TECHNICAL REPORT

ISO/IEC TR 10183-2

First edition
1993-12-15

Information technology — Text and office systems — Office Document Architecture (ODA) and interchange format — Technical Report on ISO 8613 implementation testing —

Part 2:
Framework for abstract test cases

Technologies de l'information — Bureautique — Architecture de documents de bureau (ODA) et format d'échange — Rapport technique sur la mise en application de tests de l'ISO 8613 —

Partie 2: Cadre général pour cas de tests abstraits

Reference number
ISO/IEC TR 10183-2:1993(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 10183-2, which is a Technical Report of type 3, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Sub-Committee SC 18, *Document processing and related communication*.

ISO/IEC TR 10183 consists of the following parts, under the general title *Information technology — Text and office systems — Office Document Architecture (ODA) and interchange format — Technical Report on ISO 8613 implementation testing*:

— *Part 1: Testing methodology*

— *Part 2: Framework for abstract test cases*

Information technology — Text and office systems — Office Document Architecture (ODA) and interchange format — Technical Report on ISO 8613 implementation testing —

Part 2:
Framework for abstract test cases

1 Scope

The purpose of ISO/IEC TR 10183 is to define a testing methodology and provide a framework for specifying abstract test cases for ISO 8613 implementation testing, the overall objective being the provision of a suitable base for testing the *interworking* capability of ODA implementations.

Such testing will assist in the analysis of an implementation's ability to interwork in an environment of implementations of ISO 8613 and implementations of International Standardized Profiles (ISPs) based on ISO 8613. ISPs are standardized document application profiles that have been internationally harmonized. As such they represent agreed stabilized subsets of ISO 8613 designed for the interworking of ODA systems at different levels of functionality.

In ISO 8613 the term "conformance" refers to the conformance of a data stream to the rules specified in ISO 8613. This includes the conformance of a data stream to a document application profile based on ISO 8613. Conformance testing methodology as defined in Annex G of ISO 8613-1 covers the analysis of data streams without regard to the capabilities of implementations to generate or receive conforming data streams. To achieve an environment of interworking ODA systems, it is necessary to have a testing methodology that can verify implementation support for ISO 8613 and DAP's at the semantic level as well as the data stream or syntax level.

Hence, implementation testing is additional testing that supplements the conformance testing of data streams. It increases the probability that different implementations of ISO 8613 and ISPs are able to interwork. Implementation testing is based on the concept of measuring an implementation's ability to generate and/or receive a representative set of documents. If an implementation can exhibit this capability, then it is more likely to interwork successfully with other verified implementations exchanging a wider range of documents.

The implementation testing methodology introduces the requirement for abstract test cases as well as procedures for their use in the generation and reception testing of implementations.

In establishing a framework for implementation testing, a conceptual model of ODA systems has been developed to describe, in an abstract sense, the multitude of configurations and limitations of real ODA systems.

The methodology contained in ISO/IEC TR 10183 caters for the testing of implementations of ISP's based on ISO 8613. The methodology may also be used for testing other Document Application Profiles based on ISO 8613.

This part of ISO TR 10183

- specifies a framework for the development of abstract test cases;
- specifies a test case notation used to specify abstract test cases;
- gives examples of abstract test cases.

ISO/IEC TR 10183 does not cover the testing of user interfaces in an ODA based system. Any suitable system interface

is only used as a point of control and observation to verify that ODA document transformations associated with the various ODA processes have been carried out as claimed by an implementor.

Abstract test cases created from the specifications in this part of ISO/IEC TR 10183 should be used in the definition of executable test suites and data streams for testing implementations of ISO 8613 and implementations of International Standardized Profiles based on ISO 8613.

Where appropriate, concepts and terminology described in ISO/IEC 9646 have been used. In some cases, definitions and concepts have been adapted to cater for the fact that ODA is not an OSI protocol.

2 References

The following documents are referenced within ISO/IEC TR 10183 and provide additional background information.

ISO/IEC 646:1991, *Information technology - ISO 7-bit coded character set for information interchange (3rd edition).*

ISO 2022:1986, *Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques.*

ISO/IEC 6429:1992, *Information technology - Control functions for coded character sets.*

ISO 8613:1989, *Information processing - Text and office systems - Office Document Architecture (ODA) and interchange format.*

ISO/IEC 9646-1:1991, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts.*

ISO/IEC TR 10183-1:1993, *Information technology - Text and office systems - Office Document Architecture (ODA) and Interchange Format - Technical Report on ISO 8613 implementation testing - Part 1: Testing methodology.*

ISO/IEC ISP 10610-1:1993, *Information technology - International Standardized Profile FOD11 - Open Document Format - Simple document structure - Character content architecture only - Part 1: Document Application Profile (DAP).*

ISO/IEC ISP 11181-1:1993, *Information technology - International Standardized Profile FOD26 - Open Document Format - Enhanced document structure - Character, raster graphics and geometric graphics content architectures - Part 1: Document Application Profile (DAP).*

ISO/IEC ISP 11182-1:1993, *Information technology - International Standardized Profile FOD36 - Open Document Format - Extended document structure - Character, raster graphics and geometric graphics content architectures - Part 1: Document Application Profile (DAP).*

3 Definitions

The definitions of ISO 8613 apply to ISO/IEC TR 10183. Some terms from ISO 9646 are also used where there is an equivalence of meaning. Additionally, the definitions given in ISO/IEC TR 10183-1 also apply.

4 Abbreviations

The abbreviations given in ISO/IEC TR 10183-1 apply. Additionally, the following abbreviations are used.

DAP: Document Application Profile.

ISP: International Standardized Profile.

IUT: Implementation Under Test.

PCO: Point of Control and Observation.

Pc: Process component.

Sc: System interface component.

Ic: Interchange component.

GSS: Generating Support Statement.

RSS: Receiving Support Statement.

DUT: Document Under Test

TCS: Test Case Specification (used in generation testing)

TDS: Test Data Stream (used in reception testing)

FDAR: A local representation of a formatted form (FDA) ODA document

PDAR: A local representation of a processable form (PDA) ODA document

FPDAR: A local representation of a formatted processable form (FPDA) ODA document

ATCN: Abstract Test Case Notation

TCID: Test case Identifier

TCP: Test Case Purpose

5 Framework for the development of abstract test cases

This clause presents the framework for the development of abstract test cases for generation and reception testing. The structure of abstract test cases is described in 5.1 and the design of generation and reception test cases is described in 5.2. Examples of test cases are contained in Annexes A and B.

5.1 Structure of abstract test cases

Every abstract test case has a Test Case Identifier (TCID), a Test Case Purpose (TCP) and a test case body. The TCP is a description of the objective that the test case is designed to achieve. The test case body contains a specification, written in the Abstract Test Case Notation (ATCN), for a set of ODA functional elements that are necessary to achieve the TCP and assign verdicts to the possible outcomes. In the case of reception testing, the test case body may also contain a test case script that can be used to aid analysis of a document and help in determination of the test case verdict.

A Document Under Test (DUT), used in generation testing, or a Test Data Stream (TDS), used in reception testing, may contain one or more test cases.

The requirements for ODA test cases are that they be configurable to different DAPs and different Implementations under Test (IUTs). This can be achieved by having test cases which are feature based and which contain the minimal amount of DAP dependent information. These constraints impact on the structure of the test case specification.

The test case specification part of the test case body consists of one or more constituent specifications for one or more ODA structures. The test case specification contains only the information necessary to express the purpose of the test case. One or more test case specifications do not necessarily represent a complete document specification since not all constituents, attributes and values are present. However, any DUT or TDS resulting from the compilation of one or more abstract test cases must be a conforming ODA document.

5.1.1 Relationships between constituents in a test case

The constituent specifications in a test case result in constituents that are connected in a less constraining fashion than in ODA. Two types of relationship describe the possible connections: the hierarchical relationship and the logical-layout relationship.

The *hierarchical relationship* is used to identify particular descendants of a constituent. It allows the definition of a hierarchy of constituents without having to specify all the constituents in a document structure. A constituent's descendant can be defined as being either a subordinate or an immediate subordinate. The order of appearance of descendants in relation to the sequential order can also be specified by the hierarchical relationship.

The *logical-layout relationship* is used to constrain a particular logical constituent and its subordinates to be laid out in a particular layout constituent and its subordinates. This type of relationship allows the guidance of the layout of the logical objects defined in the test case specification without having to specify layout directives which are DAP dependent or a complete document.

5.1.2 Preconditions

There is some ambiguity in allowing missing attributes and attribute values in the test case. This ambiguity occurs when a DUT or TDS is derived for the test case and can result in attributes and attribute values being in conflict with the purpose of the test case. It is necessary to prohibit the occurrence of such attributes and attribute values in any DUT or TDS containing the test case. This can be achieved using preconditions. A *precondition* is a constraint on the presence and values of attributes of particular constituents in any TDS derived for the test case. The constraint can apply to the subordinates of a particular constituent specified in the test case, to the objects which are superior (at any hierarchical level) to a particular constituent specified in the test case, or to the constituents hierarchically situated between two particular constituents belonging to the same ODA structure in the test case.

5.1.3 Attribute value range

In a test case, attribute values must be expressed in a way that allows them to be as unconstrained as possible so that they can be configurable to different DAPs. Attribute values can be specified in an unconstrained way by providing a range of possible values instead of a single value (provided more than one value is possible in the test case). The specification of a range of values in the test case is done in a similar way to the specification of DAPs using the Document Application Profile Proforma Notation in Annex F of ISO 8613-1.

5.1.4 Constituent names

It is often necessary to restrict a constituent specification in a test case to be of a particular type. Normally, in ODA, the type of a constituent is determined by a combination of attributes. In order to facilitate the specification of such a

constraint in the test case, a constraint, called a 'constituent type constraint', is introduced. This type of constraint allows the type of a constituent to be constrained without having to set particular attribute values (which might restrict the test case being configurable to different DAPs).

DAPs assign semantics to the constituents of the possible documents allowed. The constituents represent the allowed structural elements such as objects and object classes in a document for instance paragraphs, sections and columns, each having allowed properties such as indentation, numbering and size. It is also necessary to assign similar semantics to the constituents in a test case. This can be achieved by similarly associating the constituents, specified in a test case, with names identifying their basic functionality (e.g. paragraphs, sections and columns). Since the test cases need to be DAP configurable, the constituent names should represent general elements applicable to different DAPs.

5.1.5 Constraints

It is necessary to ensure that test cases provide a predictable outcome. When implementation testing involves the analysis of a view transformation, the test cases also need to consider an predictable result in the analysis of, for example, the imaged DUT containing the test case. Three types of constraint are necessary to achieve such predictability:

- constraints on the visibility of a layout object;
- constraints on the size of content and
- constraints on the available area of a layout object.

These constraints cannot be realized without knowledge of the DAP and must be expressed in a generic manner to allow the test cases to be DAP configurable. These three constraints are described in more detail in the following clauses.

5.1.5.1 Constraints on visibility

The ***visibility*** of layout objects is necessary when testing certain features such as 'New Pageset'. Visibility is necessary to ensure that particular content is laid out in particular layout objects. For example, in the case of the layout directive 'New Pageset' applied to some logical objects, it will be necessary to confirm that the content associated with one or more basic logical objects has been laid out in a new pageset. In order to verify this, it is necessary to observe the occurrence of the new pageset in the DUT. The way a layout object can be made visible may vary from DAP to DAP. In test cases, layout objects are designated visible by assigning the characteristic 'VISIBLE' to their constituent. This allows the specification of the visibility constraint in a generic manner and defers the decision on how to realize the constraint to a later phase when the DAP information is used.

5.1.5.2 Constraints on content

The testing of certain features requires the use of content information of specific sizes. In some cases, the content is used to view the dimensions of a layout object in the DUT. In other cases, the content is used to assess the layout process for a given feature in a particular context. The context must be such that the occurrence of the feature produces a predictable effect in the DUT. For example, consider the situation when a TDS for a processable document is being used to test the layout support of a receiving IUT through an imaged DUT. And consider that the TDS contains a test case for the feature that requires two logical objects to be contained in the same layout object. This feature could be observed in an imaged DUT if the processable TDS was such that a new page would occur when the particular logical objects were to be laid out. The recipient's layout process could be tested in such a context by defining three basic logical objects in the test case, the last two being those for which 'Same Layout Object' applies. The content associated with the basic logical objects must be designed in such a way that they do not fit together in the body of one page. It is not possible to achieve this using fixed pieces of content since the available area in the page body, which is DAP dependent, is not known when the generic test case is designed. It is therefore necessary to bind the relative size of the content portions associated with the basic logical objects to the available area of the page body and leave the generation

of the actual pieces of content and hence their size to a later stage when the available area is determined. This type of content is termed *relative content*. In the example, the desired effect can be obtained using relative content by defining three basic logical objects with content occupying respectively one third, one third and one half of the page body's available area. The constraint on the size of the content associated with the basic logical objects ought to force the layout process to generate a new page since the three pieces of content together exceed the page body's available area.

5.1.5.3 Constraints on available area

It is necessary to have a mechanism that effectively determines the size of the available area within particular layout objects. Such control is required in order to ensure that content associated with particular basic logical objects can be laid out in layout objects which were designed, in a given test case, to receive them. It is also necessary for test cases with raster or geometric graphics content to make sure that the available area of the associated layout object is of sufficient size to allow observation of the visible content. In the example in 5.1.5.2, where it was required to ensure that some logical content could be laid out in a layout object, there was no need to define constraints on the available area of the layout object since the relative content could be used. However, there are situations where it is more appropriate to use fixed pieces of content. The use of fixed content allows the incorporation of readable text in imaged DUTs. The readable text can be used as a guide during analysis and determination of the verdict of the test case. When fixed content is specified, there is no link between the size of the content and the size of the receiving layout object's available area, and therefore no guarantee that the content can be laid out in the layout object. It is therefore necessary to include a restriction on the layout object's available area to ensure that the content can actually be laid out in the object. A constraint on the available area of a layout object in a test case specification consists of a minimum and/or maximum size for the horizontal and/or vertical directions of the area.

5.2 Design of abstract test cases

The design of test cases for generation testing and for reception testing is different since the former must create a document specification language which is understandable by an implementor/operator of the IUTs proprietary system. Reception testing must cater for the analysis of DUTs to determine the verdict of test cases. 5.2.1 and 5.2.2 describe the design of generation and reception test cases.

5.2.1 Generation test case design

In generation testing, each test case includes a feature name and a feature description. The feature name is expressed in document processing terms (e.g paragraph, footnote, column) used in clause 6 of DAPs defined in accordance with the Document Application Profile Proforma Notation in ISO 8613-1. The feature description expresses the feature in the abstract test case notation (see clause 7) using ODA constituents and attribute specifications. In the case where a feature can be described using different combinations of ODA functional elements, a set of descriptions can be associated with the feature.

Examples of feature descriptions are contained in Annex A. The examples include test case feature descriptions for 'Paragraph with text and graphic' (A.2) and 'Page with header or footer frame' (A.3).

5.2.2 Reception Test Case Design

A reception test case can be viewed as a constraint on a TDS. The notation used to express the constraint is similar to the one used for generation test cases. However, some functionality is added to the notation to cater for the specific requirements of reception testing.

The reception test cases can be divided in three categories:

- Test cases which check that the IUT can receive and interpret the semantics of attributes and attribute values correctly. The semantics of attributes are specified in ISO 8613;

- Test cases which check that the IUT can receive and correctly process combinations of attributes which affect each other. The result of such interactions between attributes is also specified in ISO 8613;
- Test cases which check that the IUT follows the ISO 8613 defaulting mechanism correctly.

Hence, reception test cases check that the IUT can process test data streams in accordance with the semantics of the ODA architectural model.

In reception testing, the results of the tests are determined from analysis of the DUTs provided by the IUT. Each test case in a TDS must result in a predictable outcome in a DUT. In the design of such test cases effects on the layout process need to be anticipated and controlled to some extent.

5.2.2.1 Algorithm for designing a reception test case

When designing a reception test case for a particular ODA requirement, the first step is to determine an environment suitable for the test case. A test case environment is a collection of constituents and attributes, organized in such a manner that, following reception of the associated TDS, the effect of the test case is predictable in a DUT produced by the IUT.

The second step involves representing the test environment in the test case notation. The test case must also express any necessary conditions needed to make the test environment valid. Each condition can identify disallowed attributes or attribute values for certain parts of the document if their inclusion would cause a conflict with the purpose of the test case.

The third optional step consists of writing a script indicating how to determine the verdict from DUT analysis. Scripts should indicate how to determine the verdict of a test case in a way that is, as far as possible, independent of the DUT representation. Because of the differences of equipment and applications, scripts should be written in general and flexible terms stating only the minimum requirements necessary to satisfy the purpose of the test cases.

Some further principles should be followed when designing reception test cases:

- A reception test case should not specify a complete document or document structure. The test case specification should only include what is needed to test the purpose and should be as general as the requirement will allow. For instance, a test case for the binding reference 'B_REF (SUP (CURR_OBJ)) ("binding name")' should not specify that the binding "binding name" be in an immediately superior object but rather in any superior of an object. This approach allows the test case to be more flexible.
- Scripts should indicate how to determine the verdict of a test case in a way that is, as far as possible, independent of the DUT representation. Expected results should be described in a relative manner by comparison rather than in absolute terms. In the case where the results must be described in an absolute manner, the script should specify a range of acceptable values rather than precise measures. This tolerance margin is necessary because of equipment precision differences and DAP tolerances. For example, when assessing a DUT containing a test case for the attribute "Position", the script might state "object B is below and to the left of object A" rather than "object B is positioned at 200, 500".

6 Abstract Test Case Proforma

This clause defines a proforma for ODA abstract test cases. The objectives of the proforma and notation are:

- to allow the specification of non-ambiguous implementation testing abstract test cases;
- to allow the configuration of abstract test cases to different DAPs;
- to provide test cases which are both human-readable and machine-processable.

The test case proforma consists of up to four sections:

- Section 1 Abstract Test Case Identifier
- Section 2 Abstract Test Case Purpose
- Section 3 Abstract Test Case Specification
- Section 4 Abstract Test Case Script

6.1 Abstract Test Case Identifier

This section contains an identifier that uniquely identifies the abstract test case.

6.2 Abstract Test Case Purpose

This section gives a human-readable description of the abstract test case. It is intended to help understand the test case and hence facilitate its use. The purpose also provides the test case intention and any assumptions. This information is useful when verifying the correctness of the test case.

6.3 Abstract Test Case Specification

This section provides the notational specification of the test case using the ATCN as specified in clause 7. The ATCN is machine processable and therefore suitable in a test document generation process that can generate complete documents containing one or more test cases. These documents can then be used during reception testing as part of an implementation test system that can generate Test Data Streams.

6.4 Abstract Test Case Script

This section is optional and used only for reception test cases. It provides a human readable description of the test case that can be used to determine the test case verdict. Such scripts should contain both general and specific requirements. The general requirement for an IUT is that the provision of a suitable system interface for the test operator in order that the DUT can be verified correctly. The specific requirements for the IUT are based on the particular test case under consideration.

The content of the text should include a list of checkpoints. Each checkpoint describes what predictable effects are expected in a DUT and indicates how these effects impact on the verdict of the test case.

7 Abstract Test Case Notation

There is some commonality between the specification of DAPs and the specification of test cases. Both types of specification define constraints on what is allowed by ODA. The Abstract Test Case Notation used in ISO/IEC TR 10183 is based on the DAPPN in Annex F of ISO 8613-1. Two types of revisions have been made to the DAPP notation in deriving the ATCN. The first removed functionality which was not applicable to test case specifications and the second added extensions based on constructs already defined in the DAPPN. Unless otherwise stated, constructs used in the DAPPN have the same semantics when used in the ATCN apart from the fact that they are constraints on document specifications rather than DAP specifications.

7.1 Overview

A test case specification consists of a set of constraints on an ODA document. The constraints can be used by an implementation test system during reception testing as part of the process that determines a test document or during generation testing as part of the process that checks the presence of test cases in a DUT received from the IUT. The

constraints are specified using one or more constraint definitions. There are two types of constraint definitions: constituent constraints and factor constraints.

A constituent constraint is composed of two elements: an environment section and an attribute constraint specification. The purpose of the attribute constraint specification is to define constraints on the presence and values of attributes in the constituent. The purpose of the environment section is to set necessary restrictions on the relationship of the constituent to other constituents in any document containing the test case. The restrictions can take the form of prescribed relationships between the constituent and other constituents, references to preconditions on the presence and values of attributes of one or more constituents in a document and any other characteristics necessary to achieve predictability of the DUT. Additionally, the environment section can also be used to place restrictions on the type of a constituent.

Factor constraints are used to specify preconditions or to factorize information. A factor constraint consists of an attribute constraint section, as specified in constituent constraints. The factorization of information in a test case specification is achieved by referencing one or more factor constraints using the factor reference mechanism.

A precondition is a set of constraints used by one or more constituents in the test case to restrict their subordinates, objects which are superior to them, or constituents hierarchically situated between two constituents of the test case. The precondition applies not only to the constituents which are defined in the test case specification but also to the constituents which are added to the test case specification to obtain a complete document. The constraints associated with the precondition are specified using the environment section and the attribute constraint specification of a factor constraint. A factor constraint can be referenced by one or more constituent constraints.

7.2 Definition of the symbols used in the notation

7.2.1 Symbols used in the meta-language

The symbols listed and defined below are used in the production rules contained in 7.3

::=	Used to specify that the string of symbols on its right side is to be substituted for the non-terminal symbol on its left side.
|	Used to separate alternatives.
< >	Used as a pair of symbols to delimit a non-terminal symbol.
-- --	Used as a pair of symbols to delimit a comment string.
( )	Used as a pair of symbols to delimit a syntactical unit.
[ ]	Used as a pair of symbols to delimit an optional syntactical unit.
...	Used following a syntactical unit to indicate that the syntactical unit may be occur one or more times.
' '	Used as a pair of symbols to delimit a terminal symbol.

7.2.2 Terminal symbols used in the notation

The following is a list of terminal symbols which are used in the notation.

...	Used to specify that an attribute or parameter value may be repeated one or more times.
{ }	Used as a pair of symbols to delimit a syntactical unit.
[ ]	Used to delimit an optional syntactical unit.
( )	Used to delimiter parameters in functions and macros.
< >	Used as a pair of symbols to delimit a non-terminal symbol.
-- --	Used as a pair of symbols to delimit a comment in an abstract test case.
|	Used to separate alternatives.
,	Used to separate attribute value specifications.
..	Used to specify a range of integers.
#	Used to announce a parameter.
" "	Used as a pair of symbols to delimit a printable character string. If a double quotation mark is to be used within the character string then it shall be announced with another double quotation mark.
` '	Used as a pair of symbols to delimit token words such as enumerated attribute values and terminal symbols in ISO 8613.
$	Used to announce a macro substitution.
>	Used to specify greater than.
<	Used to specify less than.
>=	Used to specify greater than or equal to.

<=	Used to specify less than or equal to.
\\	Used as a pair of symbols to delimit a non-printable character string.
/	Used to separate two hexadecimal numbers representing an octet.
ABSTRACT	Used to specify a constraint on the name of a constituent abstract feature associated with the constraint definition.
AGG	Used to specify an unordered set of elements with no additional elements.
AGG*	Used to specify an unordered set of elements with additional elements allowed.
ALT	Used to specify possible construction expressions that result in an instance from a choice of the specified object classes, where the instance cannot be preceded by or followed by other objects.
ALT*	Used to specify possible construction expressions that result in an instance from a choice of the specified object classes, where the instance can be preceded by or followed by other objects.
ANNOUNCER	Used to specify an announcer as defined in ISO 2022.
ANY_INTEGER	Used to specify any integer value allowed by ODA and an associated DAP.
ANY_LEVEL	Used to specify a subordinate of an object or object class description where each intermediate subordinate has no siblings.
ANY_LEVEL*	Used to specify a subordinate of an object or object class description where each intermediate subordinate may have siblings.
ANY_STRING	Used to specify any character string allowed by ODA and an associated DAP.
ANY_VALUE	Used to denote that any attribute or (sub-) parameter value may be specified that is permitted in ISO 8613 and an associated DAP.
ASN.1	Used to announce an ASN.1 value.
ATTRIBUTE	Used to announce a particular attribute or parameter in a particular constituent.
AVAILABLE_AREA	Used to specify a constraint on the available area of a layout object associated with the constraint definition.
BMU	Used to announce a generic measurement.
BOUNDED_PRE_CONDITION	Used to specify pre-conditions which apply to the objects hierarchically situated between two constituents associated with two particular constituent constraints.
CASE ... OF	Used to specify a conditional selection that depends on some referenced value.
CHARACTER	Used to denote any character content information allowed by ODA and an associated DAP.

CHO	Used to specify a set of alternative elements.
CONSTITUENT_TYPE	Used to specify a constraint on the type of constituents associated with the constraint definition.
CONTENT	Used to specify a piece of content the size of which is dependent on the size of the area which the piece of content must fill when it is laid out.
CONTENT_ID_OF	Used to specify implicitly the content portion identifiers of the set of instances of a particular constituent constraint.
DEFINE	Used to announce a string substitution macro.
DIS	Used to specify that a defaultable or non-mandatory attribute/parameter must not be specified in an associated constituent/attribute.
ENVIRONMENT	Used to specify restrictions on the environment of the constituents associated with the constraint definition.
EXCEPT	Used to specify values not allowed for a particular attribute or parameter.
FACTOR	Used to announce a common set of constraints that apply to one or more constituent constraints either directly or via the ODA defaulting mechanism.
FIXED	Used to specify a fixed measurement for the size of an area occupied by content information.
GEOMETRIC	Used to denote any geometric content information allowed by ODA and an associated DAP.
HORIZONTAL	Used to announce the horizontal dimension of an area.
ITER	Used to specify possible construction expressions that result in one or more instances of the specified object class, where the instances cannot be preceded by, interspersed with or followed by other objects.
ITER*	Used to specify possible construction expressions that result in one or more instances of the specified object class, where the instances can be preceded by, interspersed with or followed by other objects.
LAID_OUT_IN	Used to specify a relationship between a constituent in the logical structure and a constituent in the layout structure. That is, the logical constituent will be laid out in the specified layout constituent.
NOVR	Used to specify that the attribute value is specified on another constituent and its value may not be overridden.
MAX	Used to announce the specification of the maximum size of the horizontal or vertical dimension of an area.
MIN	Used to announce the specification of the minimum size of the horizontal or vertical dimension of an area.
MUL	Used to specify a group of parameters that occur one or more times.
OBJECT_CLASS_ID_OF	Used to specify any object class identifier from the set of instances of a particular constituent constraint.

OBJECT_ID_OF	Used to specify any object identifier from the set of instances of a particular constituent constraint.
PERM	Used to specify that a defaultable or non-mandatory attribute/parameter may be specified in the associated constituent/attribute.
PMUL	Used to specify a group of parameters that occur zero, one or more times.
RASTER	Used to denote any raster content information allowed by ODA and an associated DAP.
RELATIVE	Used to specify a relative measurement for the size of an area occupied by content information that is to occupy a percentage of the available area of a given layout object.
REP	Used to specify an element which can be repeated which cannot be preceded by, interspersed with or followed by other elements.
REP*	Used to specify an element which can be repeated which can be preceded by, interspersed with or followed by other elements.
REQ	Used to specify attributes or parameters that must be directly specified for, or applied to, a constituent or attribute.
REVISION	Used to specify the revision number of a character set as defined in ISO 2022.
SEQ	Used to specify an ordered set of elements with no additional elements.
SEQ*	Used to specify an ordered set of elements with additional elements allowed.
SER	Used to specify possible construction expressions that result in a sequence of instances of the specified object classes, where the instances cannot be preceded by, interspersed with or followed by other objects.
SER*	Used to specify possible construction expressions that result in a sequence of instances of the specified object classes, where the instances can be preceded by, interspersed with or followed by other objects.
SET	Used to specify possible construction expressions that result in an aggregate of instances of the specified object classes, where the instances cannot be preceded by, interspersed with or followed by other objects.
SET*	Used to specify possible construction expressions that result in an aggregate of instances of the specified object classes, where the instances can be preceded by, interspersed with or followed by other objects.
STYLE_ID_OF	Used to specify any style identifier from the set of instances of a particular style constraint.
SUB_ID_OF	Used to specify implicitly the object identifiers of the set of instances of the subordinates of a particular constituent constraint.

SUBORDINATES	Used to specify constraints on the subordinates of the constituent associated with the constraint definition.
SUBORDINATES_OF	Used to denote the subordinates, at any hierarchical level, of a constituent associated with the constituent constraint.
SUBORDINATES_PRE_CONDITION	Used to specify pre-conditions which apply to the subordinates of the constituent associated with the constraint definition.
SUPERIOR	Used when referencing a value to specify the first superior constituent constraint containing the value.
SUPERIORS_OF	Used to denote the superiors, at any hierarchical level, of a constituent associated with the constituent constraint.
SUPERIORS_PRE_CONDITION	Used to specify pre-conditions which apply to the objects which are superior (at any hierarchical level) to the constituent associated with the constraint definition.
TC_CONSTITUENT_NAME	Used to specify a constraint on the name of constituents associated with the constraint definition.
TcId	Used to announce an abstract test case identifier.
TcPurpose	Used to announce an abstract test case purpose.
TcNotation	Used to announce an abstract test case specification.
TcScript	Used to announce an abstract test case script.
VALUE	Used to announce the value of an attribute.
VERTICAL	Used to announce the vertical dimension of an area.
VIRTUAL	Used to specify that the attribute value will be specified in another constraint.
VISIBLE	Used to specify that the constituent associated with the constraint definition must be visible in the DUT.
VOID	Used when referencing a value to specify that the attribute is absent.

7.3 Description of the Notation

This clause describes how the notation is developed from a set of production rules. Some of the production rules are repeated in different subclauses of 7.3 in order to help the reader. A complete summary list of production rules is contained in 7.4.

Each Abstract Test Case Definition consists of an Abstract Test Identifier followed by a Test Case Purpose, an Abstract Test Case Specification and an optional Abstract Test Case Script. The format of an Abstract Test Case Definition is as follows:

```
<test-case-definition>          ::= <test-case-identifier>
                                    <test-case-purpose>
                                    <test-case-notation>
                                    [<test-case-script>]

<test-case-identifier>          ::= 'TcId' '{' <tc-id> '}'

<tc-id>                         ::= <asn1-object-id> | <tcid-c-f-name>
```

<tcid-c-f-name> -- A unique name that identifies a Test Case or Test Case Specification constituent or factor constraint. It consists of a sequence of one or more characters. The first character shall be an uppercase letter ('A'..'Z'). The remaining characters may be letters ('A'..'Z', 'a'..'z'), digits ('0'..'9') or hyphens ('-'). --

<test-case-purpose> ::= 'TcPurpose' '{' <printable-string> '}'

<test-case-notation> ::= 'TcNotation' '{' <test-case-specification> '}'

<test-case-script> ::= 'TcScript' '{' <printable-string> '}'

A test case specification, in the test case proforma notation, consists of intermixed macro definitions and constraint definitions. The format of the macro definition is described in 7.3.1. 7.3.2 defines the format of the constraint definitions. The format of a test case specification is as follows:

<test-case-specification> ::= { <macro-definition> | <constraint-definition> }...

7.3.1 Macro definition

The macro facility provides a shorthand mechanism for the notation. For each macro definition of the form DEFINE(S,".."), corresponding text or additional macros are substituted for all strings of the form: $S where S is a unique macro name identifying the macro. The string substitution macros must be fully expanded to a defined single string of characters lexically inserted into the corresponding part of the notation.

The format of a macro definition is as follows:

<macro-definition> ::= 'DEFINE' '{' <macro-name> '.' <macro-string> '}'

<macro-name> -- A unique name that identifies a Test Case Specification macro name. It consists of a sequence of one or more characters. The first character shall be a letter ('a'..'z', 'A'..'Z'). The remaining characters may be letters ('a'..'z', 'A'..'Z'), digits ('0'..'9') or hyphens ('-'). Uppercase and lowercase letters are not significant but their use throughout a Test Case Specification should be consistent. --

<macro-string> ::= <printable-string>

Macros shall be defined before their macro names are used. Macros may be repeated for readability but shall not be redefined.

7.3.2 Constraint definitions

Two types of constraint definitions can be specified in a test case: constituent constraints and factor constraints.

A constituent constraint is used to specify constraints on a constituent in a document containing the test case. It defines restrictions on the environment of the constituent as well as constraints on the presence and values of attributes of the constituent.

A factor constraint specifies constraints which are common to one or more constituent constraints.

The format of a constraint definition is as follows:

<constraint-definition> ::= <constituent-constraint> | <factor-constraint>

A constituent constraint consists of a constituent name followed by a constraint. The format of a constituent constraint is as follows:

<constituent-constraint> ::= <constituent-name> '{'<c-constraint> '}'

A factor constraint consists of a factor name followed by a constraint. The format of a factor constraint is as follows:

<factor-constraint> ::= <factor-name> '{' <f-constraint> '}'

7.3.3 Constraint names

There are two types of constraint names: constituent names and factor names. These types of names are used to identify respectively constituent constraints and factor constraints within a test case specification.

A test case constituent name consist of either one field or two fields separated by the terminal symbol ':'. The first field is a name which identifies the constituent (tcid-c-f-name). The second field (if used) identifies a factor constraint that specifies additional constraints imposed on that constituent.

A factor name is structured similarly to a constituent name. The first field in the name is used to identify the factor constraint (tcid-c-f-name). The second field of the factor name (if used) identifies another factor constraint which contains additional constraints that are applicable to that factor constraint.

<constituent-name> ::= <tcid-c-f-name> [<factor-reference>]

<factor-reference> ::= ':' <tcid-c-f-name>

<factor-name> ::= 'FACTOR' <tcid-c-f-name> [<factor-reference>]

7.3.4 Constraint specifications

A constraint specification consists of either a constituent constraint or a factor constraint. A constituent constraint consists of an optional environment section followed by an attribute constraint section. A factor constraint consists of an attribute constraint section.

The environment section allows restrictions to be made on the environment of particular constituents when present in a resulting document. The attribute constraint section is used to define constraints on the presence and values of attributes of particular constituents in a document. The constrained constituents are those which will be associated with the constraint definition.

The attribute constraint section may also be used in a factor constraint to factorize information applicable to several constituent constraints.

The environment section and the attribute constraint section are described respectively in 7.3.5 and 7.3.15.

The format of the constraint specifications is as follows:

<c-constraint> ::= [<environment-section>] <attr-constraint-section>

<f-constraint> ::= <attr-constraint-section>

7.3.5 Environment section

The environment section allows restrictions to be made on the environment of particular constituents when embedded

in a resulting document. The environment section consists of a constituent type constraint followed by zero, one or more environment constraints (7.3.7). A constituent type constraint (7.3.6) determines to which constituent type the constraint applies.

The format of the environment section is as follows:

<environment-section> ::= 'ENVIRONMENT'
<constituent-type-constraint>
[<environment-constraint>]...

7.3.6 Constituent type constraint

The purpose of this constraint is to identify the type of constituent to which a particular constituent constraint may be applied, without assigning values to attributes such as "object type" and "object identifier" which could be restricted by a given DAP.

The constraint on the constituent type of the constituent is expressed by specifying the keyword 'CONSTITUENT_TYPE' followed by a constituent type as specified in ISO 8613.

The format of a constituent type specification is as follows:

<constituent-type-constraint> ::= 'CONSTITUENT_TYPE' '(' <constituent-type> ')'

<constituent-type> ::= 'document-profile'
| 'layout-object-class'
| 'layout-object'
| 'content-portion'
| 'logical-object-class'
| 'logical-object'
| 'presentation-style'
| 'layout-style'

EXAMPLE

CONSTITUENT_TYPE (content-portion)

7.3.7 Environment constraint

Seven types of specifications may appear in an environment constraint. A test case constituent name constraint (7.3.8) determines to which constituent names the constraint applies.

A subordinates constraint (7.3.9) is used to specify relationships between a given constituent and other constituents in any document containing the test case. The specification indicates which constituents must be subordinate to a given constituent. In any resulting document the relationship must be supported by the constituents to which these constraints apply.

A reference to layout constraint (7.3.10) is used to specify that a given constituent in the logical structure must be laid out in a particular constituent in the layout structure in any document containing the test case. In such a document the relationship must be supported by the constituents to which these constraints apply.

A reference to precondition constraint (7.3.11) is used to impose constraints on the presence and values of attributes of particular constituents in any document containing the test case. Preconditions are referenced in constituent constraints to constrain the objects which are superior (at any level) to the given constituent, the subordinates of the given constituent or the constituents situated between two particular constituents belonging to the same ODA structure (these two constituents are identified in a test case using two constituent constraints).

A visibility constraint (7.3.12) is used if a constituent should be observable in the DUT. The available area constraint (7.3.13) is used to specify constraints on the available area of layout objects. The content constraint (7.3.14) is used as a constraint on some content by specifying the size of the area that the content must occupy when it is laid out.

The format of the environment section is as follows:

```
<environment-constraint>    ::=   <tc-constituent-name-constraint>
                                | <subordinates-constraint>
                                | <reference-to-layout>
                                | <reference-to-precondition>
                                | <visibility-constraint>
                                | <available-area-constraint>
                                | <content-constraint>
```

7.3.8 Constituent name constraint

The format of a constituent name constraint specification is as follows:

```
<tc-constituent-name-constraint>    ::= 'TC_CONSTITUENT_NAME'
                                        '(' <tc-constituent-name> [ '|' <tc-constituent-name> ]... ')'

<tc-constituent-name>               ::= <actual-feature-name>
                                        | 'ABSTRACT' '(' <abstract-feature-name> ')'

<actual-feature-name>                   -- String representing the name of a feature associated with a
                                           constituent identifier specified in one or more DAPs. --

<abstract-feature-name>                 -- String representing the name of an abstract feature
                                           independent of any particular DAP. --
```

7.3.9 Subordinates

This type of specification is used to define relationships between constituents in any document containing the test case. The relationship specifies which constituents are subordinate to a particular constituent. This type of relationship also allows general structures to be specified that are configurable to different DAPs.

In a test case, the constituents to which this relationship applies are identified using constituent constraints. The relationship must be supported by the constituents to which these constraints will be applied in any resulting document.

A reference to subordinates is announced by the keyword 'SUBORDINATES'. The keyword is followed by the specification of either a specific subordinate expression (which satisfies a set of construction expressions for object descriptions) or a generic subordinate expression (defined for object class descriptions).

The format of a reference to subordinates specification is as follows:

```
<subordinates-constraint>           ::= 'SUBORDINATES' '(' <subordinates-constraint-expr> ')'

<subordinates-constraint-expr>      ::= <generic-subord-expr> | <specific-subord-expr>
```

7.3.9.1 Specific subordinate expressions

Subordinate constraints are applicable to object descriptions and are specified using construction expressions. The construction expressions constrain the values allowed for the attribute "Subordinates" of the object description and objects which are subordinate to the object description (in the case of the operators ANY_LEVEL and

ANY_LEVEL*). In addition to the operators defined for construction expressions in the DAPPN, the following operators are also used: AGG*, SEQ*, REP*, ANY_LEVEL, ANY_LEVEL*. The operators AGG*, SEQ* and REP* may be used anywhere the operators AGG, SEQ and REP are allowed. The operators ANY_LEVEL and ANY_LEVEL* can be applied to any construction factor. An integer can optionally occur after the operators REP and REP* to indicate the required number of instances of the resulting object descriptions. The additional operators are defined in 7.2.2.

The format of a specific subordinate expression is as follows:

<specific-subord-expr> ::= <specific-subord-term>
|'AGG' '(' <specific-subord-list> ')'
|'SEQ' '(' <specific-subord-list> ')'
|'CHO' '(' <specific-subord-list> ')'
|'AGG*' '(' <specific-subord-list> ')'
|'SEQ*' '(' <specific-subord-list> ')'

<specific-subord-term> ::= 'REQ' <specific-subord-factor>
|'REP' [<integer>] <specific-subord-factor>
|'REP*' [<integer>] <specific-subord-factor>

<specific-subord-factor> ::= <any-level-factor> | <simple-factor>

<any-level-factor> ::= 'ANY_LEVEL' '(' <simple-factor> ')'
|'ANY_LEVEL*' '(' <simple-factor> ')'

<simple-factor> ::= <specific-subord-expr> | 'SUB_ID_OF' '(' <tcid-c-f-name> ')'

<specific-subord-list> ::= <specific-subord-term>[',' <specific-subord-term>]...

EXAMPLES

SUBORDINATES (AGG (REQ SUB_ID_OF (Text), REQ SUB_ID_OF (Footnote)))
allows subordinates {Text, Footnote} and {Footnote, Text} for example

SUBORDINATES (SEQ* (REQ SUB_ID_OF (Text), REQ SUB_ID_OF (Footnote)))
allows subordinates {Text, Footnote}, {x, Text, y, Footnote} and {x, y, Text, Footnote, z} for example, where x, y and z are any objects.

7.3.9.2 Generic subordinate expressions

Subordinate constraints applicable to object class descriptions specify a set of production rules. The production rules constrain the values allowed for the attribute "Generator-For-Subordinates" of the object class and also of subordinate object classes (in the case of the operators ANY_LEVEL and ANY_LEVEL*). In addition to the terminal symbols allowed in the DAPPN, the following meta-operators are also allowed in the production rules: SET, SET*, SER, SER*, ITER, ITER*, ALT, ALT*, ANY_LEVEL, ANY_LEVEL*. The constraints imposed by these operators are expressed by describing constraints on the specific structure. An integer can optionally occur after the operators ITER and ITER* to indicate the required number of instances of the resulting object descriptions. The additional operators are defined in 7.2.2.

The format of a generic subordinate expression is as follows:

<generic-subord-expr> ::= { <production-rule> }...

<production-rule> ::= <non-terminal-symbol> '::=' <value-expr-a> ';'

<non-terminal-symbol> -- An Abstract Test Case unique non-terminal symbol. It consists of

a sequence of one or more characters enclosed in angle brackets ('<' and '>'). The first character shall be a lowercase letter ('a'..'z'). The remaining characters may be lowercase letters ('a'..'z'), digits ('0'..'9') or hyphens ('-'). --

<value-expr-a> ::= <value-expr-b> ['|' <value-expr-b>]...

<value-expr-b> ::= <value-expr-c>
|<value-expr-c> <value-expr-c>...
|<value-expr-c> { ',' <value-expr-c> }...

<value-expr-c> ::= '{' <value-expr-a> '}' ['...']
|'[' <value-expr-a> ']' ['...']
|<value-type> ['...']

<value-type> ::= <non-terminal-symbol>
|<enumerated-type>
|<integer-value>
|<character-string>
|<escape-sequence>...
|<asn1-object-id>
|<reference-to-value>
|<reference-to-object-id>
|<reference-to-object-class-id>
|<reference-to-style-id>
|<reference-to-subordinates>
|<reference-to-content-portions>
|<expr-function>
|'ANY_VALUE'

<expr-function> ::= <basic-unit> | <test-case-unit>

<basic-unit> ::= <expr-token-word-0>
|<expr-token-word-1a> <one-parameter-a>
|<expr-token-word-1b> <one-parameter-b>
|<expr-token-word-2a> <two-parameter-a>
|<expr-token-word-2b> <two-parameter-b>

<one-parameter-a> ::= <value-type>
|'{' <value-expr-a> '}'

<one-parameter-b> ::= '(' <value-expr-a> ')'

<two-parameter-a> ::= '(' <value-expr-c> ',' <value-expr-c> ')'

<two-parameter-b> ::= '(' <value-expr-a> ')' '(' <value-expr-a> ')'

<test-case-unit> ::= 'ITER' [<integer>] '(' <single-parameter> ')'
|'ITER*' [<integer>] '(' <single-parameter> ')'
|'SET' '(' <parameter-list> ')'
|'SET*' '(' <parameter-list> ')'
|'SER' '(' <parameter-list> ')'
|'SER*' '(' <parameter-list> ')'
|'ANY_LEVEL' '(' <single-parameter> ')'
|'ANY_LEVEL*' '(' <single-parameter> ')'
|'ALT' '(' <parameter-list> ')'

|'ALT*' '(' <parameter-list> ')'

<single-parameter> ::= <value-type>

<parameter-list> ::= <value-type> { ',' <value-type> }...

EXAMPLES

SUBORDINATES (<subordinates> ::= ITER* (OBJECT_CLASS_ID_OF (Text));)

allows construction expressions such as:

REP OBJECT_CLASS_ID_OF (Text)
SEQ (REQ OBJECT_CLASS_ID_OF (Text), REQ OBJECT_CLASS_ID_OF (Text))
AGG (REQ OBJECT_CLASS_ID_OF (x), OPT OBJECT_CLASS_ID_OF (Text),
REQ OBJECT_CLASS_ID_OF (Text))
SEQ (REQ OBJECT_CLASS_ID_OF (Text), OPT OBJECT_CLASS_ID_OF (y),
REQ OBJECT_CLASS_ID_OF (Text), REQ OBJECT_CLASS_ID_OF (z))

where x, y and z are any object classes.

SUBORDINATES (<subordinates> ::= ALT (OBJECT_CLASS_ID_OF (Text),
OBJECT_CLASS_ID_OF (Figure));)

allows construction expressions such as:

REQ OBJECT_CLASS_ID_OF (Text)
REQ OBJECT_CLASS_ID_OF (Figure)
SEQ (REQ OBJECT_CLASS_ID_OF (Text))
AGG (REQ OBJECT_CLASS_ID_OF (Figure))
CHO (REQ OBJECT_CLASS_ID_OF (Text), OBJECT_CLASS_ID_OF (Figure))

7.3.10 Reference to layout

This type of specification is used to establish a relationship between a constituent in the logical structure and a constituent in the layout structure in any resulting document containing the test case. The purpose of the relationship is to indicate that the constituent in the logical structure should be laid out in a particular constituent in the layout structure. In a test case, the relationship is expressed using a constituent constraint defined for the logical structure and a constituent constraint defined for the layout structure. The relationship must be supported by the constituents to which these constituent constraints will be applied in any resulting document.

This type of relationship can be used to control the layout of a logical structure without explicitly specifying layout directives in the test case. The use of layout directives in a test case should be avoided, if possible, since such directives are usually DAP-dependent.

A reference to layout is announced by the keyword 'LAID_OUT_IN'. The keyword is followed by a constituent constraint name identifying the layout constituent constraint. The constituent constraint name is enclosed between round brackets.

The format of a reference to the layout structure is as follows:

<reference-to-layout> ::= 'LAID_OUT_IN' '(' <tcid-c-f-name> ')'
-- the tcid-c-f-name identifies a layout constituent constraint --

EXAMPLE

LAID_OUT_IN (Sub_LayObj)

7.3.11 Reference to pre-condition

A reference to a pre-condition is used to impose constraints on the presence and values of attributes of particular constituents in a test document. Pre-conditions are referenced in constituent constraints to constrain the objects which are superior (at any hierarchical level) to the constituent to which the constituent constraint is applied, the subordinates of the constituent to which the constituent constraint is applied or the constituents hierarchically situated between but not including two particular constituents belonging to the same structure (these two constituents are identified in the test case using two constituent constraints). Pre-conditions are used to protect a test case from the use of attribute values in any resulting DAP-specific test document that would conflict with its purpose.

A pre-condition is expressed using a factor constraint. A reference to a pre-condition is announced in a constraint definition using one of the following keywords:

- 'SUBORDINATES_PRE_CONDITION';
- 'SUPERIORS_PRE_CONDITION';
- 'BETWEEN_PRE_CONDITION'.

The first two keywords are followed by a field providing the name of a factor constraint specifying the pre-condition. The third keyword is followed by three fields. The first field is the name of a factor constraint specifying the pre-condition. The second and third fields provide the name of two constituent constraints delimiting the range of the pre-condition.

The format of a pre-condition specification is as follows:

<reference-to-pre-condition> ::= 'SUBORDINATES_PRE_CONDITION' '(' <tcid-c-f-name> ')'

| 'SUPERIORS_PRE_CONDITION' '(' <tcid-c-f-name>')'

| 'BETWEEN_PRE_CONDITION' '('<tcid-c-f-name> ','
<tcid-c-f-name> ','
<tcid-c-f-name> ')'

EXAMPLES

SUBORDINATES_PRE_CONDITION (PRECONDITION_1)

BETWEEN_PRE_CONDITION (PRECONDITION_2, Object1, Object2)

7.3.12 Visibility constraint

The visibility constraint indicates that properties of the constituent to which this constraint applies should be capable of being observed in the DUT. The properties to which this constraint applies will be determined when the test case is configured to a particular DAP. For example, properties represented by attributes such as "border" and "offset". Particular aspects of the feature can be described in the visibility description. The required abstract effect produced in the DUT is described in the script of the test case when it is configured to a particular DAP. In an abstract test case, the characteristic "visible" is announced using the keyword 'VISIBLE'.

The format of visibility specification is as follows:

<visibility-constraint> ::= 'VISIBLE' '(' <visibility-description> ')'

<visibility-description> -- A natural language description of visible aspects of the feature --

7.3.13 Available area constraint

The available area constraint is used to specify constraints on the available area of layout objects. It is used to ensure that the available area of a particular layout object will be large enough to contain particular content information. The characteristic is announced by the keyword 'AVAILABLE_AREA'. The keyword is followed by an area specification. The area specification is used to specify a minimum and/or maximum size in for the horizontal and/or vertical directions of the area. The size can be specified as an integer representing a value in SMU's or can be specified using the function 'SMU'. The function 'SMU' takes as parameter an integer value in BMU's and returns the corresponding value in SMU for the scaling unit selected during the DAP integration.

The format of available area specification is as follows:

```
<available-area-constraint>   ::= 'AVAILABLE_AREA' '(' <available-area> ')'

<available-area>              ::= 'HORIZONTAL' '(' <area-extent> ')'
                              |   'VERTICAL' '(' <area-extent> ')'
                              |   'HORIZONTAL' '(' <area-extent> ')' ',' 'VERTICAL' '('
                                      <area-extent> ')'

<area-extent>                 ::= 'MIN' '(' <measure> ')'
                              |   'MAX' '(' <measure> ')'
                              |   'MIN' '(' <measure> ')' ',' 'MAX' '(' <measure> ')'

<measure>                     ::= <smu-measure> | <generic-measure>

<smu-measure>                     -- An integer value specifying a measure in SMUs --

<generic-measure>             ::= 'SMU' '(' <bmu-measure> ')'

<bmu-measure>                     -- An integer value specifying a measure in BMUs --
```

EXAMPLE

AVAILABLE_AREA (HORIZONTAL (MIN (SMU (500)), MAX (SMU(2000))), VERTICAL (MIN (SMU(1000))))

7.3.14 Content constraint

The content constraint is used as a constraint on some content by specifying the size of the area that the content must occupy when it is laid out. The specification consists of the keyword 'CONTENT' followed by a content spec. The content spec, that is the size of the area, can be specified in absolute terms or in relative terms. In the former case, the specification of the size of the area consists of the keyword 'FIXED' followed by two fields. The first field is the horizontal size of the area and the second field the vertical size of the area. Both sizes are specified either as an integer value in SMU's or with the function 'SMU' (see previous paragraph). In the latter case, the size of the area is expressed in relation to the size of the available area of a particular layout object. The size of the area is specified using the keyword 'RELATIVE' followed by two fields. The first field consists of a constituent constraint name identifying the layout object. The second field provides a percentage number which is relative to the size of the layout object identified by the first field.

The format of content specification is as follows:

```
<content-constraint>          ::= 'CONTENT' '(' <content-spec> ')'
```

<content-spec> ::= 'FIXED' '(' <horizontal-measure> ',' <vertical-measure> ')'
| 'RELATIVE' '(' <tcid-c-f-name> ',' <area-percentage> ')'

<horizontal-measure> ::= <measure>

<vertical-measure> ::= <measure>

<area-percentage> -- Integer between 0 and 100 representing the percentage of the area which should be filled with content. --

EXAMPLE

CONTENT (RELATIVE (LayObj, 20))

7.3.15 Attribute constraint section

The attribute constraint section is used to specify constraints on the presence and values of attributes and attribute groups of particular constituents in any resulting document containing the test case.

The format of the attribute constraint section is as follows:

<attr-constraint-section> ::= [<attribute-expr-a>]

<attribute-expr-a> ::= <attribute-expr-b> ['|' <attribute-expr-b>]...

<attribute-expr-b> ::= <attribute-expr-c> [',' <attribute-expr-c>]...

An attribute expression type c can take one of two forms :

- case attribute value range specification;
- simple attribute value range specification.

The format of an attribute expression type c specification is as follows :

<attribute-expr-c> ::= <case-attribute> | <simple-attribute>

<case-attribute> ::= 'CASE' <reference-type> 'OF' '{' <single-case-attribute>... '}'

<reference-type> ::= <functional-reference> | <direct-reference>

<functional-reference> ::= <function> '(' <direct-reference> ')'

<function> ::= 'SUPERIOR'

<direct-reference> ::= <tcid-c-f-name> '(' <attribute-reference> ')'

<attribute-reference> ::= <attribute-name> ['#' <parameter-name>]...

<single-case-attribute> ::= <case-value> ':' <attribute-expr-a>

<case-value> ::= <attribute-value-except> | 'VOID'

The simple attribute specification begins with one of one of the following keywords REQ, PERM, DIS:

NOTE1 - If the attribute/parameter is specified as PERM and is defaultable then the attribute/parameter must have a value within the specified range, either explicitly specified in the constituent/attribute or derived using the defaulting rules specified in ISO 8613-2. If the attribute/parameter is non-mandatory then the constraint on the value is only applicable if

the attribute/parameter is specified in the constituent/attribute.

NOTE 2 - If the attribute/parameter is specified as DIS and is defaultable then the attribute/parameter must have a value, derived using the defaulting rules specified in ISO 8613-2., within the specified range. If the attribute/parameter is non-mandatory then no constraint on the value should be specified.

The second field is the name of the attribute which is being constrained. The third field contains an attribute value except specification. An attribute value can be empty or consist of a parameter expression type a, a keyword, content information or 'NOVR' followed by an attribute value expression.

The format of the simple attribute specification is as follows:

```
<simple-attribute>          ::= { 'REQ' | 'PERM' | 'DIS' } <attribute-name> [<attribute-value-except>]

<attribute-name>            -- a name that identifies an attribute from ISO 8613 (see Table 1 in
                               Annex F of ISO 8613-1) --

<attribute-value-except>    ::= '{' <attribute-value> '}' | 'EXCEPT' '{' <attribute-value> '}'

<attribute-value>           ::= -- empty --
                              | <parameter-expr-a>
                              | <keyword>
                              | <content-information>
                              | [ 'NOVR' ] <attribute-value-expr>

<attribute-value-expr>      ::= <production-rule>...
                              | <value-expr-a>

<parameter-expr-a>          ::= <parameter-expr-b> [ '|' <parameter-expr-b> ]...

<parameter-expr-b>          ::= <parameter-expr-c>
                              | <parameter-expr-c> <parameter-expr-c>...
                              | <parameter-expr-c> { ',' <parameter-expr-c> }...

<parameter-expr-c>          ::= <case-parameter>
                              | <simple-parameter>
                              | 'REQ' '{' <parameter-expr-a> '}'
                              | 'PERM' '{' <parameter-expr-a> '}'
                              | 'MUL' '{' <parameter-expr-a> '}'
                              | 'PMUL' '{' <parameter-expr-a> '}'

<case-parameter>            ::= 'CASE' <reference-type> 'OF' '{' <single-case-parameter>... '}'

<single-case-parameter>     ::= <case-value> ':' <attribute-value-except>

<simple-parameter>          ::= { 'REQ' | 'PERM' | 'DIS' } '#' <parameter-name>
                                [<attribute-value-except>]

<parameter-name>            -- a name that identifies a (sub-) parameter from ISO 8613 (see Table 1 in
                               Annex F of ISO 8613-1) --
```

7.3.15.1 Keywords

The allowed keywords are 'ANY_VALUE' and 'VIRTUAL'. The keyword 'ANY_VALUE' denotes that the attribute or parameter value may be any value allowed in ISO 8613 and the DAP specification. The keyword 'VIRTUAL' is used in factor constraints to denote that the attribute or parameter value will be determined in a constraint definition

that references that factor constraint.

The format of the keyword specification is as follows:

<keyword> ::= 'ANY_VALUE' | 'VIRTUAL'

EXAMPLE

```
REQ     Layout-object-class     {ANY_VALUE};
```

7.3.15.2 Content information constraint

This constraint is used to specify the type of content allowed in the content information attribute.

The format of the content information constraint is as follows:

<content-information> ::= <content-type> [<character-string>...]

<content-type> ::= 'CHARACTER'
| 'RASTER'
| 'GEOMETRIC'

7.3.15.3 Attribute value type

An attribute expression value type specification consists of one of fourteen different types of specification: non-terminal symbol, enumerated value, integer value, character string, escape sequence, asn1 object identifier, reference to an attribute or parameter value, reference to an object identifier, reference to an object class identifier, reference to a style identifier, reference to subordinates, reference to content portions, expression function and any attribute expression value allowed by ODA or an associated DAP.

The format of the attribute value type specification is as follows:

<value-type> ::= <non-terminal-symbol>
| <enumerated-type>
| <integer-value>
| <character-string>...
| <escape-sequence>...
| <asn1-object-id>
| <reference-to-value>
| <reference-to-object-id>
| <reference-to-object-class-id>
| <reference-to-style-id>
| <reference-to-subordinates>
| <reference-to-content-portions>
| <expr-function>
| 'ANY_VALUE'

EXAMPLES

```
REQ   Content-information          {"ccccc"};
REQ   Layout-style                 { STYLE_ID_OF (LStyle) };
PERM  Graphic-rendition            EXCEPT {'variable-spacing'};
```

7.3.15.4 Enumerated data specification

The enumerated data specification is used when an attribute or parameter consists of an enumerated data type as defined in ISO 8613. The specification consists of an enumerated data type value.

The format of the enumerated data specification is as follows:

<enumerated-type> -- Any enumerated data type value specified in ISO 8613 enclosed in single quotes (see Table 1 in Annex F of ISO 8613-1) --

EXAMPLE

REQ Layout-path {'90-degrees'};

7.3.15.5 Integer value specification

The integer value specification consists of one of four different types of specification. One type is the keyword 'ANY_INTEGER' which is used to specify any integer value allowed by ISO 8613. The three other types of specification are a choice of integers, a relational specification and the specification of an inclusive range between two integers.

The format of the integer value specification is as follows:

<integer-value> ::= <integer>
| { <relational-operator> <integer> }
| { <integer> '..' <integer> }
| 'ANY_INTEGER'

<integer> -- Any integer constant --

<relational-operator> ::= '>' | '<' | '>=' | '<='

EXAMPLES

REQ Dimensions {#horizontal-dimension {#fixed-dimension {<=9240} },
#vertical-dimension {#fixed-dimension {20..12400}}};

7.3.15.6 Character string specification

This form of specification is used to specify the value of an attribute or parameter of type character string. A character string can consist of the keyword 'ANY_STRING', a printable string or a non-printable string. The keyword 'ANY_STRING' is used to specify any string allowed by ODA and an associated DAP.

When ANY_STRING is used in a content information constraint such as:

Content-information {CHARACTER ANY_STRING "abc" ANY_STRING "def"};

ANY_STRING implies a string of one or more characters. If there is a choice, ANY_STRING is the longest, leftmost string that satisfies a match. If more than one ANY_STRING appears in the constraint then this rule is applied in the leftmost order.

A printable string consists of characters taken from the ISO/IEC 646 IRV (ASCII) character set of printable characters, enclosed in double quotes. A non-printable string is a string enclosed in "\" symbols and is composed of a sequence of octetstrings and/or control functions.

An octetstring is composed of a list of octets where each octet consists of two hexadecimal numbers (between 0 and 15) separated by the symbol '/'.

Two types of control functions can be specified in a non-printable string: ODA control functions and code extension control functions.

The format of the character string specification is as follows:

```
<character-string>          ::= 'ANY_STRING'
                                |<printable-string>
                                |<non-printable-string>

<printable-string>          -- One or more characters taken from the set of printable
                               ISO/IEC 646 IRV (ASCII) characters enclosed in double quotes. A
                               double quote in the string is announced by another double quote --

<non-printable-string>      ::= '\' { <octetstring> | <control-function> }... '\'

<octetstring>               ::= { <octet> }...

<octet>                     ::= <integer> '/' <integer>
                                --where each integer is in the range 00..15 --

<control-function>          ::= <oda-control-function> | <code-extension-control-fct>
```

The ODA control functions are those defined in ISO 8613/6 for the character content architecture. An ODA control function can be specified by providing an appropriate mnemonic name defined in ISO 8613/6 followed by a list of parameters (if required). The parameter values can either be integers, enumerated data values or keywords, depending on the control function and the type of test case.

The format of an ODA control function is as follows:

```
<oda-control-function>      ::= <oda-control-fct-name> [ <oda-control-fct-parameters> ]

<oda-control-fct-name>      ::= 'BPH' | 'BS' | 'CR' | 'HPB' | 'HPR' | 'GCC' | 'IGS'
                                |'JFY' | 'LF' | 'NBH' | 'PLD' | 'PLU' | 'PTX' | 'SCS'
                                |'SGR' | 'SHS' | 'SACS' | 'SLS' | 'SRCS' | 'SOS' | 'SSW'
                                |'SP' | 'SRS' | 'ST' | 'STAB' | 'SUB' | 'SVS' | 'VPB' | 'VPR'

<oda-control-fct-parameters> ::= '(' <oda-ctl-fct-parameter> [ ',' <oda-ctl-fct-parameter> ]... ')'

<oda-ctl-fct-parameter>     ::= <integer> | <value-type>
```

The code extension control functions are those defined in ISO 2022 and ISO 6429 for the designation and invocation of graphic character sets. There are four types of code extension control functions: announcer control functions, revision control functions, character set designation control function and character set invocation control functions. An announcer control function is specified by providing the keyword 'ANNOUNCER' followed by an octet enclosed between round brackets. A revision control function is used to identify the revision number of the character set and is specified by providing the keyword 'REVISION' followed by an integer enclosed between round brackets. A character set designation control function is used to designate a character set and is specified by an escape defined in ISO 2022. The list of allowed character set names is given below. The list of parameters is specified as a non-printable string. A character set invocation control function is used to invoke a character set. It is specified by providing a keyword representing a shift function. The list of allowed keywords for shift function is given below.

The format of a code extension control function is as follows:

```
<code-extension-control-fct>  ::= <announcer> | <revision> | <character-set-designation>
```

<announcer> ::= 'ANNOUNCER' '(' <octet> ')'

<revision> ::= 'REVISION' '(' <integer> ')'

<character-set-designation> ::= {<escape-sequence> | <invocation-control-function>} ...

EXAMPLES

REQ Content-information {CHARACTER "arc"\LF\"x"\PLD\"2"\PLU\"yz"\CR LF\ };
REQ Content-information {CHARACTER "ask"\SGR (3, 4)\"Bill" };
REQ Content-information {CHARACTER "gar"\ESC 02/13 04/01 LS1R\"çon" };

7.3.15.7 Escape sequence

The escape sequence specification is used when an attribute expression value consists of an escape sequence used for control purposes. Escape sequences are described in terms of the ESC control function followed by one or more characters defined by row and column positions in a character code table. The format of the escape sequence constraint is as follows:

<escape-sequence> ::= { 'ESC'<octet>...}

EXAMPLE

REQ Profile-character-sets {ESC 02/13 04/01}; Designating ISO 8859-1 to G1

7.3.15.8 Invocation control function

This form of specification is used when an attribute value allows the use of invocation control functions. Invocation control functions are specified by providing a keyword that represents a shift function. The format of the invocation control function constraint is as follows:

<invocation-control-function> ::= {'SI' | 'SO' | 'LS0' | 'LS1' | 'LS1R' | 'LS2' | 'LS2R' | 'LS3' | 'LS3R' | 'SS2' | 'SS3'}

EXAMPLE

REQ Content-information {CHARACTER "gar"\LS1R\"çon"};

NOTE - The invocation control function is specified by means of the escape sequence "ESC 07/14"

7.3.15.9 ASN.1 object identifier constraint

This form of specification is used to specify the value of an attribute or parameter of type ASN.1 object identifier. The value is specified as a sequence of integers. The format of the ASN.1 object identifier constraint is as follows:

<asn1-object-id> ::= 'ASN.1' '{' <integer>... '}'

EXAMPLES

REQ Content-architecture-class {ASN.1 { 2 8 2 6 1 } };

7.3.15.10 Reference to attribute values

This form of specification is used to specify the value of an attribute or parameter by referring to the value of a particular attribute or parameter in one or more constituents of the test document. The function 'VALUE' is used for this form of specification. The function has two parameters. The first parameter identifies the attribute or parameter, the value of which will be extracted from the designated constituents of the test document. The second parameter identifies the constituents in which the values must be taken. The constituents are identified by either providing a list of constituent constraint names or by using one of the following functions:'SUBORDINATES_OF' or 'SUPERIORS_OF'. The function 'SUBORDINATES_OF' refers to the constituents which are subordinates of a particular constituent. The function 'SUPERIORS_OF' refers to the constituents which are superior (at any hierarchical level) to a particular constituent. The constituent is identified in the parameter of the function using a constituent constraint name. The function returns the values of the attribute or parameter (identified by the first parameter) allowed by the designated constituents (identified by the second parameter).

The format of the reference to attribute values specification is as follows:

<reference-to-value> ::= 'VALUE' '(' <reference-to-attr> ')'

<reference-to-attr> ::= 'ATTRIBUTE' '(' <attr-or-parameter-name> ',' <ref-to-constituents> ')'

<attr-or-parameter-name> ::= <attribute-name> ['#' <parameter-name>]...

<ref-to-constituents> ::= '{' <tcid-c-f-name> [',' <tcid-c-f-name>]... '}'
| 'SUBORDINATES_OF' '(' <tcid-c-f-name> ')'
| 'SUPERIORS_OF' '(' <tcid-c-f-name> ')'

EXAMPLES

```
REQ     New-layout-object    {
          #to-layout-object  {VALUE (ATTRIBUTE(Object-class-identifier, {LayObj}))}};
PERM    Layout-object-class  EXCEPT {VALUE (ATTRIBUTE(Object-class-identifier,
                                      SUBORDINATES_OF (LayObj)))};
```

7.3.15.11 References to object, object classes, styles, subordinates and content portions

The references to objects, object classes, styles, subordinates and content portions are used when the value of an attribute or parameter consists of an object, object class, layout or presentation style or the sequence of numeric strings that implicitly identify subordinates or content portions. The specification consists of a keyword followed by a constituent constraint name.

The format of the reference to objects, object classes, styles and content portions is as follows:

<reference-to-object-id> ::= 'OBJECT_ID_OF' '(' <tcid-c-f-name> ')'

<reference-to-object-class-id> ::= 'OBJECT_CLASS_ID_OF' '(' <tcid-c-f-name> ')'

<reference-to-style-id> ::= 'STYLE_ID_OF' '(' <tcid-c-f-name> ')'

<reference-to-subordinates> ::= 'SUB_ID_OF' '(' <tcid-c-f-name> ')' [<integer>]

<reference-to-content-portions> ::= 'CONTENT_ID_OF' '(' <tcid-c-f-name> ')' [<integer>]

EXAMPLE

```
REQ     Layout-style        { STYLE_ID_OF (LStyle-1) };
```

7.3.15.12 Constraints on expression functions

There are two types of constraints on expression functions: basic units and test case units. The constraint on basic unit expression functions is used to specify object identifier expression constraints, string expression constraints, binding pair constraints and construction expression constraints. Each of these constraints is specified the same way as in Annex F of ISO 8613-1. The test case unit is used to constrain construction expressions (see 7.3.9.2)

The format of the constraint on expression functions is as follows:

```
<expr-function>        ::= <basic-unit> | <test-case-unit>

<basic-unit>           ::= <expr-token-word-0>
                         | <expr-token-word-1a> <one-parameter-a>
                         | <expr-token-word-1b> <one-parameter-b>
                         | <expr-token-word-2a> <two-parameter-a>
                         | <expr-token-word-2b> <two-parameter-b>

<one-parameter-a>      ::= <value-type>
                         | '{' <value-expr-a> '}'

<one-parameter-b>      ::= '(' <value-expr-a> ')'

<two-parameter-a>      ::= '(' <value-expr-c> ',' <value-expr-c> ')'

<two-parameter-b>      ::= '(' <value-expr-a> ')' '(' <value-expr-a> ')'
```

The expression token words are those words used as terminal symbols in A.2 in ISO 8613-2 apart from the values of the non-terminal <object type>. The association of these words is as follows :

<expr-token-word-0> :

'CURR-OBJ'	'CURRENT-OBJECT'		

<expr-token-word-1a> :

'OPT'	'OPT REP'	'REP'

<expr-token-word-1b> :

'AGG'	'AGGREGATE'	'CHO'	'CHOICE'
'DEC'	'DECREMENT'	'INC'	'INCREMENT'
'L-ALPHA'	'LOWER-ALPHA'	'L-ROM'	'LOWER-ROMAN'
'MK-STR'	'MAKE-STRING'	'ORD'	'ORDINAL'
'PREC'	'PRECEDING'	'PREC-OBJ'	'PRECEDING-OBJECT'
'SEQ'	'SEQUENCE'	'SUP'	'SUPERIOR'
'SUP-OBJ'	'SUPERIOR-OBJECT'	'U-ALPHA'	'UPPER-ALPHA'
'U-ROM'	'UPPER-ROMAN'		

<expr-token-word-2a> :

'CURR-INST'	'CURRENT-INSTANCE'

<expr-token-word-2b> :

'B_REF'	'BINDING_REFERENCE'

EXAMPLE

```
REQ        Bindings { REQ #binding-identifier {"PGnum"},
                      REQ #binding-value{INC(B_REF(PREC(CURR-OBJ))("PGnum")) };
```

7.4 Summary of the production rules

<test-case-definition> ::= <test-case-identifier>
<test-case-purpose>
<test-case-notation>
[<test-case-script>]

<test-case-identifier> ::= 'TcId' '{' <tc-id> '}'

<tc-id> ::= <asn1-object-id> | <tcid-c-f-name>

<tcid-c-f-name> -- A unique name that identifies a Test Case or Test Case Specification constituent or factor constraint. It consists of a sequence of one or more characters. The first character shall be an uppercase letter ('A'..'Z'). The remaining characters may be letters ('A'..'Z', 'a'..'z'), digits ('0'..'9') or hyphens ('-'). --

<test-case-purpose> ::= 'TcPurpose' '{' <printable-string> '}'

<test-case-notation> ::= 'TcNotation' '{' <test-case-specification> '}'

<test-case-script> ::= 'TcScript' '{' <printable-string> '}'

<test-case-specification> ::= { <macro-definition> | <constraint-definition> }...

<macro-definition> ::= 'DEFINE' '(' <macro-name> ',' <macro-string> ')'

<macro-name> -- A unique name that identifies a Test Case Specification macro name. It consists of a sequence of one or more characters. It consists of a sequence of one or more characters. The first character shall be a letter ('a'..'z', 'A'..'Z'). The remaining characters may be letters ('a'..'z', 'A'..'Z'), digits ('0'..'9') or hyphens ('-'). Uppercase and lowercase letters are not significant but their use throughout a Test Case Specification should be consistent. --

<macro-string> ::= <printable-string>

<constraint-definition> ::= <constituent-constraint> | <factor-constraint>

<constituent-constraint> ::= <constituent-name> '{' <c-constraint> '}'

<constituent-name> ::= <tcid-c-f-name> | <factor-reference>

<factor-reference> ::= ':' <tcid-c-f-name>

<c-constraint> ::= [<environment-section>] <attr-constraint-section>

<factor-constraint> ::= <factor-name> '{ <f-constraint> '}'

<factor-name> ::= 'FACTOR' <tcid-c-f-name> [<factor-reference>]

```
<f-constraint>                  ::= <attr-constraint-section>

<environment-section>           ::= 'ENVIRONMENT'
                                    <constituent-type-constraint>
                                    [ <environment-constraint> ]...

<constituent-type-constraint>   ::= 'CONSTITUENT_TYPE' '(' <constituent-type> ')'

<constituent-type>              ::= 'document-profile'
                                  | 'layout-object-class'
                                  | 'layout-object'
                                  | 'content-portion'
                                  | 'logical-object-class'
                                  | 'logical-object'
                                  | 'presentation-style'
                                  | 'layout-style'

<environment-constraint>        ::= <tc-constituent-name-constraint>
                                  | <subordinates-constraint>
                                  | <reference-to-layout>
                                  | <reference-to-precondition>
                                  | <visibility-constraint>
                                  | <available-area-constraint>
                                  | <content-constraint>

<tc-constituent-name-constraint>::='TC_CONSTITUENT_NAME'
                                    '(' <tc-constituent-name> [ '|' <tc-constituent-name> ]... ')'

<tc-constituent-name>           ::= <actual-feature-name>
                                  | 'ABSTRACT' '(' <abstract-feature-name> ')'

<actual-feature-name>             -- String representing the name of a feature associated with a
                                    constituent identifier specified in one or more DAPs. --

<abstract-feature-name>           -- String representing the name of an abstract feature
                                    independant of any particular DAP. --

<subordinates-constraint>       ::= 'SUBORDINATES' '(' <subordinates-constraint-expr> ')'

<subordinates-constraint-expr>::= <generic-subord-expr> | <specific-subord-expr>

<generic-subord-expr>           ::= { <production-rule> }...

<specific-subord-expr>          ::= <specific-subord-term>
                                  | 'AGG' '(' <specific-subord-list> ')'
                                  | 'SEQ' '(' <specific-subord-list> ')'
                                  | 'CHO' '(' <specific-subord-list> ')'
                                  | 'AGG*' '(' <specific-subord-list> ')'
                                  | 'SEQ*' '(' <specific-subord-list> ')'

<specific-subord-term>          ::= 'REQ' <specific-subord-factor>
                                  | 'REP' [<integer>] <specific-subord-factor>
                                  | 'REP*' [<integer>] <specific-subord-factor>

<specific-subord-factor>        ::= <any-level-factor> | <simple-factor>
```

<any-level-factor> ::= 'ANY_LEVEL' '(' <simple-factor> ')'
| 'ANY_LEVEL*' '(' <simple-factor> ')'

<simple-factor> ::= <specific-subord-expr> | 'SUB_ID_OF '(' <tcid-c-f-name> ')'

<specific-subord-list> ::= <specific-subord-term> [',' <specific-subord-term>]...

<reference-to-layout> ::= 'LAID_OUT_IN' '(' <tcid-c-f-name> ')'

<reference-to-precondition> ::= 'SUBORDINATES_PRE_CONDITION' '(' <tcid-c-f-name> ')'
| 'SUPERIORS_PRE_CONDITION' '(' < tcid-c-f-name> ')'
| 'BOUNDED_PRE_CONDITION'
'(' <tcid-c-f-name> ',' <tcid-c-f-name> ',' <tcid-c-f-name> ')'

<visibility-constraint> ::= 'VISIBLE' '(' <visibility-description> ')'

<visibility-description> -- A natural language description of visible aspects of the feature --

<available-area-constraint> ::= 'AVAILABLE_AREA' '(' <available-area> ')'

<available-area> ::= 'HORIZONTAL' '(' <area-extent> ')'
| 'VERTICAL' '(' <area-extent> ')'
| 'HORIZONTAL' '(' <area-extent> ')' ',' 'VERTICAL' '(' <area-extent> ')'

<area-extent> ::= 'MIN' '(' <measure> ')'
| 'MAX' '(' <measure> ')'
| 'MIN' '(' <measure> ')' ',' 'MAX' '(' <measure> ')'

<measure> ::= <smu-measure> | <generic-measure>

<smu-measure> -- An integer value specifying a measure in SMUs --

<generic-measure> ::= 'SMU' '(' <bmu-measure> ')'

<bmu-measure> -- An integer value specifying a measure in BMUs --

<content-constraint> ::= 'CONTENT' '(' <content-spec> ')'

<content-spec> ::= 'FIXED' '(' <horizontal-measure> ',' <vertical-measure> ')'
| 'RELATIVE' '(' <tcid-c-f-name> ',' <area-percentage> ')'

<horizontal-measure> ::= <measure>

<vertical-measure> ::= <measure>

<area-percentage> -- Integer between 0 and 100 representing the percentage of
the area which should be filled with content. ---

<attr-constraint-section> ::= [<attribute-expr-a>]

<attribute-expr-a> ::= <attribute-expr-b> ['|' <attribute-expr-b>]...

<attribute-expr-b> ::= <attribute-expr-c> [',' <attribute-expr-c>]...

<attribute-expr-c> ::= <case-attribute> | <simple-attribute>

<case-attribute> ::= 'CASE' <reference-type> 'OF' '{' <single-case-attribute>... '}'

```
<reference-type>            ::= <functional-reference> | <direct-reference>

<functional-reference>      ::= <function> '(' <direct-reference> ')'

<function>                  ::= 'SUPERIOR'

<direct-reference>          ::= <tcid-c-f-name> '(' <attribute-reference> ')'

<attribute-reference>       ::= <attribute-name> [ '#' <parameter-name> ]...

<single-case-attribute>     ::= <case-value> ':' <attribute-expr-a>

<case-value>                ::= <attribute-value-except> | 'VOID'

<simple-attribute>          ::= { 'REQ' | 'PERM' | 'DIS' } <attribute-name> [<attribute-value-except>]

<attribute-name>             -- A name that identifies an attribute from ISO 8613 (see Table 1 in
                                Annex F of ISO 8613-1) --

<attribute-value-except>    ::= '{' <attribute-value> '}' [ 'EXCEPT' '{' <attribute-value> '}' ]

<attribute-value>           ::= -- empty --
                              | <parameter-expr-a>
                              | <keyword>
                              | <content-information>
                              | [ 'NOVR' ] <attribute-value-expr>

<parameter-expr-a>          ::= <parameter-expr-b> [ '|' <parameter-expr-b> ]...

<parameter-expr-b>          ::= <parameter-expr-c>
                              | <parameter-expr-c> <parameter-expr-c>...
                              | <parameter-expr-c> { ',' <parameter-expr-c> }...

<parameter-expr-c>          ::= <case-parameter>
                              | <simple-parameter>
                              | 'REQ' '{' <parameter-expr-a> '}'
                              | 'PERM' '{' <parameter-expr-a> '}'
                              | 'MUL' '{' <parameter-expr-a> '}'
                              | 'PMUL' '{' <parameter-expr-a> '}'

<case-parameter>            ::= 'CASE' <reference-type> 'OF' '{' <single-case-parameter>... '}'

<single-case-parameter>     ::= <case-value> ':' <attribute-value-except>

<simple-parameter>          ::= { 'REQ' | 'PERM' | 'DIS' } '#' <parameter-name>
                                [<attribute-value-except>]

<parameter-name>             -- A name that identifies a (sub-) parameter from ISO 8613 (see Table 1 in
                                Annex F of ISO 8613-1) --

<keyword>                   ::= 'ANY_VALUE' | 'VIRTUAL'

<content-information>       ::= <content-type> [<character-string>...]

<content-type>              ::= 'CHARACTER'
                              | 'RASTER'
                              | 'GEOMETRIC'
```

```
<character-string>              ::= 'ANY_STRING'
                                  | <printable-string>
                                  | <non-printable-string>

<printable-string>               -- One or more characters taken from the set of printable ascii characters
                                    enclosed in double quotes. A double quote in the string is announced by
                                    another double quote --

<non-printable-string>          ::= '\' { <octetstring> | <control-function> }... '\'

<octetstring>                   ::= { <octet> }...

<octet>                         ::= <integer> '/' <integer>
                                  -- where each integer is in the range 00..15 --

<control-function>              ::= <oda-control-function> | <code-extension-control-fct>

<oda-control-function>          ::= <oda-control-fct-name> [ <oda-control-fct-parameters> ]

<oda-control-fct-name>          ::= 'BPH' | 'BS' | 'CR' | 'HPB' | 'HPR' | 'GCC' | 'IGS'
                                  | 'JFY' | 'LF' | 'NBH' | 'PLD' | 'PLU' | 'PTX' | 'SCS'
                                  | 'SGR' | 'SHS' | 'SACS' | 'SLS' | 'SRCS' | 'SOS' | 'SSW'
                                  | 'SP' | 'SRS' | 'ST' | 'STAB' | 'SUB' | 'SVS' | 'VPB' | 'VPR'

<oda-control-fct-parameters>    ::= '(' <oda-ctl-fct-parameter> [ ',' <oda-ctl-fct-parameter> ]... ')'

<oda-ctl-fct-parameter>         ::= <integer>
                                  | <value-type>

<code-extension-control-fct>    ::= <announcer>
                                  | <revision>
                                  | <character-set-designation>

<announcer>                     ::= 'ANNOUNCER' '(' <octet> ')'

<revision>                      ::= 'REVISION' '(' <integer> ')'

<character-set-designation>     ::= { <escape-sequence>| <invocation-control-function>}...

<escape-sequence>               ::= 'ESC'<octet>...

<invocation-control-function>   ::= {'SI' | 'SO' | 'LS0' | 'LS1' | 'LS1R' | 'LS2' | 'LS2R' |
                                    'LS3' | 'LS3R' | 'SS2' | 'SS3'}

<attribute-value-expr>          ::= <production-rule>...
                                  | <value-expr-a>

<production-rule>               ::= <non-terminal-symbol> '::=' <value-expr-a> ';'

<non-terminal-symbol>            -- An Abstract Test Case unique non-terminal symbol. It consists of a
                                    sequence of one or more characters enclosed in angle brackets ('<' and
                                    '>'). The first character shall be a lowercase letter ('a'..'z'). The remaining
                                    characters may be lowercase letters ('a'..'z'), digits ('0'..'9') or hyphens
                                    ('-'). --

<value-expr-a>                  ::= <value-expr-b> [ '|' <value-expr-b> ]...
```