ETSI EG 203 130 V1.1.1 (2013-04)

ETSI Guide

Methods for Testing and Specification (MTS);
Model-Based Testing (MBT);
Methodology for standardized test specification development

Reference

DEG/MTS-142 MBT_methodology

Keywords

methodology, model, testing

Contents

Intellectual Property Rights

Foreword

This ETSI Guide (EG) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

1 Scope

The present document provides a set of guidelines for the use of Model Based Testing in standardized test development. This includes model creation for test generation and test selection. It also defines a process for development and review of models and generated tests.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TR 102 840 (V1.2.1): "Methods for Testing and Specifications (MTS); Model-based testing in standardisation".

[i.2] ETSI ES 202 951 (V1.1.1): "Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Requirements for Modelling Notations".

[i.3] DTR/MTS-00141: "MBT Case Studies".

[i.4] ISO/IEC 9646-1: "Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework -- Part 1: General concepts".

[i.5] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[i.6] ETSI TS 102 871-2 (V1.1.1): "Intelligent Transport Systems (ITS); Testing; Conformance test specifications for GeoNetworking ITS-G5; Part 2: Test Suite Structure and Test Purposes (TSS&TP)".

[i.7] ETSI ES 202 553: "Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes".

[i.8] ETSI EG 202 237: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic approach to interoperability testing".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

abstract test case: complete and independent specification of the actions required to achieve a specific test purpose, defined at the level of abstraction of a particular abstract test method, starting in a stable testing state and ending in a stable testing state

NOTE: See ISO/IEC 9646-1 [i.4].

abstract test method: description of how an IUT is to be tested, given at an appropriate level of abstraction to make the description independent of any particular realization of a means of testing, but with enough detail to enable abstract test cases to be specified for this method

NOTE: See ISO/IEC 9646-1 [i.4].

Abstract Test Suite (ATS): test suite composed of abstract test cases

NOTE: See ISO/IEC 9646-1 [i.4].

adaptation of the generated test suite: extending the generated code so that the extension lowers the abstraction level in order to get and Abstract Test Suite (ATS)

NOTE: See ISO/IEC 9646-1 [i.4].

executable test case: realization of an abstract test case

NOTE: See ISO/IEC 9646-1 [i.4].

Implementation Conformance Statement (ICS): statement made by the supplier of an implementation or system claimed to conform to a given specification, stating which capabilities have been implemented

NOTE: See ISO/IEC 9646-1 [i.4].

Implementation eXtra Information for Testing (IXIT): statement made by a supplier or implementor of an IUT which contains or references all of the information (in addition to that given in the ICS) related to the IUT and its testing environment, which will enable the test laboratory to run an appropriate test suite against the IUT

NOTE: See ISO/IEC 9646-1 [i.4].

Implementation Under Test (IUT): implementation of one or more standards in an adjacent user/provider relationship, being the part of real open system which is to be studied by testing

Interoperable Function Statement (IFS):document that defines (a) the standardised functions which an IUT supports, (b) the functions which are optional and (c) those that are conditional on the support of other functions

NOTE: See EG 202 237 [i.8].

means of testing: combination of equipment and procedures that can perform the derivation, selection, parameterization and execution of test cases, in conformance with a reference standardized ATS and can produce a conformance log

NOTE: See ISO/IEC 9646-1 [i.4].

Model-Based Testing (MBT): umbrella of approaches that generate tests from models

NOTE: See TR 102 840 [i.1].

system model: computer-readable behavioural model that describes the intended external operational characteristics of a system, i.e. how the system being modelled interacts with its environment

NOTE: See TR 102 840 [i.1].

System Under Test (SUT): the real open system in which the IUT resides

NOTE: See ISO/IEC 9646-1 [i.4].

Test Case (TC): abstract or executable test case

NOTE: See ISO/IEC 9646-1 [i.4].

Test Description (TD): prose description of a test case

Test Purpose (TP): prose description of a well defined objective of testing, focusing on a single conformance requirement or a set of related conformance requirements as specified in the appropriate standard

Test Suite Structure (TSS): document defining (hierarchical) grouping of test cases according to some rules

transformation of the generated test suite: modifying the generated code in order to improve its quality, but without changing its level of abstraction and its behaviour

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
ASN	Abstract Syntax Notation
ATM	Automated Teller Machine
ATS	Abstract Test Suite
ICS	Implementation Conformance Statement
IFS	Interoperable Function Statement
ITS	Intelligent Transportation Systems
IUT	Implementation Under Test
IXIT	Implementation eXtra Information for Testing
MBT	Model-Based Testing
OSI	Open Systems Interconnection
STF	Specialist Task Force
SUT	System Under Test
TC	Test Case
TD	Test Description
TP	Test Purpose
TSS	Test Suite Structure
TTCN	Tree and Tabular Combined Notation

4 Scope and purpose of these guidelines

The present document provides general guidelines for using model based testing with automated test generation in conformance test development for ETSI standards.

All the guidelines presented are tool-independent. The present document does not provide any enumeration of MBT tools, characterization or comparison of MBT tools, recommendations on using specific MBT tools, or tips on how to perform conformance test development tasks with the help of specific MBT tools.

5 Process

5.1 ETSI test development process

The ETSI test development process is sketched in Figure 1. In the figure, the boxes describe development steps and the ellipses describe the corresponding outcomes, e.g. "Requirements Catalogue" is the result of the step "Identification of Requirements".

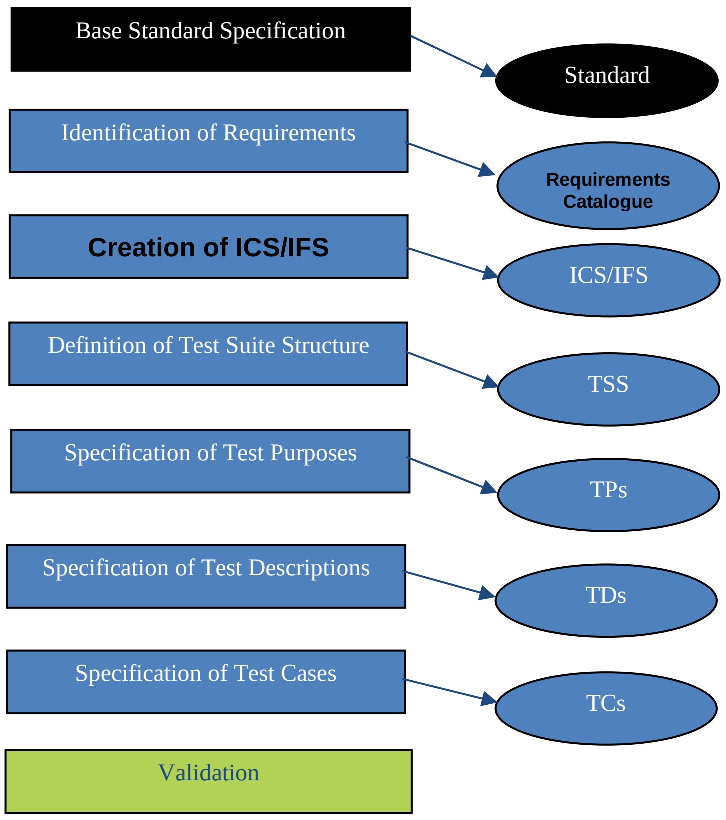
Base Standard Specification
Standard
Identification of Requirements
Requirements Catalogue
Creation of ICS/IFS
ICS/IFS
Definition of Test Suite Structure
TSS
Specification of Test Purposes
TPs
Specification of Test Descriptions
TDs
Specification of Test Cases
TCs
Validation

Figure 1: ETSI test development process

The "Cataloguing of Requirements" comprises the identification of requirements by analysis of the base standard. It is recommended to collect and structure the identified requirements in a catalogue.

The "Creation of ICS/IFS" step develops Implementation Conformance Statement (ICS) document or Interoperable Function Statement (IFS) document. ICS/IFS will be presented as templates to be filled by product vendors declaring which of the optional features in the standard their product is supporting, i.e. implementing. The ICS/IFS templates make the relationship between features rather clear and easy to follow. Some features are defined to be mandatory in the standard, the ICS/IFS document should reflect that and the vendor is supposed to declare that their product supports all mandatory features if he wants to claim conformance to the standard. Some features will be optional and the vendor in question has full freedom to choose whether they support an optional feature or not. Other features may be conditional, e.g. if you have selected some option then some other options have to follow. For example, if a protocol message type is declared to be supported, the data fields in that message need to be supported as well. ICS/IFS are supposed to reflect what is already written in the standard and should not change the requirements but just express the same thing in a more clear, concise and precise way. In practice, this is not always the case and ICS/IFS are at times used to effectively create a profile of a standard. ICS/IFS information from the filled templates is as a rule used to drive the test case selection and the parameterisation of test cases.

The test suite structure is a tree structure where the root represents the whole test suite and the leaves are the test purposes/test cases. The nodes represent test groups which either contain subgroups (i.e. other non-leaf nodes) or test purposes/cases (i.e. leaf nodes).

The usual approach is that the test suite structure follows from the features to be tested and the kind of testing. For example, test cases for a feature are grouped together and the test cases testing one feature may be further structured in groups testing valid, invalid or inopportune behaviour.

Test purposes (TPs) are meant to describe what needs to be tested under which conditions, avoiding going into details on how the tests are to be performed. Almost as a rule TPs are developed as a dedicated effort in an STF or some other team. However, the Technical Committees review the TPs quite carefully. On that level, the delegates can understand quite well the text describing the TPs and can relate it well both to the standards and to the products that their companies are building. With the classical (manual) approach to test development, the interaction between TP developers and reviewers is most important and readability of the TPs is crucial for that.

A TP has a unique identifier that is derived from the test suite structure. A TP has to contain the initial and post-conditions and the requirement to be tested including pass criteria, preferably with references to the standard or to the requirements catalogues. The TPs can be expressed in the dedicated language TPlan ES 202 553 [i.7] or in plain text. TPs in plain text are often structured in tables so that above elements can clearly be seen.

Following the TP development is the stage of selecting the tests that are to be developed as test cases written in TTCN-3 language ES 201 873-1 [i.5] with the intention of being used in some conformance certification scheme. This phase is very important to companies and this is where a lot of their attention and energy is going. This level of discussion is more often than not based on company interest at a given point in time.

For some technologies the TP development is followed by a detailed textual description of test cases denoted as TDs. Such TDs are very carefully reviewed by standardization delegates before they get approved and published. In fact, all that was previously said for the TP review would apply to Test Description review. Such detailed TDs are used as input in writing TTCN-3 test cases. It is important to note that development of 3GPP test specifications includes the development of detailed test descriptions. As this is by far the biggest test specification effort in ETSI, it would be important to look at that aspect when it comes to MBT use.

The "Specification of Test Cases" comprises the implementation of TPs (and at times TDs) by TTCN-3 code. In general, each TP is implemented by one TTCN-3 test case. Almost as a rule TTCN-3 test cases are developed manually as a dedicated effort in an STF. In contrast to TPs and TDs, the Technical Committee walkthrough review of the code is rather superficial and the work of testing experts is accepted as a starting point for the test case validation.

Current practice is to structure the test cases into three distinct parts of the code. One is a preamble that drives the implementation into the initial condition for the test. The other is the test body that implements the behaviour testing the corresponding TP, including setting of the verdicts. The third part is the postamble that is supposed to drive the implementation back into a stable testing state such that other tests could be run from that state.

In the "Validation" step, the TTCN-3 test cases are made executable and are executed on test equipment against more than one SUT provided by ETSI members. If the results obtained are the same, the test case in question can be declared validated. If there are differences, the analysis of respective traces needs to determine the cause of differences. The source of the problem may be in the implementation that did not respect the standard, in the test case code that may either be wrong or incomplete or in the standard that may be imprecise, incomplete, ambiguous or containing conflicting requirements.

5.2 Integration of MBT into the ETSI test development process

A vision on how MBT could be integrated into the ETSI test development process is shown in Figure 2.

The MBT step "Modelling for Testing" is based on the standard and the corresponding requirements contained in the standard. As the modelling formalizes behaviour described in the standard and relates requirements to the model, it can be seen as an additional validation step for the standard itself and the requirements. Problems and ambiguities may directly be fed back to the responsible committees at ETSI. The result of the modelling step is a model, which serves as input for the test generation.

The "Test Generation" step comprises the tool-based generation of test cases. Tools provide different strategies for test generation and for defining the termination criterion for the generator, e.g. certain coverage should be achieved. However, during generation problems related to state space explosion or reaching the termination criterion may be detected. For example, certain requirements cannot be reached or the generator does not terminate. Reasons for such failures may be related to ambiguities in the standard or inappropriate modelling of requirements. As a result, feedback on requirements and standard may be given and the test generation will be repeated with an updated model and/or other termination criteria.

The model should contain only elements that affect the behaviour that is required and will be tested by generated tests. For example, payload that does not influence the behaviour may not be modelled. As a consequence, the "Transformation & Adaptation" step bridges the gap between automatically generated test cases and ETSI TTCN-3 test cases. In addition, further documents like TSS, TP and TD descriptions may have to be generated. Whereas the generation of TSS, TP and TD descriptions may be straightforward, closing the abstraction gap may require to add information, e.g. about test environment and the SUT, and to restructure the test cases descriptions, e.g. test behaviour may have to be parameterized and behaviour may be structured into preamble, test body and postamble.

Whenever possible, the "Transformation & Adaptation" step should be done automatically to avoid inconsistencies between the documents needed in an MBT-based ETSI test development process.

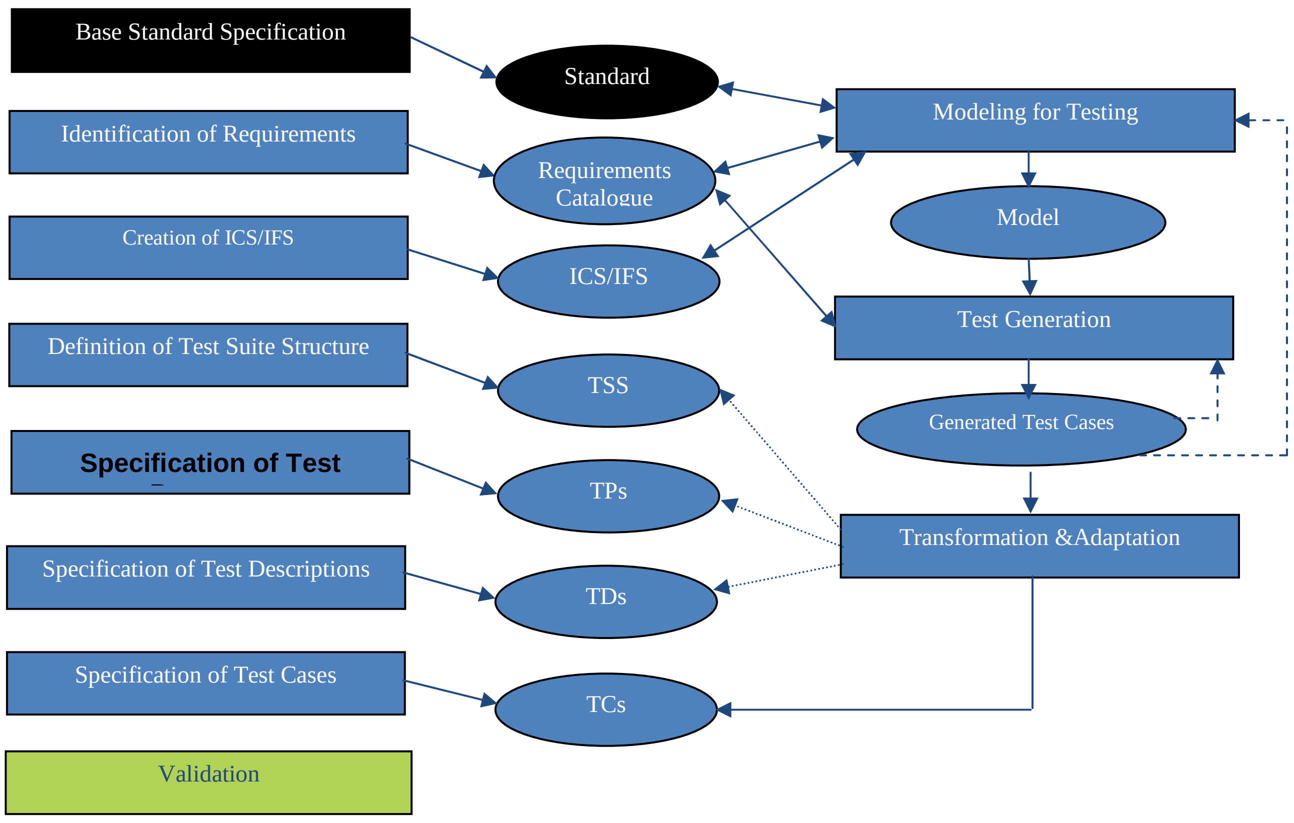
Base Standard Specification
Identification of Requirements
Creation of ICS/IFS
Definition of Test Suite Structure
Specification of Test
Specification of Test Descriptions
Specification of Test Cases
Validation
Standard
Requirements Catalogue
ICS/IFS
TSS
TPs
TDs
TCs
Modeling for Testing
Model
Test Generation
Generated Test Cases
Transformation &Adaptation

Figure 2: Using MBT within the ETSI test development process

Test cases derived using MBT approach would have to be validated just like test cases that are manually developed.

6 Modelling for testing

Model-based conformance test development includes creation of a behaviour model of a system to be tested. This model is used further as an input for test generation. This clause presents guidelines on model creation.

6.1 Identification of requirements

To be used in conformance test development process, the model for testing should adequately reflect the requirements of the standard, for which conformance tests are to be developed. It is precisely the standard requirements that need to be checked by the tests generated from the model, so they are the main input for model development.

To make this input available and to provide base for accurate traceability of both the model and the generated tests, one should identify the requirements, and preferably collect them in a requirements catalogue. Requirements in the catalogue should posses the following characteristics.

- Each requirement is usually an excerpt (most often, textual, maybe, including formulas or diagrams) from the standard text, containing usually one specific statement on the conforming system behaviour.
- Implicit requirements implied by the standard text and usage context of conforming systems are made explicit and put in the catalogue as others.
- Each requirement has a unique identifier for referencing. If the standard under consideration uses references to others and requirements may originate from different documents, it is recommended that requirement identifier contains information on source document.
- To facilitate management of large numbers of requirements, they can be grouped according to various criteria (functionality, source documents, roles, various kinds of requirements, etc.). Those groups can make up a tree-like hierarchy (or several hierarchies based on different criteria).

- Requirements stating effectively the same should be united in a special group (possibly under the single identifier) provided with the most precise statement expressing them all.

- Applicability context of each requirement - the full set of conditions when this requirement is applicable - should be specified explicitly.

- Each catalogue has a set of requirements attributes. Each requirement should have a value for each attribute. The following attributes are used most often.

  - Role(s) to apply to:

    Components of complex systems are usually play some roles; each role corresponds to a set of functions performed. A requirement usually provides restrictions on specific role(s), not on all.

  - Modality:

    A requirement can be obligatory (specified using words like "shall", "must", "always", "definitely", "never", "required", etc.), recommended (specified using words like "should", "ought to", "recommended", etc.), or optional (specified using words like "may", "might", "possible", "permitted", etc.).

  - Testability:

    Testable requirement has a definite finite procedure, which allows an arbitrary adequately experienced person to determine whether this requirement holds or not. Non-testable requirements are sometimes admissible. They usually contain some general statements, which is hard to check in an objective way.

- Requirements applicability conditions can include some options, e.g. each time an optional requirement A holds, a requirement B should also hold. Requirements, which applicability depends on some options, are called conditional. Usually each option being implemented implies applicability of a certain set of requirements. These dependencies between requirements should also be extracted from the standard and clearly stated in the catalogue.

- Other relations between requirements, e.g. one requirement refines another, formulates alternative behaviour, is similar, but differs in some detail, etc., can be stated in the catalogue. They become useful when the standard is changed and consistent modifications should be performed in different parts of the text. Understanding of such relations also helps adequate understanding of the standard as a whole.

6.2 Requirements modelling

To produce a correct model of the requirements, a modeller should adequately understand their meaning and accurately implement it in the modelling language. Adequate understanding of requirements requires thorough analysis of their statements and interrelations, taking into account also usage context and domain rules and restrictions.

Possible issues and problems in requirements representation, which may cause misunderstanding, modelling errors and so inadequacy and incorrectness of the generated tests, include the following:

- Ambiguities of requirement statements, possibility of different interpretations of the standard text (different ways of modelling, which leads to externally observable differences in behaviour). Each ambiguity should be analyzed for its reasons and possibility to remove it.
  It may mean just an intentional non-determinism leaving reasonable freedom for standard implementers. In this case the model should allow the same non-determinism and the tests generated should accurately distinguish all possible correct behaviours from incorrect ones. Another possible resolution in such a situation is to make the model more abstract to remove details distinguishing different results of non-deterministic behaviour, so the resulting model becomes deterministic.
  In other cases, when ambiguities are problematic, they should be removed or refined to unambiguous statements. To do this a modeller may use more specific related requirements from other parts of the standard (for this reason dependencies between requirements from different clauses are useful), understanding of the context and domain rules, and additional information from standard developers and domain experts.

- Inability to design a clear checking procedure for a requirement marked as testable. This is a sign of a problem either with requirement statement, or with the abstraction level chosen for a model. To resolve it the same actions can be performed as for ambiguity of a requirement.

- Incompleteness of requirements, absence of behaviour-related descriptions for specific situations. Such incompleteness can also be deliberate and mean that no restrictions are put on the corresponding behaviour, or it can be a problem to be fixed. The same actions as for ambiguities are helpful.

- Inconsistencies between different requirements statements. Again, some inconsistencies can be just seeming and be caused by inadequate understanding of standard details or contexts of contradicting statements, others are problematic and should be resolved by taking into account context and domain knowledge, or additional information from standard authors and experts.

- Lack of reasonable integrity between different parts of the standard, when very similar things are represented differently, the same procedure is described several times in different sections, etc. Usually this means that the standard text is not mature. Resolution of this issue and making adequate decisions on what things are actually the same and what should be distinguished usually requires deep understanding of the domain and discussions with standard developers and other experts.

Of course, any issues with standard requirements discovered during modelling and recognized by standard developers and experts should be fixed in the standard text to prevent further misunderstanding and to save time and effort of other people, who will work with the standard.

6.3 Identification of modelling strategy

Modelling strategy includes the decisions of the model scope, interface, abstraction level, modular structure and architecture. These decisions have a significant influence on the results of modelling and the effectiveness of the model in test generation. Changing such decisions on later phases of model development may be quite laborious and increase the risks of introducing modelling errors.

This clause provides some guidelines on making these decisions:

- **Model scope**

  Standards often define rather complex behaviour with a lot of separate elements and scenarios. To make modelling and test development more manageable a model scope - a part of conforming system behaviour to be modelled - should be selected as a first step. Usually this decision selects a part of the system behaviours and corresponding set of externally observable events.
  If possible, it is preferable to separate a complex behaviour described in a standard into several parts and to model them separately.
  Such a selection or separation should be guided by two factors: necessity to model the behaviour to be checked by the resulting tests and the cohesion of elements of conforming system behaviour - the selected separate part should usually have weak relations with behaviour that is left out of the model.
  Sometimes the alternative is to model the complete behaviour related with a set of external events (all possible system reactions on all possible combinations and sequences of these events) or only some scenarios of conforming system behaviour related to these events. The first approach is preferable, since it provides more accurate coverage of system behaviour and result in more adequate tests generated. The second approach may however be chosen due to lack of budget and resources for complete modelling. In this case modeller should realize possible lack of coverage in the resulting tests of specific situations not touched by the modelled set of scenarios, so the selection of scenarios to model should be very accurate.

- **Model interface**

  Any standard always specifies reference point(s) between communicating entities, where their behaviour can be observed and checked. Such points of control and observation are usually described in terms of ports, operations or methods that can be invoked, commands that can be given, results that can be achieved, messages or events that can be processed. Representation of all these things in a model is a model interface.
  The interface can be extracted from the standard by selecting operations, commands, or events created by an environment, on which a conforming system should somehow react, and operation results or externally visible events created by a system.
  The set of operations and events chosen as interface ones depend on the model scope. Their specific composition and data structures used also depend on the abstraction level chosen and on the modularization of the model.
  Operations and events related to the behaviour outside of the chosen model scope are not to be included in the model interface. Correct processing of events left out of the model scope may have to be ensured on the adaptation phase.

One more factor to be considered for interface definition is the effort needed to connect the chosen interface to an implementation under test during test execution. A modeller should examine possible adapters and try to choose model interface in such a way that minimize of their development effort.

- **Model abstraction level**

  Abstraction level of a model determines the details of modelled part of conforming system behaviour that are presented in the model. While the model scope determines the mere set of behaviour elements included in the model, the abstraction level determines the details of their presentation. These details are mostly related with composition and processing of data structures used as types of interface operation and events parameters and results, and also as types of internal data used to store history-related information.
  The choice of the abstraction level is usually prescribed by the constraints to be checked by the resulting tests - precisely that details should be modelled that help to check the desired properties in tests, and all the details of behaviour, which has no relation to the properties to be checked, should be omitted. The abstraction level chosen for a model further becomes the abstraction level of the generated tests.

- **Modular structure of a model**

  In case of high complexity of behaviour modelled developing a model as a single monolithic module increases the risk of making errors and complicates further modifications. Designing an appropriate modular structure of a model helps to cope with these problems. The modular structure defines specific set of modules comprising the model and possible interactions between them.
  Decomposition of a model into modules, along with selecting the part of the behaviour to model, and choosing the appropriate abstraction level are all the techniques to cope with model complexity. Decomposition into modules is also a way to organize collaborative parallel work of several developers on one model.
  Usually the modular structure can be borrowed from the modular structure of the standard - if it describes a complex behaviour, it has to be modularized somehow. However, it may appear that the same or very similar constraints are stated in different requirements of the standard and concern different (at least, at the first look) situations. In this case it is obligation of a modeller to make a decision whether to put their description in one module or in different ones. Such a decision should take into account effectiveness of model analysis performed by the test generation tool used, and also information obtained from experts and standard authors on future development of the standard to make introduction of most probable changes in the model as easy as possible.
  One more situation where decomposition into modules can be suggested by external factors is use of parts of the modelled behaviour in other standards - in this case such parts become obvious candidates to be separated into modules, and these modules may be useful for reuse in other projects.
  Additional generic guidelines for choosing modular structure of a model are the same as in general software engineering - anticipate most probable future changes, write one knowledge in one place, partition complex tasks, etc.
  When deciding the model structure, consider any limitations that the test generation tool may have.

Less influential decisions that can be made and changed later in the process of model development, but has great significance for the results of modelling, include model instantiation and parameterization:

- **Model instantiation**

  Model instantiation determines the number of instances of model objects or modules used in test generation process. It is defined on the basis of model modular structure, the desired test coverage (since the set of situations reachable depends on the objects used in test generation) and capabilities of used test generator (since complex object structures can be very hard to analyse).
  Test configurations developed for the standard under consideration can be a good input for making decisions on model instantiation.
  For example, in ITS case study considered in DTR/MTS-00141 [i.3] the protocol unit model can have only one instance used for test generation, or can be represented by several interacting instances. In the first case the set of reachable behaviour scenarios is smaller and is completely determined by the variety of messages used as inputs for the single model instance. In the second case the variety of possible scenarios grows rapidly with the number of instances used, because of possible asynchrony in their communications. So, the second case is much harder for automatic test generation. Nevertheless, the ITS test purposes developed by TS 102 871-2 [i.6] use 4 manually defined test configurations, each consisting of several protocol unit instances.

- **Model parameters and options**

  Definition of possible model parameters and options, which may be transferred to the generated test suite, should be based on possible variety of conforming implementations. Optional or conditional requirements can be modelled under control of model parameters, which values can be set by a special probe test examining the corresponding features of the implementation under test before execution of main test cases.
  In such a way options related with ICS and parameters related with IXIT can be included in a model.
  Possibility to define such parameterization of a model depends on the test generation tool used.

6.4 Annotating models with references to requirements

Referencing requirements from a model contributes considerably to the quality of the model development as the model developer or readers can quickly check that all the requirements are modelled and there are no model elements for which no related requirement can be identified. To support an accurate referencing along with traceability of tests and failures found by them to the initial standard requirements, the model should be annotated with references to the requirements. Model-based testing tools complying with ES 202 951 standard on modelling notation [i.2] should support such annotations.

The annotation with reference to a requirement should be placed on the appropriate model element. Specific element to choose (state, transition, action, constraint, etc.) depends on the tool capabilities; in any case such an element should be the most relevant one among the elements that allow such annotations. For tools supporting coverage targeted test generation or coverage measurement, it is recommended to place annotations in such a way that the coverage measured by the tool according to the annotated elements is as close as possible to coverage of requirements (see clause 7.1).

Reference to a requirement may have a form of its identifier in requirements catalogue, or corresponding section and paragraph numbers, or full citation of the corresponding element of standard text, or some mixed or intermediate form. The specific form can be selected on the base of tool capabilities and possible increase or decrease of effort of test maintenance and test execution report analysis. Sometimes it is better to use only requirements identifiers, because texts make tests and test reports too large and unmanageable, sometimes it is preferable to have full text in tests and reports, because they remain compact enough and developers spend less effort on their maintenance and analysis, if they have no need to look often into other documents.

For better maintainability and readability the chosen form of requirement reference should be the same throughout the whole model and throughout the set of models if there are several models describing different parts and aspects of the same standard.

Accurate referencing requirements from a model can help in detecting complex issues. For example, if an execution of a model exists, which doesn’t touch any model element with annotation referencing to a requirement, this situation requires additional analysis. It may be a modelling error, which should be somehow fixed, or it may also be caused by an incompleteness of standard requirements.

6.5 Modelling guidelines

Model maintainability effort depends on model readability; the less readability implies the greater effort needed to make any modification. The usual guidelines increasing readability of texts and graphical models include the following recommendations:

- Names of various elements of a model (states, actions, variables, parameters, functions, etc.) should be self-explanatory, close to the names of the corresponding entities (if any exist) in standard requirements, and at the same time as compact as possible.

- To decrease the effort of understanding and analysis of models, it is recommended to obey the following restrictions:

  - Number of states or any other solid graphical elements (excluding linear ones) on a diagram should be not greater than 6. Such solid elements should not overlap, unless it is required by their semantics.

  - Transitions or other lines on a graphical diagram should be positioned with as small number of intersections as possible. They should not lie above or below solid graphical elements, unless it is required by their semantics.

    - Functions or methods should have not greater than 5 parameters; it is preferable to decrease as possible the number of functions with more than 2 parameters.

    - Textual description of a single function or method should not exceed 100 lines and should not contain more than 5 branching statements or more than 3 loops.

    - Complex expressions consisting of more than 5 operands should be avoided and replaced with auxiliary functions if possible. If expression contains only uniform operators (only conjunctions, or additions, etc.) this boundary can be increased.

  However, these recommendations may be hard to adhere, if the complexity of a modelled behaviour is high. In such a case a modeller should decide which restrictions should be obeyed and which may be alleviated on the base of interests of model and test users and maintainers.

- Usual path of human perception of such a model tends to be left-to-right and top-to-down (in Western culture). So, the sequence of execution of model elements or the sequence of their most natural understanding should be laid close to those directions. Again, complexity of the model may require breaking this rule of thumb.

- The chosen rules and policies of model elements representation, naming, and layout should be used homogeneously as possible through the entire model and, maybe, through the set of models describing different parts and aspects of the same standard.

Additional recommendation concerning improvement of model maintainability is related with logging of test events. Logging can be very helpful in debugging the complex behaviour. In case of sufficient logging support in the tool used, use of tool capabilities is preferred. Otherwise, if logging can help in model debugging, even in case of future modifications, it is recommended that a modeller designs a logging of main test steps and events on the model level, so that the interpretation of logs in terms of model is easy and natural.

6.6 Model quality

Model quality is determined by attributes related with the ability of the model to be used for effective conformance test generation and to be maintained and evolved with preservation of this ability and with adequate effort:

- Characteristics, related with model ability to be used for effective test generation, include the following ones:

    - Correctness: whether a model correctly represents the standard requirements and other rules and constraints of the domain.

    - Functional completeness: whether a model describe the full set of the constraint on the conforming system behaviour on the chosen abstraction level, or only a part of them; whether it describes all the possible situations of conforming system interaction with environment, or only a part of them.

    - Traceability to standard requirements: whether a model contains adequate references to standard requirements, and what part of model elements has such references.

    - Processability: whether a model can be processed effectively by tools to generate necessary tests.

- Characteristics, related with model maintainability and evolvability:

    - Readability and intelligibility for human readers. Includes adherence to suitable naming conventions, demonstrativeness of graphical diagrams, and clarity of textual parts of model.

    - Maintainability related with effort required to fix errors, make minor changes, and low-level refactoring, and also with effort required to add new features and modules, to make high-level restructuring of a model.

    - Integrity related with consistency of representation of different model element and consistency of rules and policies, which should be obeyed to make correct modifications and additions in different parts of a model.

The presented list is not complete, some other attributes and characteristics that are significant for model quality may be extracted and added to it.

It is hard to provide general guidelines for objective measuring or achieving higher levels of these quality attributes. Some guidelines of the second kind are presented above, but the specifics of the project and domain, complexity of the modelled behaviour, and project environment influence all these attributes significantly, so any general recommendation may fail in some specific situation.

7 Automatic test generation

This clause provides guidelines on test generation in the process of model-based conformance test development.

7.1 Defining test coverage

Test coverage achieved is one of the main characteristics of the tests, and test coverage criterion chosen as a target for test generation is the main means of developer control over test generation process and the main input to this process in addition to the model used.

Test coverage plays two roles in test development: a final measure of the completeness and adequacy of the tests, and a target value of such a measure to be achieved by test development. In both cases there are no absolute expression of test coverage, specific test coverage value can be achieved according to some coverage criterion. Such a criterion should provide some classification of possible system behaviours, so that one can evaluate the extent of a set of behaviours observed during testing with respect to all possible behaviours. Usually this classification is based on (classes of) situations - each (class of) situations correspond to some set, sequence, or more complex aggregate of events occurred during system operation.

Coverage criterion is a rule, which can be applied to some class of systems to define a finite set of situations related with system behaviour. Each scenario of system behaviour can be further considered as corresponding to one or several situations defined by a criterion. Test coverage achieved according to such a criterion is the percentage of these situations actually realized during test execution. Since coverage criterion is applicable to many systems, occasionally some of situations it defines for specific system cannot be achieved in any possible its execution. In this case such unreachable situations are removed and test coverage achieved is calculated using only the set of reachable situations. This rule ensures that less than 100 % coverage according to a criterion always means that some reachable behaviour is not covered and makes understanding of coverage reports easier.

When a coverage criterion is used as a target for test development, developers try to ensure that a certain value (e.g. 100 %) of test coverage according to this criterion will be achieved by the tests developed.

The following list provides non-exhaustive set of coverage criteria examples:

- A situation corresponds to realization of applicability conditions of some requirement. The full set of situations corresponds to the full set of requirements. This is so called requirements coverage criterion.
- A situation corresponds to execution of a transition in a state machine model. The full set of situations corresponds to all transitions. This is so called transition coverage criterion.
- A situation corresponds to execution of a statement of textual code. The full set of situations corresponds to all statements. This is so called statement coverage criterion.
- Cards, which can be inserted into an ATM, can be valid or invalid. This gives possibility to use coverage criterion classifying all possible interactions with ATM in two classes: whether a valid or invalid card is used. This is an example of coverage criterion based on input data partitioning.
- If a system code has an expression x + y, its behaviours may be classified as follows: values of x and y used are such that the result of x + y is different from 0; x + y is different from x; x +y is different from y; x + y is different from x - y; x + y is different from x * y; x + y is different from x / y. This is an example of mutation-based coverage criterion - one designs a set of possible simple faults called mutations and measures test suite adequacy by its capability to detect such faults.

Several coverage criteria can be combined in a mixed one in many ways. Such mixed coverage criteria are more suitable to evaluate test adequacy for complex systems with many different aspects of behaviour:

- A mixture coverage criterion contains all the situations, which are defined by initial coverage criteria. A set of situations of this criterion is simply a union of sets of situations of original criteria.
  For example, one may decide to measure test adequacy in testing based on state machine model by combination of requirements coverage and transition coverage: 100 % coverage here means that all requirements are covered and all transitions are executed.

- A product coverage criterion contains all possible combinations of situations of the initial criteria.
  For example, one may partition ATM cards into invalid ones, valid ones with zero balance, valid ones with small (< $100) balance, and valid ones with significant (>= $100) balance. Further one may try to cover all possible combinations of transitions performed in ATM state machine model and card classes. In this example not all theoretically possible combinations can be realized - usually one cannot request money with an invalid card. Usage of product coverage criteria often requires accurate detection of such unreachable situations, which should be removed from the criterion.

- Sequences coverage criterion contains all possible sequences of certain length of situations of one initial coverage criterion.
  For example, to test rather complex behaviour one can use pairs of transitions criterion - a situation in this criterion corresponds to an execution of a pair of consecutive transitions in state machine model. Again, like with product criteria, not all theoretically possible sequences can be reachable in practice; such unreachable situations should be detected and removed from coverage criterion.

The choice of specific coverage criterion as a test construction target is one of the main decisions during test development. In conformance testing it should be based on accurate understanding of conforming system behaviour and possible serious breaches of the standard, which may prevent interoperability of systems in the domain. Requirements coverage criterion is recommended as a starting point. If necessary developer may add more details from model to be taken into account, until the coverage criterion becomes a good reflection of what is considered as really adequate tests.

Choosing coverage criterion (both as a measurement and as a target) in model-based testing projects should take into account effort needed to evaluate it adequately. Use of model-based testing can lead to large numbers of tests and may involve rather complex models. In such situation it is important to have some reliable automation for coverage measurement, or, at least, it should not use too laborious manual procedures, lest evaluating coverage achieved will require a lot of effort and may get unreliable results. So, the choice of coverage criteria in case of working with complex models and large amounts of tests should be based on reliable tools capable to measure it or may include only as simple as possible manual procedures (also for the possibility to double-check evaluated coverage if necessary).

7.2 Generating test cases

In most cases, test generation with the help of an MBT tool is rather straightforward: a developer provides a model, chooses target coverage criterion, and performs test generation by the tool. However, model complexity or some other issues may create obstacles on this way. This section provides recommendations on possible overcomes of such issues.

A situation when the test generation tool used directly supports generation targeted on the coverage criterion chosen is the most simple. In case of lack of such support the following solutions are possible:

- Choose the most close to the chosen one coverage criterion supported by the tool as a target for test generation.

- If the used tool supports test generation targeted at labelled element coverage, test developer can design and distribute in the model a set of labels corresponding to the desired coverage criterion.
  This way may require inserting additional model elements or making instrumented descriptions of particular scenarios (i.e. insert auxiliary variables serving as counters and flags and statements checking values of these variables) just for the sake of enabling the tool to cover them.

- If the used tool supports test generation targeted not by a coverage criterion, but some other technique, the developer should use this technique in a way that gives results as close as possible to satisfy the criterion chosen. The skills needed for this are related with deep knowledge of tool capabilities.
  For example, test generation may be controlled by specification of the set of generic or parameterized scenarios chosen from the whole model behaviour, which becomes the base scenarios of possible tests. In this case the developer should design a set of scenarios so that it contains all the situations to be covered targets the tool to generation of the desired set of tests.

In case when model complexity makes generation of proper tests problematic the techniques for coping with model complexity presented in clause 6.3 should be applied: partitioning of a model into several separate models, choosing more high abstractions to represent the desired behaviour, decomposition of a model into modules.

When complex objects are used as input data in the model, additional work may be required to design the necessary test data. In particular case when test generation is hindered by complex constraints on input data it is useful to create a set of test data pools for complex data types used in input events or operations and to force the tool to use those data pools in test data generation (sometimes re-working of model interface may be required to implement this approach). Of course, this way requires resolving manually at least some of the test data constraints ensuring achievement of desired coverage goals.

7.3 Quality of generated test cases

Quality of generated tests is determined by attributes related with the ability of the generated tests to be used for effective conformance testing and to be maintained and evolved with preservation of this ability and with adequate effort:

- Characteristics, related with test suite ability to be used for effective conformance testing, include the following ones:
    - Correctness: whether a test suite correctly checks the standard requirements and other rules and constraints of the domain.
    - Functional completeness: whether a test suite checks the full set of the constraint on the conforming system behaviour on the chosen abstraction level, or only a part of them.
    - Test adequacy or test completeness: whether a test suite provides complete testing according to a chosen coverage criterion, or some situations are missing.
    - Traceability to standard requirements: whether a test suite contains adequate references to standard requirements, and what part of tests has such references.
    - Portability: whether a test suite can be used to check any implementation of the standard under consideration, or has some restrictions on implementations under test.
    - Usability: how much effort, what skills and knowledge are needed to run the tests, to develop additional adapters, if needed, and to understand their results adequately.
    - Readability and intelligibility for human readers in case of analysis of test behaviour related with failures detected is required.
    - Report completeness: whether the reported data contains all the necessary information on test runs (is a run successful or not, what part of tests are executed, and what aren't and why, what are the failures detected and in what specific situations, what is the resulting test coverage according to some criteria).
    - Report readability: whether the reports created are easy enough to comprehend.
- Characteristics related with test suite maintainability and evolvability. The main benefits of model-based testing are related with transfer of most of maintenance tasks to models. Nevertheless tests suites should be maintainable, and maintenance tasks implemented manually for them should be minimized:
    - Readability and intelligibility for human readers in case of analysis related with possible adaptation or transformation is required. This attribute is closely related with suitable and uniform structure of generated tests, traceability of generated code to requirements and model elements, adherence to naming conventions for functions and variables used in tests, convenient commenting of generated code, etc.
    - Effectiveness of debugging the generated test if such need occurs. Since adaptation and transformation can introduce errors in the tests, their structure and organization should support effective detection and localization of such errors. One of the suitable means for that is logging, which can be switched on if necessary and generate logs with information needed to trace and locate various issues.

The presented list is not complete, some other attributes and characteristics that are significant for test suite quality may be extracted and added to it.

8 Transformation and adaptation of the generated test cases

Model-based testing (MBT) tools generate a set of tests from the model. The tools allow the users to automatically transform generated test cases in different useful formats in order to document the test cases and to be able to execute them. Regardless of the chosen testing language the generated code needs further adaptation until it becomes executable. The reason for this is that the models are usually on a high abstraction level focusing on data and behaviour that is important for the particular test goals, consequently the generated tests will also be on the same abstraction level and details that were left out during modelling need to be added later to get an executable test suite.

Transforming the generated code can be performed not only for adapting the generated tests to get executable tests, but also to improve their quality. For example, the readability of the generated code can be improved by some renaming, restructuring of the test cases can enhance maintainability and introducing parameterization makes the executable test suite more usable.

8.1 Transformation and adaptation steps

Please note, that the transformation and adaptation steps mentioned here are not always needed and are therefore optional. The order of the clauses does not impose any order for the execution of the steps.

8.1.1 Adaptation to the target test environment

The output of the MBT tools is a generated test suite, where each test case has its test logic that is working with messages to be sent and expected answers. Since the models are usually on a high abstraction level, the generated test cases will be on this high abstraction level as well. Some automatic code generation is supported in most cases by the tools. In order to get an executable test suite the level of abstraction of the generated code should be lowered by making them more concrete, for example by adding some extensions.

In order to keep the consistency between the model and the generated code, it is not recommended to directly modify the generated code. The reasons are obvious: the modifications will be lost, when a new test (and code) generation is performed. In order to avoid this situation some tools are generating code which is using a clear API, and the extension should implement this API. If it is not the case, it is still practical to design such an API between the generated test logic and the required extensions in order to keep them as clearly separated as possible. These extensions with a well-defined API are often called test frameworks. Using the OSI terminology ISO/IEC 9646-1 [i.4] the Abstract Test Suite (ATS) is built from the generated code and (an optional) test framework.

A test framework may have several responsibilities. Typically they are performing a two way transformation between the data structures of the messages on a high abstraction level in the generated tests and the data structures that are representing concrete, "real-life" messages. This transformation may include adding values to fields that were not modelled, or converting the generated values of fields that were differently modelled to keep the abstraction level of the model high. For example, the data structures that are used for concrete messages can be described in ASN.1. In communication protocol testing the test framework often implements the encoding and decoding of the low level data structures and usually has some means to use a transport protocol to send them out, or receive the incoming messages.

Test frameworks are typically manually developed, but provided that the API between the generated code and the test framework is not changed they can be reused without modification for further test generations. Although the test frameworks are usually project specific, if their provided API is suitable for models in other projects, they can be reused.

8.1.2 Parameterization

Parameterization of the generated test cases improves their adaptation to different contexts.

A parameter is a variable for which the value will be provided at testing run time. Parameters can be introduced both in the model, and they will then appear in the generated test suite, or in the test framework.

The method to add a parameter to the test framework is naturally depending on the language which is used for implementing the test framework. Some languages have sophisticated means and language structures to support parameterization, some others needs extra development (e.g. parsing a configuration file). An example for a test framework parameter can be TTCN-3 module parameters.

In order to get parameterization in the generated test suite code, the modelling language and the model based testing tool have to support this feature. In case there is no support for parameterization in the model, then it is still possible to assign some special values to the variables in the model. These special values will be present in the generated code of the test cases and therefore during the data transformation in the test framework the special generated values can be replaced with the values of parameters that are coming from an external source. Naturally, it is also possible to add parameters manually to the generated code, but this has to be done very carefully, because it is possible, that the manual additions will be lost after a new code generation.

As an example IXIT (Implementation eXtra Information for Testing) has information about the environment and the SUT (System Under Test), which is inevitable for an executable test suite. The IXIT parameters can appear either as parameters in the generated code or parameters in the test framework.

8.1.3 Renaming of identifiers

Normally the code generated by an MBT tool is meant to be transparent and not for the eye of the human users, but in some cases the readability of the generated code can be an important issue. Standardized test specification development for example is an area, where the readability of the generated code has to be seriously considered, since today this is the result that will be standardized.

Good naming of the identifiers promotes comprehending the generated code. Suitable names of the test cases help understanding the goal of the test case.

8.1.4 Structuring of test behaviour

Structuring the test behaviour can improve readability and reusability. For example, in the OSI conformance testing methodology ES 202 553 [i.7] the tests are divided into three sections: preamble, test body and postamble. The preamble drives the SUT into the state required to perform the test body, then the test body is executed to verify the test goal, finally the postamble cleans up and restores the SUT to its initial state.

There are MBT tools that support generating a test suite so that the test cases are organized as preamble, test body and postamble. In some cases the tools don’t have this feature and therefore some further steps need to be taken to transform the generated test suite and structure it according to the desired way.

8.1.5 Parallelization

Using concurrent test configurations, where the test behaviour is executed in parallel, has several benefits: it enables to handle some SUT non-determinisms (for example there can be cases where the order of the expected answers is hard to guarantee) and identifying behaviours executable in parallel often improves readability and reusability.

TTCN-3 allows creating concurrent test configurations, where the test components are executed in parallel. As of today the MBT tools are normally not supporting code generation for concurrent test execution. In case it is required the generated code can be either transformed, or the test framework can be developed in a way, that it takes care of the creation of parallel test processes and handles their synchronization.

8.1.6 Logging

Log statements are not only making the generated code more readable, but they also help during the analysis of the test results. Adding logging to the following actions is recommended:

- Sending/receiving messages.
- Verdict changes.
- "Executing" requirement annotations in the model during test execution.

8.2 Quality of the transformed and adapted test suite

Basically the quality characteristics described for the generated test cases (see clause 7.3) apply here as well.

It is important that the adaptation and transformation of the generated test suite should not decrease the quality of the generated test cases.

From maintainability point of view it is obvious that any manual (not automatic) transformation of the generated code can decrease maintainability, since the modification of the model, or the (re-)generation of the test suite may overwrite the manually added transformations. But in the case of manual transformation the consistency of the transformed code and the model (and the test suite, etc.) should somehow be preserved.

History

Document history		
V1.1.1	February 2013	Membership Approval Procedure MV 20130402: 2013-02-01 to 2013-04-02
V1.1.1	April 2013	Publication