

# Comments to Draft 1 of ARINC Project Paper 653 Part 3, Conformity Test Suite

Aliki Ott  
Universität Bremen  
Department for Mathematics/Computer Science  
Research Group Operating Systems and Distributed Systems  
tsio@tzi.uni-bremen.de

June 11, 2005

## Abstract

This document summarizes comments on draft 1 of ARINC Project Paper 653 Part 3 *Conformity Test Suite* ([1]). When considering the behavior of API service calls, draft 2 of supplement 2 to ARINC 653 ([2]) is used to determine how the OS is expected to behave. It is assumed that for each API service first the error section is considered and then the normal behavior section.

Changes to draft 2 of supplement 2 to ARINC 653 (part 1) that have been proposed at the Lisbon meeting (June 7-9, 2005) are not considered.

Questions and comments regarding [2] are denoted in blue.

Comments whose solution depends on questions to be answered first are denoted in red.

## Contents

<b>1</b>	<b>Comments to Compliance Test Concept</b>	<b>3</b>
1.1	Finding adequate configurations . . . . .	3
1.2	Capturing test results . . . . .	3
1.3	Restrictions of the Concept . . . . .	3
<b>2</b>	<b>Comments to 3.9 Test Sequence</b>	<b>4</b>
<b>3</b>	<b>Comments to Test Procedures</b>	<b>4</b>
3.1	General comments . . . . .	4
3.2	Comments to 3.1.2 (API/OS Management Types) . . . . .	4
3.2.1	Test Procedure T-API-GEN-001 . . . . .	4
3.2.2	Test Procedure T-API-GEN-001 . . . . .	5
3.3	Comments to 3.2.1.2 (Partition Management Types) . . . . .	5
3.4	Comments to 3.3.1.2 (Process Management Types) . . . . .	5
3.5	Comments to 3.2.2.1.2 (GET_PARTITION_STATUS) . . . . .	5
3.5.1	Test Procedure T-API-PART-0220:0010 . . . . .	5
3.6	Comments to 3.2.2.2.1 and 3.2.2.2.2 (SET_PARTITION_MODE) . . . . .	5
3.6.1	Test assertion table 3.2.2.2.1 . . . . .	5

3.6.2	Test Procedure T-API-PART-0230:0010 . . . . .	6
3.6.3	Test Procedure T-API-PART-0230:0020 . . . . .	6
3.6.4	Test Procedure T-API-PART-0230:0030 . . . . .	6
3.6.5	Test Procedure T-API-PART-0230:0035 . . . . .	7
3.7	Comments to 3.3.2.1.2 and 3.3.2.1.4 (GET_PROCESS_ID) . . . . .	7
3.7.1	Test procedure T-API-PROC-0220:0011 . . . . .	7
3.7.2	Service macro CHECKGET_PROCESS_ID . . . . .	7
3.7.3	Additional tests . . . . .	7
3.8	Comments to 3.3.2.2.2, 3.3.2.2.3 and 3.3.2.2.4 (GET_PROCESS_STATUS) . . . . .	8
3.8.1	Test Procedure T-API-PROC-0220:0011 . . . . .	8
3.8.2	Test Procedure T-API-PROC-0220:0012 . . . . .	8
3.8.3	Service macro CHECKGET_PROCESS_STATUS . . . . .	8
3.9	Comments to 3.3.2.3.1 and 3.3.2.3.2 (CREATE_PROCESS) . . . . .	8
3.9.1	Test assertion table conditions . . . . .	8
3.9.2	Test procedure T-API-PROC-0230:0011 . . . . .	8
3.9.3	Test procedure T-API-PROC-0230:0015 . . . . .	9
3.9.4	Test procedure T-API-PROC-0230:0012 . . . . .	9
3.9.5	Test procedure T-API-PROC-0230:0020 . . . . .	9
3.10	Comments to 3.3.2.4.1 and 3.3.2.4.2 (SET_PRIORITY) . . . . .	9
3.10.1	Test assertion tables . . . . .	9
3.10.2	Test procedure T-API-PROC-240:0016 . . . . .	10
3.10.3	Test procedure T-API-PROC-240:0026 . . . . .	10
3.10.4	Test procedure T-API-PROC-240:0030 . . . . .	10
3.10.5	Test procedure T-API-PROC-240:0023 . . . . .	10
3.11	Comments to 3.3.2.5.2 (SUSPEND_SELF) . . . . .	10
3.11.1	Test procedure T-API-PROC-0250:0020 . . . . .	10
3.11.2	Test procedure T-API-PROC-0250:0010 . . . . .	11
3.12	Comments to 3.3.2.6.2 (SUSPEND) . . . . .	11
3.12.1	Test procedure T-API-PROC-0260:0015 . . . . .	11
3.13	Comments to 3.3.2.7.2 and 3.3.2.7.3 (RESUME) . . . . .	11
3.13.1	Test procedure T-API-PROC-0270:0030 . . . . .	11
3.13.2	Test procedure T-API-PROC-0270:0065 . . . . .	11
3.13.3	Test procedure T-API-PROC-0270:0070 . . . . .	12
3.14	Comments to 3.3.2.8.2 (STOP_SELF) . . . . .	12
3.14.1	Test procedure T-API-PROC-0280:0020 . . . . .	12
3.14.2	Test procedure T-API-PROC-0280:0030 . . . . .	13
3.14.3	Test procedure T-API-PROC-0280:0040 . . . . .	13
3.15	Comments to 3.3.2.9 - 3.6.2.2.3 . . . . .	14
3.16	Comments to 3.6.2.2.4 (GET_QUEUEING_PORT_ID) . . . . .	14
3.16.1	Test procedure T-API-PROC-0600:0011 . . . . .	14
3.16.2	Test procedure T-API-PROC-0600:0012 . . . . .	14
3.16.3	Service macro CHECKGETQUEUEINGPORTID . . . . .	14
3.17	Comments to 3.6.2.2.5 (GET_QUEUEING_PORT_STATUS) . . . . .	15
3.17.1	Test procedure T-API-PROC-0610:0012 . . . . .	15
3.18	Comments to 3.7 and 3.8 . . . . .	15

# 1 Comments to Compliance Test Concept

## 1.1 Finding adequate configurations

In the compliance test concept, it is denoted (on p. 2) that a “representative set of configuration data” shall be used. It remains unclear if this means *exactly one configuration* or *a set of different configurations*. One configuration cannot be representative for all possible solutions. Furthermore, only very few data are provided with respect to the configuration data to be used. If different configurations shall be used, it should be considered that this requires a tool chain for compiling, linking and data loading which should allow automated execution of the conformance test suite. Furthermore, the effort for generating and choosing adequate configuration tables should not be underestimated.

In general, there are some elements that are not explicitly configured in the configuration tables (e.g., processes, buffers, etc.) but share the partition’s memory space. As a consequence, it may not be possible with a given configuration to create a set of processes and additionally a set of buffers or blackboards.

Suggested Solution: Explicitly state how many configurations shall be used and add more information about relevant (non-implementation dependent) configuration data. With respect to the competition of different elements for the partition’s configured memory space, a commentary or note should comment these configuration parts.

## 1.2 Capturing test results

It remains unclear how test results are logged outside the SUT. In particular, it is denoted (on p. 3) that “specific test instrumentations and details to enable the execution of the Test Application and to capture its results” should be contained in the test plan. It is unclear if this means that a specific API for test instrumentation services is required.

Suggested Solution: *At the Lisbon meeting it has been clarified that details with respect to logging of test results is outside the scope of the conformance test suite. Nevertheless, this should be stated explicitly.*

## 1.3 Restrictions of the Concept

It is stated on p. 3 that testing timing constraints and I/O behavior of the platform are not the scope of the compliance process. Nevertheless, the respective systems are usually hard real-time reactive systems which means that timing and I/O are important issues. I/O cannot be considered by this conformance test suite since a software test approach is used which means that specific interface drivers are not part of the SUT (i.e., the test candidate).

With respect to timing and performance, it should be considered to add adequate test procedures. In particular, since the list of service functionalities (on p. 9) already contains an item PERF for performance test procedures.

Suggested Solution: For considering timing and performance issues, it is suggested to define constant names for the minimum, maximum and mean execution time of each API service and state that the respective values are implementation dependent. These constants can then be used in the defined test procedures (e.g., if waiting or blocking shall take more time than a sequence of defined API service calls) and in additional performance test procedures.

*It has been stated in the Lisbon meeting that getting the values for the minimum, maximum and mean execution time is very difficult and depends on many parameters. Nevertheless, this is an implementation dependent detail.*

## 2 Comments to 3.9 Test Sequence

This section defines four sequences of testing the test procedures defined in Sect. 3.1 to 3.8. It is not stated explicitly that

- these phases shall be executed in a sequence since the first phase creates the elements necessary in the second phase, etc. and that
- the sequence of test procedures within one phase is defined in such a way that allows to execute the test procedures one after the other.

In particular, it is not stated that it is not necessary to reset the module by means of a hardware reset or `SET_PARTITION_MODE(COLD_START)` after each test procedure.

Suggested Solution: Explicit commentary to explain this.

## 3 Comments to Test Procedures

### 3.1 General comments

Some API services are not tested in partition's operating mode `NORMAL` and others not in partition's operating mode `COLD_START/WARM_START`. This contradicts the general possibility to execute all API services in all possible situations which is reflected by respective error/return codes. Furthermore, one objective of the test suite is to show full compliance with A653 ("In fact the test suite is a means to declare that the OS is fully compliant with the A653, being this status obtained when all the tests are OK.", [1], p. 393) which – to my understanding – is not ensured if such situations are not tested.

- API services not tested in partition's operating mode `NORMAL`:  
e.g., `GET_PROCESS_ID`, `GET_PROCESS_STATUS`, `GET_MY_ID`, `GET_*_PORT_ID`, `GET_*_PORT_STATUS`, `GET_BUFFER_ID`, `GET_BUFFER_STATUS`, `GET_BLACKBOARD_ID`, `GET_BLACKBOARD_STATUS`, `GET_SEMAPHORE_ID`, `GET_SEMAPHORE_STATUS`, `GET_EVENT_ID`, `GET_EVENT_STATUS`
- API services not tested in partition's operating mode `COLD_START/WARM_START`:  
e.g., `SET_PRIORITY`, `SUSPEND_SELF`, `SUSPEND`, `RESUME`

Suggested Solution: Two solutions can be suggested:

1. To add test procedures that perform these API services also in the other modes, (preferred by the author of this document) or
2. to examine if some of the defined test procedures can be executed in another mode with minor changes to the test description (Lisbon compromise).

### 3.2 Comments to 3.1.2 (API/OS Management Types)

#### 3.2.1 Test Procedure T-API-GEN-001

The objective of the test procedure is to test `RETURN_CODE_TYPE`, but the test description describes to instantiate a variable of type `OPERATING_MODE_TYPE`.

Suggested Solution: Rewrite the test description in the following way: "Instantiate a variable of the `RETURN_CODE_TYPE` enumeration type assigning it the following values iteratively: `NO_ERROR`, `NO_ACTION`, `NOT_AVAILABLE`, `INVALID_PARAM`, `INVALID_CONFIG`, `INVALID_MODE`, and `TIMED_OUT`."

### 3.2.2 Test Procedure T-API-GEN-001

Test procedure only checks that the enumeration type *contains* the required elements, but

- does not show the correct sequence of the elements in the enumeration type (which is necessary if a coding standard does not forbid to use numerical values instead of the element names) and
- does not show that no other elements can be used (which is important to ensure that the API services cannot return other return codes).

Suggested Solution: Add a comment that for showing that the enumeration type is identical to the one given in the ARINC 653 specification, additional code reviews are required that can show that no additional elements are contained in the enumeration type and that the sequence of elements is correct.

### 3.3 Comments to 3.2.1.2 (Partition Management Types)

See 3.2.2.

Suggested Solution: See suggested solution in 3.2.2

### 3.4 Comments to 3.3.1.2 (Process Management Types)

See 3.2.2.

Suggested Solution: See suggested solution in 3.2.2

### 3.5 Comments to 3.2.2.1.2 (GET\_PARTITION\_STATUS)

#### 3.5.1 Test Procedure T-API-PART-0220:0010

The test description requires that as a result of calling `GET_PARTITION_STATUS` it is expected to get “partition ID = 1”. On the other hand `PARTITION_ID_TYPE` is denoted as implementation dependent (cf. p. 10) and furthermore partition IDs are probably defined in the configuration table (which means that some valid and unique value has to be used, but not necessarily “1”).

Suggested Solution: Expected value is that the returned value complies with the value defined in the configuration table.

### 3.6 Comments to 3.2.2.2.1 and 3.2.2.2.2 (SET\_PARTITION\_MODE)

#### 3.6.1 Test assertion table 3.2.2.2.1

For requirement API-PART-0230:0030 it is allowed to have “ANY” as the current operating mode when requesting to switch to operating mode `WARM_START`. ANY means probably “any possible operating mode, i.e., `NORMAL`, `COLD_START`, `WARM_START`, `IDLE`”. In the description of this test assertion it is expected that the partition switches to `WARM_START`. But this is only possible if ANY does not mean `COLD_START` or `IDLE`. Particularly, since `COLD_START` as current operating mode and `WARM_START` as parameter is tested by API-PART-0230:0060.

Suggested Solution: Change ANY for API-PART-0230:0030 to “`NORMAL` or `WARM_START`”.

### 3.6.2 Test Procedure T-API-PART-0230:0010

1. In the environment scenario it is stated that “MasterPartition is in initialization state”. This is not a valid operating mode.
2. The service macro CHECKSET\_PARTITION\_MODE shall check that the expected error code is NO\_ERROR. It is not possible to check the return code after successfully switching to NORMAL mode since then (probably) other processes than the one in COLD\_START/WARM\_START are running (see comment [A653P1], p. 45) Furthermore, this means that the service macro cannot be used to check that the new operating mode is NORMAL, other processes are running and are not aware that they have to continue with the service macro.

#### Suggested Solution:

1. Add a comment that this means COLD\_START or WARM\_START.
2. Rewrite the test description to handle this situation by providing
  - (a) pseudo code for the process ProcessMain which calls SET\_PARTITION\_MODE(NORMAL) and checks that the return code (if this check is executed the test is failed)
  - (b) pseudo code for process Master\_Test running in NORMAL mode which calls GET\_PARTITION\_STATUS and checks its output.

### 3.6.3 Test Procedure T-API-PART-0230:0020

1. In the environment scenario it is stated that one sampling port called S1 is created. It is then used in the test description for writing and reading and additionally by different partitions. Both shall not be possible.
2. In the assumption, constraint and comments section some necessary services are missing: CREATE\_PROCESS and CREATE\_SAMPLING\_PORT.

#### Suggested Solution:

1. Change the environment scenario to contain two sampling ports which are connected in the config table, one for writing and the other one for reading, each has to be created by the respective partition. Furthermore, to increase readability, the names of these ports should be different (although this is not necessary).
2. Add the respective API services to the list of tested services.

### 3.6.4 Test Procedure T-API-PART-0230:0030

1. See 3.6.3, comment 1.
2. Process MessageSender is one of two processes running in partition AuxPartition which is in NORMAL mode at the beginning of the test procedure, i.e., MessageSender cannot be running during COLD\_START/WARM\_START. Therefore, it is not possible that the return value of GET\_PARTITION\_STATUS when called by process MessageSender is WARM\_START or COLD\_START.

#### Suggested Solution:

1. See suggested solution for 3.6.3, comment 1.
2. When the partition is in initialization mode (i.e., either COLD\_START or WARM\_START), the respective process calls GET\_PARTITION\_STATUS and saves the returned operating mode in the variable GVI which can then be used by process MessageSender during NORMAL mode.

### 3.6.5 Test Procedure T-API-PART-0230:0035

1. See 3.6.3, comment 1.
2. See 3.6.4, comment 2.
3. The objective of this test procedure is to test the service SET\_PARTITION\_MODE to COLD\_START. In the test description the process MessageInterpreter shall call SET\_PARTITION\_MODE with argument WARM\_START (which contradicts the objective and is already tested by T-API-PART-0230:0030).

#### Suggested Solution:

1. See suggested solution for 3.6.3, comment 1.
2. See suggested solution for 3.6.4, comment 2.
3. Correct argument value to COLD\_START.

### 3.7 Comments to 3.3.2.1.2 and 3.3.2.1.4 (GET\_PROCESS\_ID)

#### 3.7.1 Test procedure T-API-PROC-0220:0011

In the environment scenario, process Proc1 shall be created according to “”.

Suggested Solution: Provide the necessary parameters explicitly.

#### 3.7.2 Service macro CHECKGET\_PROCESS\_ID

In the service macro GET\_PROCESS\_ID is called but the returned process ID is not checked. In particular, this does not ensure that the process ID returned by CREATE\_PROCESS has not been changed.

Suggested Solution: Check additionally that the returned process ID is identical to the one which had been returned by CREATE\_PROCESS to ensure that the process ID is changed after creation of the process. Therefore

1. change the test description of T-API-PROC-0220:0011 and T-API-PROC-0220:0012 to call CREATE\_PROCESS with the given parameters and to save the returned process ID in a variable PID.
2. add an additional input parameter to the service macro CHECKGET\_PROCESS\_ID called In-ProcId of type PROCESS\_ID\_TYPE. Compare this value to the one returned by the service call.
3. change the test descriptions of T-API-PROC-0220:0011 and T-API-PROC-0220:0012 to provide as further argument the value saved in PID.

#### 3.7.3 Additional tests

Create more than one process and check that GET\_PROCESS\_ID returns for each the correct process ID to ensure that the service works correctly not only for the first created process.

### 3.8 Comments to 3.3.2.2.2, 3.3.2.2.3 and 3.3.2.2.4 (GET\_PROCESS\_STATUS)

#### 3.8.1 Test Procedure T-API-PROC-0220:0011

The environment scenario implies that processes are “defined in the Master Partition Configuration Table”, but processes shall not be defined in configuration tables. Furthermore, a reference to an unknown test procedure is given.

Suggested Solution: Provide the parameter values for the process Proc1.

#### 3.8.2 Test Procedure T-API-PROC-0220:0012

In the expected results section it is said that the test procedure confirms “the inexistence of any process”. This is not true because only one invalid process ID has been checked and there might be other valid ones.

Suggested Solution: Change the description to “confirming the inexistence of a process with the tested invalid process ID”.

#### 3.8.3 Service macro CHECKGET\_PROCESS\_STATUS

In the description’s enumeration item 1 denotes that the service GET\_PROCESS\_STATUS shall be called with the “process *name*” as an input parameter. This is not a possible input parameter to the respective service.

Suggested Solution: Change the description to “process *ID*”

### 3.9 Comments to 3.3.2.3.1 and 3.3.2.3.2 (CREATE\_PROCESS)

#### 3.9.1 Test assertion table conditions

Condition #2 says “enough storage for *file*, no max. created procs.”.

Suggested Solution: Change *file* to *process*.

#### 3.9.2 Test procedure T-API-PROC-0230:0011

1. See 3.8.1
2. The environment scenario gives concrete values to be used when creating a process. The parameters STACK\_SIZE, PERIOD and TIME\_CAPACITY depend on the configuration of the respective partition and the defined scheduling. The concrete values as used here may be not valid for the used configuration.

Suggested Solution:

1. See suggested solutions for 3.8.1 and next solution
2. Use constants or a description to denote what kind of values should be used (e.g., “valid stack size”, “valid period”, “valid time capacity”).

### 3.9.3 Test procedure T-API-PROC-0230:0015

1. See 3.8.1
2. It is not possible to successfully create a process with the same name several times as suggested in the test description.

#### Suggested Solution:

1. See suggested solutions for 3.8.1
2. Three solutions are possible: (a) Use different process names and ensure that enough memory is available to successfully create the processes, (b) reset in between (hardware reset), (c) reset by switching to COLD\_START and thus delete the already created process which allows to create it again.

### 3.9.4 Test procedure T-API-PROC-0230:0012

1. See 3.8.1
2. In the expected results section, it is implied that the last create process call was successful and created a process. But there is no such call of CREATE\_PROCESS (either directly nor via CHECKCREATE\_PROCESS).
3. In the assumptions section it is stated that a “negative entry point creates an insufficient storage capacity situation”. **It is unclear what a negative entry point is (since the entry point is a function pointer).**

#### Suggested Solution:

1. See suggested solutions for 3.8.1
2. Possible solutions are: (a) Add “Call procedure CHECKCREATE\_PROCESS with valid arguments”, (b) delete the respective part of the sentence in the expected results section (preferred).
3. **Delete the sentence with this implementation detail in the assumptions section (preferred) or explain what it means.**

### 3.9.5 Test procedure T-API-PROC-0230:0020

1. See 3.8.1

#### Suggested Solution:

1. See suggested solutions for 3.8.1

## 3.10 Comments to 3.3.2.4.1 and 3.3.2.4.2 (SET\_PRIORITY)

### 3.10.1 Test assertion tables

For API-PROC-0240:0030 “ANY” is used for table column priority which implies that the priority can be valid or out of range. Since the returned error code INVALID\_PARAM is the same for API-PROC-0240:0030 and API-PROC-0240:0040, it is not clear for API-PROC-0240:0030 that the non-existing process is the cause for the error code.

Suggested Solution: Instead of “ANY” use “within range” to ensure that the non-existing process ID is the cause.

### 3.10.2 Test procedure T-API-PROC-240:0016

If SET\_PRIORITY with preemption enabled is successful (i.e., returns no error) and results in two processes with the same priority, it depends on the implementation of the scheduler which process is chosen after process scheduling. Nevertheless, the test purpose description and the test description say that the previously running process “will not be preempted by” the other process.

This results in the following question to be answered: What exactly means “ask for process scheduling” in the description of SET\_PRIORITY with respect to preemption of the currently running process. In particular, if two processes have the same priority. I assume that the scheduler’s behavior in case of two processes with the same priority is implementation dependent.

Suggested Solution: No solution can be suggested as long as this question has not been answered. It is assumed that this test procedure has to be rewritten.

### 3.10.3 Test procedure T-API-PROC-240:0026

See 3.10.2.

Note that here the test purpose description and the test description are inconsistent and the test description denotes that the currently running process will be preempted.

Suggested Solution: A solution depends on the answer for the above question.

### 3.10.4 Test procedure T-API-PROC-240:0030

In the expected results section it is expected that GV1=TRUE and GV2=FALSE. This seems to be wrong because P1 is running and is not preempted by P2 because preemption is disabled. Then it calculates the result for GV2 and thereby checks that GV1 is FALSE (i.e., the initialized value) to show that P2 has not been executed yet. The calculated result must be GV2=TRUE. Then the process stops itself which unlocks preemption and process P2 is running and sets variable GV1=TRUE.

Suggested Solution: Add explicitly STOP\_SELF in the pseudo code of process P1 and add comment that this unlocks preemption. Correct expected results section to GV1=TRUE and GV2=TRUE.

### 3.10.5 Test procedure T-API-PROC-240:0023

In the expected results section it is expected that GV1=TRUE and GV2=FALSE. This seems to be wrong because P1 is running and is not preempted by P2 because preemption is disabled. Then it calculates the result for GV2 and thereby checks that GV1 is FALSE (i.e., the initialized value) to show that P2 has not been executed yet. The calculated result must be GV2=TRUE. Then the process stops itself and process P2 is running and sets variable GV1=TRUE.

Suggested Solution: Correct the expected results section to GV1=TRUE and GV2=TRUE.

## 3.11 Comments to 3.3.2.5.2 (SUSPEND\_SELF)

### 3.11.1 Test procedure T-API-PROC-0250:0020

In the test description for process Master\_Test there are two points to read the variables set by the processes P1 and P2. Each time only one variable is read.

Suggested Solution: Improve the test description and/or the expected results section to “at first read point GV1=TRUE and GV2=FALSE, at second point (i.e., after timed wait) GV1=GV2=TRUE”.

### 3.11.2 Test procedure T-API-PROC-0250:0010

In the test description of process P1 the arguments for calling DISPLAY\_BLACKBOARD consider two messages depending on the previous results, but do not consider that these different messages have different length.

Suggested Solution: Instead of “Size is 2 bytes” use either “Size is 2 bytes if message is ‘OK’, size is 4 bytes if message is ‘FAIL’ ” or use “size corresponds to the number of characters used for the message” (or simpler “size is the length of the message”).

### 3.12 Comments to 3.3.2.6.2 (SUSPEND)

#### 3.12.1 Test procedure T-API-PROC-0260:0015

In the expected results section it is stated that process P1 is suspended although the test description only contains calls to SUSPEND with argument P2.

Suggested Solution: Correct the expected results section to “The process P2 is suspended (functional test), making the call again nothing happens (robustness test).”

### 3.13 Comments to 3.3.2.7.2 and 3.3.2.7.3 (RESUME)

#### 3.13.1 Test procedure T-API-PROC-0270:0030

In the environment scenario it is denoted that process P1 is *periodic*. In the test description of process P1 this process shall suspend itself and then is resumed by process Master\_Test. It is expected that these calls to SUSPEND\_SELF and RESUME are successful meaning that SUSPEND\_SELF returns no return code and RESUME returns NO\_ERROR. But it is not allowed to suspend or resume periodic processes and the expected return values should be NO\_ACTION. Nevertheless, this is probably not the test objective (testing RESUME of a periodic process is addressed by T-API-PROC-0270:0090). Moreover the test assertion table for RESUME denotes that for this test procedure the process to be resumed is not periodic.

Suggested Solution: Correct environment scenario section by changing “is periodic with period 5” to “is aperiodic”.

#### 3.13.2 Test procedure T-API-PROC-0270:0065

The test description of process P1 uses the constant REGULAR\_TIME\_WAIT for a TIMED\_WAIT call. While P1 is waiting, process Master\_Test shall execute a sequence of different service calls without being preempted by process P1 (which has a higher priority). This is only possible if the concrete value used for waiting is longer than needed for executing the respective sequence of calls.

Suggested Solution: Add to the assumptions, constraints and comments section “The concrete value used for REGULAR\_TIME\_WAIT is implementation dependent and should be chosen to allow that the given sequence of calls can be executed by process Master\_Test without being preempted by process P1.”

### 3.13.3 Test procedure T-API-PROC-0270:0070

This test procedure tests assertions API-PROC-0270:0070, API-PROC-0270:0080, API-PROC-0270:0090, and API-PROC-0270:0100 (in this sequence).

1. In the test description part related to **API-PROC-0270:0090**, the test objective is to test RESUME with a periodic process (i.e., with process P2) that is *not* dormant. The test description says that process Master\_Test starts the periodic process which has a higher priority and is preempted immediately. This is not necessarily true since the start of a periodic process depends on when its next release point is. Finally, when process P2 is running it stops itself calling STOP\_SELF which sets the process to DORMANT (which was not intended to test here!). Then process Master\_Test is running again and checks RESUME and expects the return code INVALID\_MODE. (Note that this error code is also returned for dormant processes not only for periodic processes!)
2. In the test description part related to **API-PROC-0270:0100**, the test objective is to show that if a process that has not been suspended shall be resumed, no actions have to be performed. In the test description of process P3 it shall resume itself which will return INVALID\_PARAM but not NO\_ACTION as expected.

#### Suggested Solution:

1. The following changes should be done:
  - (a) Change the test description for Master\_Test to have an additional TIMED\_WAIT before calling service procedure CHECK\_RESUME to allow that process P2 has already preempted process Master\_Test.
  - (b) Change the test description of process P2 to call SET\_PRIORITY with argument P2 and priority 1 which results in immediate preemption of process P2 by Master\_Process and ensures that process P2 is not dormant.
2. Two solutions are possible:
  - (a) Either change test description of process P3 and call CHECK\_RESUME with argument ID of Master\_Test. (Master\_Test is READY and thus return code is NO\_ACTION as expected.) (preferred) or
  - (b) Change the priority of process P3 to be lower than the one of Master\_Test, start the process P3 in Master\_Test (which will not preempt the running process), then call CHECK\_RESUME with argument ID of P3 and expected value NO\_ACTION. Additionally, delete the call for CHECK\_RESUME form P3.

### 3.14 Comments to 3.3.2.8.2 (STOP\_SELF)

#### 3.14.1 Test procedure T-API-PROC-0280:0020

1. In the environment scenario, the error handler is not mentioned but shall be running.
2. In the test description process Master\_Test starts process P2 and is preempted by it. Process P2 locks preemption and calls GET\_PROCESS\_STATUS of process P1 and expects that process P1 is READY (but actually it has not yet been started and thus is DORMANT). P2 then starts P1 and checks that its status is *still* (?) READY. P2 then unlocks preemption and is immediately preempted by P1. P1 then checks the process status of P2 by calling GET\_PROCESS\_STATUS and expects the status to be DORMANT (although P2 is only preempted not stopped and thus should still be READY).

Suggested Solution:

1. Add to the environment scenario: “Error Handler is created”.
2. Change the test description for process P1 and P2 as follows:
  - (a) P1 calling GET\_PROCESS\_STATUS for P2 shall expect *READY*.
  - (b) P2 calling GET\_PROCESS\_STATUS for P1 shall first expect *DORMANT*.
  - (c) Delete “still” in second call of GET\_PROCESS\_STATUS for P1 by P2.

**3.14.2 Test procedure T-API-PROC-0280:0030**

1. In the environment scenario, the error handler process and process P2 are not mentioned but shall be created.
2. In the test description, process P1 disables preemption and then “triggers” the error handler. The error handler then stops its callee (i.e., process P1) which resets the LOCK\_LEVEL counter. To achieve the test situation (i.e., error handler and preemption disabled), the error handler shall call LOCK\_PREEMPTION expecting a no error return value. But if LOCK\_PREEMPTION is called by the error handler, NO\_ACTION is returned, which means that FALSE is assigned to GV3. (Nevertheless, GV3 is not checked by any other process.)  
[Question to \[2\], p. 55 and 56: What is the exact meaning of “reset the LOCK\\_LEVEL counter;” used in the pseudo code of API service STOP and STOP\\_SELF. Does it mean to enable preemption? If yes, this seems to contradict the commentary given for service STOP?](#)  
 Additionally, the expected results section denotes that after stopping the error handler control returns to process P1 which had been stopped before.

Suggested Solution:

1. Add to the environment scenario: “Process P2 is created with priority ?? and is aperiodic” and “Error Handler process is created”.
2. **A solution how to change the test description depends on the answers of the above questions.** Additionally, GV3 should be evaluated by Master\_Test process. (if necessary)  
 Other sections to be considered/corrected: Expected results section.

**3.14.3 Test procedure T-API-PROC-0280:0040**

1. In the environment scenario, the error handler is not mentioned but shall be running.
2. In the test description of the error handler process, the error handler shall call TIMED\_WAIT and expect no error return code. But if TIMED\_WAIT is called by the error handler, INVALID\_MODE is returned, i.e., the variable GV3 is set to FALSE. (Nevertheless, GV3 is not checked by any other process.)

Suggested Solution:

1. Add to the environment scenario: “Error Handler is created”.
2.
  - (a) Delete call of TIME\_WAIT in the error handler pseudo code **or** delete the call and use “error handler performs any processing for some time (longer than REGULAR\_TIME\_WAIT ms)”.
  - (b) In the pseudo-code for error handler process set GV3=TRUE (without condition).
  - (c) In the pseudo-code for Master\_Test process add evaluation of variable GV3.

### 3.15 Comments to 3.3.2.9 - 3.6.2.2.3

These sections and all sections in between have not been examined.

### 3.16 Comments to 3.6.2.2.4 (GET\_QUEUING\_PORT\_ID)

#### 3.16.1 Test procedure T-API-PROC-0600:0011

1. In the test description, macro CHECKGETQUEUINGPORTID is called with two parameters but requires three.
2. In the expected results section, macro CHECKGETQUEUINGPORTID shall return a CREATE\_SAMPLING\_PORT\_ERROR structure which is not correct according to the parameter types of the macro.

#### Suggested Solution:

1. Add as second parameter “expected port ID”.
2. Correct expected results such that the macro returns a CHECK\_GET\_QUEUING\_PORT\_ID\_ERROR structure.

#### 3.16.2 Test procedure T-API-PROC-0600:0012

1. In the test description, macro CHECKGETQUEUINGPORTID is called with two parameters but requires three.
2. In the expected results section, macro CHECKGETQUEUINGPORTID shall return a CREATE\_SAMPLING\_PORT\_ERROR structure which is not correct according to the parameter types of the macro.

#### Suggested Solution:

1. Add as second parameter “invalid port ID”.
2. Correct expected results such that the macro returns a CHECK\_GET\_QUEUING\_PORT\_ID\_ERROR structure.

#### 3.16.3 Service macro CHECKGETQUEUINGPORTID

1. In the description, it is written at the beginning of item 5 “Return a but structure...”.
2. In the description, for item 5.b is denoted that GET\_QUEUING\_PORT\_STATUS has an output parameter port ID. This is wrong. The respective parameter is an input to the service.

#### Suggested Solution:

1. Correct “but structure” to “bit structure”.
2. Change to “Expected port ID equal to output port ID of the service GET\_QUEUING\_PORT\_ID”.

### **3.17 Comments to 3.6.2.2.5 (GET\_QUEUING\_PORT\_STATUS)**

#### **3.17.1 Test procedure T-API-PROC-0610:0012**

In the expected results section, macro CHECKCREATEQUEUINGPORT shall return a CREATE\_QUEING\_PORT\_ERROR structure which is not correct according to the parameter types of the macro.

Suggested Solution: Correct expected results such that the macro returns a CHECK\_GET\_QUEUING\_PORT\_STATUS\_ERROR structure.

### **3.18 Comments to 3.7 and 3.8**

These sections have not been examined.

## **References**

- [1] Airlines Electronic Engineering Committee. Draft 1 of ARINC Project Paper 653: Avionics Application Software Standard Interface, Part 3 – Conformity Test Specification. Aeronautical Radio Inc., May 2005. Available at [http://www.arinc.com/aeec/draft\\_documents/653p3\\_d1.pdf](http://www.arinc.com/aeec/draft_documents/653p3_d1.pdf).
- [2] Airlines Electronic Engineering Committee. Draft 2 of Supplement 2 to ARINC Specification 653-1: Avionics Application Software Standard Interface. Aeronautical Radio Inc., May 2005.