول في السابق طبقاً لتوقيع إجراء. وتستخدم عملية **reply** فقط عند منفذ قائم على إجراء (أو مختلط). ويتضمن تعريف نمط المنفذ اسم الإجراء الذي ينتمي لعملية **reply**.

ملاحظة – لا يمكن دائماً التأكد سكونياً من العلاقة بين نداء مقبول وعملية **reply**. وللاختبار، يسمح بتحديد عملية **reply** دون عملية **reply** المتصاحبة.

يتألف جزء القيمة لعملية **reply** من مرجع توقيع مع قائمة معلمات فعلية متصاحبة وقيمة عودة (خيارية). ويمكن تعريف التوقيع إما في شكل مقاس توقيع أو يمكن تعريفه في الخط. ويكون لجميع معلمات **out** و**inout** لتوقيع قيمة محددة، أي، لا يسمح باستخدام آليات مواءمة مثل *AnyValue*.

يمكن إرسال استجابات لعملية واحدة أو أكثر لعملية **call** إلى كيان واحد أو عديد من الكيانات أو إلى جميع نظراء الكيانات الموصلة بالمنفذ المتناول. ويمكن تحديد هذا بنفس الطريقة الواردة في 1.1.2.23. ويعني هذا أن متغير شرط **to** لعملية **reply** هو، لاستجابات unicast، عنوان كيان مستقبَل؛ ولاستجابات multicast، قائمة عناوين لمجموعة مستقبلات؛ ولاستجابات broadcast، الكلمات المفتاحية **all component**.

في حالة توصيلات من واحد إلى واحد، يمكن حذف شرط **to**، بسبب أن الكيان المستقبَل هو معرف بشكل وحيد بواسطة بنية النظام.

إذا تعين إعادة قيمة إلى الطرف الطالب، يذكر ذلك صراحة باستخدام الكلمة المفتاحية **value**.

مثـــال:

```
MyPort.reply(MyProc2:{ - ,5});           // Replies to an accepted call of MyProc2.

MyPort.reply(MyProc2:{ - ,5}) to MyPeer; // Replies to an accepted call of MyProc2 from MyPeer

MyPort.reply(MyProc2:{ - ,5}) to (MyPeer1, MyPeer2); // Multicast reply to MyPeer1 and MyPeer2

MyPort.reply(MyProc2:{ - ,5}) to all component;// Broadcast reply to all entities connected
                                               // to MyPort

MyPort.reply(MyProc3:{5,MyVar} value 20);// Replies to an accepted call of MyProc3.
```

4.3.23 عملية Getreply

0.4.3.23 عـــام

تستخدم عملية **getreply** لمناولة ردود من إجراء مطلوب في السابق. وتستخدم عملية **getreply** فقط عند منفذ قائم على إجراء (أو مختلط). ويتضمن تعريف النمط اسم الإجراء الذي ينتمي لعملية **getreply**.

تزيل عملية **getreply** رد القمة من صف انتظار منفذ واصل إذا، وإذا فقط، لبت معايير المواءمة المتصاحبة مع عملية **getreply**. وتتعلق معايير المواءمة هذه بتوقيع الإجراء الذي يعالج وشريك الاتصالات. ويمكن تحديد معايير المواءمة لتوقيع إما في الخط أو تشتق من مقاس توقيع.

يمكن تحديد مواءمة مقابل قيمة عودة مستقبَلة باستخدام الكلمة المفتاحية **value**.

يمكن أن تقتصر عملية **getreply** على شريك اتصالات معين في حالة توصيلات من واحد إلى كثيرين. ويدل على هذا التقييد استخدام الكلمة المفتاحية **from**.

المثـــال 1:

```
MyPort.getreply(MyProc:{5, ?} value 20); // Accepts a reply of MyProc with two out or
                                         // inout parameters and a return value of 20

MyPort.getreply(MyProc2:{ - , 5}) from MyPeer; // Accepts a reply of MyProc2 from MyPeer
```

لا يستخدم متغير توقيع لعملية **getreply** ليمر في أسماء متغير لمعلمات **out** و**inout**. ويتم تخصيص قيم معلمة لمتغيرات في جزء تخصيص عملية **out** و**inout**. ويسمح هذا باستخدام مقاسات توقيع في عمليات **getreply** بنفس الطريقة مثل المقاسات التي تستخدم للأنماط.

يتألف جزء التخصيص (الخياري) لعملية **getreply** من تخصيص قيم معلمة **out** و**inout** لمتغيرات واسترداد عنوان المرسل للرد. وتستخدم الكلمة المفتاحية **value** لاسترداد قيم عودة وتستخدم الكلمة المفتاحية **param** لاسترداد قيم معلمة لرد. وتستخدم الكلمة المفتاحية **sender** عندما يطلب استرداد عنوان المرسل.

المثـــال 2:

```
MyPort.getreply(MyProc1:{?, ?} value ?) -> value MyRetValue param(MyPar1,MyPar2);
// The returned value is assigned to variable MyRetValue and the value
// of the two out or inout parameters are assigned to the variables MyPar1 and MyPar2.

MyPort.getreply(MyProc1:{?, ?} value ?) -> value MyRetValue param( - , MyPar2) sender
MySender;
// The value of the first parameter is not considered for the further test execution and
// the address of the sender component is retrieved and stored in the variable MySender.

// The following examples describe some possibilities to assign out and inout parameter values
// to variables. The following signature is assumed for the procedure which has been called

signature MyProc2(in integer A, integer B, integer C, out integer D, inout integer E);

MyPort.getreply(ATemplate) -> param( - , - , - , MyVarOut1, MyVarInout1);

MyPort.getreply(ATemplate) -> param(MyVarOut1:=D, MyVarOut2:=E);

MyPort.getreply(MyProc2:{ - , - , - , 3, ?}) -> param(MyVarInout1:=E);
```

Get any reply 1.4.3.23

تزيل عملية **getreply** ليس لها قائمة متغيرات لمعايير مواءمة توقيع رسالة الرد على قمة صف انتظار منفذ واصل (إن وُجد) إذا تم تلبية جميع معايير الملاءمة الأخرى. ولا تخصص معلمات أو قيم عودة لاستجابات مقبولة من *GetAnyReply* لمتغير. وإذا استخدم *GetAnyReply* في الاستجابة وجزء مناولة استثناء لعملية **call**، تعامل الردود فقط من الإجراء المنفذ بواسطة العملية **call**.

مثـــال:

```
MyPort.getreply;              // Removes the top reply from MyPort.

MyPort.getreply from MyPeer;  // Removes the top reply received from MyPeer from MyPort.

MyPort.getreply -> sender MySenderVar;  // Removes the top reply from MyPort and retrieves the
                                        // address of the sender entity
```

Get a reply على أي منفذ 2.4.3.23

للحصول على رد على أي منفذ، استخدم الكلمات المفتاحية **any port**.

مثـــال:

```
any port.getreply(Myproc)
```

5.3.23 عملية Raise

تستخدم عملية **raise** للحصول على استثناء. ويطلب استثناء فقط عند منفذ قائم على إجراء (أو مختلط). والاستثناء هو رد فعل لنداء إجراء مقبول تؤدي نتيجته إلى حدث استثنائي. ويحدد نمط الاستثناء في توقيع إجراء مطلوب. ويتضمن تعريف النمط لمنفذ في قائمة نداءات إجراءات مقبولة يكون اسم الإجراء يخص الاستثناء.

ملاحظة – لا يمكن دائماً التأكد سكونياً من العلاقة بين نداء مقبول وعملية **raise**. وللاختبار، يسمح بتحديد عملية **raise** دون عملية **getcall** المتصاحبة.

يتألف جزء القيمة لعملية **raise** من مرجع توقيع تتبعه قيمة الاستثناء.

تحدد الاستثناءات مثل الأنماط. ولهذا، يمكن أن تشتق قيمة الاستثناء إما من مقاس أو أن تكون قيمة ناتجة من تعبير (يمكن أن يكون قيمة صريحة بالطبع). ويستخدم مجال نمط خياري في مواصفة القيمة لعملية **raise** في حالات، حيث من الضروري تجنب أي غموض لنمط القيمة الذي يرسَل.

يمكن إرسال استثناءات لعملية واحدة أو أكثر لعملية **call** إلى كيان واحد أو عديد من الكيانات أو إلى جميع نظراء الكيانات الموصلة بالمنفذ المتناول. ويمكن تحديد هذا بنفس الطريقة الواردة في 1.1.2.23. ويعني هذا أن متغير شرط **to** لعملية **raise** هو، لاستثناءات unicast، عنوان كيان مستقبل واحد؛ ولاستثناءات multicast، قائمة عناوين لمجموعة مستقبلات؛ ولاستثناءات broadcast ، الكلمات المفتاحية **all component**.

في حالة توصيلات من واحد إلى واحد، يمكن حذف شرط **to**، بسبب أن الكيان المستقبل هو معرف بشكل وحيد بواسطة بنية النظام.

مثـــال:

```
MyPort.raise(MySignature, MyVariable + YourVariable - 2);
// Raises an exception with a value which is the result of the arithmetic expression
// at MyPort

MyPort.raise(MyProc, integer:5});  // Raises an exception with the integer value 5 for MyProc

MyPort.raise(MySignature, "My string") to MyPartner;
// Raises an exception with the value "My string" at MyPort for MySignature and
// send it to MyPartner

MyPort.raise(MySignature, "My string") to (MyPartnerOne, MyPartnerTwo);
// Raises an exception with the value "My string" at MyPort and sends it to MyPartnerOne and
// MyPartnerTwo (i.e., multicast communication)

MyPort.raise(MySignature, "My string") to all component;
// Raises an exception with the value "My string" at MyPort for MySignature and sends it
// to all entites connected to MyPort (i.e., broadcast communication)
```

6.3.23 عملية Catch

0.6.3.23 عام

تستخدم عملية **catch** للحصول على استثناءات طلبها مكون اختبار أو SUT كرد فعل لنداء إجراء وتستخدم عملية **catch** فقط عند منافذ قائمة على إجراءات (أو مختلطة). ويحدد نمط الاستثناء الذي تم الحصول عليه في توقيع الإجراء الذي طلب الاستثناء. وتحدد الاستثناءات مثل الأنماط ومن ثم يمكن معاملتها مثل رسائل، مثلاً، يمكن أن تستخدم مقاسات للتمييز بين قيم مختلفة لنفس نمط الاستثناء.

تزيل عملية **catch** استثناء القمة من صف انتظار منفذ واصل إذا، وإذا فقط، لبّى استثناء تلك القمة جميع معايير المواءمة المتصاحبة مع عملية **catch**. ولا يحدث ربط للقيم الواصلة لشروط التعبير أو المقاس. ويتم تخصيص قيم الاستثناء لمتغير في جزء تخصيص عملية **catch**.

يمكن أن تقتصر عملية **catch** على شريك اتصالات معين في حالة توصيلات من واحد إلى كثيرين. ويتم هذا التقييد باستخدام الكلمة المفتاحية **from**.

المثـــال 1:

```
MyPort.catch(MyProc, integer: MyVar);   // Catches an integer exception of value
                                        // MyVar raised by MyProc at port MyPort.

MyPort.catch(MyProc, MyVar);            // Is an alternative to the previous example.

MyPort.catch(MyProc, A<B);              // Catches a boolean exception

MyPort.catch(MyProc, MyType:{5, MyVar}); // In-line template definition of an exception value.

MyPort.catch(MyProc, charstring:"Hello")from MyPeer; // Catches "Hello" exception from MyPeer
```

يتألف جزء التخصيص (الخياري) لعملية **catch** من تخصيص قيمة تنفيذ واسترداد عنوان مكون طالب. وتستخدم الكلمة المفتاحية **value** لاسترداد قيمة استثناء وتستخدم الكلمة المفتاحية **sender** عندما تطلب لاسترداد عنوان مرسل.

المثـــال 2:

```
MyPort.catch(MyProc, MyType:?) from MyPartner -> value MyVar;
// Catches an exception from MyPartner and assigns its value to MyVar.

MyPort.catch(MyProc, MyTemplate(5)) -> value MyVarTwo sender MyPeer;
// Catches an exception, assigns its value to MyVarTwo and retrieves the
// address of the sender.
```

يمكن أن تكون عملية **catch** جزءاً من الاستجابة وجزء مناولة استثناء لعملية **call** أو تستخدم لتحديد بديل في بيان **alt**. وإذا استخدمت عملية **catch** في جزء القبول لعملية **call**، تكون المعلومات حول اسم منفذ ومرجع توقيع ليدل على الإجراء الذي طلب الاستثناء إطنابية بسبب أن هذه المعلومات تتبع من عملية **call**. ومع ذلك، ولأجل قابلية القراءة، (مثلاً، في حالة بيانات **call** معقدة)، تتكرر هذه المعلومات.

1.6.3.23 استثناء الإمهال

هناك استثناء واحد **timeout** خاص يمكن الحصول عليه بواسطة عملية **catch**. والاستثناء **timeout** هو حالة طوارئ للخروج في حالات، حيث الإجراء المطلوب لا يجيب ولا يطلب استثناء في وقت محدد مسبقاً (انظر 2.1.3.23).

مثـــال:

```
MyPort.call(MyProc:{5,MyVar}, 20E-3) {
  [] MyPort.getreply(MyProc:{?, ?}) { }
  [] MyPort.catch(timeout) {            // timeout exception after 20 ms
      setverdict(fail);
      stop;
     }
}
```

ويقتصر الحصول على استثناءات **timeout** على جزء مناولة استثناء لنداء. ولا يسمح بمعايير مواءمة إضافية (بما في ذلك جزء **from**) ولا بجزء تخصيص لعملية **catch** يتناول استثناء **timeout**.

2.6.3.23 الحصول على أي استثناء

تسمح عملية **catch** ليس لها قائمة متغيرات بالحصول على أي استثناء صالح. والحالة العامة هي دون استخدام الكلمة المفتاحية **from**. ولا تخصص قيم استثناء مقبولة من *CatchAnyException* لمتغير. وإذا استخدم *CatchAnyException* في الاستجابة وجزء مناولة استثناء لعملية **call**، يعامل فقط الاستثناءات التي طلبها الإجراء الذي نفذته عملية **call**. ويحصل *CatchAnyException* أيضاً على استثناء **timeout**.

مثـــال:

```
MyPort.catch;

MyPort.catch from MyPartner;

MyPort.catch -> sender MySenderVar;
```

3.6.3.23 Catch على أي منفذ

لـ **catch** استثناء على أي منفذ، استخدم الكلمة المفتاحية **any**.

مثـــال:

```
any port.catch;
```

4.23 عملية Check

0.4.23 عـــام

إن عملية **check** هي عملية تنوعية تسمح نفاذ قراءة عنصر القمة لصفوف انتظار منفذ واصلة قائمة على رسالة وقائمة على إجراء دون إزالة عنصر القمة من صف الانتظار. ويتعين على عملية **check** أن تناول قيم نمط معين عند منافذ قائمة على رسالة وتميز بين نداءات لتقبل واستثناء للحصول عليها وإجابات من نداءات سابقة عند منافذ قائمة على إجراء.

تستخدم عمليات استقبال **receive** و**getcall** و**getreply** و**catch**، مع أجزاء مواءمتها وتخصيصها، عملية **check** لتعريف الشرط الذي يتعين التأكد منه واستخراج القيمة أو قيم معلماته، إذا طلبت.

يكون عنصر *القمة* لصف انتظار منفذ واصل هو الذي يتم التأكد منه (ليس من الممكن النظر في صف الانتظار). وإذا كان صف الانتظار فارغاً، تفشل عملية **check**. وإذا لم يكن صف الانتظار فارغاً، تؤخذ نسخة من عنصر القمة وتؤدي عملية استقبال محددة في عملية **check** على النسخة. وتفشل عملية **check** إذا فشلت عملية الاستقبال، أي، لم يتم تلبية معايير المواءمة. وفي هذه الحالة، يتم تجاهل نسخة عنصر القمة لصف الانتظار ويستمر تنفيذ الاختبار بطريقة عادية، أي، يقيم البيان التالي أو البديل لعملية التأكد. وتكون عملية **check** ناجحة إذا كانت عملية الاستقبال ناجحة.

إن استخدام عملية **check** بطريقة خاطئة، مثلاً، التأكد من استثناء عند منفذ قائم على رسالة، يسبب خطأ اختبار مجرد.

ملاحظة – في معظم الحالات، يمكن التأكد من الاستخدام الصحيح لعملية **check** سكونياً، أي، قبل/خلال التجميع.

مثـــال:

```
MyPort1.check(receive(5)); // Checks for an integer message of value 5.

MyPort2.check(getcall(MyProc:{5, MyVar}) from MyPartner);
// Checks for a call of MyProc at port MyPort2 from MyPartner

MyPort2.check(getreply(MyProc:{5, MyVar} value 20));
// Checks for a reply from procedure MyProc at MyPort2 where the returned value is 20 and
// the values of the two out or inout parameters are 5 and the value of MyVar.

MyPort2.check(catch(MyProc, MyTemplate(5, MyVar)));

MyPort2.check(getreply(MyProc1:{?, MyVar} value *) -> value MyReturnValue param(MyPar1,-));

MyPort.check(getcall(MyProc:{5, MyVar}) from MyPartner -> param (MyPar1Var, MyPar2Var));

MyPort.check(getcall(MyProc:{5, MyVar}) -> sender MySenderVar);
```

1.4.23 عملية Check any

تسمح عملية **check** ليس لها قائمة متغيرات التأكد من أن شيئاً ينتظر المعالجة في صف انتظار منفذ واصل. وتسمح عملية **check** any بالتمييز بين مرسلين مختلفين (في حالة توصيلات من واحد إلى واحد) باستخدام شرط، ولاسترداد المرسل باستخدام اختزال جزء التخصيص مع شرط **from**. وفي حالة منافذ مختلطة، تتأكد عملية **sender** من كل من صفوف دخل قائمة على رسالة وقائمة على إجراء للمنفذ المختلط. وإذا كانت عملية **check** تتواءم على كل من صفوف انتظار دخل لمنفذ مختلط، تحظى المعلومات المتعلقة بصف انتظار قائم على إجراء بالأولوية، أي، تعود كنتيجة لعملية **check**. فمثلاً، إذا كانت صفوف انتظار دخل قائمة على رسالة وقائمة على إجراء لمنفذ مختلط ليست فارغة، وينبغي استرداد معلومات مرسل بواسطة عملية **check**، يُعاد مرسل النداء أو الرد أو الاستثناء في صف انتظار دخل قائم على إجراء.

ملاحظة – يمكن استرداد المعلومات المتعلقة بصف انتظار دخل قائم على رسالة لمنفذ مختلط بسهولة باستخدام عملية **check** مع عملية **receive**، مثل:

```
MyPort.check(receive) -> sender Mysender.
```

مثـــال:

```
MyPort.check;

MyPort.check(from MyPartner);

MyPort.check(-> sender MySenderVar);
```

Check on any port 2.4.23

لـ **check** على أي منفذ، استخدم الكلمة المفتاحية **any port**. وفي حالة عملية **check** دون متغير، يتم التأكد من صفوف انتظار دخل لمنافذ مختلطة كما ورد في 1.4.23.

مثـــال:

```
any port.check;
```

5.23 التحكم في منافذ اتصالات

0.5.23 عـــام

إن عمليات TTCN-3 للتحكم في منافذ قائمة على رسالة وقائمة على إجراء ومنافذ مختلطة هي:

- **clear**: تزيل محتويات صف انتظار منفذ واصل؛
- **start**: تزيل محتويات صف انتظار منفذ واصل وتقر صلاحية عمليات إرسال واستقبال عند المنفذ؛
- **stop**: تخمّد إرسال ولا تسمح بعمليات استقبال لتتواءم مع منفذ؛
- **halt**: تخمّد عمليات إرسال عند المنفذ مباشرة ولا تسمح بعمليات استقبال للتواؤم مع رسائل/نداءات/ردود/استثناءات جديدة تدخل صف انتظار المنفذ بعد أداء عملية halt. ويمكن معالجة المداخل الموجودة فعلاً في صف الانتظار.

1.5.23 عملية Clear port

تزيل عملية **clear** محتويات صف انتظار واصل لمنفذ محدد. وإذا كان صف الانتظار المنفّذ فارغاً، لن يكون لهذه العملية أي تأثير.

مثـــال:

```
MyPort.clear;  // clears port MyPort
```

2.5.23 عملية Clear port

إذا عُرّف منفذ على أنه يسمح لعمليات استقبال مثل **receive**، **getcall** وما إلى ذلك، تحرر عملية **start** صف الانتظار الواصل لمنفذ مسمى ويبدأ في الاستماع للحركة عبر المنفذ. وإذا عُرّف المنفذ ليسمح بعمليات إرسال، فإن عمليات مثل **send**، **call**، **raise** وما إلى ذلك، يسمح بأدائها أيضاً عند المنفذ.

مثـــال:

```
MyPort.start;  // starts MyPort
```

بالتغيب، تبدأ جميع منافذ مكون صراحة عندما يخلق مكون. وتسبب عملية start port منافذ غير متوقفة لكي يعاد بدؤها بواسطة إزالة جميع الرسائل المنتظرة في صف انتظار واصل.

3.5.23 عملية Start port

إذا عُرّف منفذ على أنه يسمح بعمليات استقبال مثل **receive** و**getcall**، تسبب عملية **stop** توقف الاستماع عند منفذ مسمى. وإذا عُرّف المنفذ على أنه يسمح بعمليات إرسال، فإن منفذ **stop** لا يسمح بأداء عمليات مثل **raise** و**call** و**send** وما إلى ذلك.

المثـــال 1:

```
MyPort.stop;   // stops MyPort
```

ملاحظة – يعني التوقف عن الاستماع عند المنفذ أن جميع عمليات الاستقبال المعرّفة قبل عملية **stop** تؤدى بالكامل قبل توقف تشغيل المنفذ.

المثـــال 2:

```
MyPort.receive (MyTemplate1) -> value RecPDU;
                              // the received value is decoded, matched against
                              // MyTemplate1 and the matching value is stored
                              // in the variable RecPDU
MyPort.stop;                  // No receiving operation defined following the stop
                              // operation is executed (unless the port is restarted
                              // by a subsequent start operation)
MyPort.receive (MyTemplate2); // This operation does not match and will block (assuming
                              // that no default is activated)
```

4.5.23 عملية halt port

إذا سمح منفذ عمليات استقبال مثل **receive**، **trigger**، **getcall**، لا تسمح عملية **halt** لعمليات استقبال بالنجاح لرسائل وعناصر نداء إجراء تدخل صف انتظار المنفذ بعد أداء عملية **halt** عند ذلك المنفذ. إن رسائل وعناصر نداء إجراء التي كانت في صف الانتظار قبل عملية **halt** يمكن معالجتها مع عمليات الاستقبال. وإذا سمح المنفذ بعمليات إرسال، لا يسمح منفذ **halt** مباشرة بأداء عمليات إرسال مثل **send**، **call**، **raise**، وما إلى ذلك. ولا يكون لعمليات halt أي تأثير على حالة المنفذ أو صف انتظاره.

الملاحظة 1 – تضع عملية **halt** لمنفذ واسم بعد آخر مدخل في صف انتظار مستقبل عندما تؤدي العملية. ويمكن معالجة المداخل قبل الواسم بطريقة عادية. وبعد معالجة جميع المداخل في صف الانتظار السابق للواسم، تكون حالة المنفذ مكافئة لحالة توقف.

الملاحظة 2 – إذا تم أداء عملية **stop** لمنفذ على منفذ متوقف قبل جميع المداخل في صف انتظار سابق للواسم، لا يسمح بعمليات استقبال إضافية مباشرة (أي، ينقل الواسم إلى قمة صف الانتظار).

الملاحظة 3 – إذا حررت عملية **start** لمنفذ على منفذ متوقف في صف الانتظار بغض النظر عن ما إذا كانت قد وصلت قبل أو بعد أداء عملية **halt** لمنفذ. وتزيل أيضا الواسم.

الملاحظة 4 – تحرر عملية **clear** لمنفذ على منفذ متوقف جميع المداخل في صف الانتظار بغض النظر عن ما إذا كانت قد وصلت قبل أو بعد أداء عملية **halt** لمنفذ. وتضع أيضا الواسم عند قمة صف الانتظار.

مثـــال:

```
MyPort.halt;                    // No sending allowed on Myport from this moment on;
                                // processing of messages in the queue still possible.
MyPort.receive (MyTemplate1);   // If a message was already in the queue before the halt
                                // operation and it matches MyTemplate1, it is processed;
                                // otherwise the receive operation blocks.
```

6.23 استخدام any and all مع منافذ

يمكن أن تستخدم الكلمات المفتاحية **any** و**all** مع عمليات تشكيل واتصالات كما ورد في الجدول 18.

الجدول Z.140/18 – Any وAll مع منافذ

مثال	مسموح		عملية
	all	**any**	
any port.receive		نعم	**receive, trigger, getcall, getreply, catch, check**)
			connect/map
all port.start	نعم		**start, stop, clear, halt**

24 عمليات مؤقت

0.24 عـــام

يدعم TTCN-3 عدداً من عمليات مؤقت. ويمكن أن تستخدم هذه العمليات في اختبارات مجردة ووظائف وaltsteps وتحكم وحدة.

يفترض أن كل وحدة منظور TTCN-3 يعلن فيها عن مؤقتات تحتفظ بـ *running-timers list* و*timeout-list* الخاصة بها، أي، قائمة جميع المؤقتات التي تنفذ وقائمة جميع المؤقتات التي لها إمهال. وتكون قوائم الإمهال جزء من اللقطات التي أخذت عند تنفيذ اختبار مجرد. وتحين قائمة الإمهال إذا بدأ مؤقت وحدة منظور أو توقف أو أمهل أو تنفذ عملية **timeout**.

الملاحظة 1 – إن قائمة مؤقتات تنفيذ وقائمة إمهال هي قوائم مفهومية ولا تقيد تنفيذ مؤقتات. ويمكن استخدام أيضاً بنيات معطيات أخرى مثل مجموعة، حيث النفاذ إلى أحداث إمهال غير مقيد بواسطة، مثلاً، ترتيب حدوث أحداث الإمهال.

الملاحظة 2 – يفترض أن لكل مكون اختبار قائمة مؤقتات تنفيذ خاصة، وقائمة إمهال للخروج تقوم بمناولة بدء/وقف مؤقت وأحداث إمهال لمؤقتات معلنة في تعريف نمط مكون متناظر.

عندما ينتهي مؤقت (مفهومياً مباشرة قبل معالجة لقطة لمجموعة أحداث بديلة)، يوضع حدث الإمهال في قائمة إمهال وحدة المنظور الذي أُعلن فيها المؤقت. ويصبح المؤقت غير نشط فوراً. ويمكن أن يظهر مدخل واحد فقط لمؤقت معين في قائمة الإمهال لوحدة منظور التي أعلن فيها المؤقت في أي وقت من الأوقات.

تُلغى جميع المؤقتات المنفذة أوتوماتياً عندما يتوقف المكون ضمنياً أو صراحة.

الجدول Z.140/19 – نظرة شاملة على عمليات مؤقت TTCN-3

عمليات مؤقت	
بيان	**الكلمة المفتاحية أو الرمز المتصاحب**
ابدأ مؤقتاً	**start**
أوقف مؤقتاً	**stop**
اقرأ وقت الانقضاء	**read**
تأكد من أن المؤقت ينفذ	**running**
حدث إمهال	**timeout**

1.24 عملية Start timer

تستخدم عملية timer **start** لتدل على وجوب أن يبدأ المؤقت التنفيذ. وتكون قيم مؤقت أعداد **float** غير سالبة (أي، أكبر من أو مساوية لـ 0.0) وعندما يبدأ مؤقت، يضاف اسمه إلى قائمة مؤقتات التنفيذ (لوحدة المنظور المعنية).

مثـــال:

```
MyTimer1.start;         // MyTimer1 is started with the default duration
MyTimer2.start(20E-3); // MyTimer2 is started with a duration of 20 ms

// Elements of timer arrays may also be started in a loop, for example
timer t_Mytimer [5];
var float v_timerValues [5];

for (var integer i := 0; i<=4; i:=i+1)
  { v_timerValues [i] := 1.0 }

for (var integer i := 0; i<=4; i:=i+1)
  {t_Mytimer [i].start ( v_timerValues [i])}
```

تستخدم معلمة قيمة مؤقت خيارية إذا لم تتوفر مدة بالتغيب، أو إذا كانت هناك رغبة في تجاهل قيمة بالتغيب محددة في إعلان المؤقت. وعند تجاهل مدة مؤقت، تنطبق القيمة الجديدة فقط على المطابق الحالي للمؤقت، تستخدم، أي عمليات **start** لاحقة لهذا المؤقت، لا تحدد مدة، مدة بالتغيب.

يعني بدء مؤقت بقيمة مؤقت 0.0 أن المؤقت يمهل فوراً. وبدء مؤقت بقيمة مؤقت سلبية، مثلاً، تكون قيمة المؤقت هي نتيجة تعبير، أو دون قيمة مؤقت محددة، يسبب خطأ في وقت التنفيذ.

تنفذ ميقاتية المؤقت من قيمة float صفر (0.0) حتى الحد الأقصى الذي تقرره معلمة المدة.

يمكن تطبيق عملية **start** على مؤقت تنفيذ، يكون فيها المؤقت قد توقف وأعيد بدؤه. ويُزال أي مدخل في قائمة إمهال من قائمة الإمهال.

2.24 عملية Stop timer

تستخدم عملية **stop** لوقف مؤقت ينفذ ولإزالته من قائمة مؤقتات تنفذ. ويصبح مؤقت متوقف غير نشط ويضبط وقت انقضائه على قيمة float صفر (0.0).

إن وقف مؤقت غير نشط هي عملية صالحة، بالرغم من عدم وجود أي تأثير لها. ويسبب وقف مؤقت منتهي إزالة مدخل لهذا المؤقت في قائمة الإمهال. ويمكن استخدام الكلمة المفتاحية **all** لوقف جميع المؤقتات المرئية في وحدة المنظور التي طلبت فيها عملية **stop**.

مثـــال:

```
MyTimer1.stop;     // stops MyTimer1
all timer.stop;    // stops all running timers
```

3.24 عملية Read timer

تستخدم عملية **read** لاسترداد الوقت الذي انقضى منذ أن بدأ المؤقت المحدد. وتكون القيمة المعادة من نمط **float**.

مثـــال:

```
var float Myvar;
MyVar:= MyTimer1.read; // assign to MyVar the time that has elapsed since MyTimer1 was started
```

عند تطبيق عملية **read** على مؤقت غير نشيط، أي، على مؤقت لا يرد في قائمة مؤقتات منفذة، يعيد قيمة float صفر (0.0).

4.24 عملية Running timer

تستخدم عملية **running** للتأكد ما إذا كان المؤقت موجوداً أم لا في قائمة مؤقتات تنفذ لوحدة منظور معينة (أي، أنه بدأ و لم يمهل أو يتوقف). وتعيد العملية قيمة **true** إذا كان المؤقت موجوداً في القائمة، وإلا تعيد **false**.

مثـــال:

```
if (MyTimer1.running) { … }
```

5.24 عملية Timeout

تسمح عملية **timeout** بالتأكد من انتهاء صلاحية مؤقت، أو جميع المؤقتات، في وحدة منظور لمكون اختبار أو تحكم وحدة طلبت فيها عملية إمهال.

عندما تعالج عملية **timeout**، إذا ذكر اسم مؤقت، يجري البحث عن قوائم الإمهال طبقاً لقواعد منظور TTCN-3. وإذا كان هناك حدث إمهال يتواءم مع اسم المؤقت، يزال ذلك الحدث من قائمة الإمهال، وتنجح عملية **timeout**. ولا يستخدم **timeout** في تعبير **boolean**، ولكن يمكن أن يستخدم لتحديد بديل في بيان **alt** أو كبيان بمفرده في وصف سلوك. وفي الحالة الأخيرة، تعتبر عملية **timeout** اختزالاً لبيان **alt** مع بديل واحد فقط، أي، له علم دلالات سد، وبالتالي يوفر المقدرة على الانتظار السلبي لإمهال مؤقت (مؤقتات).

المثـــال 1:

```
MyTimer1.timeout; // checks for the timeout of the previously started timer MyTimer1
```

تنجح الكلمة المفتاحية **any** المستخدمة مع عملية **timeout** (بدلاً من مؤقت مسمّى صراحة) إذا لم تكن قائمة الإمهال فارغة.

المثـــال 2:

```
any timer.timeout; // checks for the timeout of any previously started timer
```

6.24 موجز استخدام any وall مع مؤقتات

يمكن أن تستخدم الكلمات المفتاحية **any** و**all** مع عمليات مؤقت كما ورد في الجدول 20.

الجدول Z.140/20 – any وall مع مؤقتات

عملية	مسموح		مثال
	any	all	
start			
stop		نعم	all timer.stop
read			
running	نعم		if (any timer.running) {…}
timeout	نعم		any timer.timeout

25 عملية Test verdict

0.25 عـــام

تسمح عمليات Verdict أن ترسل وتستقبل أحكاماً باستخدام عمليتي **setverdict** و**getverdict** على التوالي. وتستخدم العمليتان فقط في اختبارات مجردة وaltsteps ووظائف.

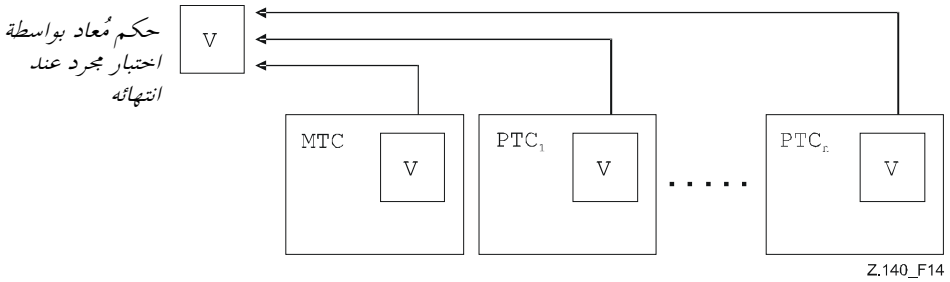
الجدول Z.140/21 – نظرة شاملة على عمليات حكم اختبار TTCN-3

عمليات حكم اختبار	
بيان	**الكلمة المفتاحية أو الرمز المتصاحب**
اضبط حكماً محلياً	**setverdict**
احصل على حكم محلي	**getverdict**

يحتفظ كل مكون اختبار لتشكيل نشيط بحكمه المحلي. والحكم المحلي هو شيء يخلق لكل مكون اختبار وقت خلقه. ويستخدم لتتبع الحكم الفردي في كل مكون اختبار (أي، في MTC وفي كل PTC).

1.25 حكم اختبار مجرد

بالإضافة إلى ذلك، يوجد حكم اختبار مجرد عالمي مستطبق ومتناول بواسطة نظام الاختبار المُحيَّن عندما ينهي كل مكون اختبار التنفيذ (أي، MTC وكل PTC). وهذا الحكم لا يمكنه النفاذ إلى عمليات **getverdict** و**setverdict**. وتعاد قيمة هذا الحكم بواسطة اختبار مجرد عند انتهاء التنفيذ. وإذا لم يحفظ الحكم العائد صراحة في جزء التحكم (مثلاً، مخصص لمتغير)، فتتم خسارته.



الشكل Z.140/14 – توضيح العلاقة بين الأحكام

ملاحظة – لا يحدد TTCN-3 الآليات الفعلية التي تؤدي تحيين أحكام محلية واختبار مجرد. وتنفذ هذه الآليات على نحو مستقل.

2.25 قيم حكم وقواعد الكتابة الفوقية

0.2.25 عـــام

يمكن أن يكون لحكم خمس قيم مختلفة: **pass** و**fail** و**inconc** و**none** و**error**، أي، القيم المميزة لـ **verdicttype** (انظر 1.6).

ملاحظة – تعني **inconc** حكماً غير نهائي.

تستخدم عملية **setverdict** فقط مع قيم **pass** و**fail** و**inconc** و**none**.

المثـــال 1:

```
setverdict(pass);
setverdict(inconc);
```

يمكن استرداد قيمة الحكم المحلي باستخدام عملية **getverdict**.

المثـــال 2:

```
MyResult := getverdict; // Where MyResult is a variable of type verdicttype
```

عندما يُستطبق مكون اختبار، يُخلق شيء حكمه المحلي ويُضبط على قيمة **none**.

عند تغيير قيمة حكم محلي (أي، باستخدام عملية **setverdict**)، يتبع تأثير هذا التغيير قواعد الكتابة الفوقية الواردة في الجدول 22. ويُحيَّن حكم اختبار مجرد ضمنياً عند انتهاء مكون الاختبار. ويتبع تأثير العملية الضمنية هذه قواعد الكتابة الفوقية الواردة في الجدول 22.

الجدول Z.140/22 – قواعد الكتابة الفوقية للحكم

القيمة الحالية لحكم	قيمة تخصيص حكم جديد			
	pass	**inconc**	**fail**	**none**
none	**pass**	**inconc**	**fail**	**none**
pass	**pass**	**inconc**	**fail**	**pass**
inconc	**inconc**	**inconc**	**fail**	**inconc**
fail	**fail**	**fail**	**fail**	**fail**

المثـــال 3:

```
 :
setverdict(pass); // the local verdict is set to pass
 :
setverdict(fail); // until this line is executed, which will result in the value
 :                // of the local verdict being overwritten to fail
 :                // When the ptc terminates the test case verdict is set to fail
```

1.2.25 حكم خطأ

إن حكم **error** هو خاص في أن نظام الاختبار يضعه ليدل على أن خطأ اختبار مجرد (أي، وقت التنفيذ) قد حدث. ولا تضعه عملية **setverdict** ولا تعيده عملية **getverdict**. ولا يمكن لقيمة حكم آخر أن تتجاهل حكم **error**. ويعني هذا أن حكم **error** يمكن أن يكون نتيجة فقط لعملية اختبار مجرد **execute**.

26 أعمال خارجية

في بعض حالات الاختبار، يمكن أن يكون سطح بيني (أسطح بينية) لـ SUT غائبة أو غير معروفة بديهياً (مثلاً، سطح بيني الإدارة) ولكن يمكن أن يكون من الضروري حث SUT على تنفيذ بعض الأعمال (مثلاً، إرسال رسالة إلى نظام الاختبار). وأيضاً، يمكن أن تكون بعض الأعمال مطلوبة من الموظفين المنفذين للاختبار (مثلاً، لتغيير الأوضاع البيئية للاختبار مثل درجة الحرارة وفُلطية التغذية الآلية وما إلى ذلك).

يمكن وصف العمل المطلوب كتعبير سلسلة، أي، استخدام سلاسل حرفية ومتغيرات سلسلة منمطة ومعلمات وما إلى ذلك، ويسمح بأي تسلسل بعد ذلك.

مثـــال:

```
var charstring myString:= " now."
action("Send MyTemplate on lower PCO" & myString);  // Informal description of the
                                                    // external action
```

لا توجد مواصفة لما يتم أو بواسطة SUT لبدء هذا العمل، فقط وصف غير رسمي للعمل المطلوب نفسه.

يمكن استخدام أعمال خارجية في اختبارات مجردة ووظائف وaltsteps وتحكّم وحدة.

27 جزء تحكم وحدة

0.27 عـــام

تُعرّف اختبارات مجردة في جزء تعاريف الوحدة بينما يدير جزء تحكم الوحدة تنفيذها. وتمر جميع المتغيرات المعرّفة (إن وُجدت) في جزء التحكم لوحدة إلى اختبار مجرد بواسطة معلمية إذا استخدمت في تعريف سلوك لذلك الاختبار المجرد، أي، لا يدعم TTCN-3 المتغيرات العالمية لأي نوع.

في بداية كل اختبار مجرد، يُعاد ضبط تشكيل الاختبار. ويعني هذا أن جميع المكونات والمنافذ التي قامت بها عمليات **create** و**connect** في اختبار مجرد سابق قد تم تدميرها عندما توقف الاختبار المجرد (ومن ثم، هي غير مرئية للاختبار المجرد الجديد).

1.27 تنفيذ اختبارات مجردة

يطلب اختبار مجرد باستخدام بيان **execute**. ونتيجة لتنفيذ اختبار مجرد، يعاد حكم اختبار مجرد سواء **none** أو **pass** أو **inconc** أو **fail** أو **error** ويمكن أن يخصص لمتغير لمزيد من المعالجة.

وخيارياً، يسمح بيان **execute** بالإشراف على اختبار مجرد بواسطة مدة مؤقت (انظر 5.27).

مثـــال:

```
execute(MyTestCase1());                    // executes MyTestCase1, without storing the
                                           // returned test verdict and without time
                                           // supervision

MyVerdict := execute(MyTestCase2());       // executes MyTestCase2 and stores the resulting
                                           // verdict in variable MyVerdict

MyVerdict := execute(MyTestCase3(),5E-3); // executes MyTestCase3 and stores the resulting
                                           // verdict in variable MyVerdict. If the test case
                                           // does not terminate within 5ms, MyVerdict will
                                           // get the value 'error'
```

2.27 إنهاء اختبارات مجردة

ينتهي اختبار مجرد مع انتهاء MTC. وعند انتهاء MTC (ضمنياً أو صراحة)، تُزال جميع مكونات الاختبار المتوازية المنفذة بواسطة نظام الاختبار.

الملاحظة 1 – إن الآلية المحسوسة لوقف جميع PTCs هي أداة محددة، وبالتالي هي خارج مدى هذه التوصية.

يُحسب الحكم النهائي لاختبار مجرد على أساس الأحكام المحلية النهائية لمكونات اختبار مختلفة طبقاً للقواعد المعرّفة في القسم 25. ويصبح الحكم المحلي الفعلي لمكون اختبار حكمه المحلي النهائي عندما ينهي مكون الاختبار نفسه أو يتوقف بنفسه، أو مكون اختبار آخر أو بواسطة نظام الاختبار.

الملاحظة 2 – لتجنب الشروط الحرجة لحساب أحكام اختبار نتيجة للتوقف المتأخر لـ PTCs، ينبغي أن يضمن MTC أن جميع PTCs قد توقفت (بواسطة بيان **done** أو **killed**).

3.27 التحكم في تنفيذ اختبارات مجردة

يمكن استخدام بيانات برنامج، المقتصرة على المعرفة في الجدولين 11 و12، في جزء التحكم لوحدة لتحديد أشياء مثل الترتيب الذي تنفذ به الاختبارات المجردة أو عدد المرات التي ينبغي أن ينفذها الاختبار المجرد.

مثـــال:

```
module MyTestSuite () {
   :
  control {
       :
      // Do this test 10 times
      count:=0;
      while (count < 10)
      {    execute (MySimpleTestCase1());
       count := count+1;
      }
  }
}
```

إذا لم تستخدم بيانات برمجة، فإن الاختبارات المجردة تنفذ، بالتغيب، بالترتيب التتابعي الذي تظهر به في تحكم الوحدة.

ملاحظة – لا يستثني هذا إمكانية أن بعض الأدوات يمكن أن تتجاهل الترتيب بالتغيب لتسمح للمستعمل أو الأداة اختيار ترتيب تنفيذ مختلف.

ويمكن استخدام اختيار وعدم اختيار اختبارات مجردة للتحكم في تنفيذ اختبارات مجردة (انظر 4.27).

4.27 اختيار اختبارات مجردة

توجد طرق مختلفة في TTCN-3 لاختيار وعدم اختيار اختبارات مجردة. فمثلاً، يمكن استخدام تعبيرات بولانية لاختيار ولعدم اختيار أي اختبارات مجردة تنفذ. ويشمل هذا، بالطبع، استخدام وظائف تعيد قيمة **boolean**.

المثـــال 1:

```
module MyTestSuite () {
    :
  control {
       :
      if (MySelectionExpression1()) {
       execute(MySimpleTestCase1());
       execute(MySimpleTestCase2());
       execute(MySimpleTestCase3());
      }
      if (MySelectionExpression2()) {
       execute(MySimpleTestCase4());
       execute(MySimpleTestCase5());
       execute(MySimpleTestCase6());
      }
       :
  }
}
```

وطريقة أخرى لتنفيذ اختبارات مجردة كزمرة هي جمعها في وظيفة، وتنفيذ الوظيفة من تحكم وحدة.

المثـــال 2:

```
function MyTestCaseGroup1()
{  execute(MySimpleTestCase1());
   execute(MySimpleTestCase2());
   execute(MySimpleTestCase3());
}
function MyTestCaseGroup2()
{  execute(MySimpleTestCase4());
   execute(MySimpleTestCase5());
   execute(MySimpleTestCase6());
}
 :
control
{  if (MySelectionExpression1()) { MyTestCaseGroup1(); }
   if (MySelectionExpression2()) { MyTestCaseGroup2(); }
    :
}
```

ونظراً لأن اختبار مجرد يعيد قيمة وحيدة لنمط **verdicttype**، من الممكن أيضاً التحكم في ترتيب تنفيذ اختبار مجرد يعتمد على ناتج اختبار مجرد. واستخدام **verdicttype** TTCN-3 هو طريقة أخرى لاختيار اختبارات مجردة.

المثـــال 3:

```
if ( execute (MySimpleTestCase()) == pass )
 { execute (MyGoOnTestCase()) }
else
 { execute (MyErrorRecoveryTestCase()) };
```

5.27 استخدام مؤقتات في التحكم

يمكن استخدام مؤقتات للإشراف على تنفيذ اختبار مجرد. ويتم هذا باستخدام إمهال صريح في بيان **execute**. وإذا لم ينتهِ الاختبار المجرد في المدة هذه، تكون نتيجة تنفيذ الاختبار المجرد حكم خطأ وينهي نظام الاختبار الاختبارَ المجرد. ويكون المؤقت المستخدم للإشراف على اختبار مجرد هو مؤقت نظام ولا يحتاج للبدء أو الإعلان عنه.

المثـــال 1:

```
MyReturnVal := execute (MyTestCase(), 7E-3);
// Where the return verdict will be error if MyTestCase does not complete execution
// within 7 ms
```

يمكن أن تستخدم أيضاً عمليات مؤقت صراحة للتحكم في تنفيذ اختبار مجرد.

المثـــال 2:

```
// Example of the use of the running timer operation
while (T1.running or x<10) // Where T1 is a previously started timer
{ execute(MyTestCase());
  x := x+1;
}
```

```
// Example of the use of the start and timeout operations
timer T1 := 1.0;
 :
execute(MyTestCase1());
T1.start;
T1.timeout; // Pause before executing the next test case
execute(MyTestCase2());
```

28 تحديد نعوت

0.28 عـــام

يمكن أن تتصاحب نعوت مع عناصر لغة TTCN-3 بواسطة بيان **with**. وتُعرّف قواعد تركيب متغير لبيان **with** (أي، النعوت الفعلية) كسلسلة نص حر.

هناك أربعة أنواع من النعوت:

أ) **display**: يسمح لمواصفة بعرض نعوت متعلقة بأنساق تقديم محددة؛

ب) **encode**: يسمح بمراجع لقواعد تشفير محددة؛

ج) **variant**: يسمح بمراجع لمتغيرات تشفير محددة؛

د) **extension**: يسمح بمواصفة لنعوت معرفة لمستعمل.

1.28 عرض نعوت

يمكن أن يكون لجميع عناصر لغة TTCN-3 نعوت **display** لتحديد كيفية عرض عناصر لغة معينة، مثلاً، في نسق جدول.

توجد سلسلة نعوت خاصة متعلقة بعرض نعوت لنسق تقديم (مطابقة) جدول في [1] ITU-T Z.141.

توجد سلسلة نعوت خاصة متعلقة بعرض نعوت لنسق تقديم بياني في [2] ITU-T Z.142.

يمكن أن يُعرّف مستعمل نعوت **display** أخرى.

ملاحظة – بسبب أن النعوت المعرّفة لمستعمل ليست معيارية، يمكن أن يختلف تفسير هذه النعوت بين أدوات أو لا يمكن حتى دعمها.

2.28 تشفير قيم

0.2.28 عـــام

تُعرّف قواعد التشفير قيمة ومقياس معين وما إلى ذلك وتشفر وترسل عبر **port** اتصالات وكيفية تشفير الإشارات المستقبَلة. ولا يوجد لدى TTCN-3 آلية تشفير بالتغيب. ويعني هذا أن قواعد التشفير أو تعليمات التشفير معرّفة بطريقة خارجية لـ TTCN-3.

في TTCN-3، يمكن أن تحدد قواعد تشفير عامة أو خاصة باستخدام نعوت **encode** و**variant**.

1.2.28 نعوت Encode

يسمح نعت **encode** بتصاحب قاعدة تشفير مرجعية أو توجيه تشفير لتعريف TTCN-3.

إن الطريقة التي تُعرّف بها قواعد التشفير الفعلية (مثلاً، نثر ووظائف وما إلى ذلك) هي خارج مدى هذه التوصية. وإذا لم تكن قواعد معينة مرجعية، فإن التشفير يعتمد على التنفيذ الفردي.

في معظم الحالات، تستخدم نعوت التشفير بطريقة تراتبية. ويكون المستوى الأعلى هو وحدة كاملة، ويكون المستوى التالي هو زمرة، والمستوى الأدنى هو نمط فردي أو تعريف:

أ) **module**: ينطبق التشفير على جميع الأنماط المعرفة في الوحدة، بما في ذلك أنماط TTCN-3 (أنماط مبنية)؛

ب) **group**: ينطبق التشفير على زمرة تعاريف نمط معرفة لمستعمل؛

ج) **type** أو **definition**: ينطبق التشفير على نمط أو تعريف معرف لمستعمل؛

د) **field**: ينطبق التشفير على مجال في **record** أو **set** أو **template**.

مثـــال:

```
module MyTTCNmodule
{ :
  import from MySecondModule {
      type MyRecord
  }
  with { encode "MyRule 1" } // Instances of MyRecord will be encoded according to MyRule 1

  :
  type charstring MyType;  // Normally encoded according to the 'Global encoding rule
  :
  group MyRecords
  {   :
      type record MyPDU1
      {
       integer field1,      // field1 will be encoded according to 'Rule 3
       boolean field2,      // fild2 will be encoded according to 'Rule 3'
       Mytype     field3    // field3 will be encoded according to 'Rule 2
      }
      with { encode (field1, field2) "Rule 3" }
      :
  }
  with { encode "Rule 2" }

}
with { encode "Global encoding rule" }
```

2.2.28 نعوت Variant

لتنقيح تخطيط تشفير محدد حالي بدلاً من إحلاله، يستخدم نعت **variant**. إن نعوت variant مختلفة عن النعوت الأخرى بسبب أنها تتعلق عن قرب بنعوت encode. ولهذا، بالنسبة لنعوت variant، تنطبق قواعد كتابة فوقية إضافية (انظر 1.5.28).

مثـــال:

```
module MyTTCNmodule1
{ :
  type charstring MyType;  // Normally encoded according to the 'Global encoding rule'
  :
  group MyRecords
  {   :
      type record MyPDU1
      {
       integer field1,      // field1 will be encoded according to 'Rule 2'
                            // using encoding variant 'length form 3'
       Mytype field3        // field3 will be encoded according to 'Rule 2'
                            // using any possible length encoding format
      }
      with { variant (field1) "length form 3" }
      :
  }
  with { encode "Rule 2" }

}
with { encode "Global encoding rule" }
```

3.2.28 سلاسل خاصة

إن السلاسل التالية هي نعوت **variant** معرّفة مسبقاً (معيارية) لأنماط أساسية بسيطة (انظر 1.2.E):

أ) "8 bit" و"unsigned 8 bit" تعني، عندما تنطبق على صحيح وأنماط معددة، يتم مناولة قيمة الصحيح أو أعداد الصحيح المتصاحبة مع تعديدات كما لو قدمت على 8 بتات (بايتة وحيدة) في النظام.

ب) "16 bit" و"unsigned 16 bit" تعني، عندما تنطبق على صحيح وأنماط معددة، يتم مناولة قيمة الصحيح أو أعداد الصحيح المتصاحبة مع تعديدات كما لو قدمت على 16 بتة (بايتتان) في النظام.

ج) "32 bit" و"unsigned 32 bit" تعني، عندما تنطبق على صحيح وأنماط معددة، يتم مناولة قيمة الصحيح أو أعداد الصحيح المتصاحبة مع تعديدات كما لو قدمت على 32 بتة (أربع بايتات) في النظام.

د) "64 bit" و"unsigned 64 bit" تعني، عندما تنطبق على صحيح وأنماط معددة، يتم مناولة قيمة الصحيح أو أعداد الصحيح المتصاحبة مع تعديدات كما لو قدمت على 64 بتة (8 بايتات) في النظام.

هـ) "IEEE754 extended float"، "IEEE754 float","IEEE754 double" تعني، عندما تنطبق على نمط طليق، تُشفّر القيمة ويُفكك تشفيرها طبقاً لمعيار IEEE 754 (انظر البيبليوغرافيا).

إن السلاسل التالية هي نعوت **variant** معرّفة مسبقاً (معيارية) لـ **charstring** و**universal charstring** (انظر 2.2.E):

أ) "UTF-8" تعني، عندما تنطبق على نمط سلسلة سمات عالمية، تُشفر ويُفكك تشفير كل سمة قيمة فردياً طبقاً لـ UCS Transformation Format 8 (UTF-8) كما عُرّف في الملحق R من [10] ISO/IEC 10646.

ب) "UCS-2" تعني، عندما تنطبق على نمط سلسلة سمات عالمية، تُشفر ويُفكك تشفير كل سمة قيمة فردياً طبقاً لـ UCS-2 coded representation form (انظر 1.14 من [10] ISO/IEC 10646).

ج) "UTF-16" تعني، عندما تنطبق على نمط سلسلة سمات عالمية، تُشفر ويُفكك تشفير كل سمة قيمة فردياً طبقاً لـ UCS Transformation Format 16 (UTF-16) كما عُرّف في الملحق Q من [10] ISO/IEC 10646.

د) "8 bit" تعني، عندما تنطبق على نمط سلسلة سمات عالمية، تُشفر ويُفكك تشفير كل سمة قيمة فردياً طبقاً للتقديم المشفر كما حدد في ISO/IEC 8859 (تشفير 8 بتات).

إن السلسلة التالية هي نعت **variant** مُعرّف مسبقاً (معياري) لأنماط مبنية (انظر 3.2.E):

أ) "IDL:fixed FORMAL/01-12-01 v.2.6" يعني، عندما ينطبق على نمط سجل، يتم مناولة القيمة باعتبارها قيمة عشرية لنقطة ثابتة IDL (انظر البيبليوغرافيا).

يمكن استخدام نعوت variant في مركب مع نعوت مشفرة عامة أكثر محددة للمستوى الأعلى. فمثلاً، يسبب **universal charstring** محدد مع نعت **variant** "UTF-8" في وحدة لها نعت تشفير عالمي انظر 2.12 من التوصية ITU-T Z.146 [6] "BER:1997" لكل سمة للقيمة في سلسلة تشفر أولاً لقواعد UTF-8 التالية ثم تشفر قيمة UTF-8 هذه متبعة قواعد BER الأكثر عالمية.

4.2.28 تشفيرات غير صالحة

من المرغوب تحديد قواعد تشفير غير صالحة وبالتالي تحدد هذه في مصدر قابل للإشارة إليه خارجي عن الوحدة بنفس الطريقة التي يشار إليها لقواعد التشفير الصالحة.

3.28 نعوت تمديد

يمكن أن يكون لعناصر لغة TTCN-3 نعوت **extension** يحددها المستعمل.

ملاحظة – بسبب أن النعوت المعرّفة لمستعمل ليست معيارية، يمكن أن يختلف تفسير هذه النعوت بين أدوات يوردها بائعون مختلفون أو لا يمكن حتى دعمها.

4.28 منظور لنعوت

يمكن أن يتصاحب بيان **with** مع نعوت لعنصر واحد للغة. ومن الممكن أن تصاحب نعوت عدد من عناصر لغة بواسطة، مثلاً، الاستماع إلى مجالات نمط مبني في بيان نعت متصاحب مع تعريف نمط وحيد أو متصاحب مع بيان **with** لوحدة منظور محيطة أو **group** لعناصر لغة.

مثـــال:

```
// MyPDU1 will be displayed as PDU
type record MyPDU1 { … } with { display "PDU"}

// MyPDU2 will be displayed as PDU with the application specific extension attribute MyRule
type record MyPDU2 { … }
with
{
  display "PDU";
  extension "MyRule"
}

// The following group definition …
group MyPDUs {
  type record MyPDU3 { … }
  type record MyPDU4 { … }
}
with {display "PDU"}   // All types of group MyPDUs will be displayed as PDU
```

```
// is identical to
group MyPDUs {
   type record MyPDU3 { … } with { display "PDU"}
   type record MyPDU4 { … } with { display "PDU"}
}
```

5.28 قواعد الكتابة الفوقية لنعوت

يتجاهل تعريف نعت في وحدة منظور دنيا تعريف نعت عام في منظور أعلى. وتُعرّف قواعد كتابة فوقية إضافية لنعوت variant في 1.5.28.

المثـــال 1:

```
type record MyRecordA
{
   :
} with { encode "RuleA" }

// In the following, MyRecordA is encoded according to RuleA and not according to RuleB
type record MyRecordB
{
    :
   field    MyRecordA
} with { encode "RuleB" }
```

إن بيان **with** الموضوع داخل منظور لبيان **with** آخر يتجاهل **with** في أقصى الخارج. وينطبق هذا أيضاً على استخدام بيان **with** مع زمرات. وينبغي إيلاء العناية عندما يستخدم تخطيط كتابة فوقية مع مراجع لتعاريف وحيدة. والقاعدة العامة هي أن تخصص النعوت وتتم الكتابة الفوقية طبقاً لترتيب حدوثها.

```
// Example of the use of the overwriting scheme of the with statement
group MyPDUs
{
   type record MyPDU1 { … }
   type record MyPDU2 { … }

   group MySpecialPDUs
   {
       type record MyPDU3 { … }
       type record MyPDU4 { … }
   }
   with {extension "MySpecialRule"} // MyPDU3 and MyPDU4 will have the application
                                    // specific extension attribute MySpecialRule
}
with
{
   display "PDU";       // All types of group MyPDUs will be displayed as PDU and
   extension "MyRule";  // (if not overwritten) have the extension attribute MyRule
}

// is identical to …
group MyPDUs
{
   type record MyPDU1 { … } with {display "PDU"; extension "MyRule" }
   type record MyPDU2 { … } with {display "PDU"; extension "MyRule" }
   group MySpecialPDUs {
       type record MyPDU3 { … } with {display "PDU"; extension "MySpecialRule" }
       type record MyPDU4 { … } with {display "PDU"; extension "MySpecialRule" }
   }
}
```

يمكن كتابة فوقية لتعريف نعت في منظور أدنى لأي منظور أعلى باستخدام توجيه **override**.

المثـــال 2:

```
type record MyRecordA
{
    :
} with { encode "RuleA" }

// In the following, MyRecordA is encoded according to RuleB
type record MyRecordB
{
    :
   fieldA  MyRecordA
} with { encode override "RuleB" }
```

يجبر توجيه **override** جميع الأنماط المحتوية في جميع منظورات دنيا على أن تفرض على نعت محدد.

1.5.28 قواعد كتابة فوقية لنعوت variant إضافية

إن نعت **variant** يتعلق دائماً بنعت **encode**. بينما متغير تشفير يمكن أن يتغير، لا يتغير تشفير دون كتابة فوقية لجميع نعوت variant. ولهذا، تنطبق قواعد الكتابة الفوقية التالية على نعوت variant:

- يُكتب فوقياً نعت **variant** على نعت **variant** الحالي طبقا للقاعدة المعرفة في 5.28؛
- إن نعت **encoding**، الذي يكتب فوقياً نعت **encoding** الحالي طبقا للقاعدة المعرفة في 5.28، يكتب فوقياً أيضاً نعت **variant** الحالي المتناظر، أي، لا يتوفر نعت **variant** جديد، ولكن يصبح نعت **variant** الحالي غير نشط؛
- إن نعت **encoding**، الذي يغير نعت **encoding** الحالي لعنصر لغة مستوردة طبقا للقواعد المعرفة في 6.28، يغير أيضاً نعت **variant** الحالي المتناظر، أي، لا يتوفر نعت **variant** جديد، ولكن يصبح نعت **variant** الحالي غير نشط.

مثـــال:

```
module MyVariantEncodingModule {
   :
   type charstring MyCharString;     // Normally encoded according to "Encoding 1"
   :
   group MyVariantsOne {
       :
       type record MyPDUone
       {
       integer field1,  // field1 will be encoded according to "Encoding 2" only.
             // "Encoding 2" overwrites "Encoding 1" and variant 'Variant 1'
       Mytype field3     // field3 will be encoded according to "Encoding 1" with
             // variant "Variant 1".
       }
       with { encoding (field1) "Encoding 2" }
       :
   }
   with { variant "Variant 1" }

   group MyVariantsTwo
   {   :
       type record MyPDUtwo
       {
       integer  field1, // field1 will be encoded according to "Encoding 3"
                // using encoding variant 'Variant 3'
       Mytype   field3  // field3 will be encoded according to "Encoding 3"
                // using encoding variant "Variant 2"
       }
       with { variant (field1) "Variant 3" }
       :
   }
   with { encode "Encoding 3"; variant 'Variant 2'}

}
with { encode "Encoding 1" }
```

6.28 تغيير نعوت لعناصر لغة مستوردة

بشكل عام، يستورد عنصر لغة مع نعوته. وفي بعض الحالات، يتعين تغيير هذه النعوت عند استيراد عنصر لغة، مثلاً، يمكن عرض نمط في وحدة واحدة باعتباره ASP، ثم يستورد بواسطة وحدة أخرى حيث ينبغي أن يعرض باعتباره PDU. وفي هذه الحالات، يسمح بتغيير النعوت في بيان **import**.

مثـــال:

```
import from MyModule {
   type MyType
}
with { display "ASP" }  // MyType will be displayed as ASP

import from MyModule {
   group MyGroup
}
with {
   display "PDU";   // By default all types will be displayed as PDU
   extension "MyRule"
}
```

الملحق A

BNF وعلم الدلالات السكوني

TTCN-3 BNF 1.A

0.1.A عـــام

يُعرف هذا الملحق علم الدلالات لـ TTCN-3 باستخدام BNF ممدد (ويسمى هنا BNF فقط).

1.1.A تحويلات لوصف علم الدلالات

يعرف الجدول 1.A الترميز التحويلي المستخدم لتحديد قواعد BNF ممدد لـ TTCN-3

الجدول Z.140/1.A – ترميز تحويلي تركيبي

::=	is defined to be
abc xyz	abc followed by xyz
|	alternative
[abc]	0 or 1 instances of abc
{abc}	0 or more instances of abc
{abc}+	1 or more instances of abc
(...)	textual grouping
Abc	the non-terminal symbol abc
"abc"	a terminal symbol abc

2.1.A رموز مُنهي بيان

بشكل عام، تنتهي تركيبات لغة TTCN-3 (أي، تعاريف وإعلانات وبيانات وعمليات) بفاصلة منقوطة (؛). والفاصلة المنقوطة خيارية إذا انتهى تركيب اللغة بـ (}) على الجانب الأيمن أو الرمز التالي بـ ;(}) على الجانب الأيمن، أي، يكون تركيب اللغة هو آخر بيان في فدرة بيانات وعمليات وإعلانات.

3.1.A مُعرفات

إن معرفات TTCN-3 حساسة للحروف الكبيرة والصغيرة وقد تحتوى فقط على حروف صغيرة (z-a) وحروف كبيرة (Z-A) وأرقام عددية (0-9). واستخدام رمز (_) مسموح به أيضاً. ويبدأ معرف بحرف (أي، ليس عدداً ولا رمز (_).

4.1.A تعليقات

يمكن أن تظهر تعليقات مكتوبة في نص حر في أي مكان من مواصفة TTCN-3.

وتكون تعليقات فدرة مفتوحة بواسطة الرمز */ وتغلق بواسطة الرمز /*.

المثـــال 1:

```
/* This is a block comment
spread over two lines */
```

تعليقات الفدرة يجب أن لا تكون متداخلة.

```
/* This is not /* a legal */ comment */
```

وتفتح تعليقات خط بواسطة الرمز */ وتغلق بواسطة /*.

المثـــال 2:

```
// This is a line comment
// spread over two lines
```

ويمكن أن تتبع تعليقات خط بيانات برنامج TTCN-3 ولكن لا تُدمج في بيان.

المثـــال 3:

```
// The following is not legal
const // This is MyConst integer MyConst := 1;

// The following is legal
const integer MyConst := 1; // This is MyConst
```

5.1.A مطاريف TTCN-3

ترد رموز مطراف TTCN-3 وكلمات محجوزة في الجدولين 2.A و3.A.

الجدول 140.Z/2.A – قائمة رموز مطراف خاص لـ TTCN-3

رموز فدرة بداية/نهاية	{ }
رموز قائمة بداية/نهاية	( )
رموز بديلة	[ ]
ترميز (في مدى)	..
تعليقات خط وتعليقات فدرة	// /* */
رمز منهي خط/بيان	;
رموز مشغل حسابي	+ / -
رمز مشغل تسلسل سلسلة	&
رموز مشغل تكافؤ	!= == >= <=
رموز احتواء سلسلة	" '
رموز سمة واحدة/مواءمة	? *
رمز تخصيص	:=
تخصيص عملية اتصالات	->
قيم سلسلة ثنائية وسلسلة ستة عشرية وسلسلة أثمونات	O H B
أُس طليق	E

تعامل معرفات وظيفة معرفة مسبقاً المعرفة في الجدول 10 والواردة في الملحق C باعتبارها كلمات محجوزة.

الجدول Z.140/3.A – قائمة مطاريف TTCN-3 التي هي كلمات محجوزة

action	fail	noblock	select
activate	false	none	self
address	float	not	send
alive	for	not4b	sender
all	from	nowait	set
alt	function	null	setverdict
altstep			signature
and	getverdict	octetstring	start
and4b	getcall	of	stop
any	getreply	omit	subset
anytype	goto	on	superset
	group	optional	system
bitstring		or	
boolean	hexstring	or4b	template
		out	testcase
case	if	override	timeout
call	ifpresent		timer
catch	import	param	to
char	in	pass	trigger
charstring	inconc	pattern	true
check	infinity	port	type
clear	inout	procedure	
complement	integer		union
component	interleave	raise	universal
connect		read	unmap
const	kill	receive	
control	killed	record	value
create			valueof
	label	rem	var
deactivate	language	repeat	variant
default	length	reply	verdicttype
disconnect	log	return	
display		running	while
do	map	runs	with
done	match		
	message		xor
else	mixed		xor4b
encode	mod		
enumerated	modifies		
error	module		
except	modulepar		
exception	mtc		
execute			
extends			
extension			
external			

إن مطاريف TTCN-3 الواردة في الجدول 3.A لا تستخدم كمعرفات في وحدة TTCN-3. وتكتب هذه المطاريف بالحروف الصغيرة.

6.1.A إنتاج TTCN-3 لقواعد تركيب BNF

0.6.1.A وحدة TTCN-3

```
1. TTCN3Module ::= TTCN3ModuleKeyword TTCN3ModuleId
                   "{"
                   [ModuleDefinitionsPart]
                   [ModuleControlPart]
                   "}"
                   [WithStatement] [SemiColon]
2. TTCN3ModuleKeyword ::= "module"
3. TTCN3ModuleId ::= ModuleId
4. ModuleId ::= GlobalModuleId [LanguageSpec]
/* STATIC SEMANTICS - LanguageSpec may only be omitted if the referenced module contains TTCN-3
notation */
5. GlobalModuleId ::= ModuleIdentifier
6. ModuleIdentifier ::= Identifier
7. LanguageSpec ::= LanguageKeyword FreeText
8. LanguageKeyword ::= "language"
```

1.6.1.A جزء تعاريف وحدة

0.1.6.1.A عـــام

```
9. ModuleDefinitionsPart ::= ModuleDefinitionsList
10. ModuleDefinitionsList ::= {ModuleDefinition [SemiColon]}+
11. ModuleDefinition ::= (TypeDef |
                          ConstDef |
                          TemplateDef |
                          ModuleParDef |
                          FunctionDef |
                          SignatureDef |
                          TestcaseDef |
                          AltstepDef |
                          ImportDef |
                          GroupDef |
                          ExtFunctionDef |
                          ExtConstDef) [WithStatement]
```

1.1.6.1.A تعاريف Typedef

```
12. TypeDef ::= TypeDefKeyword TypeDefBody
13. TypeDefBody ::= StructuredTypeDef | SubTypeDef
14. TypeDefKeyword ::= "type"
15. StructuredTypeDef ::= RecordDef |
                          UnionDef |
                          SetDef |
                          RecordOfDef |
                          SetOfDef |
                          EnumDef |
                          PortDef |
                          ComponentDef
16. RecordDef ::= RecordKeyword StructDefBody
17. RecordKeyword ::= "record"
18. StructDefBody ::= (StructTypeIdentifier [StructDefFormalParList] | AddressKeyword)
                      "{" [StructFieldDef {"," StructFieldDef}] "}"
19. StructTypeIdentifier ::= Identifier
20. StructDefFormalParList ::= "(" StructDefFormalPar {"," StructDefFormalPar} ")"
21. StructDefFormalPar ::=   FormalValuePar
/* STATIC SEMANTICS - FormalValuePar shall resolve to an in parameter */
22. StructFieldDef ::= (Type | NestedTypeDef) StructFieldIdentifier [ArrayDef] [SubTypeSpec]
                       [OptionalKeyword]
23. NestedTypeDef ::= NestedRecordDef |
                      NestedUnionDef |
                      NestedSetDef |
                      NestedRecordOfDef |
                      NestedSetOfDef |
                      NestedEnumDef
24. NestedRecordDef ::= RecordKeyword "{" [StructFieldDef {"," StructFieldDef}] "}"
25. NestedUnionDef ::= UnionKeyword "{" UnionFieldDef {"," UnionFieldDef} "}"
26. NestedSetDef ::= SetKeyword "{" [StructFieldDef {"," StructFieldDef}] "}"
27. NestedRecordOfDef ::= RecordKeyword [StringLength] OfKeyword (Type | NestedTypeDef)
28. NestedSetOfDef ::= SetKeyword [StringLength] OfKeyword (Type | NestedTypeDef)
29. NestedEnumDef ::= EnumKeyword "{" EnumerationList "}"
30. StructFieldIdentifier ::= Identifier
31. OptionalKeyword ::= "optional"
32. UnionDef ::= UnionKeyword UnionDefBody
33. UnionKeyword ::= "union"
34. UnionDefBody ::= (StructTypeIdentifier [StructDefFormalParList] | AddressKeyword)
```

```
                    "{" UnionFieldDef {"," UnionFieldDef} "}"
35. UnionFieldDef ::= (Type | NestedTypeDef) StructFieldIdentifier [ArrayDef] [SubTypeSpec]
36. SetDef ::= SetKeyword StructDefBody
37. SetKeyword ::= "set"
38. RecordOfDef ::= RecordKeyword [StringLength] OfKeyword StructOfDefBody
39. OfKeyword ::= "of"
40. StructOfDefBody ::= (Type | NestedTypeDef) (StructTypeIdentifier | AddressKeyword) [SubTypeSpec]
41. SetOfDef ::= SetKeyword [StringLength] OfKeyword StructOfDefBody
42. EnumDef ::= EnumKeyword (EnumTypeIdentifier | AddressKeyword)
                "{" EnumerationList "}"
43. EnumKeyword ::= "enumerated"
44. EnumTypeIdentifier ::= Identifier
45. EnumerationList ::= Enumeration {"," Enumeration}
46. Enumeration ::= EnumerationIdentifier ["("[Minus] Number ")"]
47. EnumerationIdentifier ::= Identifier
48. SubTypeDef ::= Type (SubTypeIdentifier | AddressKeyword)  [ArrayDef] [SubTypeSpec]
49. SubTypeIdentifier ::= Identifier
50. SubTypeSpec ::= AllowedValues [StringLength] | StringLength
/* STATIC SEMANTICS - AllowedValues shall be of the same type as the field being subtyped */
51. AllowedValues ::= "(" (ValueOrRange {"," ValueOrRange}) | CharStringMatch ")"
52. ValueOrRange ::= RangeDef | ConstantExpression
/* STATIC SEMANTICS - RangeDef production shall only be used with integer, charstring, universal
charstring or float based types */
/* STATIC SEMANTICS - When subtyping charstring or universal charstring range and values shall not
be mixed in the same SubTypeSpec */
53. RangeDef ::= LowerBound ".." UpperBound
54. StringLength ::= LengthKeyword "(" SingleConstExpression [".." UpperBound] ")"
/* STATIC SEMANTICS - StringLength shall only be used with String types or to limit set of and
record of. SingleConstExpression and UpperBound shall evaluate to non-negative integer values (in
case of UpperBound including infinity) */
55.   LengthKeyword ::= "length"
56. PortType ::= [GlobalModuleId Dot] PortTypeIdentifier
57. PortDef ::= PortKeyword PortDefBody
58. PortDefBody ::= PortTypeIdentifier PortDefAttribs
59. PortKeyword ::= "port"
60. PortTypeIdentifier ::= Identifier
61. PortDefAttribs ::= MessageAttribs | ProcedureAttribs | MixedAttribs
62. MessageAttribs ::= MessageKeyword
                       "{" {MessageList [SemiColon]}+ "}"
63. MessageList ::= Direction AllOrTypeList
64. Direction ::= InParKeyword | OutParKeyword | InOutParKeyword
65. MessageKeyword ::= "message"
66. AllOrTypeList ::= AllKeyword | TypeList
/* NOTE: The use of AllKeyword in port definitions is deprecated */
67. AllKeyword ::= "all"
68. TypeList ::= Type {"," Type}
69. ProcedureAttribs ::= ProcedureKeyword
                         "{" {ProcedureList  [SemiColon]}+ "}"
70. ProcedureKeyword ::= "procedure"
71. ProcedureList ::= Direction AllOrSignatureList
72. AllOrSignatureList ::= AllKeyword | SignatureList
73. SignatureList ::= Signature {"," Signature}
74. MixedAttribs ::= MixedKeyword
                     "{" {MixedList [SemiColon]}+ "}"
75. MixedKeyword ::= "mixed"
76. MixedList ::= Direction ProcOrTypeList
77. ProcOrTypeList ::= AllKeyword | (ProcOrType {"," ProcOrType})
78. ProcOrType ::= Signature | Type
79. ComponentDef ::= ComponentKeyword ComponentTypeIdentifier
                     [ExtendsKeyword ComponentType {"," ComponentType}]
                     "{" [ComponentDefList] "}"
80. ComponentKeyword ::= "component"
81. ExtendsKeyword ::= "extends"
82. ComponentType ::= [GlobalModuleId Dot] ComponentTypeIdentifier
83. ComponentTypeIdentifier ::= Identifier
84. ComponentDefList ::= {ComponentElementDef [SemiColon]}
85. ComponentElementDef ::= PortInstance | VarInstance | TimerInstance | ConstDef
86. PortInstance ::= PortKeyword PortType PortElement {"," PortElement}
87. PortElement ::= PortIdentifier [ArrayDef]
88. PortIdentifier ::= Identifier
```

2.1.6.1.A تعاريف ثابت

```
89. ConstDef ::= ConstKeyword Type ConstList
/* STATIC SEMANTICS - Type shall follow the rules given in clause 9.*/
90. ConstList ::= SingleConstDef  {"," SingleConstDef}
91. SingleConstDef ::= ConstIdentifier [ArrayDef] AssignmentChar ConstantExpression
/* STATIC SEMANTICS - The Value of the ConstantExpression shall be of the same type as the stated
type for the constants */
92. ConstKeyword ::= "const"
```

```
93. ConstIdentifier ::= Identifier
```

3.1.6.1.A تعاريف مقاس

```
94. TemplateDef ::= TemplateKeyword BaseTemplate  [DerivedDef] AssignmentChar TemplateBody
95. BaseTemplate ::= (Type | Signature) TemplateIdentifier  ["(" TemplateFormalParList ")"]
96. TemplateKeyword ::= "template"
97. TemplateIdentifier ::= Identifier
98. DerivedDef ::= ModifiesKeyword TemplateRef
99. ModifiesKeyword ::= "modifies"
100. TemplateFormalParList ::= TemplateFormalPar {"," TemplateFormalPar}
101. TemplateFormalPar ::= FormalValuePar | FormalTemplatePar
/* STATIC SEMANTICS - FormalValuePar shall resolve to an in parameter */
102. TemplateBody ::= (SimpleSpec | FieldSpecList | ArrayValueOrAttrib) | [ExtraMatchingAttributes]
/* STATIC SEMANTICS - Within TeplateBody the ArrayValueOrAttrib can be used for array, record,
record of and set of types. */
103. SimpleSpec ::= SingleValueOrAttrib
104. FieldSpecList ::= "{"[FieldSpec {"," FieldSpec}] "}"
105. FieldSpec ::= FieldReference AssignmentChar TemplateBody
106. FieldReference ::= StructFieldRef | ArrayOrBitRef | ParRef
/* STATIC SEMANTICS - Within FieldReference ArrayOrBitRef can be used for record of and set of
templates/template fields in modified templates only*/
107. StructFieldRef ::= StructFieldIdentifier| PredefinedType | TypeReference
/* STATIC SEMANTICS - PredefinedType and TypeReference shall be used for anytype value notation
only. PredefinedType shall not be AnyTypeKeyword.*/

108. ParRef ::= SignatureParIdentifier
/* STATIC SEMANTICS - SignatureParIdentifier shall be a formal parameter Identifier from the
associated signature definition */
109. SignatureParIdentifier ::= ValueParIdentifier
110. ArrayOrBitRef ::= "[" FieldOrBitNumber "]"
/* STATIC SEMANTICS - ArrayRef shall be optionally used for array types and ASN.1 SET OF and
SEQUENCE OF and TTCN-3 record of and set of. The same notation can be used for a Bit reference
inside an ASN.1 or TTCN-3 bitstring type */
111. FieldOrBitNumber ::= SingleExpression
/* STATIC SEMANTICS - SingleExpression will resolve to a value of integer type */
112. SingleValueOrAttrib ::= MatchingSymbol |
                             SingleExpression |
                             TemplateRefWithParList
/* STATIC SEMANTIC - VariableIdentifier (accessed via singleExpression) may only be used in in-line
template definitions to reference variables in the current scope */
113. ArrayValueOrAttrib ::= "{" ArrayElementSpecList "}"
114. ArrayElementSpecList ::= ArrayElementSpec {"," ArrayElementSpec}
115. ArrayElementSpec ::= NotUsedSymbol | PermutationMatch | TemplateBody
116. NotUsedSymbol ::= Dash
117. MatchingSymbol ::= Complement |
                        AnyValue |
                        AnyOrOmit |
                        ValueOrAttribList |
                        Range |
                        BitStringMatch |
                        HexStringMatch |
                        OctetStringMatch |
                        CharStringMatch |
                        SubsetMatch |
                        SupersetMatch
118. ExtraMatchingAttributes ::= LengthMatch | IfPresentMatch | LengthMatch  IfPresentMatch
119. BitStringMatch ::= "'" {BinOrMatch} "'" "B"
120. BinOrMatch ::= Bin | AnyValue | AnyOrOmit
121. HexStringMatch ::= "'" {HexOrMatch} "'" "H"
122. HexOrMatch ::= Hex | AnyValue | AnyOrOmit
123. OctetStringMatch ::= "'" {OctOrMatch} "'" "O"
124. OctOrMatch ::= Oct | AnyValue | AnyOrOmit
125. CharStringMatch ::= PatternKeyword Cstring
126. PatternKeyword ::= "pattern"
127. Complement ::= ComplementKeyword ValueList
128. ComplementKeyword ::= "complement"
129. ValueList ::= "(" ConstantExpression {"," ConstantExpression} ")"
130. SubsetMatch ::= SubsetKeyword ValueList
/* STATIC SEMANTIC - Subset matching shall only be used with the set of type */
131. SubsetKeyword ::= "subset"
132. SupersetMatch ::= SupersetKeyword ValueList
/* STATIC SEMANTIC - Superset matching shall only be used with the set of type */
133. SupersetKeyword ::= "superset"
134. PermutationMatch ::= PermutationKeyword PermutationList
135. PermutationKeyword ::= "permutation"
136. PermutationList ::= "(" TemplateBody { "," TemplateBody } ")"
/* STATIC SEMANTICS - Restrictions on the content of TemplateBody are given in clause B.1.3.3 */
137. AnyValue ::= "?"
138. AnyOrOmit ::= "*"
```

```
139. ValueOrAttribList ::= "(" TemplateBody {"," TemplateBody}+ ")"
140. LengthMatch ::= StringLength
141. IfPresentMatch ::= IfPresentKeyword
142. IfPresentKeyword ::= "ifpresent"
143. Range ::= "(" LowerBound ".." UpperBound ")"
144. LowerBound ::= SingleConstExpression | Minus InfinityKeyword
145. UpperBound ::= SingleConstExpression | InfinityKeyword
/* STATIC SEMANTICS - LowerBound and UpperBound shall evaluate to types integer, charstring,
universal charstring or float. In case LowerBound or UpperBound evaluates to types charstring or
universal charstring, only SingleConstExpression may be present and the string length shall be 1*/
146. InfinityKeyword ::= "infinity"
147. TemplateInstance ::= InLineTemplate
148. TemplateRefWithParList ::= [GlobalModuleId Dot] TemplateIdentifier [TemplateActualParList] |
                                TemplateParIdentifier
149. TemplateRef ::= [GlobalModuleId Dot] TemplateIdentifier | TemplateParIdentifier
150. InLineTemplate ::= [(Type | Signature) Colon] [DerivedRefWithParList AssignmentChar]
                        TemplateBody
/* STATIC SEMANTICS - The type field may only be omitted when the type is implicitly unambiguous */
151. DerivedRefWithParList ::= ModifiesKeyword TemplateRefWithParList
152. TemplateActualParList ::= "(" TemplateActualPar {"," TemplateActualPar} ")"
153. TemplateActualPar ::= TemplateInstance
/* STATIC SEMANTICS - When the corresponding formal parameter is not of template type the
TemplateInstance production shall resolve to one or more SingleExpressions */
154. TemplateOps ::= MatchOp | ValueofOp
155. MatchOp ::= MatchKeyword "(" Expression "," TemplateInstance")"
/* STATIC SEMANTICS - The type of the value returned by the expression must be the same as the
template type and each field of the template shall resolve to a single value */
156. MatchKeyword ::= "match"
157. ValueofOp ::= ValueofKeyword "(" TemplateInstance")"
158. ValueofKeyword ::= "valueof"
```

4.1.6.1.A تعاريف وظيفة

```
159. FunctionDef ::= FunctionKeyword FunctionIdentifier
                     "("[FunctionFormalParList] ")" [RunsOnSpec] [ReturnType]
                     StatementBlock
160. FunctionKeyword ::= "function"
161. FunctionIdentifier ::= Identifier
162. FunctionFormalParList ::= FunctionFormalPar {"," FunctionFormalPar}
163. FunctionFormalPar ::= FormalValuePar |
                           FormalTimerPar |
                           FormalTemplatePar |
                           FormalPortPar
164. ReturnType ::= ReturnKeyword [TemplateKeyword] Type
/* STATIC SEMANTICS - The use of the template keyword shall conform to restrictions in
clause 16.1.0 */
165. ReturnKeyword ::= "return"
166. RunsOnSpec ::= RunsKeyword OnKeyword ComponentType
167. RunsKeyword ::= "runs"
168. OnKeyword ::= "on"
169. MTCKeyword ::= "mtc"
170. StatementBlock ::= "{" [FunctionStatementOrDefList] "}"
171. FunctionStatementOrDefList ::= {FunctionStatementOrDef [SemiColon]}+
172. FunctionStatementOrDef ::= FunctionLocalDef |
                                FunctionLocalInst |
                                FunctionStatement
173. FunctionLocalInst ::= VarInstance | TimerInstance
174. FunctionLocalDef ::= ConstDef | TemplateDef
175. FunctionStatement ::= ConfigurationStatements |
                           TimerStatements |
                           CommunicationStatements |
                           BasicStatements |
                           BehaviourStatements |
                           VerdictStatements |
                           SUTStatements
176. FunctionInstance ::= FunctionRef "(" [FunctionActualParList] ")"
177. FunctionRef ::= [GlobalModuleId Dot] (FunctionIdentifier | ExtFunctionIdentifier ) |
                     PreDefFunctionIdentifier
178. PreDefFunctionIdentifier ::= Identifier
/* STATIC SEMANTICS - The Identifier will be one of the pre-defined TTCN-3 Function Identifiers from
Annex C */
179. FunctionActualParList ::= FunctionActualPar {"," FunctionActualPar}
180. FunctionActualPar ::= TimerRef |
                           TemplateInstance |
                           Port |
                           ComponentRef
/* STATIC SEMANTICS - When the corresponding formal parameter is not of template type the
TemplateInstance production shall resolve to one or more SingleExpressions i.e. equivalent to the
Expression production */
```

5.1.6.1.A تعاريف توقيع

```
181. SignatureDef ::= SignatureKeyword SignatureIdentifier
                      "("[SignatureFormalParList] ")" [ReturnType | NoBlockKeyword]
                    [ExceptionSpec]
182. SignatureKeyword ::= "signature"
183. SignatureIdentifier ::= Identifier
184. SignatureFormalParList ::= SignatureFormalPar {"," SignatureFormalPar}
185. SignatureFormalPar ::=  FormalValuePar
186. ExceptionSpec ::= ExceptionKeyword "(" ExceptionTypeList ")"
187. ExceptionKeyword ::= "exception"
188. ExceptionTypeList ::= Type {"," Type}
189. NoBlockKeyword ::= "noblock"
190. Signature ::= [GlobalModuleId Dot] SignatureIdentifier
```

6.1.6.1.A تعاريف Testcase

```
191. TestcaseDef ::= TestcaseKeyword TestcaseIdentifier
                     "("[TestcaseFormalParList] ")" ConfigSpec
                     StatementBlock
192. TestcaseKeyword ::= "testcase"
193. TestcaseIdentifier ::= Identifier
194. TestcaseFormalParList ::= TestcaseFormalPar {"," TestcaseFormalPar}
195. TestcaseFormalPar ::= FormalValuePar |
                           FormalTemplatePar
196. ConfigSpec ::= RunsOnSpec [SystemSpec]
197. SystemSpec ::= SystemKeyword ComponentType
198. SystemKeyword ::= "system"
199. TestcaseInstance ::= ExecuteKeyword "(" TestcaseRef "(" [TestcaseActualParList] ")"
                          ["," TimerValue] ")"
200. ExecuteKeyword ::= "execute"
201. TestcaseRef ::= [GlobalModuleId Dot] TestcaseIdentifier
202. TestcaseActualParList ::= TestcaseActualPar {"," TestcaseActualPar}
203. TestcaseActualPar ::= TemplateInstance
/* STATIC SEMANTICS - When the corresponding formal parameter is not of template type the
TemplateInstance production shall resolve to one or more SingleExpressions i.e. equivalent to the
Expression production */
```

7.1.6.1.A تعاريف Altstep

```
204. AltstepDef ::= AltstepKeyword AltstepIdentifier
                    "("[AltstepFormalParList] ")" [RunsOnSpec]
                    "{" AltstepLocalDefList AltGuardList "}"
205. AltstepKeyword ::= "altstep"
206. AltstepIdentifier ::= Identifier
207. AltstepFormalParList ::= FunctionFormalParList
/* STATIC SEMANTICS - altsteps that are activated as defaults shall only have in parameters, port
parameters, or timer parameters */
/* STATIC SEMANTICS - altsteps that are only invoked as an alternative in an alt statement or as
stand-alone statement in a TTCN-3 behaviour description may have in, out and inout parameters. */
208. AltstepLocalDefList ::= {AltstepLocalDef [SemiColon]}
209. AltstepLocalDef ::= VarInstance | TimerInstance | ConstDef | TemplateDef
/*STATIC SEMANTICS - AltstepLocalDef shall conform to restrictions in clause 16.2.2.1 */
210. AltstepInstance ::= AltstepRef "(" [FunctionActualParList] ")"
/* STATIC SEMANTICS - all timer instances in FunctionActualParList shall be declared as component
local timers (see also production ComponentElementDef) */
211. AltstepRef ::= [GlobalModuleId Dot] AltstepIdentifier
```

8.1.6.1.A تعاريف استيراد

```
212. ImportDef ::= ImportKeyword ImportFromSpec (AllWithExcepts | ("{" ImportSpec "}"))
213. ImportKeyword ::= "import"
214. AllWithExcepts ::= AllKeyword [ExceptsDef]
215. ExceptsDef ::= ExceptKeyword "{" ExceptSpec "}"
216. ExceptKeyword ::= "except"
217. ExceptSpec ::= {ExceptElement [SemiColon]}
/* STATIC SEMANTICS - Any of the production components (ExceptGroupSpec, ExceptTypeDefSpec etc.) may
be present only once in the ExceptSpec production */
218. ExceptElement ::= ExceptGroupSpec |
                      ExceptTypeDefSpec |
                      ExceptTemplateSpec |
                      ExceptConstSpec |
                      ExceptTestcaseSpec |
                      ExceptAltstepSpec |
                      ExceptFunctionSpec |
                      ExceptSignatureSpec |
                      ExceptModuleParSpec
219. ExceptGroupSpec ::= GroupKeyword (ExceptGroupRefList | AllKeyword)
220. ExceptTypeDefSpec ::= TypeDefKeyword (TypeRefList | AllKeyword)
```

```
221. ExceptTemplateSpec ::= TemplateKeyword (TemplateRefList | AllKeyword)
222. ExceptConstSpec ::= ConstKeyword (ConstRefList | AllKeyword)
223. ExceptTestcaseSpec ::= TestcaseKeyword (TestcaseRefList | AllKeyword)
224. ExceptAltstepSpec ::= AltstepKeyword (AltstepRefList | AllKeyword)
225. ExceptFunctionSpec ::= FunctionKeyword (FunctionRefList | AllKeyword)
226. ExceptSignatureSpec ::= SignatureKeyword  (SignatureRefList | AllKeyword)
227. ExceptModuleParSpec ::= ModuleParKeyword  (ModuleParRefList | AllKeyword)
228. ImportSpec ::= {ImportElement [SemiColon]}
229. ImportElement ::= ImportGroupSpec |
                       ImportTypeDefSpec |
                       ImportTemplateSpec |
                       ImportConstSpec |
                       ImportTestcaseSpec |
                       ImportAltstepSpec |
                       ImportFunctionSpec |
                       ImportSignatureSpec |
                       ImportModuleParSpec
230. ImportFromSpec ::= FromKeyword ModuleId [RecursiveKeyword]
/* NOTE - The use of RecursiveKeyword is deprecated*/
231. RecursiveKeyword ::= "recursive"
232. ImportGroupSpec ::= GroupKeyword (GroupRefListWithExcept | AllGroupsWithExcept)
233. GroupRefList ::= FullGroupIdentifier {"," FullGroupIdentifier}
234. GroupRefListWithExcept ::= FullGroupIdentifierWithExcept {"," FullGroupIdentifierWithExcept}
235. AllGroupsWithExcept ::= AllKeyword [ExceptKeyword GroupRefList]
236. FullGroupIdentifier ::= GroupIdentifier {Dot GroupIdentifier}
237. FullGroupIdentifierWithExcept ::= FullGroupIdentifier [ExceptsDef]
238. ExceptGroupRefList ::= ExceptFullGroupIdentifier {"," ExceptFullGroupIdentifier}
239. ExceptFullGroupIdentifier ::= FullGroupIdentifier
240. ImportTypeDefSpec ::= TypeDefKeyword (TypeRefList | AllTypesWithExcept)
241. TypeRefList ::= TypeDefIdentifier {"," TypeDefIdentifier}
242. AllTypesWithExcept ::= AllKeyword [ExceptKeyword TypeRefList]
243. TypeDefIdentifier ::= StructTypeIdentifier |
                          EnumTypeIdentifier |
                          PortTypeIdentifier |
                          ComponentTypeIdentifier |
                          SubTypeIdentifier
244. ImportTemplateSpec ::= TemplateKeyword (TemplateRefList | AllTemplsWithExcept)
245. TemplateRefList ::= TemplateIdentifier {"," TemplateIdentifier}
246. AllTemplsWithExcept ::= AllKeyword [ExceptKeyword TemplateRefList]
247. ImportConstSpec ::= ConstKeyword (ConstRefList | AllConstsWithExcept)
248. ConstRefList ::= ConstIdentifier {"," ConstIdentifier}
249. AllConstsWithExcept ::= AllKeyword [ExceptKeyword ConstRefList]
250. ImportAltstepSpec ::= AltstepKeyword (AltstepRefList | AllAltstepsWithExcept)
251. AltstepRefList ::= AltstepIdentifier {"," AltstepIdentifier}
252. AllAltstepsWithExcept ::= AllKeyword [ExceptKeyword AltstepRefList]
253. ImportTestcaseSpec ::= TestcaseKeyword (TestcaseRefList | AllTestcasesWithExcept)
254. TestcaseRefList ::= TestcaseIdentifier {"," TestcaseIdentifier}
255. AllTestcasesWithExcept ::= AllKeyword [ExceptKeyword TestcaseRefList]
256. ImportFunctionSpec ::= FunctionKeyword (FunctionRefList | AllFunctionsWithExcept)
257. FunctionRefList ::= FunctionIdentifier {"," FunctionIdentifier}
258. AllFunctionsWithExcept ::= AllKeyword [ExceptKeyword FunctionRefList]
259. ImportSignatureSpec ::= SignatureKeyword  (SignatureRefList | AllSignaturesWithExcept)
260. SignatureRefList ::= SignatureIdentifier {"," SignatureIdentifier}
261. AllSignaturesWithExcept ::= AllKeyword [ExceptKeyword SignatureRefList]
262. ImportModuleParSpec ::= ModuleParKeyword  (ModuleParRefList | AllModuleParWithExcept)
263. ModuleParRefList ::= ModuleParIdentifier {"," ModuleParIdentifier}
264. AllModuleParWithExcept ::= AllKeyword [ExceptKeyword ModuleParRefList]
```

9.1.6.1.A تعاريف زمرة

```
265. GroupDef ::= GroupKeyword GroupIdentifier
                  "{" [ModuleDefinitionsPart] "}"
266. GroupKeyword ::= "group"
267. GroupIdentifier ::= Identifier
```

10.1.6.1.A تعاريف وظيفة خارجية

```
268. ExtFunctionDef ::= ExtKeyword FunctionKeyword ExtFunctionIdentifier
                        "("[FunctionFormalParList] ")" [ReturnType]
269. ExtKeyword ::= "external"
270. ExtFunctionIdentifier ::= Identifier
```

11.1.6.1.A تعاريف ثابت خارجي

```
271. ExtConstDef ::= ExtKeyword ConstKeyword Type ExtConstIdentifier
/* STATIC SEMANTICS - Type shall follow the rules given in clause 9.*/
272. ExtConstIdentifier ::= Identifier
```

12.1.6.1.A تعاريف معلمة وحدة

```
273. ModuleParDef ::= ModuleParKeyword ( ModulePar | ("{" MultitypedModuleParList "}"))
274. ModuleParKeyword ::= "modulepar"
275. MultitypedModuleParList ::= { ModulePar SemiColon }+
276. ModulePar ::= ModuleParType ModuleParList
/* STATIC SEMANTICS - The Value of the ConstantExpression shall be of the same type as the stated
type for the Parameter */
277. ModuleParType ::= Type
/* STATIC SEMANTICS - Type shall not be of component, default or anytype. Type shall only resolve to
address type if a definition for the address type is defined within the module */
278. ModuleParList ::= ModuleParIdentifier [AssignmentChar ConstantExpression]
                      {","ModuleParIdentifier [AssignmentChar ConstantExpression]}
279. ModuleParIdentifier ::= Identifier
```

2.6.1.A جزء التحكم

0.2.6.1.A عـــام

```
280. ModuleControlPart ::= ControlKeyword
                           "{" ModuleControlBody "}"
                           [WithStatement] [SemiColon]
281. ControlKeyword ::= "control"
282. ModuleControlBody ::= [ControlStatementOrDefList]
283. ControlStatementOrDefList ::= {ControlStatementOrDef [SemiColon]}+
284. ControlStatementOrDef ::= FunctionLocalDef |
                               FunctionLocalInst |
                               ControlStatement
285. ControlStatement ::= TimerStatements |
                          BasicStatements |
                          BehaviourStatements |
                          SUTStatements |
                          StopKeyword
```

/* STATIC SEMANTICS - Restrictions on use of statements in the control part are given in Table 11 */

1.2.6.1.A استطباق متغيّر

```
286. VarInstance ::= VarKeyword ((Type VarList) | (TemplateKeyword Type TempVarList))
287. VarList ::= SingleVarInstance {"," SingleVarInstance}
288. SingleVarInstance ::= VarIdentifier [ArrayDef] [AssignmentChar VarInitialValue]
289. VarInitialValue ::= Expression
290. VarKeyword ::= "var"
291. VarIdentifier ::= Identifier
292. TempVarList ::= SingleTempVarInstance {"," SingleTempVarInstance}
293. SingleTempVarInstance ::= VarIdentifier [ArrayDef] [AssignmentChar TempVarInitialValue]
294. TempVarInitialValue ::= TemplateBody
295. VariableRef ::= (VarIdentifier | ValueParIdentifier) [ExtendedFieldReference]
```

2.2.6.1.A استطباق مؤقت

```
296. TimerInstance ::= TimerKeyword TimerList
297. TimerList ::= SingleTimerInstance{"," SingleTimerInstance}
298. SingleTimerInstance ::= TimerIdentifier [ArrayDef] [AssignmentChar TimerValue]
299. TimerKeyword ::= "timer"
300. TimerIdentifier ::= Identifier
301. TimerValue ::= Expression
/* STATIC SEMANTICS - When Expression resolves to SingleExpression it must resolve to a value of
type float. Expression shall only resolve to CompoundExpression in the initialization in default
timer value assignment for timer arrays */
302. TimerRef ::= (TimerIdentifier | TimerParIdentifier) {ArrayOrBitRef}
```

3.2.6.1.A عمليات مكون

```
303. ConfigurationStatements ::= ConnectStatement |
                                 MapStatement |
                                 DisconnectStatement |
                                 UnmapStatement |
                                 DoneStatement |
                                 KilledStatement |
                                 StartTCStatement |
                                 StopTCStatement |
                                 KillTCStatement
304. ConfigurationOps ::= CreateOp | SelfOp | SystemOp | MTCOp | RunningOp | AliveOp
305. CreateOp ::= ComponentType Dot CreateKeyword ["(" SingleExpression ")"] [AliveKeyword]
/* STATIC SEMANTICS - Restrictions on SingleExpression see in clause 22.1 */
306. SystemOp ::= SystemKeyword
307. SelfOp ::= "self"
308. MTCOp ::= MTCKeyword
309. DoneStatement ::= ComponentId Dot DoneKeyword
310. KilledStatement ::= ComponentId Dot KilledKeyword
```

```
311. ComponentId ::= ComponentOrDefaultReference | (AnyKeyword | AllKeyword) ComponentKeyword
312. DoneKeyword ::= "done"
313. KilledKeyword ::= "killed"
314. RunningOp ::= ComponentId Dot RunningKeyword
315. RunningKeyword ::= "running"
316. AliveOp ::= ComponentId Dot AliveKeyword
317. CreateKeyword ::= "create"
318. AliveKeyword ::= "alive"
319. ConnectStatement ::= ConnectKeyword SingleConnectionSpec
320. ConnectKeyword ::= "connect"
321. SingleConnectionSpec ::= "(" PortRef "," PortRef ")"
322. PortRef ::= ComponentRef Colon Port
323. ComponentRef ::= ComponentOrDefaultReference | SystemOp | SelfOp | MTCOp
324. DisconnectStatement ::= DisconnectKeyword [SingleOrMultiConnectionSpec]
325. SingleOrMultiConnectionSpec ::= SingleConnectionSpec |
                                    AllConnectionsSpec |
                                    AllPortsSpec |
                                    AllCompsAllPortsSpec]
326. AllConnectionsSpec ::= "(" PortRef ")"
327. AllPortsSpec ::= "(" ComponentRef ":" AllKeyword PortKeyword ")"
328. AllCompsAllPortsSpec ::= "(" AllKeyword ComponentKeyword ":" AllKeyword PortKeyword ")"
329. DisconnectKeyword ::= "disconnect"
330. MapStatement ::= MapKeyword SingleConnectionSpec
331. MapKeyword ::= "map"
332. UnmapStatement ::= UnmapKeyword [SingleOrMultiConnectionSpec]
333. UnmapKeyword ::= "unmap"
334. StartTCStatement ::= ComponentOrDefaultReference Dot StartKeyword "(" FunctionInstance ")"
/* STATIC SEMANTICS - the Function instance may only have in parameters */
/* STATIC SEMANTICS - the Function instance shall not have timer parameters */
335. StartKeyword ::= "start"
336. StopTCStatement ::= StopKeyword | (ComponentReferenceOrLiteral Dot StopKeyword) |
                        (AllKeyword ComponentKeyword Dot StopKeyword)
337. ComponentReferenceOrLiteral ::= ComponentOrDefaultReference | MTCOp | SelfOp
338. KillTCStatement ::= KillKeyword | (ComponentIdentifierOrLiteral Dot KillKeyword) |
                         (AllKeyword ComponentKeyword Dot KillKeyword)
339. ComponentOrDefaultReference ::= VariableRef | FunctionInstance
/* STATIC SEMANTICS - The variable associated with VariableRef or the return type associated with
FunctionInstance must be of component type when used in configuration statements and shall be of
default type when used in the deactivate statement. */
340. KillKeyword ::= "kill"
```

4.2.6.1.A عمليات منْفَذ

```
341. Port ::= (PortIdentifier | PortParIdentifier) {ArrayOrBitRef}
342. CommunicationStatements ::= SendStatement |
                                 CallStatement |
                                 ReplyStatement |
                                 RaiseStatement |
                                 ReceiveStatement |
                                 TriggerStatement |
                                 GetCallStatement |
                                 GetReplyStatement |
                                 CatchStatement |
                                 CheckStatement |
                                 ClearStatement |
                                 StartStatement |
                                 StopStatement
343. SendStatement ::= Port Dot PortSendOp
344. PortSendOp ::= SendOpKeyword "(" SendParameter ")" [ToClause]
345. SendOpKeyword ::= "send"
346. SendParameter ::= TemplateInstance
347. ToClause ::= ToKeyword AddressRef |
                  AddressRefList |
                  AllKeyword ComponentKeyword
/* STATIC SEMANTICS - AddressRef should not contain matching mechanisms */
348. AddressRefList ::= "(" AddressRef {"," AddressRef} ")"
349. ToKeyword ::= "to"
350. AddressRef ::= TemplateInstance
/* STATIC SEMANTICS - TemplateInstance must be of address or component type */
351. CallStatement ::= Port Dot PortCallOp  [PortCallBody]
352. PortCallOp ::= CallOpKeyword "(" CallParameters ")" [ToClause]
353. CallOpKeyword ::= "call"
354. CallParameters ::= TemplateInstance ["," CallTimerValue]
/* STATIC SEMANTICS - only out parameters may be omitted or specified with a matching attribute */
355. CallTimerValue ::= TimerValue | NowaitKeyword
/* STATIC SEMANTICS - Value must be of type float */
356. NowaitKeyword ::= "nowait"
357. PortCallBody ::= "{" CallBodyStatementList "}"
358. CallBodyStatementList ::= {CallBodyStatement [SemiColon]}+
359. CallBodyStatement ::= CallBodyGuard  StatementBlock
```

```
360. CallBodyGuard ::= AltGuardChar CallBodyOps
361. CallBodyOps ::= GetReplyStatement | CatchStatement
362. ReplyStatement ::= Port Dot PortReplyOp
363. PortReplyOp ::= ReplyKeyword "(" TemplateInstance [ReplyValue]")" [ToClause]
364. ReplyKeyword ::= "reply"
365. ReplyValue ::= ValueKeyword Expression
366. RaiseStatement ::= Port Dot PortRaiseOp
367. PortRaiseOp ::= RaiseKeyword "(" Signature "," TemplateInstance ")" [ToClause]
368. RaiseKeyword ::= "raise"
369. ReceiveStatement ::= PortOrAny Dot PortReceiveOp
370. PortOrAny ::= Port | AnyKeyword PortKeyword
371. PortReceiveOp ::= ReceiveOpKeyword ["(" ReceiveParameter ")"] [FromClause] [PortRedirect]
/* STATIC SEMANTICS - the PortRedirect option may only be present if the ReceiveParameter option is
also present */
372. ReceiveOpKeyword ::= "receive"
373. ReceiveParameter ::= TemplateInstance
374. FromClause ::= FromKeyword AddressRef
375. FromKeyword ::= "from"
376. PortRedirect ::= PortRedirectSymbol (ValueSpec [SenderSpec] | SenderSpec)
377. PortRedirectSymbol ::= "->"
378. ValueSpec ::= ValueKeyword VariableRef
379. ValueKeyword ::= "value"
380. SenderSpec ::= SenderKeyword VariableRef
/* STATIC SEMANTICS - Variable ref must be of address or component type */
381. SenderKeyword ::= "sender"
382. TriggerStatement ::= PortOrAny Dot PortTriggerOp
383. PortTriggerOp ::= TriggerOpKeyword ["(" ReceiveParameter ")"] [FromClause] [PortRedirect]
/* STATIC SEMANTICS - the PortRedirect option may only be present if the ReceiveParameter option is
also present */
384. TriggerOpKeyword ::= "trigger"
385. GetCallStatement ::= PortOrAny  Dot PortGetCallOp
386. PortGetCallOp ::= GetCallOpKeyword ["(" ReceiveParameter ")"] [FromClause]
                       [PortRedirectWithParam]
/* STATIC SEMANTICS - the PortRedirectWithParam option may only be present if the ReceiveParameter
option is also present */
387. GetCallOpKeyword ::= "getcall"
388. PortRedirectWithParam ::= PortRedirectSymbol RedirectWithParamSpec
389. RedirectWithParamSpec ::= ParamSpec [SenderSpec] |
                               SenderSpec
390. ParamSpec ::= ParamKeyword ParamAssignmentList
391. ParamKeyword ::= "param"
392. ParamAssignmentList ::= "(" (AssignmentList | VariableList) ")"
393. AssignmentList ::= VariableAssignment {"," VariableAssignment}
394. VariableAssignment ::= VariableRef AssignmentChar ParameterIdentifier
/* STATIC SEMANTICS - the parameterIdentifiers must be from the corresponding signature
definition */
395. ParameterIdentifier ::= ValueParIdentifier
396. VariableList ::= VariableEntry  {"," VariableEntry}
397. VariableEntry ::= VariableRef | NotUsedSymbol
398. GetReplyStatement ::= PortOrAny Dot PortGetReplyOp
399. PortGetReplyOp ::= GetReplyOpKeyword ["(" ReceiveParameter [ValueMatchSpec] ")"]
                        [FromClause] [PortRedirectWithValueAndParam]
/* STATIC SEMANTICS - the PortRedirectWithParam option may only be present if the ReceiveParameter
option is also present */
400. PortRedirectWithValueAndParam ::= PortRedirectSymbol RedirectWithValueAndParamSpec
401. RedirectWithValueAndParamSpec ::= ValueSpec [ParamSpec] [SenderSpec] |
                                       RedirectWithParamSpec
402. GetReplyOpKeyword ::= "getreply"
403. ValueMatchSpec ::= ValueKeyword TemplateInstance
404. CheckStatement ::= PortOrAny Dot PortCheckOp
405. PortCheckOp ::= CheckOpKeyword ["(" CheckParameter ")"]
406. CheckOpKeyword ::= "check"
407. CheckParameter ::= CheckPortOpsPresent | FromClausePresent | RedirectPresent
408. FromClausePresent ::= FromClause [PortRedirectSymbol SenderSpec]
409. RedirectPresent ::= PortRedirectSymbol SenderSpec
410. CheckPortOpsPresent ::= PortReceiveOp | PortGetCallOp | PortGetReplyOp | PortCatchOp
411. CatchStatement ::= PortOrAny Dot PortCatchOp
412. PortCatchOp ::= CatchOpKeyword ["("CatchOpParameter ")"] [FromClause] [PortRedirect]
/* STATIC SEMANTICS - the PortRedirect option may only be present if the CatchOpParameter option is
also present */
413. CatchOpKeyword ::= "catch"
414. CatchOpParameter ::= Signature "," TemplateInstance | TimeoutKeyword
415. ClearStatement ::= PortOrAll Dot PortClearOp
416. PortOrAll ::= Port | AllKeyword PortKeyword
417. PortClearOp ::= ClearOpKeyword
418. ClearOpKeyword ::= "clear"
419. StartStatement ::= PortOrAll Dot PortStartOp
420. PortStartOp ::= StartKeyword
421. StopStatement ::= PortOrAll Dot PortStopOp
422. PortStopOp ::= StopKeyword
```

```
423. StopKeyword ::= "stop"
424. AnyKeyword ::= "any"
```

5.2.6.1.A عمليات مؤقت

```
425. TimerStatements ::= StartTimerStatement | StopTimerStatement | TimeoutStatement
426. TimerOps ::= ReadTimerOp | RunningTimerOp
427. StartTimerStatement ::= TimerRef Dot StartKeyword ["(" TimerValue ")"]
428. StopTimerStatement ::= TimerRefOrAll Dot StopKeyword
429. TimerRefOrAll ::= TimerRef | AllKeyword TimerKeyword
430. ReadTimerOp ::= TimerRef Dot ReadKeyword
431. ReadKeyword ::= "read"
432. RunningTimerOp ::= TimerRefOrAny  Dot RunningKeyword
433. TimeoutStatement ::= TimerRefOrAny  Dot TimeoutKeyword
434. TimerRefOrAny ::= TimerRef | AnyKeyword TimerKeyword
435. TimeoutKeyword ::= "timeout"
```

3.6.1.A نمـــط

```
436. Type ::= PredefinedType | ReferencedType
437. PredefinedType ::= BitStringKeyword |
                        BooleanKeyword |
                        CharStringKeyword |
                        UniversalCharString |
                        IntegerKeyword |
                        OctetStringKeyword |
                        HexStringKeyword |
                        VerdictTypeKeyword |
                        FloatKeyword |
                        AddressKeyword |
                        DefaultKeyword |
                        AnyTypeKeyword
438. BitStringKeyword ::= "bitstring"
439. BooleanKeyword ::= "boolean"
440. IntegerKeyword ::= "integer"
441. OctetStringKeyword ::= "octetstring"
442. HexStringKeyword ::= "hexstring"
443. VerdictTypeKeyword ::= "verdicttype"
444. FloatKeyword ::= "float"
445. AddressKeyword ::= "address"
446. DefaultKeyword ::= "default"
447. AnyTypeKeyword ::= "anytype"
448. CharStringKeyword ::= "charstring"
449. UniversalCharString ::= UniversalKeyword CharStringKeyword
450. UniversalKeyword ::= "universal"
451. ReferencedType ::= [GlobalModuleId Dot] TypeReference [ExtendedFieldReference]
452. TypeReference ::= StructTypeIdentifier[TypeActualParList] |
                       EnumTypeIdentifier |
                       SubTypeIdentifier |
                       ComponentTypeIdentifier
453. TypeActualParList ::= "(" TypeActualPar {"," TypeActualPar} ")"
454. TypeActualPar ::= ConstantExpression
455. ArrayDef ::= {"[" ArrayBounds [".." ArrayBounds] "]"}+
456. ArrayBounds ::= SingleConstExpression
/* STATIC SEMANTICS - ArrayBounds will resolve to a non-negative value of integer type */
```

4.6.1.A قيمة

```
457. Value ::= PredefinedValue | ReferencedValue
458. PredefinedValue ::= BitStringValue |
                         BooleanValue |
                         CharStringValue |
                         IntegerValue |
                         OctetStringValue |
                         HexStringValue |
                         VerdictTypeValue |
                         EnumeratedValue |
                         FloatValue |
                         AddressValue |
                         OmitValue
459. BitStringValue ::= Bstring
460. BooleanValue ::= "true" | "false"
461. IntegerValue ::= Number
462. OctetStringValue ::= Ostring
463. HexStringValue ::= Hstring
464. VerdictTypeValue ::= "pass" | "fail" | "inconc" | "none" | "error"
465. EnumeratedValue ::= EnumerationIdentifier
466. CharStringValue ::= Cstring | Quadruple
467. Quadruple ::= CharKeyword "(" Group "," Plane "," Row "," Cell ")"
```

```
468. CharKeyword ::= "char"
469. Group ::= Number
470. Plane ::= Number
471. Row ::= Number
472. Cell ::= Number
473. FloatValue ::= FloatDotNotation | FloatENotation
474. FloatDotNotation ::= Number Dot DecimalNumber
475. FloatENotation ::= Number [Dot DecimalNumber] Exponential [Minus] Number
476. Exponential ::= "E"
477. ReferencedValue ::= ValueReference [ExtendedFieldReference]
478. ValueReference ::= [GlobalModuleId Dot] (ConstIdentifier | ExtConstIdentifier |
                        ModuleParIdentifier ) |
                        ValueParIdentifier |
                        VarIdentifier
479. Number ::= (NonZeroNum {Num}) | "0"
480. NonZeroNum ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
481. DecimalNumber ::= {Num}+
482. Num ::= "0" | NonZeroNum
483. Bstring ::= "'" {Bin} "'" "B"
484. Bin ::= "0" | "1"
485. Hstring ::= "'" {Hex} "'" "H"
486. Hex ::= Num | "A" | "B" | "C" | "D" | "E" | "F"| "a" | "b" | "c" | "d" | "e" | "f"
487. Ostring ::= "'" {Oct} "'" "O"
488. Oct ::= Hex Hex
489. Cstring ::= """ {Char} """
490. Char ::= /* REFERENCE - A character defined by the relevant CharacterString type. For
charstring a character from the character set defined in ITU-T Rec. T.50. For universal charstring a
character from any character set defined in ISO/IEC 10646 */
491. Identifier ::= Alpha{AlphaNum | Underscore}
492. Alpha ::= UpperAlpha | LowerAlpha
493. AlphaNum ::= Alpha | Num
494. 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"
495. 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"
496. ExtendedAlphaNum ::= /* REFERENCE - A graphical character from the BASIC LATIN or from the
LATIN-1 SUPPLEMENT character sets defined in ISO/IEC 10646  (characters from char (0,0,0,32) to char
(0,0,0,126), from char (0,0,0,161) to char (0,0,0,172) and from char (0,0,0,174) to char
(0,0,0,255)) */
497. FreeText ::= """ {ExtendedAlphaNum} """
498. AddressValue ::= "null"
499. OmitValue ::= OmitKeyword
500. OmitKeyword ::= "omit"
```

5.6.1.A معلمية

```
501. InParKeyword ::= "in"
502. OutParKeyword ::= "out"
503. InOutParKeyword ::= "inout"
504. FormalValuePar ::= [(InParKeyword | InOutParKeyword | OutParKeyword)] Type
ValueParIdentifier
505. ValueParIdentifier ::= Identifier
506. FormalPortPar ::= [InOutParKeyword] PortTypeIdentifier PortParIdentifier
507. PortParIdentifier ::= Identifier
508. FormalTimerPar ::= [InOutParKeyword] TimerKeyword TimerParIdentifier
509. TimerParIdentifier ::= Identifier
510. FormalTemplatePar ::= [(InParKeyword | OutParKeyword | InOutParKeyword )]
                           TemplateKeyword Type TemplateParIdentifier
511. TemplateParIdentifier ::= Identifier
```

6.6.1.A مع بيان

```
512. WithStatement ::= WithKeyword WithAttribList
513. WithKeyword ::= "with"
514. WithAttribList ::= "{" MultiWithAttrib "}"
515. MultiWithAttrib ::= {SingleWithAttrib [SemiColon]}
516. SingleWithAttrib ::= AttribKeyword  [OverrideKeyword] [AttribQualifier] AttribSpec
517. AttribKeyword ::= EncodeKeyword |
                       VariantKeyword |
                       DisplayKeyword |
                       ExtensionKeyword
518. EncodeKeyword ::= "encode"
519. VariantKeyword ::= "variant"
520. DisplayKeyword ::= "display"
521. ExtensionKeyword ::= "extension"
522. OverrideKeyword ::= "override"
523. AttribQualifier ::= "(" DefOrFieldRefList ")"
524. DefOrFieldRefList ::= DefOrFieldRef {"," DefOrFieldRef}
```

```
525. DefOrFieldRef ::= DefinitionRef | FieldReference | AllRef
/* STATIC SEMANTICS - the DefOrFieldRef must refer to a definition or field which is within the
module, group or definition to which the with statement is associated */
526. DefinitionRef ::= StructTypeIdentifier |
                       EnumTypeIdentifier |
                       PortTypeIdentifier |
                       ComponentTypeIdentifier |
                       SubTypeIdentifier |
                       ConstIdentifier |
                       TemplateIdentifier |
                       AltstepIdentifier |
                       TestcaseIdentifier |
                       FunctionIdentifier |
                       SignatureIdentifier |
                       VarIdentifier |
                       TimerIdentifier |
                       PortIdentifier |
                       ModuleParIdentifier |
                       FullGroupIdentifier
527. AllRef ::= ( GroupKeyword AllKeyword [ExceptKeyword "{" GroupRefList "}"]) |
                ( TypeDefKeyword AllKeyword [ExceptKeyword "{" TypeRefList "}"]) |
                ( TemplateKeyword AllKeyword [ExceptKeyword "{" TemplateRefList "}"]) |
                ( ConstKeyword AllKeyword [ExceptKeyword "{" ConstRefList "}"]) |
                ( AltstepKeyword AllKeyword [ExceptKeyword "{" AltstepRefList "}"]) |
                ( TestcaseKeyword AllKeyword [ExceptKeyword "{" TestcaseRefList "}"]) |
                ( FunctionKeyword AllKeyword [ExceptKeyword "{" FunctionRefList "}"]) |
                ( SignatureKeyword AllKeyword [ExceptKeyword "{" SignatureRefList "}"]) |
                ( ModuleParKeyword AllKeyword [ExceptKeyword "{" ModuleParRefList "}"])
528. AttribSpec ::= FreeText
```

7.6.1.A بيانات سلوك

```
529. BehaviourStatements ::= TestcaseInstance |
                             FunctionInstance |
                             ReturnStatement |
                             AltConstruct |
                             InterleavedConstruct |
                             LabelStatement |
                             GotoStatement |
                             RepeatStatement |
                             DeactivateStatement |
                             AltstepInstance |
                             ActivateOp
/* STATIC SEMANTICS - TestcaseInstance shall not be called from within an existing executing
testcase or function chain called from a testcase i.e. testcases can only be instantiated from the
control part or from functions directly called from the control part */
/* STATIC SEMANTICS - ActivateOp shall not be called from within the module control part */
530. VerdictStatements ::= SetLocalVerdict
531. VerdictOps ::= GetLocalVerdict
532. SetLocalVerdict ::= SetVerdictKeyword "(" SingleExpression ")"
/* STATIC SEMANTICS - SingleExpression must resolve to a value of type verdict */
/* STATIC SEMANTICS - the SetLocalVerdict shall not be used to assign the Value error */
533. SetVerdictKeyword ::= "setverdict"
534. GetLocalVerdict ::= "getverdict"
535. SUTStatements ::= ActionKeyword "(" [ActionText ] {StringOp ActionText} ")"
536. ActionKeyword ::= "action"
537. ActionText ::= FreeText | Expression
/*STATIC SEMANTICS - Expression shall have the base type charstring or universal charstring */
538. ReturnStatement ::= ReturnKeyword  [Expression]
539. AltConstruct ::= AltKeyword "{" AltGuardList "}"
540. AltKeyword ::= "alt"
541. AltGuardList ::= {GuardStatement | ElseStatement [SemiColon]}
542. GuardStatement ::= AltGuardChar (AltstepInstance [StatementBlock] | GuardOp StatementBlock)
543. ElseStatement ::= "["ElseKeyword "]" StatementBlock
544. AltGuardChar ::= "[" [BooleanExpression] "]"
/*STATIC SEMANTICS - BooleanExpression shall conform to restrictions in clause 20.1.2 */
545. GuardOp ::= TimeoutStatement |
                ReceiveStatement |
                TriggerStatement |
                GetCallStatement |
                CatchStatement |
                CheckStatement |
                GetReplyStatement |
                DoneStatement |
                KilledStatement
/* STATIC SEMANTICS - GuardOp used within the module control part shall only contain the
timeoutStatement */
```

```
546. InterleavedConstruct ::= InterleavedKeyword "{" InterleavedGuardList "}"
547. InterleavedKeyword ::= "interleave"
548. InterleavedGuardList ::= {InterleavedGuardElement [SemiColon]}+
549. InterleavedGuardElement ::= InterleavedGuard InterleavedAction
550. InterleavedGuard ::= "[" "]" GuardOp
551. InterleavedAction ::= StatementBlock
/* STATIC SEMANTICS - the StatementBlock may not contain loop statements, goto, activate,
deactivate, stop, return or calls to functions */
552. LabelStatement ::= LabelKeyword LabelIdentifier
553. LabelKeyword ::= "label"
554. LabelIdentifier ::= Identifier
555. GotoStatement ::= GotoKeyword LabelIdentifier
556. GotoKeyword ::= "goto"
557. RepeatStatement ::= "repeat"
558. ActivateOp ::= ActivateKeyword "(" AltstepInstance ")"
559. ActivateKeyword ::= "activate"
560. DeactivateStatement ::= DeactivateKeyword ["(" ComponentOrDefaultReference ")"]
561. DeactivateKeyword ::= "deactivate"
```

8.6.1.A بيانات أساسية

```
562. BasicStatements ::= Assignment | LogStatement | LoopConstruct | ConditionalConstruct |
                         SelectCaseConstruct
563. Expression ::= SingleExpression | CompoundExpression
/* STATIC SEMANTICS - Expression shall not contain Configuration, activate operation or verdict
operations within the module control part */
564. CompoundExpression ::= FieldExpressionList | ArrayExpression
/* STATIC SEMANTICS - Within CompoundExpression the ArrayExpression can be used for Arrays, record,
record of and set of types. */
565. FieldExpressionList ::= "{" FieldExpressionSpec {"," FieldExpressionSpec} "}"
566. FieldExpressionSpec ::= FieldReference AssignmentChar NotUsedOrExpression
567. ArrayExpression ::= "{" [ArrayElementExpressionList] "}"
568. ArrayElementExpressionList ::= NotUsedOrExpression  {"," NotUsedOrExpression}
569. NotUsedOrExpression ::= Expression | NotUsedSymbol
570. ConstantExpression ::= SingleConstExpression | CompoundConstExpression
571. SingleConstExpression ::= SingleExpression
/* STATIC SEMANTICS - SingleConstExpression shall not contain Variables or Module parameters and
shall resolve to a constant Value at compile time */
572. BooleanExpression ::= SingleExpression
/* STATIC SEMANTICS - BooleanExpression shall resolve to a Value of type Boolean */
573. CompoundConstExpression ::= FieldConstExpressionList | ArrayConstExpression
/* STATIC SEMANTICS - Within CompoundConstExpression the ArrayConstExpression can be used for
Arrays, record, record of and set of types. */
574. FieldConstExpressionList ::= "{" FieldConstExpressionSpec {"," FieldConstExpressionSpec} "}"
575. FieldConstExpressionSpec ::= FieldReference AssignmentChar ConstantExpression
576. ArrayConstExpression ::= "{" [ArrayElementConstExpressionList] "}"
577. ArrayElementConstExpressionList ::= ConstantExpression {"," ConstantExpression}
578. Assignment ::= VariableRef AssignmentChar (Expression | TemplateBody)
/* STATIC SEMANTICS - The Expression on the right hand side of Assignment shall evaluate to an
explicit Value of a type compatible with the type of the left hand side for value variables and
shall evaluate to an explicit Value, template (literal or a template instance) or a matching
mechanism compatible with the type of the left hand side for template variables. */
579. SingleExpression ::= XorExpression { "or" XorExpression }
/* STATIC SEMANTICS - If more than one XorExpression exists, then the XorExpressions shall evaluate
to specific values of compatible types */
580. XorExpression ::= AndExpression { "xor" AndExpression }
/* STATIC SEMANTICS - If more than one AndExpression exists, then the AndExpressions shall evaluate
to specific values of compatible types */
581. AndExpression ::= NotExpression { "and" NotExpression }
/* STATIC SEMANTICS - If more than one NotExpression exists, then the NotExpressions shall evaluate
to specific values of compatible types */
582. NotExpression ::= [ "not" ] EqualExpression
/* STATIC SEMANTICS - Operands of the not operator shall be of type boolean (TTCN or ASN.1) or
derivatives of type Boolean. */
583. EqualExpression ::= RelExpression { EqualOp RelExpression }
/* STATIC SEMANTICS - If more than one RelExpression exists, then the RelExpressions shall evaluate
to specific values of compatible types */
584. RelExpression ::= ShiftExpression [ RelOp ShiftExpression ]
/* STATIC SEMANTICS - If both ShiftExpressions exist, then each ShiftExpression shall evaluate to a
specific integer, Enumerated or float Value (these values can either be TTCN or ASN.1 values) or
derivatives of these types */
585. ShiftExpression ::= BitOrExpression { ShiftOp BitOrExpression }
/* STATIC SEMANTICS - Each Result shall resolve to a specific Value. If more than one Result exists
the right-hand operand shall be of type integer or derivatives and if the shift op is '<<' or '>>'
then the left-hand operand shall resolve to either bitstring, hexstring or octetstring type or
derivatives of these types. If the shift op is '<@' or '@>' then the left-hand operand shall be of
type bitstring, hexstring, charstring or universal charstring or derivatives of these types */
586. BitOrExpression ::= BitXorExpression { "or4b" BitXorExpression }
```

```
/* STATIC SEMANTICS - If more than one BitXorExpression exists, then the BitXorExpressions shall
evaluate to specific values of compatible types */
587. BitXorExpression ::= BitAndExpression { "xor4b" BitAndExpression }
/* STATIC SEMANTICS - If more than one BitAndExpression exists, then the BitAndExpressions shall
evaluate to specific values of compatible types */
588. BitAndExpression ::= BitNotExpression { "and4b" BitNotExpression }
/* STATIC SEMANTICS - If more than one BitNotExpression exists, then the BitNotExpressions shall
evaluate to specific values of compatible types */
589. BitNotExpression ::= [ "not4b" ] AddExpression
/* STATIC SEMANTICS - If the not4b operator exists, the operand shall be of type bitstring,
octetstring or hexstring or derivatives of these types. */
590. AddExpression ::= MulExpression { AddOp MulExpression }
/* STATIC SEMANTICS - Each MulExpression shall resolve to a specific Value. If more than one
MulExpression exists and the AddOp resolves to StringOp then the MulExpressions shall resolve to
same type which shall be of bitstring, hexstring, octetstring, charstring or universal charstring or
derivatives of these types. If more than one MulExpression exists and the AddOp does not resolve to
StringOp then the MulExpression shall both resolve to type integer or float or derivatives of these
types.*/
591. MulExpression ::= UnaryExpression { MultiplyOp UnaryExpression }
/* STATIC SEMANTICS - Each UnaryExpression shall resolve to a specific Value. If more than one
UnaryExpression exists then the UnaryExpressions shall resolve to type integer or float or
derivatives of these types. */
592. UnaryExpression ::= [ UnaryOp ] Primary
/* STATIC SEMANTICS - Primary shall resolve to a specific Value of type integer or float or
derivatives of these types.*/
593. Primary ::= OpCall | Value | "(" SingleExpression ")"
594. ExtendedFieldReference ::= {(Dot ( StructFieldIdentifier | TypeDefIdentifier))
                               | ArrayOrBitRef }+
/* STATIC SEMANTIC - The TypeDefIdentifier shall be used only if the type of the VarInstance or
ReferencedValue in which the ExtendedFieldReference is used is anytype.*/
595. OpCall ::= ConfigurationOps |
                VerdictOps |
                TimerOps |
                TestcaseInstance |
                FunctionInstance |
                TemplateOps |
                ActivateOp
596. AddOp ::= "+" | "-" | StringOp
/* STATIC SEMANTICS - Operands of the "+" or "-" operators shall be of type integer or float or
derivations of integer or float (i.e. subrange) */
597. MultiplyOp ::= "*" | "/" | "mod" | "rem"
/* STATIC SEMANTICS - Operands of the "*", "/", rem or mod operators shall be of type integer or
float or derivations of integer or float (i.e. subrange). */
598. UnaryOp ::= "+" | "-"
/* STATIC SEMANTICS - Operands of the "+" or "-" operators shall be of type integer or float or
derivations of integer or float (i.e. subrange). */
599. RelOp ::= "<" | ">" | ">=" | "<="
/* STATIC SEMANTICS - the precedence of the operators is defined in Table 7 */
600. EqualOp ::= "==" | "!="
601. StringOp ::= "&"
/* STATIC SEMANTICS - Operands of the string operator shall be bitstring, hexstring, octetstring or
character string */
602. ShiftOp ::= "<<" | ">>" | "<@" | "@>"
603. LogStatement ::= LogKeyword "(" LogItem { "," LogItem } ")"
604. LogKeyword ::= "log"
605. LogItem ::= FreeText | TemplateInstance
606. LoopConstruct ::= ForStatement |
                       WhileStatement |
                       DoWhileStatement
607. ForStatement ::= ForKeyword "(" Initial SemiColon Final SemiColon Step ")"
                      StatementBlock
608. ForKeyword ::= "for"
609. Initial ::= VarInstance | Assignment
610. Final ::= BooleanExpression
611. Step ::= Assignment
612. WhileStatement ::= WhileKeyword "(" BooleanExpression ")"
                        StatementBlock
613. WhileKeyword ::= "while"
614. DoWhileStatement ::= DoKeyword StatementBlock
                          WhileKeyword "(" BooleanExpression ")"
615. DoKeyword ::= "do"
616. ConditionalConstruct ::= IfKeyword "(" BooleanExpression ")"
                              StatementBlock
                              {ElseIfClause}[ElseClause]
617. IfKeyword ::= "if"
618. ElseIfClause ::= ElseKeyword  IfKeyword "(" BooleanExpression ")"  StatementBlock
619. ElseKeyword ::= "else"
620. ElseClause ::= ElseKeyword  StatementBlock
621. SelectCaseConstruct ::= SelectKeyword "(" SingleExpression ")" SelectCaseBody
622. SelectKeyword ::= "select"
```

```
623. SelectCaseBody ::= "{" { SelectCase }+ "}"
624. SelectCase ::= CaseKeyword ( '(' TemplateInstance {"," TemplateInstance } ')' | ElseKeyword )
                    StatementBlock
625. CaseKeyword ::= "case"
```

9.6.1.A إنتاج متنوع

```
626. Dot ::= "."
627. Dash ::= "-"
628. Minus ::= Dash
629. SemiColon ::= ";"
630. Colon ::= ":"
631. Underscore ::= "_"
632. AssignmentChar ::= ":="
```

الملحق B

مواءمة قيم واصلة

1.B آليات مواءمة مقاس

0.1.B عـــام

يحدد هذا الملحق آليات المواءمة التي يمكن أن تستخدم في مقاسات TTCN-3 (وفي مقاسات فقط).

1.1.B مواءمة قيم محددة

إن القيم المحددة هي آلية مواءمة أساسية لمقاسات TTCN-3. والقيم المحددة في مقاسات هي تعبيرات لا تحتوي على أي آليات مواءمة أو سمات واحدة. وما لم يحدد غير ذلك، يتواءم مجال مقاس مع قيمة مجال واصل متناظر إذا، وإذا فقط، كان لقيمة المجال الواصل نفس القيمة بالضبط لقيمة تقيم التعبير في المقاس.

مثـــال:

```
// Given the message type definition
type record MyMessageType
{
  integer     field1,
  charstring  field2,
  boolean     field3 optional,
  integer[4]  field4
}

// A message template using specific values could be
template MyMessageType MyTemplate:=
{
  field1 := 3+2,           // specific value of integer type
  field2 := "My string",   // specific value of charstring type
  field3 := true,          // specific value of boolean type
  field4 := {1,2,3}        // specific value of integer array
 }
```

1.1.1.B حذف قيم

تدل الكلمة المفتاحية **omit** على غياب مجال مقاس خياري. ويمكن أن يستخدم في قيم لجميع الأنماط الموفرة، بشرط أن يكون مجال المقاس ذلك خيارياً.

مثـــال:

```
template Mymessage MyTemplate:=
{  :
   :
   field3 := omit,  // omit this field
   :
 }
```

2.1.B آليات مواءمة بدلاً من قيم

0.2.1.B عام

يمكن استخدام آليات المواءمة التالية مكان قيم واضحة.

1.2.1.B قائمة قيم

تحدد قوائم قيم قوائم مقبولة للقيم الواصلة. ويمكن أن تستخدم في قيم لجميع الأنماط. ومجال مقاس يستخدم قائمة قيم توائم مجال واصل متناظر إذا، وإذا فقط، كانت قيمة مجال واصل متوائمة مع أي قيمة من القيم في قائمة القيم. وتكون كل قيمة في قائمة القيم من نمط معلن عنه لمجال مقاس حيث تستخدم هذه الآلية.

مثـــال:

```
template Mymessage MyTemplate:=
{
  field1 := (2,4,6),              // list of integer values
  field2 := ("String1", "String2"),// list of charstring values
  :
  :
 }
```

2.2.1.B قائمة قيم مكملة

تدل الكلمة المفتاحية **complement** على قائمة لقيم لا تقبل كقيم واصلة (أي، أنها استكمال لقائمة قيم). ويمكن أن تستخدم لجميع الأنماط.

وتكون كل قيمة في القائمة من نمط معلن عنه لمجال مقاس يستخدم فيه الاستكمال. إن مجال مقاس يستخدم لاستكمال يتواءم مع مجال واصل متناظر إذ، وإذا فقط، لم يتواءم المجال الواصل مع أي قيمة واردة في قائمة القيم. ويمكن أن تكون قائمة القيم قيمة وحيدة، بالطبع.

مثـــال:

```
template Mymessage MyTemplate:=
{
  complement (1,3,5),  // list of unacceptable integer values
  :
  field3 not(true)     // will match false
  :
}
```

3.2.1.B أي قيمة

يستخدم رمز المواءمة (*AnyValue*) "?" للدلالة على أن أي قيمة واصلة صالحة مقبولة. ويمكن أن تستخدم في قيم لكل الأنماط. إن مجال مقاس يستخدم أي آلية قيمة تتواءم مع مجال واصل متناظر إذ، وإذا فقط، قيم المجال الواصل لعنصر وحيد لنمط محدد.

```
template Mymessage MyTemplate:=
{
  field1 := ?,    // will match any integer
  field2 := ?,    // will match any non-empty charstring value
  field3 := ?,    // will match true or false
  field4 := ?     // will match any sequence of integers
}
```

4.2.1.B أي قيمة أو لا شيء

يستخدم رمز المواءمة (*AnyValueOrNone*) "*" للدلالة على أن أي قيمة واصلة صالحة، بما في ذلك حذف تلك القيمة، مقبولة. ويمكن أن يستخدم في قيم لكل الأنماط، بشرط أن مجال المقاس المعلن عنه خياري.

إن مجال مقاس يستخدم هذا الرمز يتواءم مع مجال واصل متناظر إذا، وإذا فقط، قيم مجال واصل لأي عنصر لنمط محدد، أو إذا كان المجال الواصل غائباً.

مثـــال:

```
template Mymessage MyTemplate:=
{  :
  field3 := *,    // will match true or false or omitted field
  :
 }
```

5.2.1.B مدى قيم

تدل الأمدية على مدى موثق لقيم مقبولة. وعند استخدام قيم أنماط **integer** و**float** types (أنماط فرعية لصحيح أو طليق). وتكون قيمة حدود إما:

أ) لا نهائية أو –لا نهائية؛

ب) تعبيراً يقيم قيمة صحيح أو طليق محدد.

توضع الحدود الدنيا على الجانب الأيسر للمدى، والحدود العليا على الجانب الأيمن. وتكون الحدود الدنيا أقل من الحدود العليا. إن مجال المقاس الذي يستخدم مدى يتواءم مع مجال واصل متناظر إذا، وإذا فقط، كانت قيمة مجال واصل مساوية لواحدة من القيم في المدى.

وعندما تستخدم في مقاسات أو مجالات مقاسات لأنماط **charstring** و**universal charstring**، تقيم الحدود مواضع السمة الصالحة طبقاً لجدول (جداول) مجموعة سمات مشفرة (مثلاً، لا يكون موضع ما فارغاً). ولا تعتبر المواضع بين الحدود الدنيا والعليا قيم صالحة لمدى محدد.

مثـــال:

```
template Mymessage MyTemplate:=
{
  field1 := (1 .. 6),  // range of integer type
  :
  :
  :
}
// other entries for field1 might be (-infinity to 8) or (12 to infinity)
```

SuperSet 6.2.1.B

إن SuperSet هي عملية للمواءمة تستخدم فقط في قيم أنماط **set of**. ويدل على SuperSet الكلمة المفتاحية **superset**. والمجال الذي يستخدم SuperSet يتواءم مع مجال واصل متناظر إذا، وإذا فقط، احتوى مجال واصل على جميع العناصر المعرفة على الأقل في SuperSet، ويمكن أن يحتوي على أكثر. ويكون متغير SuperSet من النمط المعلن عنه لمجال تستخدم فيه آلية SuperSet.

مثـــال:

```
type set of integer MySetOfType;

template MySetOfType MyTemplate1 := superset ( 1, 2, 3 );
// any sequence of integers matches which contains at least one occurrence of the numbers
// 1, 2 and 3 in any order and positions
```

SubSet 7.2.1.B

إن SubSet هي عملية للمواءمة يمكن أن تستخدم فقط في أنماط **set of**. ويدل على SubSet الكلمة المفتاحية **subset**.

والمجال الذي يستخدم SubSet يتواءم مع مجال واصل متناظر إذا، وإذا فقط، احتوى مجال واصل فقط على عناصر معرفة في SubSet، ويمكن أن يحتوي على أقل. ويكون متغير SubSet من النمط المعلن عنه لمجال تستخدم فيه آلية SubSet.

مثـــال:

```
template MySetOfType MyTemplate1:= subset ( 1, 2, 3 );
// any sequence of integers matches which contains zero or one occurrence of the numbers
// 1, 2 and 3 in any order and positions
```

3.1.B آليات مواءمة داخل قيم

0.3.1.B عـــام

يمكن استخدام آليات المواءمة التالية داخل قيم strings وrecords وrecords of وsets وsets of وarrays.

1.3.1.B أي عنصر

يستخدم رمز المواءمة (*AnyElement*) "?" ليدل على أنه يستبدل عناصر وحيدة لسلسلة (باستثناء سلاسل سمات) **record of** أو صفيف. ويستخدم فقط في قيم لأنماط سلسلة وأنماط **set of** وأنماط **record of** ومصفوفات.

مثـــال:

```
template Mymessage MyTemplate:=
{ :
  field2 := "abcxyz",
  field3 := '10???'B,  // where each "?" may either be 0 or 1
  field4 := {1, ?, 3}  // where ? may be any integer value
}
```

ملاحظة – يمكن استخدام "?" in field4 لتفسير *AnyValue* باعتباره *AnyElement* لقيمة صحيح، أو **record of** داخل **set of** أو صفيف. ونظراً لأن كلا التفسيرين يؤديان إلى نفس المواءمة، لا تنشأ مشكلة.

1.1.3.1.B استخدام سمة وحيدة

إذا طلب التعبير عن سمة وحيدة "?" في سلاسل لسمات واحدة، تتم بواسطة استخدام مخططات سمات (انظر 5.1.B). فمثلاً، "abcdxyz"، "abcxxyz" "abccxyz" تتواءم مع كل "abccxyz" **pattern**. ومع ذلك، لا تتواءم "abcxyz"، "abcdefxyz".

2.3.1.B أي عدد لعناصر أو غياب عناصر

يستخدم رمز المواءمة (*AnyElementsOrNone*) "*" ليدل على أنه يستبدل أي عدد أو لا شيء لعناصر متتالية لسلسلة (باستثناء سلسلة سمات) أو **record of**، **set of** أو صفيف. ويتواءم الرمز "*" مع أطول تتابع لعناصر ممكنة، طبقاً للتخطيط كما حددته الرموز المحيطة بـ "*".

مثـــال:

```
template Mymessage MyTemplate:=
{ :
  field2 := "abcxyz",
  field3 := '10*11'B,  //  where "*" may be any sequence of bits (possibly empty)
  field4 := {*, 2, 3}  // where "*" may be any number of integer values or omitted
}
var charstring MyStrings[4];
MyPCO.receive(MyStrings:{"abyz", *, "abc" });
```

إذا ظهر "*" عند أعلى مستوى داخل سلسلة، **record of** و**set of** أو صفيف، يفسر على أنه *AnyElementsOrNone*.

ملاحظة – تمنع هذه القاعدة التفسير الممكن لـ "*" باعتبارها *AnyValueOrNone* حيث تستبدل عنصر داخل سلسلة أو **record of** أو **set of** أو صفيف.

1.2.3.1.B استخدام سمات متعددة لسمات وحيدة

إذا طلب التعبير عن سمة واحدة "*" في سلاسل سمات، تتم بواسطة استخدام مخططات سمات (انظر 5.1.B). فمثلاً، "abcxyz"، "abcdefxyz"، "abcabcxyz" تتواءم مع كل "abc*xyz" **pattern**.

3.3.1.B الإبدال

إن الإبدال عملية لمواءمة تستخدم فقط في قيم أنماط **record of**. ويدل على الإبدال الكلمة المفتاحية **permutation**. ويسمح بتعبيرات *AnyElement* و*AnyElementsOrNone* كعناصر إبدال. ويكون كل عنصر وارد في الإبدال نمط يستنسخه نمط **record of**.

ويعني إبدال في محل عنصر وحيد أن أي سلسلة من عناصر مقبولة على شرط أنه يحتوى على نفس العناصر كقائمة قيم في الإبدال، بالرغم أن من الممكن بترتيب مختلف. وإذا استخدم كل من الإبدال و*AnyElementsOrNone* داخل قيمة، يتم تقييمهما على نحو مشترك.

إن *AnyElementsOrNone* المستخدم داخل إبدال يحل محل لا شيء أو أي عدد من العناصر في قطع تسجيل قيمة متوائمة بواسطة إبدال. ويقيم *AnyElementsOrNone* داخل إبدال في الآخر (عندما تتواءم جميع العناصر الأخرى لقائمة الإبدال مع عنصر في قائمة مقيمة فعلاً).

الملاحظة 1 – إن *AnyElementsOrNone* المستخدم داخل إبدال له تأثير مختلف باعتباره *AnyElementsOrNone* مستخدم بالتزامن مع إبدال كما في *AnyElementsOrNone* الأخير ويحل محل عناصر متتالية فقط. فمثلاً، {permutation(1,2,*)} يكافئ ({*,2,*,1,*,2,*}، {*,1,*,2,*,2,*})، بينما {**permutation**(1,2),*} يكافئ ({1,2},{2,1},{*,*}).

الملاحظة 2 – عندما يستخدم *AnyElementsOrNone* بالتزامن مع نعت طول لإبدال يمكن أن ينطبق على *AnyElementsOrNone* ليقيد عدد العناصر المتوائمة بواسطة *AnyElementsOrNone* (انظر أيضاً 1.4.1.B). وعلى العكس، لا يضاف نعت طول إلى *AnyElementsOrNone* مستخدم داخل إبدال (ولكن يمكن أن ينطبق على كامل الإبدال بدلاً من ذلك).

مثـــال:

```
type record of integer MySequenceOfType;

template MySequenceOfType MyTemplate1 := { permutation ( 1, 2, 3 ), 5 };
// matches any of the following sequences of 4 integers: 1,2,3,5; 1,3,2,5; 2,1,3,5;
// 2,3,1,5; 3,1,2,5; or 3,2,1,5

template MySequenceOfType MyTemplate2 := { permutation ( 1, 2, ? ), 5 };
// matches any sequence of 4 integers that ends with 5 and contains 1 and 2 at least once in
// other positions

template MySequenceOfType MyTemplate3 := { permutation ( 1, 2, 3 ), * };
// matches any sequence of integers starting with 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2 or 3,2,1

template MySequenceOfType MyTemplate4 := { *, permutation ( 1, 2, 3 )};
// matches any sequence of integers ending with 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2 or 3,2,1

template MySequenceOfType MyTemplate5 := { *, permutation ( 1, 2, 3 ),* };
// matches any sequence of integers containing any of the following substrings at any
position:
// 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2 or 3,2,1

template MySequenceOfType MyTemplate6 := { permutation ( 1, 2, * ), 5 };
// matches any sequence of integers that ends with 5 and containing 1 and 2 at least once in
// other positions
```

```
template MySequenceOfType MyTemplate7 := { permutation ( 1, 2, 3 ), * length (0..5)};
// matches any sequence of three to eight integers starting with 1,2,3; 1,3,2; 2,1,3; 2,3,1;
// 3,1,2 or 3,2,1

template MySequenceOfType MyTemplate9 := { permutation ( 1, 2, *) length (3..5), 5 };
// matches any sequence of four to six integers that ends with 5 and contains 1 and 2 at least
// once in other position
```

4.1.B مواءمة نعوت قيم

0.4.1.B عام

يمكن أن تتصاحب النعوت التالية مع آليات مواءمة.

1.4.1.B تقييدات الطول

يستخدم نعت تقييد طول لتقييد طول قيم سلسلة وعدد من العناصر في **set of**، **record of** أو بنية صفيف. ويستخدم فقط كنعت للآليات التالية: *AnyValue* و*AnyValueOrNone* و*AnyElement* (وليس داخل إبدال) وإبدال ومجموعة ثانوية ومجموعة فرعية. ويمكن استخدامه أيضا بالتزامن مع آلية مواءمة مكملة ومع نعت *AnyElementsOrNone*. وتوجد قواعد التركيب لـ **length** في 3.2.6 و3.3.6.

ملاحظة – وعندما تستخدم كل من التكملة وآليات مواءمة تقييد طول لمقاس أو مجال مقاس، تنطبق التقييدات المتضمنة فيها على مقاس أو مجال مقاس بشكل مستقل.

تفسر وحدات الطول طبقاً للجدول 4 في حالة قيم سلسلة. وبالنسبة لأنماط ومصفوفات **set of** و**record of** تكون وحدة الطول نمطاً مستنسخاً. ويدل على الحدود التعبيرات التي تبين قيم **integer** غير سالبة محددة. وبديلاً عن ذلك، يمكن استخدام الكلمة المفتاحية **infinity** كقيمة للحدود العليا للدلالة على عدم جود حد أعلى للطول.

لا تتعارض مواصفات الطول لمقاس مع الطول للتقييدات (إن وجدت) لنمط متناظر. إن مجال مقاس يستخدم طول كنعت لرمز يتواءم مع مجال واصل متناظر إذا، وإذا فقط، تواءم مجال واصل مع كل من الرمز ونعته المتصاحب. ويتواءم نعت الطول إذا كان طول المجال الواصل أكبر من أو مساو لحدود دنيا محددة وأقل من أو مساو للحدود العليا. وفي حالة قيمة طول وحيدة، يتواءم نعت الطول فقط إذا كان طول المجال المستقبل هو بالضبط القيمة المحددة.

من المسموح استخدام تقييد طول بالتزامن مع القيمة الخاصة **omit**؛ ومع ذلك، في هذه الحالة، ليس لنعت الطول تأثير (أي، مع **omit** يكون ذا إطناب). ومع **AnyValueOrNone** و**ifpresent** يحل محل تقييد في القيمة الواصلة، إن وجدت.

مثـــال:

```
template Mymessage MyTemplate:=
{
  field1 := complement ({4,5},{1,4,8,9}) length (1 .. 6), // any value containing 1, 2, 3, 4,
      // 5 or 6 elements is accepted provided it is not {4,5} or {1,4,8,9}
  field2 := "ab*ab" length(13) // max length of the AnyElementsOrNone string is 9 characters
:
}
```

2.4.1.B مؤشر IfPresent

يدل **ifpresent** على حدوث مواءمة إذا كان مجالاً خيارياً موجوداً (أي، لم يحذف). ويمكن استخدام هذا النعت مع جميع آليات المواءمة، بشرط إعلان النمط على أنه خياري.

إن مجال مقاس يستخدم **ifpresent** يتواءم مع مجال واصل متناظر إذا، وإذا فقط، تواءم المجال الواصل طبقاً لآلية المواءمة المتصاحبة أو كان المجال الواصل غائباً.

مثـــال:

```
template Mymessage:MyTemplate:=
{  :
   field2 := "abcd" ifpresent, // matches "abcd" if not omitted
   :
   :
 }
```

ملاحظة – إن *AnyValueOrNone* له نفس المعنى مثل **ifpresent ?**.

5.1.B تخطيط سمة مواءمة

0.5.1.B عام

يمكن استخدام تخطيطات سمات في مقاسات لتعريف نسق لسلسة سمات مطلوب استقبالها. ويمكن استخدام تخطيطات سمات لمواءمة قيم **charstring** و**universal charstring**. وبالإضافة إلى سمات حرفية، تسمح تخطيطات سمات باستخدام سمات تحويلية (مثل ? و* في تخطيط سمة يعني مواءمة أي سمة وأي عدد لأي سمة على التوالي).

المثـــال 1:

```
template charstring MyTemplate:= pattern "ab??xyz*0"
```

ويتواءم هذا المقاس مع أي سلسلة سمات تتألف من سمات "ab" تتبعها أي سمتين وتتبعها سمات "xyz" ويتبعها أي عدد من السمات (بما في ذلك أي عدد s"0") قبل غلق السمة "0".

وإذا طلب تفسير أي سمة تحويلية حرفيا، ينبغي أن يسبقها السمة التحويلية '\'.

المثـــال 2:

```
template charstring MyTemplate:= pattern "ab?\?xyz*";
```

ويتواءم هذا المقاس مع أي سلسلة سمات تتألف من سمات 'ab' تتبعها أي سمتين وتتبعها سمات 'xyz?' ويتبعها أي عدد من السمات.

ويرد في الجدول 1.B قائمة سمات تحويلية لتخطيطات TTCN-3. ولا تحتوي سمات تحويلية على مسافات بيضاء باستثناء مسافة بيضاء تسبق سمة الخط الجديد قبل أو داخل مجموعة تعبيرات.

الجدول Z.140/1.B – قائمة سمات تحويلية لتخطيطات TTCN-3

الوصف	سمة تحويلية
تواؤم أي سمة (انظر الملاحظتين 1 و2)	?
تواؤم أي سمة صفر أو أكثر من مرة؛ يتواءم مع أكبر عدد ممكن من السمات (انظر المثال 1 أعلاه) (انظر الملاحظتين 1 و2)	*
تسبب تفسير السمات التحويلية التالية حرفياً (انظر الملاحظة 3). وعندما يسبق سمة دون سمة تحويلية معرفة يعني '\' تتواءم السمع مع السمة التالية '\' (انظر الملاحظة 4).	\
تواؤم أي سمة في مجموعة محددة (انظر 1.5.1.B لمزيد من التفاصيل)	[ ]
لها سمة تحويلية تعني داخل زوج من الأقواس المربعة ("[" and "]") فقط، باستثناء الموضعين الأول والأخير في القوسين. وتسمح بتحديد مدى سمات (انظر 1.5.1.B لمزيد من التفاصيل)	-
له سمة تحويلية تعني السمة الأولى التالية لفتح القوسين المربعين داخل زوج من الأقواس المربعة ("[" and "]") فقط، وتسبب مواءمة أي سمة تكميلية لمجموعة سمات تلي السمة التحويلية هذه (انظر 1.5.1.B لمزيد من التفاصيل)	^
توائم السمة العالمية المحددة بواسطة التربيعي	\q{group,plane,row,cell}
تدرج مستعمل مرجعي معرف لسلسلة ويفسرها باعتبارها تعبيراً عادياً (انظر 2.1.5.B لمزيد من التفاصيل)	{reference}
توائم أي سمة في مجموعة سمات، حيث المجموعة معرفة بواسطة تعريف مرجعي (انظر 4.5.1.B لمزيد من التفاصيل)	\N{reference}
توائم أي رقم عددي (مكافئ لـ [0-9])	\d
توائم أي سمة هجائية رقمية (مكافئة لـ [0-9a-zA-Z])	\w
توائم C0 لسمة تحكم HT(11) (انظر [11] ISO/IEC 6429)	\t
توائم أي من C0 لسمات تحكم: LF(10), VT(11), FF(12), CR(13) (انظر [7] ISO/IEC 6429 (تسمى سمات خط جديد على نحو مشترك)	\n
توائم C0 لسمة تحكم CR (انظر [11] ISO/IEC 6429)	\r
توائم أي سمة من C0 التالية لسمات تحكم: HT(9), LF(10), VT(11), FF(12), CR(13) CR (انظر [7] ISO/IEC 6429)، [9] ITU-T T.50 (تسمى سمات خط جديد على نحو مشترك)	\s
توائم حدود كلمة (تسبق أو تتبع أي سمة بيانية باستثناء SP أو DEL بأي سمات مسافة بيضاء أو خط جديد)	**\b**
توائم سمة علامة التنصيص	\"
توائم سمة علامة التنصيص	""
تستخدم لتدل على تعبيرين بديلين	|

الجدول Z.140/1.B – قائمة سمات تحويلية لتخطيطات TTCN-3

الوصف	سمة تحويلية
تستخدم لتجميع تعبير	( )
توائم التعبير السابق n مرات على الأقل ولكن ليس أكثر من m مرة (postfix) (انظر 3.5.1.B لمزيد من التفاصيل)	#(n, m)
توائم التعبير السابق بالضبط n مرات (حيث n هو رقم وحيد) (postfix)؛ ونفسه كما لـ #(n)	#n
توائم التعبير السابق مرة واحدة أو مرات عديدة (postfix)؛ نفسه كما لـ #(1,)	+

الملاحظة 1 – إن * and ? Metacharacters قادرة على مواءمة أي سمات لمجموعة سمات لنمط جذر مقاس أو مجال مقاس تستخدم فيها (أي، لا تعتبر تقييدات نمط مطبق). ومع ذلك، لا يغيب عن البال أن العمليات المستقبلة تتطلب التأكد من نمط الرسالة المستقبَلة قبل محاولة المواءمة. ولهذا، لا تتوفر قيم مستقبلة لا تمتثل لمواصفة نمط فرعي لمقاس أو مجال مقاس للمواءمة مطلقاً.

الملاحظة 2 – في بعض لغات/ترميزات أخرى ? و* لها معاني مختلفة باعتبارها سمات تحويلية. ومع ذلك، في TTCN-3 تستخدم هذه السمات تقليدياً للمواءمة بالمعنى المحدد في هذا الجدول.

الملاحظة 3 – وبناء على ذلك، يمكن مواءمة سمة // بواسطة سمات // دون مسافة بينهما (\\)، مثلاً، تخطيط '\\d' يتواءم مع السلسلة '\d'؛ ويمكن مواءمة فتح أو غلق قوسين مربعين بواسطة '\[' and '\]' على التوالي.

الملاحظة 4 – إن استخدام سمة ترحيلية '\' لا ينصح به لأن مزيداً من سمات ترحيلية يمكن تعريفها لاحقاً.

1.5.1.B تعبير مجموعة

تتواءم قائمة من السمات المغلقة بزوجين من '[' and ']' مع أي سمة وحيدة في تلك القائمة. ويعين حدود تعبير مجموعة الحدود بواسطة الرموز ']' '['. وبالإضافة إلى سمات حرفية، من الممكن تحديد أمدية من السمات باستخدام الشرطة '-' كفاصل. ويتألف المدى من سمة قبل الفاصل مباشرة وسمة بعده مباشرة وجميع السمات مع شفرة سمة بين شفرات لسمتين متجاورتين. وتفقد السمة ذات الشرطة '-' داخل القائمة ولكن دون سمة سابقة أو لاحقة معناها الخاص.

ويمكن أيضاً نفي تعبير مجموعة بواسطة وضع سمة '^' باعتبارها السمة الأولى بعد فتح القوسين المربعين. ويسبق نفي أمدية سمة. ولهذا، تعالج شرطة '-' التالية مباشرة العلامة '^' كسمة حرفية.

لا يسمح بقائمة فارغة ولا بقائمة نفي فارغة. ولهذا، يعالج قفل قوس مربع '[' تالي مباشرة لفتح قوس مربع ']' أو علامة ']' تالية لفتح قوس مربع وتالية مباشرة لغلق قوس مربع '[' كسمات حرفية.

إن جميع السمات الترحيلية، باستثناء الواردة أدناه، تفقد معانيها داخل القائمة:

- '[' ليس في الموضع الأول ولا يلي مباشرة عند الموضع الأول؛
- '-' ليس في الموضع الأول أو الأخير في القائمة؛
- '^' في الموضع الأول في القائمة باستثناء عندما يليه مباشرة قوس مربع غلق؛
- '\', \d, \t, \w, \r, \n, \s و\b؛
- \q{group,plane,row,cell}؛
- .**\N**{reference}

الملاحظة 1 – لا يسمح بقوائم مدمجة (مثلاً في تخطيط '[[ab[r-z]' يدل على ']' الثاني ']' على '[' حرفي، ويغلق '[' القائمة والثاني '[' يسبب خطأ حيث لا يوجد فتح قوس في التخطيط).

الملاحظة 2 – لتضمين سمة حرفية "^"، ضعها في أي مكان في الموضع الأول أو اسبقها بعلامة "-". ولتضمين شرطة حرفية "**[**"، ضعها في المكان الأول أو الأخير في القائمة أو اسبقها بـ //. ولتضمين قوس مربع لغلق حرفي "^"، ضعه في الأول أو اسبقه بـ //. وإذا كانت السمة الأولى في القائمة "^"، فإن السمات "-" و"**[**" توائم نفسها عندما تلي مباشرة "^".

مثـــال:

```
template charstring RegExp1:= pattern '[a-z]';  // this will match any character from a to z

template charstring RegExp2:= pattern '[^a-z]';  // this will match any character except a to
z

template charstring RegExp3:= pattern '[AC-E][0-9][0-9][0-9]YKE';

// RegExp3 will match a string which starts with the letter A or a letter between
// C and E (but not e.g. B) then has three  digits and the letters YKE
```

2.5.1.B تعبير مرجع

بالإضافة إلى قيم سلسلة مباشرة، من الممكن أيضاً في التخطيط استخدام مراجع لمقاسات أو ثوابت أو متغيرات أو معلمات وحدة موجودة. ويحتوي المرجع في سمات "{" "}" ويُستبان المرجع لنمط من أنماط سلسلة سمات. ويجرى مناولة محتويات مقاسات أو ثوابت أو متغيرات مرجعية كتعبير عادي. ويجري تبديل كل تعبير مرة واحدة فقط.

مثـــال:

```
const charstring MyString:= "ab?";

template charstring MyTemplate:= pattern '{MyString}';
```

يتواءم هذا المقاس مع أي سلسلة سمات تتألف من سمات 'ab'، تتبعها أي سمة. وبالفعل، تفسر أي سلسلة سمات تلي الكلمة المفتاحية **pattern** سواء صراحة أو بواسطة مرجع متبعة القواعد المعرفة في هذا القسم.

```
template universal charstring MyTemplate1:= pattern '{MyString}de\q{1, 1, 13, 7}';
```

يتواءم هذا المقاس مع أي سلسلة سمات تتألف من سمات 'ab'، تتبعها أي سمة وتتبعها السمات 'de' وتتبعها السمة في ISO/IEC 10646 مع group=1 وplane=1 وrow=13 وcell=7.

إذا أشار تعبير مرجع إلى مقاس أو ثابت أو متغير يحتوي على تعبير واحد أو أكثر، فإن المراجع في المقاس أو الثابت أو المتغير المشار إليه تبدل على نحو متكرر قبل إدراج محتوياتها في التخطيط المرجعي.

مثـــال:

```
const charstring MyConst2 := pattern "ab";
template charstring RegExp1 := pattern "{MyConst2}";
   // matches the string "ab"
template charstring RegExp2 := pattern "{RegExp1}{RegExp1}";
   // matches the string "abab"
template charstring RegExp3 := pattern "c{RegExp2}d";
   // matches the string "cababd"

template charstring RegExp4 := pattern "{Reg";
template charstring RegExp5 := pattern "Exp1}";
template charstring RegExp6 := pattern "{RegExp4}{RegExp5}";
   // matches the string "{RegExp1}" only (i.e. shall not be handled as a reference expression
   // to the template RegExp1)
```

3.5.1.B مواءمة n مرة لتعبير

لتحديد أن التعبير الأسبق ينبغي مواءمته لعدد من المرات، تستخدم إحدى قواعد التركيب التالية: '#(n, m)' أو '#(n,)' أو '#(,m)' أو '#(n)' أو 'n#' أو '+'. ويحدد شكل '#(n, m)' أن التعبير الأسبق يجب مواءمته n مرات على الأقل ولكن ليس أكثر من m مرة. وتحدد السمة الترحيلية '#(n,)' أن التعبير الأسبق يجب مواءمته n مرات على الأقل بينما يدل '#(, m)' على أن التعبير الأسبق يتواءم ليس أكثر من m مرة. وتحدد السمات الترحيلية '#(n)' و'n#' أن التعبير الأسبق يجب مواءمته n مرات بالضبط (وهي مكافئة لـ '#(n, n)'). وفي الشكل 'n#'، تكون n رقم وحيد. وتدل السمة الترحيلية '+' على أن التعبير الأسبق يجب مواءمته مرة واحدة على الأقل (مكافئ لـ '#(1,)').

مثـــال:

```
template charstring RegExp4:= pattern '[a-z]#(9, 11)'; // match at least 9 but no more than 11
                                                       // characters from a to z
template charstring RegExp5a:= pattern '[a-z]#(9)';    // match exactly 9
                                                       // characters from a to z
template charstring RegExp5b:= pattern '[a-z]#9';      // match exactly 9
                                                       // characters from a to z
template charstring RegExp6:= pattern '[a-z]#(9, )';   // match at least 9
                                                       // characters from a to z
template charstring RegExp7:= pattern '[a-z]#(, 11)';  // match no more than 11
                                                       // characters from a to z
template charstring RegExp8:= pattern '[a-z]+';        // match at least 1
                                                       // characters from a to z,
```

4.5.1.B مواءمة مجموعة سمات مرجعية

إن ترميز الشكل **"\N{reference}"**، حيث يدل مرجع على مقاس أو ثابت أو متغير أو معلمة وحدة لطول سمة واحدة، يتواءم مع السمة في قيمة أو مقاس مرجعي.

يسبب مقاس أو ثابت أو متغير أو معلمة وحدة مرجعية ليس طولها 1 خطأ.

يكون ترميز للشكل **"\N{**typereference**}"**، حيث "typereference" مرجع لنمط **charstring** أو**universal charstring**، يتواءم مع أي سمة لمجموعة سمات يدل عليها النمط المرجعي.

الملاحظة 1 – إن الحالات عندما تكون مجموعة سمات مرجعية ليست مجموعة فرعية لقيم حقيقية يسمح بها لتعريف النمط لمقاس أو مجال مقاس يستخدم له تخطيط السمة لا تعالج كخطأ (ولكن مثلاً، لا يمكن أن تحدث مواءمة إذا لم تتراكب المجموعتان).

الملاحظة 2 – إن **{charstring}**\N مكافئ لـ ? عندما ينطبق الأخير على مقاس أو مجال مقاس لنمط **charstring** ويكون **{universal charstring}**\N مكافئ لـ ? عندما ينطبق الأخير على مقاس أو مجال مقاس لنمط **universal charstring** (ولكن يسبب خطأ إذا انطبق على مقاس أو مجال مقاس لنمط **charstring**).

مثـــال:

```
type charstring MyCharRange ('a'..'z');
type charstring MyCharList ('a', 'z');
const MyCharRange myCharR := 'r';

template charstring myTempPatt1 := pattern '\N { myCharR }';
// myTempPatt1 shall match the string 'r' only

template charstring myTempPatt2 := pattern '\N { MyCharRange }';
// myTempPatt2 shall match any string containing a single character from a to z

template MyCharRange myTempPatt3 := pattern '\N { MyCharList }';
// myTempPatt3 and shall match strings 'a' and 'r' only

template MyCharList myTempPatt4 := pattern '\N { MyCharRange }';
// myTempPatt4 shall match strings 'a' and 'r' only
```

5.5.1.B قواعد مواءمة نمط لتخطيطات

لأغراض تخطيطات مرجعية (انظر 2.5.1.B) ومجموعات سمات مرجعية (انظر 4.5.1.B)، تنطبق قواعد مواءمة لنمط محدد: يمكن دائماً استخدام نمط أو مقاس أو ثابت أو متغير أو معلمة وحدة مرجعية لنمط **charstring** في مواصفة تخطيط مقاس أو مجال مقاس لنمط **universal charstring**؛ ويمكن استخدام نمط أو مقاس أو ثابت أو متغير أو معلمة وحدة مرجعية لنمط **universal charstring** في مواصفة تخطيط مقاس أو مجال مقاس لنمط **charstring** إذا كانت جميع السمات المستخدمة في المقاس أو القيمة المرجعية، ومجموعة السمات التي يسمح بها النمط المرجعي، لها سمات متناظرة في نمط **charstring** (انظر تعريف السمات المتناظرة في 1.7.6).

الملحق C

وظائف TTCN-3 المعرّفة مسبقاً

يُعرّف هذا الملحق وظائف TTCN-3 المعرفة مسبقاً.

0.C إجراءات عامة لمناولة استثناء

إن حالات الخطأ (مثل، معلمة دخل تكون خارج المدى المسموح، ومعلمة دخل تكون من نمط خاطئ وقيمة دخل تحتوي على سمة غير صحيحة وما إلى ذلك) التي ليس لها قاعدة مناولة استثناء واضحة تعرف في الأقسام ذات العلاقة بهذا الملحق، وتسبب خطأ في وقت مُصرف أو وقت تنفيذ TTCN-3. إن أي حالة خطأ تسبب خطأ وقت مُصرف وأي حالة خطأ تسبب وقت تنفيذ هي أداة تنفيذ خيارية.

1.C تحويل صحيح إلى سمة

```
int2char(integer value) return charstring
```

تحوّل هذه الوظيفة قيمة **integer** في مدى من صفر إلى 127 (تشوير 8 بتات) إلى قيمة **charstring** لطول سمة وحيدة. وتصف قيمة الصحيح تشفير 8 بتات للسمة.

2.C تحويل سمة إلى صحيح

```
char2int(charstring value) return integer
```

تحوّل هذه الوظيفة قيمة **charstring** لطول سمة وحيدة إلى قيمة صحيح في مدى من صفر إلى 127 وتصف قيمة الصحيح تشفير 8 بتات للسمة.

3.C تحويل صحيح إلى سمة عالمية

```
int2unichar(integer value) return universal charstring
```

تحوّل هذه الوظيفة قيمة **integer** في مدى 0 إلى 2 147 483 647 (تشوير 32 بتة) إلى قيمة **universal charstring** لطول سمة وحيدة. وتصف قيمة الصحيح تشفير 32 بتة للسمة.

4.C تحويل سمة عالمية إلى صحيح

```
unichar2int(universal charstring value) return integer
```

تحوّل هذه الوظيفة قيمة **universal charstring** لطول سمة وحيدة إلى قيمة صحيح في مدى من 0 إلى 2 147 483 647. وتصف قيمة الصحيح تشفير 32 بتة للسمة.

5.C تحويل سلسلة ثنائية إلى صحيح

```
bit2int(bitstring value) return integer
```

تحوّل هذه الوظيفة قيمة **bitstring** وحيدة إلى قيمة **integer** وحيدة.

لأغراض هذا التحويل، يفسر **bitstring** على أنه قاعدة موجبة لقيمة **integer** 2. وتكون البتة أقصى اليمين هي الأقل أهمية، والبتة أقصى اليسار الأهم. وتمثل البتتان صفر و1 قيم عشرية صفر و1 على التوالي.

6.C تحويل سلسلة ستة عشرية إلى صحيح

```
hex2int(hexstring value) return integer
```

تُحوّل هذه الوظيفة قيمة **hexstring** وحيدة إلى قيمة **integer** وحيدة.

لأغراض هذا التحويل، يفسر **hexstring** على أنه قاعدة موجبة لقيمة **integer** 16. ويكون الرقم الستة عشري لأقصى اليمين هو الأقل أهمية، والرقم الستة عشري لأقصى اليسار الأهم. وتمثل الأرقام الستة عشرية من صفر إلى F قيم عشرية من صفر إلى 15 على التوالي.

7.C تحويل سلسلة أثمونات إلى صحيح

```
oct2int(octetstring value) return integer
```

تُحوّل هذه الوظيفة قيمة **octetstring** وحيدة إلى قيمة **integer** وحيدة.

لأغراض هذا التحويل، يفسر **octetstring** على أنه قاعدة موجبة لقيمة **integer** 16. ويكون الرقم الستة عشري لأقصى اليمين هو الأقل أهمية، والرقم الستة عشري لأقصى اليسار الأهم. وتكون الأرقام الستة عشرية الموفرة مضاعفات لـ 2 نظراً لأن أثمونا واحداً يتألف من رقمين ستة عشرة. ويمثل الرقمان الستة عشري صفر إلى F قيم عشرية من صفر إلى 15 على التوالي.

8.C تحويل سلسلة سمات إلى صحيح

```
str2int(charstring value) return integer
```

تحول هذه الوظيفة قيمة **charstring** الممثلة لقيمة **integer** إلى **integer** مكافئ.

مثـــال:

```
str2int("66")     // will return the integer value 66
str2int("-66")    // will return the integer value -66
str2int("abc")    // will generate compiler or testcase error
str2int("0")      // will return the integer value 0
```

9.C تحويل صحيح إلى سلسلة ثنائية

```
int2bit(in integer value, in integer length) return bitstring
```

تُحوّل هذه الوظيفة قيمة **integer** وحيدة إلى قيمة **bitstring** وحيدة. وتكون السلسلة الناتجة هي length بتة طولاً.

لأغراض هذا التحويل، يفسر **bitstring** على أنه قاعدة موجبة لقيمة **integer** 2. وتكون البتة أقصى اليمين هي الأقل أهمية، والبتة أقصى اليسار الأهم.وتمثل البتتان صفر إلى 1 قيم عشرية صفر و1 على التوالي. وإذا أدى التحويل إلى قيمة مع بتات أقل من المحددة في معلمة length، فإن **bitstring** يوهن على اليسار مع أصفار.

10.C تحويل صحيح إلى سلسلة ستة عشرية

```
int2hex(in integer value, in integer length) return hexstring
```

تُحوّل هذه الوظيفة قيمة **integer** وحيدة إلى قيمة **integer** وحيدة. وتكون السلسلة الناتجة هي length رقم ستة عشري طولاً.

لأغراض هذا التحويل، يفسر **hexstring** على أنه قاعدة موجبة لقيمة **integer** 16. ويكون الرقم الستة عشري لأقصى اليمين هو الأقل أهمية، والرقم الستة عشري لأقصى اليسار الأهم. وتمثل الأرقام الستة عشري صفر إلى F قيم عشرية من صفر إلى 15 على التوالي. وإذا أدى التحويل إلى قيمة مع أرقام ستة عشرية أقل من المحددة في معلمة length، فإن **hexstring** يوهن على اليسار مع أصفار.

11.C تحويل صحيح إلى سلسلة أثمونات

```
int2oct(in integer value, in integer length) return octetstring
```

تُحوّل هذه الوظيفة قيمة **integer** وحيدة إلى قيمة **octetstring** وحيدة. وتكون السلسلة الناتجة هي length أثمونات طولاً.

لأغراض هذا التحويل، يفسر **octetstring** على أنه قاعدة موجبة لقيمة **integer** 16. ويكون الرقم الستة عشري لأقصى اليمين هو الأقل أهمية، والرقم الستة عشري لأقصى اليسار الأهم. وتكون أعداد الأرقام الستة عشرية الموفرة مضاعفات لـ 2 نظراً لأن أثمون واحد يتألف من رقمين ستة عشري. وتمثل الأرقام الستة عشرية من صفر إلى F قيم عشرية من صفر إلى 15 على التوالي. وإذا أدى التحويل إلى قيمة مع أرقام ستة عشرية أقل من المحددة في معلمة length، فإن **hexstring** يوهن على اليسار مع أصفار.

12.C تحويل صحيح إلى سلسلة سمات

```
int2str(integer value) return charstring
```

تُحوّل هذه الوظيفة قيمة صحيح إلى سلسلتها المكافئة (تكون قاعدة سلسلة العودة عشرية دائماً).

مثـــال:

```
int2str(66)    // will return the charstring value "66"
int2str(-66)   // will return the charstring value "-66"
int2str(0)     // will return the charstring value "0"
```

13.C طول نمط سلسلة

```
lengthof(any_string_type value) return integer
```

تعيد هذه الوظيفة طول قيمة من نمط **bitstring** أو **hexstring** أو **octetstring** أو أي سلسلة سمات. وتعرّف وحدات طول كل نمط سلسلة في الجدول 4.

يحسب طول **universal charstring** بواسطة عد كل سمة مركبة وسمة مقطع hangul (بما في ذلك مرشحات) بمفردها (انظر ISO/IEC 10646 [10] والقسمين 23 و24).

مثـــال:

```
lengthof('010'B) // returns 3
lengthof('F3'H) // returns 2
lengthof('F2'O) // returns 1
lengthof (universal charstring : "Length_of_Example") // returns 17
```

14.C عدد عناصر في قيمة مبنية

```
sizeof(any_type value) return integer
```

تعيد هذه الوظيفة العدد الفعلي لعناصر معلمة وحدة أو ثابت أو متغير أو **template** لنمط **record**، **record of**، **set**، **set of** أو صفيف (انظر الملاحظة). وفي حالة قيم **record of** و**set of** أو مقاسات أو مصفوفات، تكون القيمة الفعلية التي تعاد هي عدد تتابعي لآخر عنصر معرف (دليل ذلك العنصر زائداً 1).

ملاحظة – إن عناصر شيء TTCN-3 فقط، يجري حسابها لأنها معلمة وظيفة ، أي، لا تؤخذ عناصر لأنماط/قيم متداخلة في عين الاعتبار عند تحديد عودة قيمة.

مثـــال:

```
// Given
type record MyPDU
  {  boolean field1  optional,
     integer field2
  };

template MyPDU  MyTemplate
  { field1  omit,
    field2 5
  };

var integer numElements;

// then

numElements := sizeof(MyTemplate); // returns 1

// Given
type record length(0..10) of integer MyList;
```

```
var MyList MyRecordVar;
MyRecordVar := { 0, 1, omit, 2, omit };


// then
numElements := sizeof(MyRecordVar);
// returns 4 without respect to the fact, that the element MyRecordVar[2] is undefined
```

15.C وظيفة IsPresent

```
ispresent(any_type value) return Boolean
```

يُسمح بهذه الوظيفة لنمطي **record** و**set** فقط وتعيد قيمة **true** إذا، وإذا فقط، كانت قيمة المجال المرجعي موجودة في المطابق الفعلي لشيء معطيات مرجعية. ويكون متغير **ispresent** مرجعاً لمجال نمط **record** أو **set**.

```
// Given
type record MyRecord
  {   boolean  field1 optional,
      integer field2
  }
// and given that MyPDU is a template of MyRecord type
// and received_PDU is also of MyRecord type
// then
MyPort.receive(MyPDU) -> value received_PDU
ispresent(received_PDU.field1)
// returns true if field1 in the actual instance of MyPDU is present
```

16.C وظيفة IsChosen

```
ischosen(any_type value) return boolean
```

تعيد هذه الوظيفة **true** إذا، وإذا فقط، كان مرجع شيء معطيات محدد لمتغير نمط **union** مختار فعلاً لشيء معطيات ما.

مثـــال:

```
// Given
type union MyUnion
  {   PDU_type1   p1,
      PDU_type2   p2,
      PDU_type p3
  }

// and given that MyPDU is a template of MyUnion type
// and received_PDU is also of MyUnion type
// then
MyPort.receive(MyPDU) -> value received_PDU
ischosen(received_PDU.p2)
// returns true if the actual instance of MyPDU carries a PDU of the type PDU_type2
```

17.C وظيفة Regexp

```
regexp (any_character_string_type instr, charstring expression, integer groupno) return
character_string_type
```

تعيد هذه الوظيفة السلسلة الفرعية لسلسلة سمات دخل، وهي محتوى مواءمة زمرة instr لـ n-th. وفي سلسلة الدخل، يمكن أن يكون expression أي نمط سلسلة سمات. ويكون نمط سلسلة سمات معادة هو نمط جذر instr. ويكون التعبير هو تخطيط سمة كما ورد في 5.1.B. ويحدد عدد الزمرة الذي يعاد instr، ويكون صحيح موجب. وتخصص أعداد الزمرات حسب ترتيب الحدوث لقوس مفتوح لزمرة ويبدأ العد من صفر حسب الخطوة 1. وإذا لم تلبي السلسلة الفرعية الشروط (أي، التخطيط وعدد الزمرة) الموجودة في سلسلة الدخل، تعاد سلسلة فارغة.

مثـــال:

```
   // Given
var charstring mypattern2 := "
var charstring  myinput := '      date: 2001-10-20 ;  msgno: 17; exp  '
var charstring mypattern := '[ /t]#(,)date:[ \d\-]#(,);[ /t]#(,)msgno: (\d#(1,3)); [exp]#(0,1)'

// Then the expression
var charstring mystring := regexp(myinput, mypattern,1)
//will return the value '17'.
```

18.C تحويل سلسلة ثنائية إلى سلسلة سمات

```
bit2str (bitstring value) return charstring
```

تحول هذه الوظيفة قيمة **bitstring** وحيدة إلى **charstring** وحيدة. ويكون لـ **charstring** الناتجة نفس الطول مثل **bitstring** وتحتوي على سمات '0' أو '1' فقط.

ولغرض هذا التحويل، ينبغي تحويل **bitstring** إلى **charstring**. وتُحوّل كل بتة **bitstring** إلى سمة '0' أو '1' يعتمد على القيمة صفر أو 1 للبتة. ويكون الترتيب المتوالي للسمات في **charstring** الناتج هو نفسه الترتيب لبتات في **bitstring**.

مثـــال:

```
bit2str ('1110101'B) will return "1110101"
```

19.C تحويل سلسلة ستة عشرية إلى سلسلة سمات

```
hex2str (hexstring value) return charstring
```

تحول هذه الوظيفة سلسلة ستة عشرية وحيدة إلى سلسلة سمات وحيدة. ويكون لسلسلة السمات الناتجة نفس الطول كما لسلسلة ستة عشرية وتحتوي على سمات '0' إلى '9' و'A' إلى 'F' فقط.

ولغرض هذا التحويل، ينبغي تحويل **hexstring** إلى **charstring**. ويحول كل رقم ستة عشري **hexstring** إلى سمة يعتمد على القيمة من '0' إلى '9' و'A' إلى 'F' للرقم ستة عشري. ويكون الترتيب المتوالي للسمات في **charstring** الناتج هو نفسه الترتيب للأرقام في **hexstring**.

مثـــال:

```
hex2str ('AB801'H) will return "AB801"
```

20.C تحويل سلسلة أثمونات إلى سلسلة سمات

```
oct2str (octetstring invalue) return charstring
```

تحول هذه الوظيفة **octetstring** invalue إلى **charstring** تمثل مكافئ سلسلة لقيمة الدخل. وتكون **charstring** الناتجة لها نفس الطول لـ **octetstring** الواصلة.

ولغرض هذا التحويل، يحول كل رقم ستة عشري لـ invalue إلى سمة '0'، '1'، '2'، '3'، '4'، '5'، '6'، '7'، '8'، '9'، 'a'، 'b'، 'c'، 'd'، 'e'، 'f'،'A'، 'B'، 'C'، 'D'، 'E' أو 'F'، مرددا قيمة الرقم ستة عشري. ويكون الترتيب المتوالي للسمات في **charstring** الناتج هو نفسه الترتيب لأرقام ستة عشرية في **octetstring**.

مثـــال:

```
oct2str ('4469707379'O) = "4469707379"
```

21.C تحويل سلسلة سمات إلى سلسلة أثمونات

مثـــال:

```
str2oct (charstring invalue) return octetstring
```

تحول هذه الوظيفة سلسلة لنمط **charstring** إلى **octetstring**. وتحتوي سلسلة invalue على سمات عدد زوجي ويكون كل واحد من السمات البيانية '0'، '1'، '2'، '3'، '4'، '5'، '6'، '7'، '8'، '9'، 'a'، 'b'، 'c'، 'd'، 'e'، 'f'،'A'، 'B'، 'C'، 'D'، 'E' أو 'F' فقط. ويكون **octetstring** الناتج هو نفس الطول لـ **charstring** الواصل.

مثـــال:

```
str2oct ("54696E6B792D57696E6B79") = '54696E6B792D57696E6B79'O
```

22.C تحويل سلسلة ثنائية إلى سلسلة ستة عشرية

```
bit2hex (bitstring value) return hexstring
```

تحول هذه الوظيفة قيمة **bitstring** وحيدة إلى **hexstring** وحيدة. وتمثل **hexstring** الناتجة نفس القيمة كما لـ **bitstring**.

ولغرض هذا التحويل، تحول سلسلة ثنائية إلى سلسلة ستة عشرية، حيث تقسم السلسلة الثنائية إلى زمرات من أربع بتات تبدأ من البتة في أقصى اليمين. وتحوّل كل زمرة من البتات الأربع إلى رقم ستة عشري كما يلي:

'0000'B → '0'H, '0001'B → '1'H, '0010'B → '2'H, '0011'B → '3'H, '0100'B → '4'H, '0101'B → '5'H,
'0110'B → '6'H, '0111'B → '7'H, '1000'B → '8'H, '1001'B → '9'H, '1010'B → 'A'H, '1011'B → 'B'H,
'1100'B → 'C'H, '1101'B → 'D'H,'1110'B → 'E'H, and '1111'B → 'F'H.

وعندما تحتوي زمرة البتات أقصى اليسار على ما لا يقل عن 4 بتات، تملأ هذه الزمرة بـ '0'B من اليسار حتى تحتوي على 4 بتات بالضبط وتحول بعد ذلك. ويكون الترتيب المتوالي لأرقام ستة عشرية في السلسلة ستة عشرية الناتجة هو نفسه ترتيب زمرات الأربع بتات في السلسلة الثنائية.

مثـــال:

```
bit2hex ('111010111'B)= '1D7'H
```

23.C تحويل سلسلة ستة عشرية إلى سلسلة أثمونات

```
hex2oct (hexstring value) return octetstring
```

تحول هذه الوظيفة قيمة **hexstring** وحيدة إلى **octetstring** وحيدة. وتمثل **octetstring** الناتجة نفس القيمة كما لـ **hexstring**.

ولغرض هذا التحويل، تحول **hexstring** إلى **octetstring**، حيث **octetstring** يحتوي على نفس تتابع الأرقام ستة عشرية كما لـ **hexstring** عندما يكون طول مقاس 2 لـ **hexstring** هو صفر. وإلا، تحتوي **octetstring** الناتجة على صفر كما لرقم ستة عشري أقصى اليسار يتبعه نفس تتابع أرقام ستة عشرية في **hexstring**.

مثـــال:

```
hex2oct ('1D7'H)= '01D7'O
```

24.C تحويل سلسلة ثنائية إلى سلسلة أثمونات

```
bit2oct (bitstring value) return octetstring
```

تُحوّل هذه الوظيفة قيمة **bitstring** وحيدة إلى **octetstring** وحيدة. وتمثل **octetstring** الناتجة نفس القيمة كما لـ **bitstring**.

ولغرض التحويل، يحتفظ بما يلي: bit2oct(value)=hex2oct(bit2hex(value))

مثـــال:

```
bit2oct ('111010111'B)= '01D7'O
```

25.C تحويل سلسلة ستة عشرية إلى سلسلة ثنائية

```
hex2bit (hexstring value) return bitstring
```

تحول هذه الوظيفة قيمة **hexstring** وحيدة إلى **bitstring** وحيدة. وتمثل **bitstring** الناتجة نفس القيمة كما لـ **hexstring**.

لغرض هذا التحويل، تحول **hexstring** إلى **bitstring**، حيث الأرقام ستة عشرية لـ **hexstring** تحول في زمرة البتات كما يلي:

'0'H → '0000'B, '1'H → '0001'B, '2'H → '0010'B, '3'H → '0011'B, '4'H → '0100'B, '5'H → '0101'B,
'6'H → '0110'B, '7'H → '0111'B, '8'H → '1000'B, '9'H → '1001'B, 'A'H → '1010'B, 'B'H → '1011'B,
'C'H → '1100'B, 'D'H → '1101'B,'E'H → '1110'B, and 'F'H → '1111'B.

يكون الترتيب المتوالي لزمرات 4 بتات في **bitstring** الناتجة هو نفس ترتيب الأرقام ستة عشرية في **hexstring**.

مثـــال:

```
hex2bit ('1D7'H)= '000111010111'B
```

26.C تحويل سلسلة أثمونات إلى سلسلة ستة عشرية

```
oct2hex (octetstring value) return hexstring
```

تُحوّل هذه الوظيفة قيمة **octetstring** وحيدة إلى **hexstring** وحيدة. وتمثل **hexstring** الناتجة نفس القيمة كما لـ **octetstring**.

لغرض هذا التحويل، تحول **octetstring** إلى **hexstring** المحتوية على نفس تتابع أرقام ستة عشرية كما لـ **octetstring**.

مثـــال:

```
oct2hex ('1D74'O)= '1D74'H
```

27.C تحويل سلسلة أثمونات إلى سلسلة ثنائية

```
oct2bit (octetstring value) return bitstring
```

تُحوّل هذه الوظيفة قيمة **octetstring** وحيدة إلى **bitstring** وحيدة. وتمثل **bitstring** الناتجة نفس القيمة كما لـ **octetstring**.

ولغرض التحويل، يحتفظ بما يلي: oct2bit(value)=hex2bit(oct2hex(value))

مثـــال:

```
oct2bit ('01D7'O)='0000000111010111'B
```

28.C تحويل صحيح إلى طليق

```
int2float (integer value) return float
```

تحول هذه الوظيفة قيمة **integer** إلى قيمة **float**.

مثـــال:

```
int2float(4)
```

29.C تحويل طليق إلى صحيح

```
float2int (float value) return integer
```

تُحوّل هذه الوظيفة قيمة **float** إلى قيمة **integer** بواسطة إزالة الجزء النسبي للمتغير ويعيد **integer** الناتج.

مثـــال:

```
float2int(3.12345E2) = float2int(312.345) = 312
```

30.C وظيفة مولد عدد عشوائي

```
rnd ([float seed]) return float
```

تعيد وظيفة **rnd** عدد (شبه) عشوائي أقل من 1 ولكن أكبر أو مساوٍ لصفر. ويدمّث مولد عدد عشوائي بواسطة قيمة مغادرة خيارية. وبعد ذلك، إذا لم توفر مغادرة جديدة، يستخدم العدد المولد الأخير كمغادرةً للعدد العشوائي التالي. ودون تدميث سابق، تستخدم القيمة المحسوبة من وقت النظام كقيمة مغادرة عندما يستخدم **rnd** لأول مرة.

ملاحظة – في كل مرة تدمث وظيفة **rnd** مع نفس قيمة المغادرة، تكرر نفس تتابع الأعداد العشوائية.

ولإيجاد صحيح عشوائي في مدى معين، يمكن استخدام الصيغة التالية:

```
float2int(int2float(upperbound - lowerbound +1)*rnd()) + lowerbound
// Here, upperbound and lowerbound denote highest and lowest number in range.
```

31.C وظيفة سلسلة فرعية

```
substr (any_string_type value, in integer index, in integer returncount) return input_string_type
```

تعيد هذه الوظيفة سلسلة فرعية من قيمة تكون من نمط **bitstring**، **hexstring**، **octetstring** أو أي سلسلة سمات. ويكون نمط السلسلة الفرعية هو نمط جذر قيمة الدخل. وتكون نقطة السلسلة لسلسلة فرعية تعاد معرفة بواسطة الثانية في معلمة (دليل). وتبدأ الأدلة من صفر. وتعرف معلمة الدخل الثالثة طول السلسلة الفرعية الذي يعاد. وتكون وحدات الطول هي المعرفة في الجدول 4.

مثـــال:

```
substr ('00100110'B, 3, 4)      // returns '0011'B
substr ('ABCDEF'H, 2, 3)        // returns 'CDE'H
substr ('01AB23CD'O, 1, 2)      // returns 'AB23'O
substr ("My name is JJ", 11, 2) // returns "JJ"
```

32.C عدد العناصر في نمط مبني

```
sizeoftype(any_type value) return integer
```

تعيد هذه الوظيفة عدد العناصر المعلن عنها لمعلمة وحدة أو ثابت أو متغير أو **template** لنمط **record of** أو **set of** أو صفيف (انظر الملاحظة). وتنطبق هذه الوظيفة على قيم أنماط مع تقييد طول. ويكون العدد الفعلي الذي يعاد عدد متوالي لآخر عنصر دون اهتمام ما إذا كانت قيمته معرفة أم لا (أي، يكون دليل الطول الأعلى لتعريف نمط لمعلمة وظيفة قائم على زائد 1).

ملاحظة – إن عناصر شيء TTCN-3 فقط، يجري حسابها لأنها معلمة وظيفة، أي، لا تؤخذ عناصر لأنماط/قيم متداخلة في عين الاعتبار عند تحديد عودة قيمة.

مثـــال:

```
// Given
type record of integer MyPDU1;
type set length(1..8) of integer MyPDU2;
type record length(10) of integer MyPDU3;

var MyPDU1 MyRecordOfVar1;
var MyPDU2 MyRecordOfVar2;
var MyPDU3 MyRecordOfVar3;

var integer numElements;

// then
numElements := sizeoftype(MyRecordOfVar1);  // returns error as MyPDU1 is not constrained
numElements := sizeoftype(MyRecordOfVar2);  // returns 8
numElements := sizeoftype(MyRecordOfVar3);  // returns 10
```

33.C تحويل سلسلة سمات إلى طليق

```
str2float (charstring value) return float
```

تحول هذه الوظيفة **charstring** التي تتألف من عدد نقط طليقة إلى قيمة **float**. ويتبع نسق العدد في **charstring** القواعد في 1.6 مع الاستثناءات التالية:

- يسمح بالأصفار الرائدة،
- يسمح بالعلامة '+' الرائدة قبل القيم الموجبة،
- يسمح بـ '-0.0'.

```
str2float('12345.6') //  is the same as str2float('123.456E+02')
```

34.C وظيفة Replace

```
replace (in any_string_type str, in integer ind, in integer len, in any_string_type repl)
return any_string_type
```

تحل هذه الوظيفة محل السلسلة الفرعية لقيمة **str** عند دليل **ind** لطول **len** مع قيمة سلسلة **repl** وتعيد السلسلة الناتجة **str** التي لا تعدل. وإذا كان **len** هو صفر، تُدرج سلسلة **repl**. وإذا كان **ind** هو صفر، يُدرج **repl** عند بداية **str**. وإذا كان **ind** هو lengthof(**str**)، يُدرج **repl** في نهاية **str**. ويكون **repl** هو نفس نمط السلسلة ويكون له نمط قاعدة **bitstring**، **hexstring**، **octetstring** أو أي سلسلة سمات. وتكون السلسلة المعادة هي نفس نمط **str** و**repl**. ولاحظ أن الأدلة في السلسلة تبدأ من صفر.

تؤدي حالات الخطأ التالية إلى خطأ عند مُصرف أو وقت تنفيذ:

- إن **str** أو **repl** ليس من نمط سلسلة؛
- إن **str** و**rep** من نمط مختلف؛
- إن **ind** أقل من صفر أو أكبر من **(str)lengthof**؛
- إن **len** أقل من صفر أو أكبر من **(str)lengthof**؛
- إن ind+len أكبر من lengthof(str).

مثـــال:

```
replace ('00000110'B, 1, 3, '111'B)     // returns '01110110'B
replace ('ABCDEF'H, 0, 2, '123'H)       // returns '123CDEF'H
replace ('01AB23CD'O, 2, 1, 'FF96'O)    // returns '01ABFF96CD'O
replace ("My name is JJ", 11, 1, "xx")  // returns "My name is xxJ"
replace ("My name is JJ", 11, 0, "xx")  // returns "My name is xxJJ"
replace ("My name is JJ", 2, 2, "x")    // returns "Myxame is JJ",
replace ("My name is JJ", 12, 2, "xx")  // produces test case error
replace ("My name is JJ", 13, 2, "xx")  // produces test case error
replace ("My name is JJ", 13, 0, "xx")  // returns "My name is JJxx"
```

35.C تحويل سلسلة أثمونات إلى سلسلة سمات

```
oct2char (octetstring invalue) return charstring
```

تُحوّل هذه الوظيفة invalue **octetstring** إلى **charstring**. ولا تحتوي معلمة الدخل invalue على قيم أثمون أعلى من 7F. ويكون لـ **charstring** الناتجة نفس طول الدخل **octetstring** الدخل. وتفسر الأثمونات طبقاً لشفرات التوصية ITU-T T.50 [5] (طبقاً لـ IRV) وتذيل السمات الناتجة بالقيمة المعادة.

مثـــال:

```
oct2char ('4469707379'O) = "Dipsy"
```

ملاحظة – يمكن أن تحتوي سلسلة السمات المعادة على سمات غير بيانية، لا يمكن تقديمها بين علامتي تنصيص.

36.C تحويل سلسلة سمات إلى سلسلة أثمونات

```
char2oct (charstring invalue) return octetstring
```

تُحوّل هذه الوظيفة invalue **charstring** إلى **octetstring**. ويحتوي كل أثمون لـ **octetstring** على شفرات التوصية ITU-T T.50 [5] (طبقاً لـ IRV) للسمات الملائمة لـ invalue.

مثـــال:

```
char2oct ("Tinky-Winky") = '54696E6B792D57696E6B79'O
```

الملحق D (للعلم)

فارغ

ملاحظة – تم نقل محتوى هذا الملحق إلى التوصية [6] ITU-T Z.146.

الملحق E (للعلم)

مكتبة أنماط مفيدة

1.E تحديدات

ينبغي أن تكون أسماء أنماط تضاف إلى هذه المكتبة وحيدة في كامل اللغة وفي المكتبة (أي، لا ينبغي أن يكون اسم لأسماء معرفة في الملحق C). ولا ينبغي للأسماء المعرفة في هذه المكتبة أن يستخدمها مستعملو TTCN-3 كمعرفات لتعاريف أخرى غير الواردة في هذا الملحق.

ملاحظة – ولهذا، يمكن تكرار تعاريف نمط واردة في هذا الملحق في وحدات TTCN-3، ولكن لا يمكن تعريف نمط مميز من واحد محدد في هذا الملحق مع واحد من المعرفات المستخدمة في هذا الملحق.

2.E أنماط TTCN-3 مفيدة

1.2.E أنماط بسيطة مفيدة أساسية

0.1.2.E أعداد صحيحة موقعة وغير موقعة لبايتة وحيدة

تدعم هذه الأنماط قيم صحيح لمدى من -128 إلى 128 لنمط موقع ومن صفر إلى 255 لنمط غير موقع. ويكون ترميز القيمة لهذه الأنماط هو نفس ترميز القيمة لنمط صحيح. وتُشفر قيم هذه الأنماط ويُفكك تشفيرها بينما تقدم على بايتة واحدة في النظام، مستقلة عن شكل التقديم المستخدم.

ملاحظة – يمكن أن يكون تشفير قيم هذه الأنماط هو نفسه أو مختلف من واحد إلى آخر ومن تشفير نمط صحيح (نمط الجذر للأنماط المفيدة هذه) يعتمد على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتكون تعاريف نمط لهذه الأنماط هي:

```
type integer   byte      (-128 .. 127)     with { variant "8 bit" };

type integer   unsignedbyte (0 .. 255)     with { variant "unsigned 8 bit" };
```

1.1.2.E أعداد صحيحة قصيرة موقعة وغير موقعة

تدعم هذه الأنماط قيم صحيح لمدى من 768 32 – إلى 767 32 لنمط موقع ومن صفر إلى 535 65 لنمط غير موقع. ويكون ترميز القيمة لهذه الأنماط هو نفس ترميز القيمة لنمط صحيح. وتُشفر قيم هذه الأنماط ويُفكك تشفيرها بينما تقدم على بايتين في النظام، مستقلة عن شكل التقديم المستخدم.

ملاحظة – يمكن أن يكون تشفير قيم هذه الأنماط هو نفسه أو مختلف من واحد إلى آخر، ومن تشفير نمط صحيح (نمط الجذر للأنماط المفيدة هذه) يعتمد على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتكون تعاريف نمط لهذه الأنماط هي:

```
type integer   short        (-32768 .. 32767)   with { variant "16 bit" };

type integer   unsignedshort  (0 .. 65535)      with { variant "unsigned 16 bit" };
```

2.1.2.E أعداد صحيحة طويلة موقعة وغير موقعة

تدعم هذه الأنماط قيم صحيح لمدى –648 483 147 2 إلى 647 483 147 2 لنمط موقع و0 إلى 295 967 294 4 لنمط غير موقع. ويكون ترميز القيمة لهذه الأنماط هي نفس ترميز القيمة لنمط صحيح. وتُشفر قيم هذه الأنماط ويُفكك تشفيرها، بينما تقدم على أربع بايتات في النظام، مستقلة عن شكل التقديم المستخدم.

ملاحظة – يمكن أن يكون تشفير قيم هذه الأنماط هو نفسه أو مختلف من واحد إلى آخر، ومن تشفير نمط صحيح (نمط الجذر للأنماط المفيدة هذه) يعتمد على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتكون تعاريف نمط لهذه الأنماط هي:

```
type integer   long(-2147483648 .. 2147483647)
      with { variant "32 bit" };

type integer   unsignedlong(0 .. 4294967295)
      with { variant "unsigned 32 bit" };
```

3.1.2.E أعداد صحيحة مزدوجة الطول موقعة وغير موقعة

تدعم هذه الأنماط قيم صحيح لمدى -808 775 854 036 372 223 9 إلى 807 775 854 036 372 223 9 لنمط موقع و0 إلى 615 551 709 073 744 446 18 لنمط غير موقع. ويكون ترميز القيمة لهذه الأنماط هي نفس ترميز القيمة لنمط صحيح. وتُشفر قيم هذه الأنماط ويُفكك تشفيرها بينما تقدم على بايتة واحدة في النظام، مستقلة عن شكل التقديم المستخدم.

ملاحظة – يمكن أن يكون تشفير قيم هذه الأنماط هو نفسه أو مختلف من واحد إلى آخر ومن تشفير نمط صحيح (نمط الجذر للأنماط المفيدة هذه) يعتمد على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتكون تعاريف نمط لهذه الأنماط هي:

```
type integer  longlong (-9223372036854775808 .. 9223372036854775807)
      with { variant "64 bit" };

type integer  unsignedlonglong (0 .. 18446744073709551615)
      with { variant "unsigned 64 bit" };
```

4.1.2.E IEEE 754 floats

تدعم هذه الأنماط ANSI/IEEE Standard 754 (انظر البيبليوغرافيا) للحساب الاثنيني لنقطة طليقة. ويدعم النمط IEEE 754 float أعداد نقطة طليقة مع قاعدة 10 وأس لحجم 8 والجزء العشري لحجم 23 وبتة علامة. ويدعم نمط IEEE 754 double أعداد نقطة طليقة مع قاعدة 10 وأس لحجم 11 والجزء العشري لحجم 52 وبتة علامة. ويدعم نمط IEEE 754 `extfloat` أعداد نقطة طليقة مع قاعدة 10 وأس أدنى لحجم 11 والجزء العشري الأدنى لحجم 32 وبتة علامة. ويدعم نمط IEEE 754 `extdouble` أعداداً نقطة طليقة مع قاعدة 10 وأس أدنى لحجم 15 والجزء العشري الأدنى لحجم 64 وبتة علامة.

وتُشفر قيم هذه الأنماط ويُفكك تشفيرها طبقاً لتعاريف IEEE 754. ويكون ترميز قيمة هذه الأنماط هو نفسه كما لترميز قيمة لنمط طليق (قاعدة 10).

ملاحظة – يعتمد التشفير الدقيق لقيم هذا النمط على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتكون تعاريف نمط هذه الأنماط هي:

```
type  float    IEEE754float     with { variant "IEEE754 float" };
type  float    IEEE754double    with { variant "IEEE754 double" };
type  float    IEEE754extfloat  with { variant "IEEE754 extended float" };
type  float    IEEE754extdouble     with { variant "IEEE754 extended double" };
```

2.2.E أنماط سلسلة سمات مفيدة

0.2.2.E سلسلة سمات UTF-8 "utf8string"

يدعم هذا النمط مجموعة السمات بكاملها لنمط TTCN-3 **universal charstring** (انظر الفقرة د من 1.1.6). وتكون قيمته المميزة صفر أو واحد أو أكثر من سمة من هذه المجموعة. وتشفر قيم من هذا النمط بكاملها (مثل، كل سمة لقيمة فردية) ويفكك تشفيرها طبقاً لـ UCS Transformation Format 8 (UTF-8) كما عُرّفت في الملحق R من [10] ISO/IEC 10646. ويكون ترميز قيمة لهذا النمط هو نفسه كما لترميز قيمة لنمط **universal charstring**.

وتعريف النمط لهذا النمط هو:

```
type universal charstring utf8string with { variant "UTF-8" };
```

1.2.2.E سلسلة سمات BMP "bmpstring"

يدعم هذا النمط مجموعة سمات Basic Multilingual Plane (BMP) لـ [10] ISO/IEC 10646. ويمثل BMP جميع سمات مستوى 00 لزمرة Universal Multiple-octet coded Character Set 00 لـ [10] ISO/IEC 10646. وتكون قيمته المميزة صفر أو واحد أو أكثر من سمة من BMP. وتُشفر قيم من هذا النمط بكاملها (مثل، كل سمة لقيمة فردية) ويفكك تشفيرها طبقاً للتمثيل المشفر UCS-2 من (انظر 1.14 من [10] ISO/IEC 10646). ويكون ترميز قيمة لهذا النمط هو نفسه كما لترميز قيمة لنمط **universal charstring**.

ملاحظة – يدعم النمط "bmpstring" مجموعة فرعية لنمط TTCN-3 **universal charstring**.

وتعريف النمط لهذا النمط هو:

```
type universal charstring bmpstring ( char ( 0,0,0,0 ) .. char ( 0,0,255,255) )
                        with { variant "UCS-2" };
```

2.2.2.E سلسلة سمات UTF-16

يدعم هذا النمط جميع سمات مستويات 00 إلى 16 لزمرة 00 لـ Universal Multiple-octet coded Character Set (انظر ISO/IEC 10646 [10]). وتكون قيمته المميزة صفر أو واحد أو أكثر من سمة من هذه المجموعة. وتُشفر قيم من هذا النمط بكاملها (مثل، كل سمة لقيمة فردية) ويُفكك تشفيرها طبقاً لـ UCS Transformation Format 16 (UTF-16) كما عُرّف في الملحق Q من ISO/IEC 10646 [6]. ويكون ترميز قيمة لهذا النمط هو نفسه كما لترميز قيمة لنمط **universal charstring**.

ملاحظة – يدعم النمط "utf16string" مجموعة فرعية لنمط TTCN-3 **universal charstring**.

وتعريف النمط لهذا النمط هو:

```
type universal charstring utf16string   ( char ( 0,0,0,0 ) .. char ( 0,16,255,255) )
                  with { variant "UTF-16" };
```

3.2.2.E سلسلة سمات ISO/IEC 8859 "iso8859string"

يدعم هذا النمط جميع السمات في جميع الألفبائية المعرفة في معيار أطراف متعددة ISO/IEC 8859 (انظر البيبليوغرافيا). وتكون قيمته المميزة صفر أو واحد أو أكثر من سمة من مجموعة سمات ISO/IEC 8859. وتشفر قيم من هذا النمط بكاملها (مثل، كل سمة لقيمة فردية) ويفكك تشفيرها طبقاً للتمثيل المشفر كما حدد في ISO/IEC 8859 (تشفير 8 بتات). ويكون ترميز قيمة لهذا النمط هو نفسه كما لترميز قيمة لنمط **universal charstring**.

الملاحظة 1 – يدعم النمط "iso8859string" مجموعة فرعية لنمط TTCN-3 **universal charstring**.

الملاحظة 2 – في كل ألفبائية ISO/IEC 8859 يكون الجزء الأدنى من جدول مجموعة السمات (مواضع 02/00 إلى 07.14 متوائم مع مجموعة سمات ITU-T T.50 [9]. ومن ثم، تعرف جميع السمات المحددة للغة إضافية للجزء الأعلى لجدول السمات فقط (مواضع (10/00 to 15/15). وبما أن نمط "iso8859string" معرف كمجموعة فرعية لسلسلة سمات عالمية لنمط TTCN-3، يمكن تقابل أي تمثيل لسمة مشفرة لأي ألفبائية ISO/IEC 8859 مع سمة متكافئة (سمة مع نفس تمثيل مشفر عندما تشفر على 8 بتات) من جداول سمات Basic Latin أو Latin-1 Supplement لـ ISO/IEC 10646 [6].

وتعريف النمط لهذا النمط هو:

```
type universal charstring iso8859string ( char ( 0,0,0,0 ) .. char ( 0,0,0,255) )
                        with { variant "8 bit" };
```

3.2.E أنماط مفيدة مبنية

0.3.2.E حرفي عشري لنقطة ثابتة

يدعم هذا النمط استخدام حرفي عشري لنقطة ثابتة كما عرف في IDL Syntax and Semantics version 2.6 (انظر البيبليوغرافيا). ويحدد بواسطة جزء صحيح، نقطة عشرية وجزء وظيفة. ويتألف كل من جزأي الصحيح والوظيفة من تتابع أرقام عشرية (قاعدة 10). ويخزن عدد الأرقام في "digits" ويرد حجم الجزء النسبي في "scale". وتخزن الأرقام نفسها في "value_". ويكون ترميز قيمة لهذا النمط هو نفسه لترميز قيمة نمط السجل. وتُشفر قيم هذا النمط ويُفكك تشفيرها كما لقيم عشرية لنقطة ثابتة IDL.

ملاحظة – يعتمد التشفير الدقيق لقيم هذا النمط على قواعد التشفير الفعلية المستخدمة. وتفاصيل قواعد التشفير هي خارج مدى هذه التوصية.

وتعريف النمط لهذا النمط هو:

```
type record IDLfixed {
              unsignedshort  digits,
              short    scale,
              charstring   value_
               }
            with { variant "IDL:fixed FORMAL/01-12-01 v.2.6" };
```

4.2.E أنماط مفيدة لسلسلة ذرية

1.4.2.E نمط لسلسة وحيدة IRV

إنه نمط تكون قيمته المميزة سمات وحيدة لصيغة التوصية ITU-T T.50 [9] تمتثل لصيغة مرجعية دولية (IRV) كما حددت في [9] 8.2/T.50 (انظر أيضاً الملاحظة 2 من 1.1.6).

وتعريف النمط لهذا النمط هو:

```
type charstring char length (1);
```

الملاحظة 1 – إن اسم هذا النمط المفيد هو نفسه للكلمة المفتاحية لـ TTCN-3 التي تدل على قيم **universal charstring** في شكل تربيعي. وبشكل عام، لا يسمح باستخدام كلمات مفتاحية لـ TTCN-3 كمعرفات. والنمط المفيد "char" هو استثناء وحيد ويسمح به فقط لمواءمة للخلف مع صيغ سابقة لمعيار TTCN-3.

الملاحظة 2 – يمكن استخدام السلسلة الخاصة "8 bit" المعرفة في 3.2.28 مع هذا النمط لتحديد تشفير ما لقيمه. وأيضاً، يمكن تغيير خواص نمط القاعدة بواسطة استخدام آليات نعوت.

2.4.2.E نمط سمة عالمية وحيدة

إنه نمط تكون قيمته المميزة سمات وحيدة من [10] ISO/IEC 10646.

وتعريف النمط لهذا النمط هو:

```
type universal charstring uchar length (1);
```

ملاحظة – يمكن استخدام السلسلة الخاصة المعرفة في 3.2.28، باستثناء "8 bit"، مع هذا النمط لتحديد تشفير ما لقيمه. وأيضاً، يمكن تغيير خواص نمط القاعدة بواسطة استخدام آليات نعوت.

3.4.2.E نمط بتات وحيدة

إنه نمط تكون قيمته المميزة أرقام اثنينية وحيدة.

وتعريف النمط لهذا النمط هو:

```
type bitstring bit length (1);
```

4.4.2.E نمط ستة عشري وحيد

إنه نمط تكون قيمته المميزة أرقام ستة عشرية وحيدة.

وتعريف النمط لهذا النمط هو:

```
type hexstring hex length (1);
```

5.4.2.E نمط أثمون وحيد

إنه نمط تكون قيمته المميزة أزواج أرقام ستة عشرية.

وتعريف النمط لهذا النمط هو:

```
type octetstring octet length (1);
```

الملحق F (للعلم)

عمليات على أشياء نشيطة TTCN-3

1.F عـــام

يصف هذا الملحق في شكل مختصر علم دلالات عمليات على أشياء نشيطة في TTCN-3، وتكون هذه مكونات اختبار ومؤقتات ومنافذ. ويكتب هذا السلوك الدينامي في شكل حالة أوتوماتية مع:

- حالات تُسمى وتُعرّف على أنها عقد؛
- الحالة الأولية تُعرّف بواسطة سهم واصل؛
- انتقال بين توصيل حالات لحالتين (ليس بالضرورة حالات مختلفة) معرفتان بأسهم؛
- انتقال يجري وسمه مع شرط تمكين لذلك الانتقال (أي، نداءات عملية أو بيان) والشرط الناتج (مثلاً خطأ اختبار مجرد) يفصل بينهما '/':
  - إن نداءات عمليات وبيانات هي عمليات وبيانات TTCN-3 المطبقة على الشيء (مكتوبة بخط أسود)؛
  - يعني خطأ كشرط ناتج خطأ اختبار مجرد (مكتوب بخط أسود)؛
  - يعني خمود كشرط ناتج أن استثناء لتغيير حالة ممكن، ولا تنطبق نتائج أخرى (مكتوب بخط أسود)؛
  - تشير مواءمة/لا مواءمة إلى نتيجة مواءمة لانتقال (مكتوبة بخط أسود)؛
  - قيم ملموسة هي نتائج بولاني أو طليق (مكتوبة بخط مائل)؛
  - توصف جميع الشروط الناتجة الأخرى (مكتوبة بخط معياري)؛

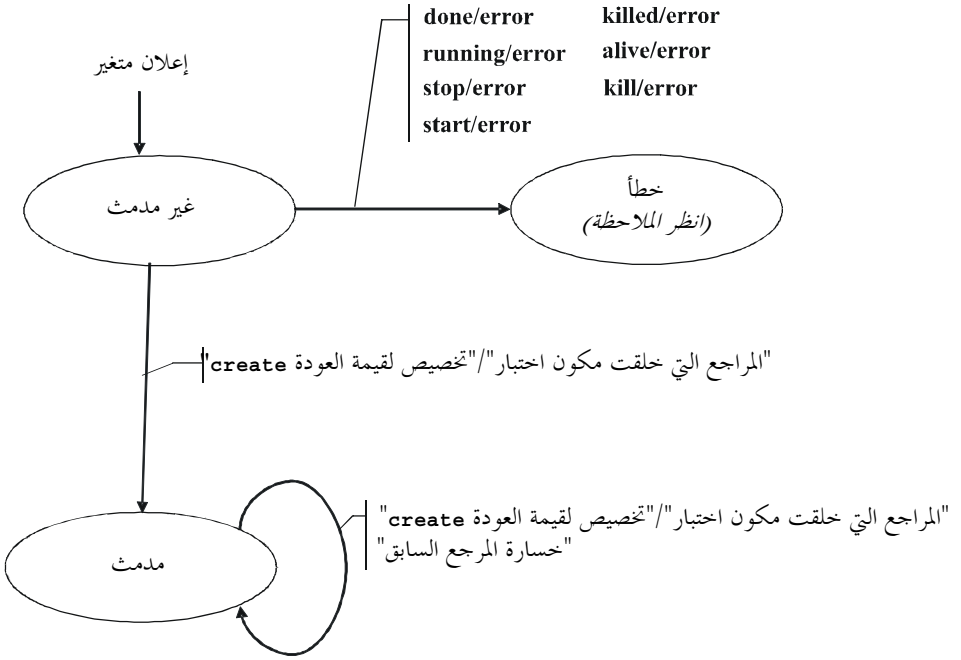
- تستخدم الملاحظات لشرح تفاصيل إضافية للحالة الأوتوماتية.

ولمزيد من التفاصيل، رجاء الرجوع إلى علم الدلالات التشغيلي لـ TTCN-3. وفي حالة أي تعارض بين هذا الملحق وعلم الدلالات التشغيلي لـ TTCN-3، يكون للأخير الأسبقية.

2.F مكونات اختبار

1.2.F مراجع مكون اختبار

تستخدم متغيرات أنماط مكون اختبار وعمليات **self** و**mtc** كمرجع مكونات اختبار. ولا تنطبق عمليات **start** و**stop** و**done** و**running** مباشرة على مكونات اختبار ولكن على مراجع مكون. ويقرر نظام الاختبار إذا كانت العملية المطلوبة تؤثر على شيء المكون نفسه أو إذا كان إجراء آخر ملائماً (مثل، حدوث خطأ عندما يستخدم مرجع stopped PTC في عملية بداية مكون). وتستخدم عملية **create** لخلق مكونات PTCs تعيد مرجع وحيد لمكون PTC قد خلق، وهو موثق نمطياً لمتغير مكون اختبار. ويرد في الشكل 1.F السلوك المتعلق بمتغيرات مكون اختبار نفسها.

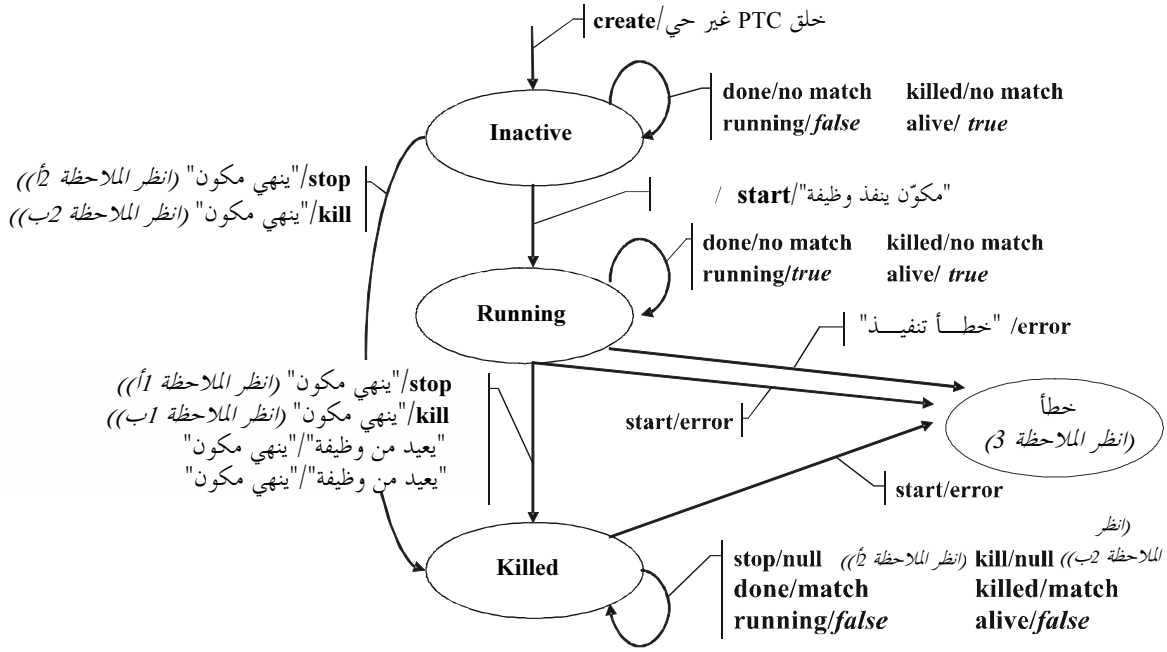


ملاحظة – عندما يدخل مكون اختبار حالة خطئه، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد وتكون النتيجة الشاملة للاختبار المجرد خطأ.

الشكل Z.140/1.F – مناولة مراجع مكون اختبار

2.2.F السلوك الدينامي لـ PTCs

يمكن PTCs أن يكون نمط غير حي أو نمط حي. ويمكن أن يكون نمط غير حي PTC في حالات Inactive وRunning وKilled. ويرد سلوكها الدينامي في الشكل 2.F.



الملاحظة 1 – أ) Stop يمكن أن يكون إما stop أو self.stop أو stop من مكون اختبار آخر؛

ب) Kill يمكن أن يكون إما kill، self.kill، kill من مكون اختبار آخر أو kill من نظام اختبار (في حالات الخطأ).

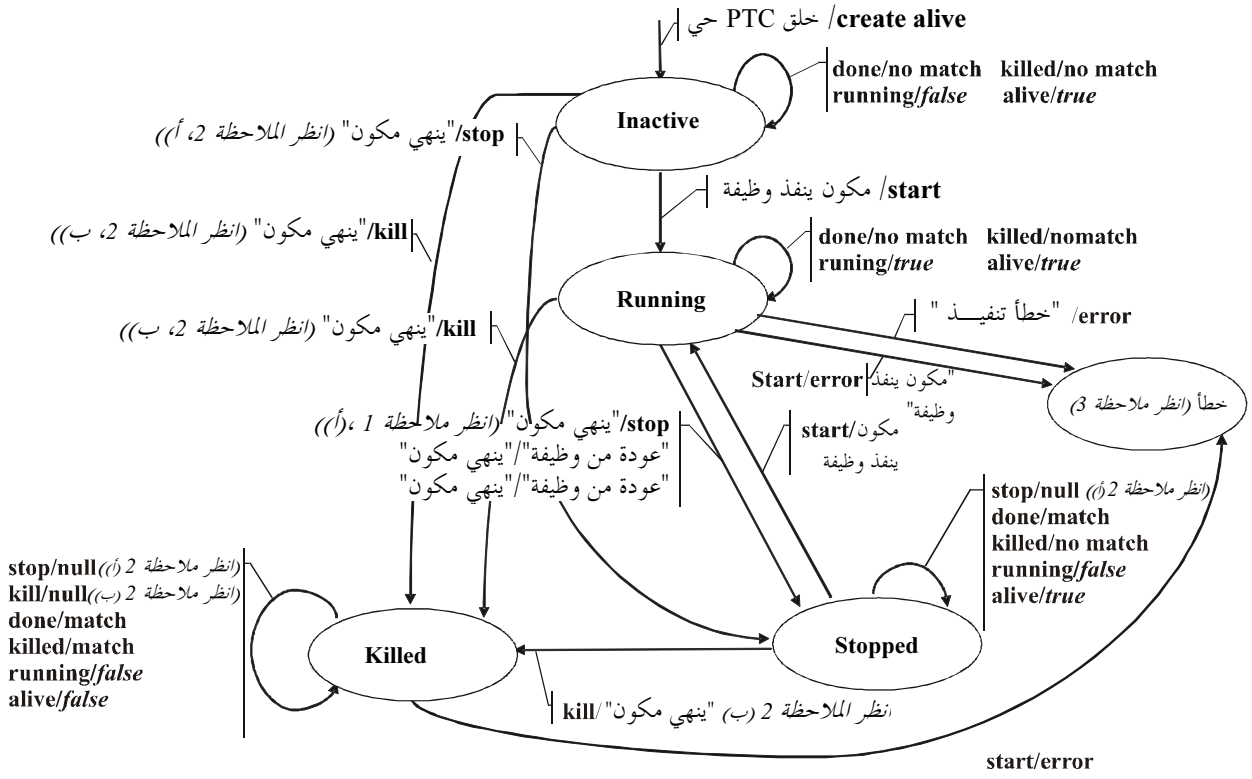
الملاحظة 2 – أ) Stop يمكن أن يكون إما من مكون اختبار آخر فقط؛

ب) Kill يمكن أن يكون إما stop أو self.kill أو kill من مكون اختبار آخر أو kill من نظام اختبار (في حالات الخطأ) فقط.

الملاحظة 3 – عندما يدخل مكون اختبار حالة خطئه، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد وتكون النتيجة الشاملة للاختبار المجرد خطأ.

الشكل Z.140/2.F – السلوك الدينامي لنمط غير حي PTCs

يمكن PTCs لنمط حي أن يكون في حالات Inactive، Running، Stopped وKilled. ويرد سلوكها الدينامي في الشكل 3.F.



الملاحظة 1 – أ) Stop يمكن أن يكون إما stop أو self.stop أو stop من مكون اختبار آخر؛

ب) Kill يمكن أن يكون إما kill، self.kill، kill من مكون اختبار آخر أو kill من نظام اختبار (في حالات الخطأ).

الملاحظة 2 – أ) Stop يمكن أن يكون إما من مكون اختبار آخر فقط؛

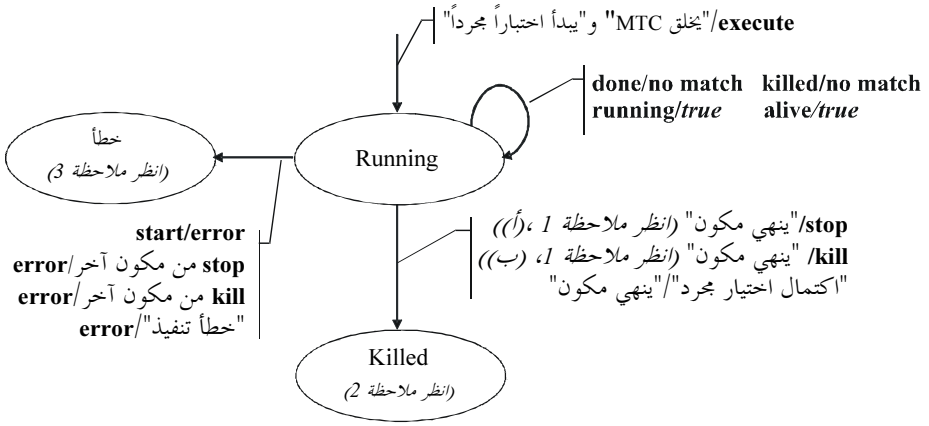
ب) Kill يمكن أن يكون إما stop أو self.kill أو kill من مكون اختبار آخر أو kill من نظام اختبار (في حالات الخطأ) فقط.

الملاحظة 3 – عندما يدخل مكون اختبار حالة خطئه، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد وتكون النتيجة الشاملة للاختبار المجرد خطأ.

الشكل Z.140/3.F – السلوك الدينامي لنمط حي PTCs

3.2.F السلوك الدينامي لـ MTC

يمكن MTC أن يكون في حالة Running أو Killed. ويرد السلوك الدينامي لـ MTC في الشكل 4.F.

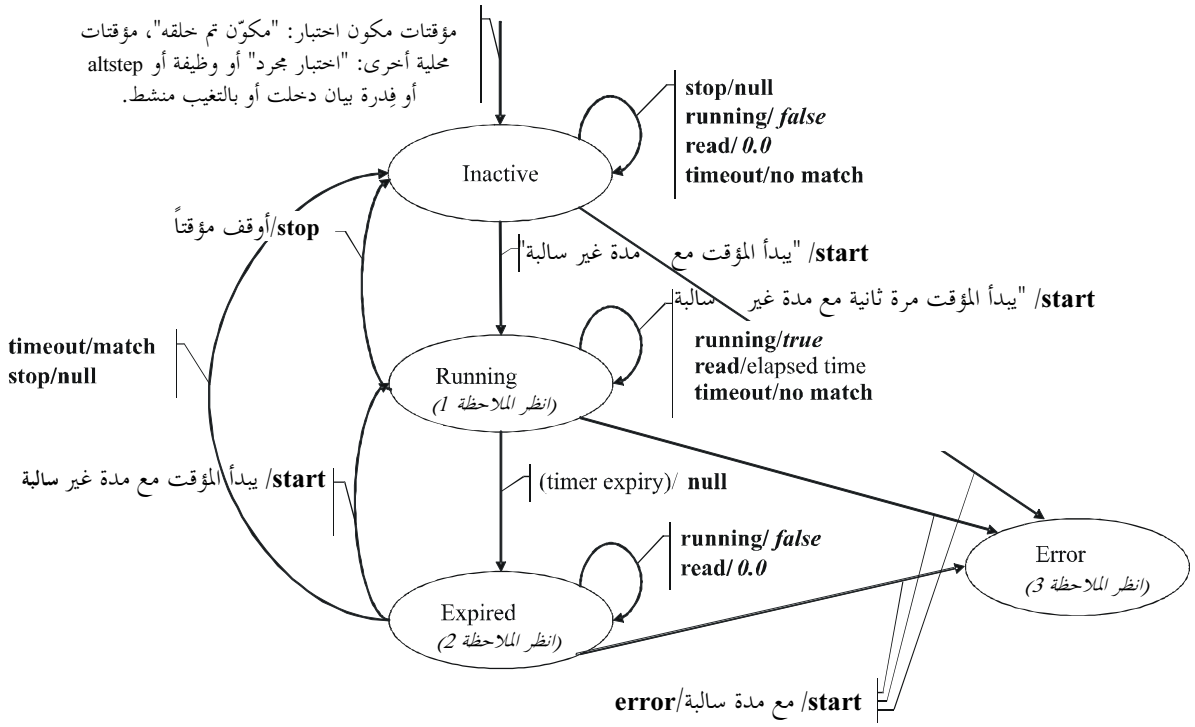


الملاحظة 1 – أ) Stop يمكن أن يكون إما stop أو self.stop أو stop من مكون اختبار آخر؛
ب) Kill يمكن أن يكون إما kill، self.kill، kill من مكون اختبار آخر أو kill من نظام اختبار (في حالات الخطأ).
الملاحظة 2 – تكون جميع PTCs الباقية killed وينتهي الاختبار المجرد.
الملاحظة 3 – عندما يدخل MTC حالة خطئه، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد خطأ.

الشكل Z.140/4.F – السلوك الدينامي لـ MTC

3.F المؤقتات

يمكن أن تكون المؤقتات في حالة Inactive أو Running أو Expired. ويرد السلوك الدينامي لمؤقت في الشكل 5.F.



الملاحظة 1 – لأي وحدة منظور، تتألف جميع المؤقتات في ذلك المنظور التي في حالة Running من قائمة مؤقت تنفيذ.
الملاحظة 2 – لأي وحدة منظور، تتألف جميع المؤقتات في ذلك المنظور التي في حالة Expired من قائمة إمهال.
الملاحظة 3– عندما يدخل مؤقت حالة خطئه، ومكون الاختبار التابع له يدخل أيضا في حالة خطئه، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد وتكون النتيجة الشاملة للاختبار المجرد خطأ.

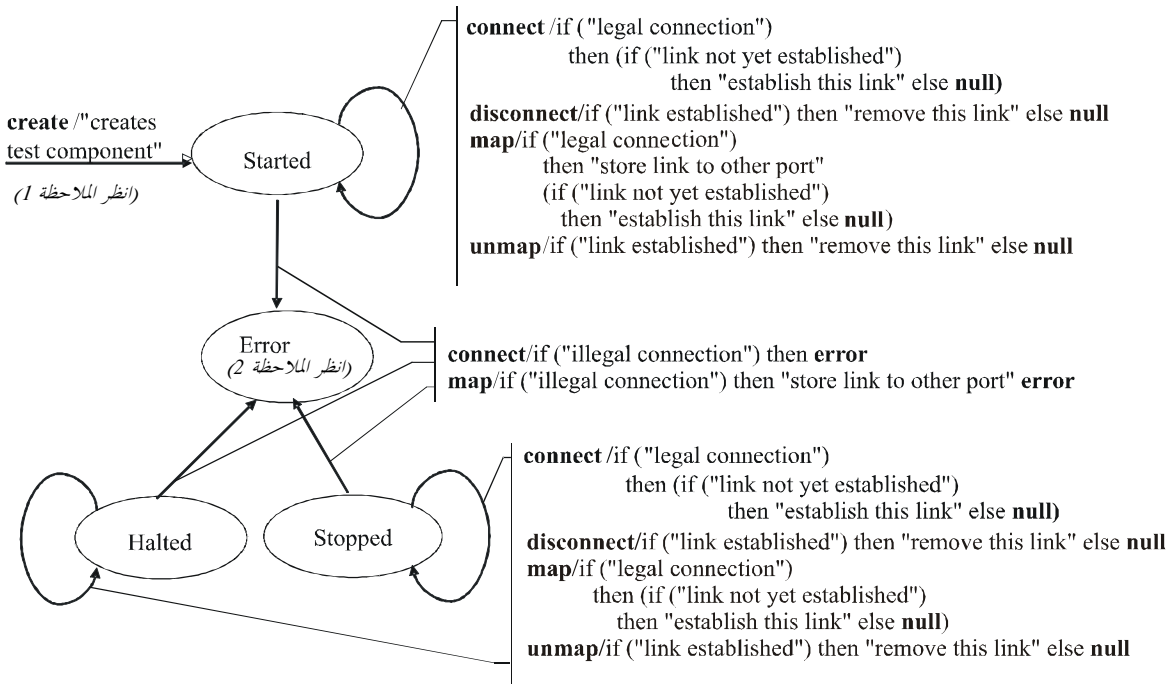
الشكل Z.140/5.F – السلوك الدينامي لمؤقتات

4.F منافذ

يمكن أن تكون منافذ في حالة Started أو Stopped. ولأن سلوكها معقد، تقسم الحالة الأوتوماتية إلى حالة أوتوماتية توفر سلوكا دينامياً لعمليات تشكيل (أي، توصيل فك توصيل وتقابل وفك تقابل) لعمليات تحكم منفذ (أي، start وstop وclear) وعمليات اتصالات (أي، send وrecieve وcall وgetcall وraise وcatch وreply وgetreply وcheck). ولأن الدافع هو لاختزال alt مع استقبال، لا ينظر فيه هنا.

1.4.F عمليات تشكيل

إن عمليات تشكيل منفذ (أي، توصيل فك توصيل وتقابل وفك تقابل connect وdisconnect وmap وunmap) لا تهتم بحالة المنفذ. ويبين الشكل 6.F سلوكها.



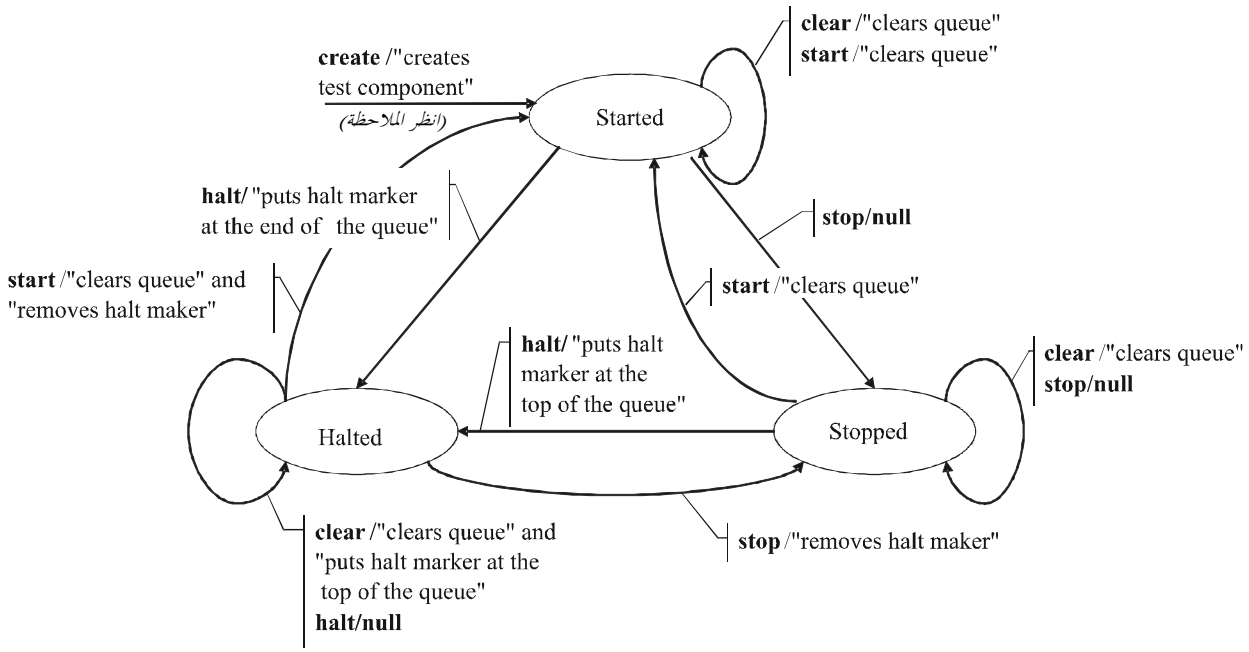
الملاحظة 1 – عند خلق PTC، يتم خلق منافذ PTC وتبدأ؛ وعند خلق MTC، يتم خلق منافذ MTC ومنافذ TSI وتبدأ.
الملاحظة 2 – عندما يدخل منفذ، تخلق منافذ ذلك وتبدأ؛ وعند خلق MTC، تخلق منافذ MTC ومنافذ TSI وتبدأ.

الشكل Z.140/6.F – السلوك الديناميي لمنافذ: عمليات تشكيل منفذ

لا يغير الانتقال الحالة الرئيسية لمنفذ، أي، يظل المنفذ في حالة Started أو Stopped.

2.4.F عمليات تحكم منفذ

يرد في الشكل 7.F نتائج عمليات تحكم منفذ.

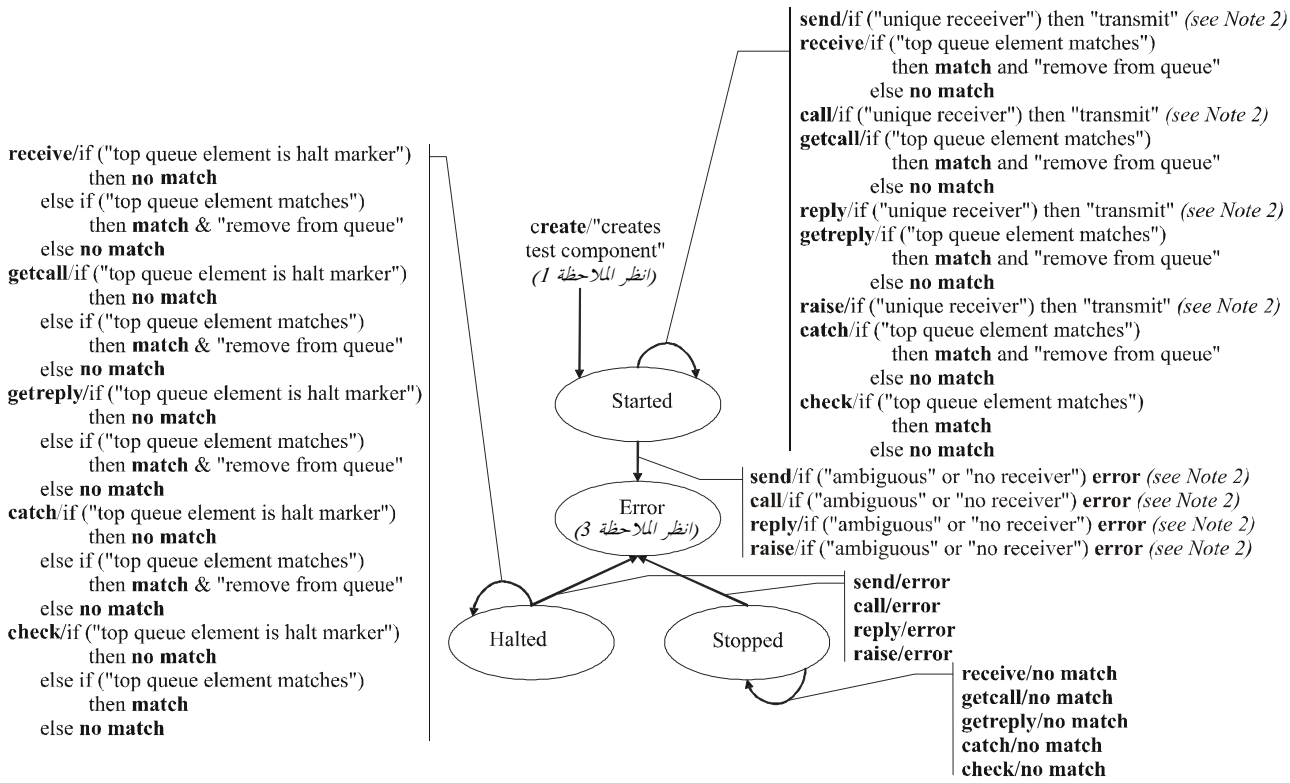


ملاحظة – عند خلق PTC، تخلق منافذ ذلك PTC وتبدأ؛ وعند خلق MTC، تخلق منافذ MTC ومنافذ TSI وتبدأ.

الشكل Z.140/7.F – السلوك الدينامي لمنافذ: عمليات تحكم منفذ

3.4.F عمليات اتصالات

يرد في الشكل 8.F نتائج عمليات اتصالات send وreceive وcall وgetcall وraise وcatch وreply وgetreply وgetreply.



الملاحظة 1 – عند خلق PTC، تخلق منافذ ذلك PTC وتبدأ؛ وعند خلق MTC، تخلق منافذ MTC ومنافذ TSI وتبدأ.

الملاحظة 2 – يوجد مستقبل وحيد إذا كانت هناك وصلة واحدة لهذا المنفذ، أو إذا كان تعبير عنوان to يشير إلى مكون اختبار يتصل منفذه بهذا المنفذ (إن مكون اختبار منتهي لا يعتبر مستقبلاً قانونياً).

الملاحظة 3 – عندما يدخل منفذ حالة خطئه، يدخل مكون الاختبار الذي يخصه أيضا حالة خطأ، يخصص حكم الخطأ لحكمه المحلي وينتهي الاختبار المجرد وتكون النتيجة الشاملة للاختبار المجرد خطأ.

الملاحظة 4 – لأن الدافع هو اختزال لـ alt مع مستقبل، لا ينظر فيه هنا.

الشكل Z.140/8.F – السلوك الديناميي لمنافذ: عمليات اتصالات

الملحق G (للعلم)

خاصيات لغة لا يُنصح بها

1.G تعريف أسلوب زمرة لمعلمات وحدة

تطلّبت الصيغة السابقة لهذه التوصية استخدام قواعد تركيب مشابهة للزمرة المبينة في المثال أدناه للإعلان عن معلمات وحدة. وتم توحيد قواعد تركيب معلمة وحدة مع قواعد تركيب إعلان ثابت ومتغير في هذه الصيغة، ولا تزال قواعد التركيب المشابهة لزمرة بالكامل لترك فترة زمنية لموفري ومستعملي الأدوات للتغيير من قواعد التركيب القديمة إلى الجديدة. ومن المخطط إزالة قواعد التركيب المشابهة لزمرة لإعلانات معلمة وحدة بالكامل في الطبعة القادمة من هذه التوصية.

مثـــال:

```
module MyModuleWithParameters
{
   modulepar { integer TS_Par0, TS_Par1 := 0;
               boolean TS_Par2 := true
             };
   modulepar { hexstring TS_Par3 };
}
```

2.G استيراد متكرر

سمحت الصيغة السابقة لهذه التوصية باستيراد تعاريف مسماة ضمنية، عبر استيراد تعاريف لنفس الوحدة باستخدامها أسلوب متكرر. وهذه الخاصية لا يُنصح بها في هذه الصيغة، ومن المخطط إزالتها بالكامل في الصيغة القادمة التي تنشر.

3.G استخدام `all` في تعاريف نمط منفذ

سمحت الصيغة السابقة لهذه التوصية باستخدام الكلمة المفتاحية **`all`** في تعاريف نمط منفذ بدلاً من قائمة واضحة لأنماط وتوقيعات مسموح بها عبر منفذ ما. وهذه الخاصية لا يُنصح بها في هذه الصيغة ومن المخطط إزالتها بالكامل في الصيغة القادمة التي تنشر.

بيبليوغرافيا

- ETSI ES 201 873-1 V1.1.2 (2001-06), *Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation version 3; Part 1: TTCN-3 Core Language*.
- ETSI ES 201 873-1 V2.2.1 (2003-02), *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*.
- ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*.
- Object Management Group (OMG): *The Common Object Request Broker: Architecture and Specification*, Chapter 3 – IDL Syntax and Semantics. Version 2.6, FORMAL/01-12-01, December 2001.
- IEEE 754 (1985), *Binary Floating-Point Arithmetic*.

سلاسل التوصيات الصادرة عن قطاع تقييس الاتصالات

السلسلة A	تنظيم العمل في قطاع تقييس الاتصالات
السلسلة D	المبادئ العامة للتعريفة
السلسلة E	التشغيل العام للشبكة والخدمة الهاتفية وتشغيل الخدمات والعوامل البشرية
السلسلة F	خدمات الاتصالات غير الهاتفية
السلسلة G	أنظمة الإرسال ووسائطه والأنظمة والشبكات الرقمية
السلسلة H	الأنظمة السمعية المرئية والأنظمة متعددة الوسائط
السلسلة I	الشبكة الرقمية متكاملة الخدمات
السلسلة J	الشبكات الكبلية وإرسال إشارات تلفزيونية وبرامج صوتية وإشارات أخرى متعددة الوسائط
السلسلة K	الحماية من التداخلات
السلسلة L	إنشاء الكبلات وغيرها من عناصر المنشآت الخارجية وتركيبها وحمايتها
السلسلة M	إدارة الاتصالات بما في ذلك شبكة إدارة الاتصالات (TMN) وصيانة الشبكات
السلسلة N	الصيانة: الدارات الدولية لإرسال البرامج الإذاعية الصوتية والتلفزيونية
السلسلة O	مواصفات تجهيزات القياس
السلسلة P	نوعية الإرسال الهاتفي والمنشآت الهاتفية وشبكات الخطوط المحلية
السلسلة Q	التبديل والتشوير
السلسلة R	الإرسال البرقي
السلسلة S	التجهيزات المطرافية للخدمات البرقية
السلسلة T	المطاريف الخاصة بالخدمات التلماتية
السلسلة U	التبديل البرقي
السلسلة V	اتصالات المعطيات على الشبكة الهاتفية
السلسلة X	شبكات المعطيات والاتصالات بين الأنظمة المفتوحة ومسائل الأمن
السلسلة Y	البنية التحتية العالمية للمعلومات وملامح بروتوكول الإنترنت وشبكات الجيل التالي
السلسلة Z	**اللغات والجوانب العامة للبرمجيات في أنظمة الاتصالات**

طُبعت في سويسرا
جنيف، 2010