Technical Document

DIAGNOSE AND RECOVERY API

Document Number:	88_02_03_00155
Version:	1.0
Status:	Draft
Approval Authority:	
Creation Date:	2004-Jan-12
Last changed:	2005-Feb-21 by Stephanie Levieil
File Name:	88_02_03_00155_RIV051_diagnose_and_recovery_API.doc

Important Notice

Change History

Date	Changed by	Approved by	Version	Status	Notes
05 Nov 2001	Stephanie Gerthoux	Cristian Livadiotti	Ver: 0.1	Approved	1
5 Feb 2002	Stephanie Gerthoux	Cristian Livadiotti	Ver: 0.2	Approved	2
29 Mar 2002	Stephanie Gerthoux	Cristian Livadiotti	Ver: 0.3	Approved	3
18 Dec 2002	Stephanie Gerthoux	Cristian Livadiotti	Ver: 0.4	Approved	4
13 May 2003	Stephanie Gerthoux	Cristian Livadiotti	Ver: 0.5	Approved	5
05 Dec 2003	Stephanie Levieil (Gerthoux)		Ver: 1.0	Draft	6

Notes:

1. Creation
2. Update the dar_diagnose_swe_filter function (update some dar_level parameters)
   Update the dar_start_watchdog_timer function (the timer_expiration_value is in milliseconds)
   Add the “2.8 Flash File” section.
3. Increase the circular buffer size
   Add “ 2. 8.2 Flash file location” section
   Add the “2.9 DAR and the trace” section.
   Minor modifications
4. Modify the parameter of the *dar_start_ watchdog_timer* from UINT8 to UINT16
5. Some modifications in the dar_diagnose_write function
   Remove the “2.9 DAR and the trace” section.
6. Add a new API function “dar_diagnose_write_emergency”

TABLE OF CONTENTS

List of Figures and Tables

List of References

[1] RIV031 Riviera Tracer Overview

[2] RIV000 Riviera Overview

[3] RIV010 Riviera Development Guide

[4] RIV041 Real Time Tracer

1 Introduction

As the Software running on a system such as GSM/GPRS platform gets more and more complex, it is necessary to provide an easy way for integrators and testers to interpret software problems even when trace information are not available, such as during field testing or after mobile production. Moreover, it is sometimes important to allow the system to recover from very critical situations in an as transparent as possible manner for the user.

Riviera Diagnose and Recovery (DAR) SW entity is in charge of providing such services in Riviera Environment. This document intends to provide an overview of DAR SW entity as well as a description on the DAR API.

For more information on Riviera, refers to [2][3].

2 Overview

2.1 Basic principle

The SW, running on a mobile, can be compared to a person. When people are feeling sick, they are providing general description of their symptoms to the doctor. They can feel tired, feel anxious, …
Based on that description, the doctor will further analyze the patient, and try to gather more information to have a better idea on the reasons of the symptoms. Potentially, based on the collected information, the doctor may define the medicine that the patient should take. It may also decide to request help from experts. Based on the doctor's preliminary investigation, it should be possible to know which expert to contact.
In parallel, while waiting for expert advices, the doctor may provide an emergency medicine, so that the patient feel better and can wait for complete recovery.

The Diagnose and Recovery SW entity can be seen as the 'doctor' of the SW system. Indeed, DAR SW entity provides further information on the reason why the mobile crashed, lost network or does not answer any longer. This information, called diagnose information, is stored by the DAR SW entity when a problem occurs and is available for the testers. Thanks to this information, the testers have a better understanding of what happened, especially before the phone crashed. The issue may be resolved based on this information. Or, if the problem is more complex, the testers should be able to know which expert to contact to resolve it.

Meanwhile, before the SW issue has been resolved, the DAR SW entity provides a way to re-start the system in a clean and as transparent as possible way: this service is called recovery. Recovery handles a clean reset of the system, is in charge of storing emergency information before resetting the system and provides a way to recover this information when the system is restarted.

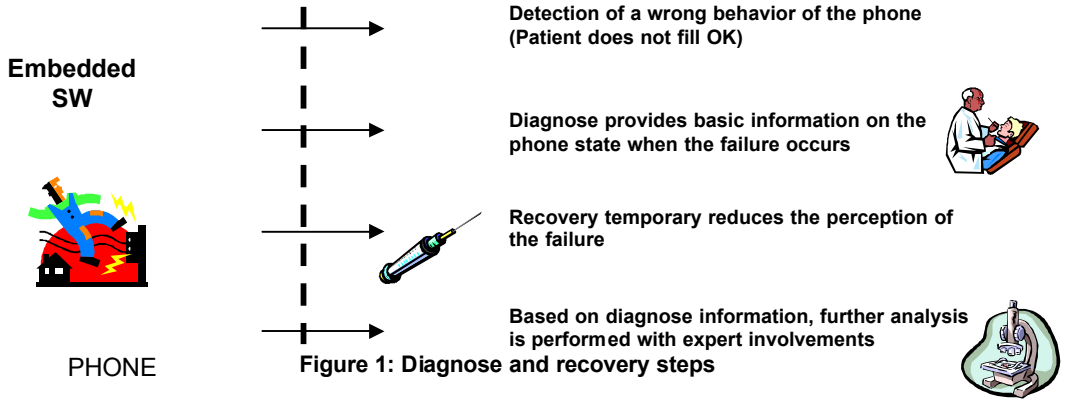


Figure 1: Diagnose and recovery steps

2.2 Implementation Overview

The Diagnose and Recovery Software entity provides services to store critical information about the system, even when trace capability is disabled. It offers a way to store this information in a file that can be downloaded from the target and further analyzed on a PC. Its objective is to be able to collect system information and to get history collection, in order to make post-mortem analysis and data.

Two kinds of information are stored by the DAR SW entity. Either information provided by other SW entity via the DAR API, either internal system information such as exceptions.

The diagnose information can be seen as a circular buffer. Diagnose information are regularly provided to the DAR SW entity (simple function call can be used by any SW entity in the system for this purpose). New information is written on old information. So the larger is the circular buffer, the longer history we can have before a problem occurred. When an error occurs, the circular buffer is stored in a permanent file (using Flash File System). This file can then be downloaded from the mobile to a PC and further analyzed. Note that the level of information as well as the group of SW entity that can access to the diagnose buffer are dynamically configurable.

The Diagnose and Recovery SW entity is part of the Riviera Scanner module, which gathers normal tracing support (RVT SWE, see [1]), Real Time Trace capability (RTT , see [4]) and the DAR SW entity. Note that DAR information can be redirected to standard trace information flow and that it is possible to store Real Time Trace information as diagnose information (not supported in current version).

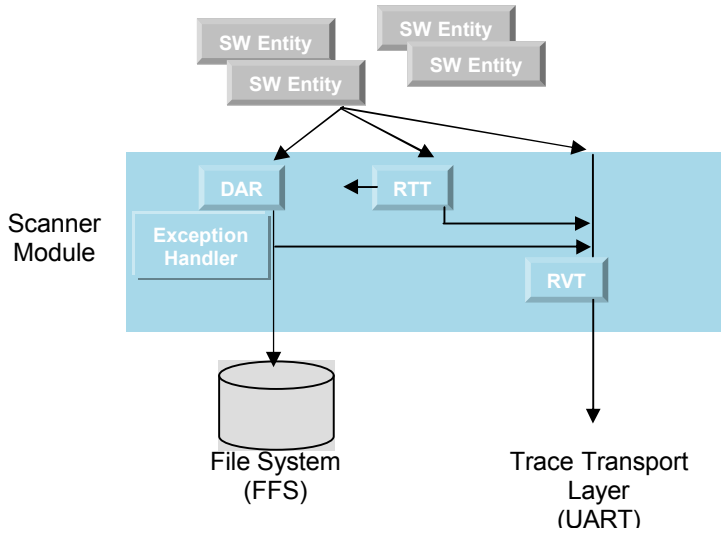


Figure 2 : Scanner Module, composed of DAR, RTT and RVT SW entities

2.3 Symptom perception

In order to clarify wordings used during diagnose process, a list of clearly identified symptoms of the mobile are listed down below. It is highly recommended to indicate in which category of a symptom is an issue, in order to have a first filtering on the cause of the problem. Basically, providing this kind of information could be done by any mobile user.

- **Freeze Category**

Description of the problem: When freeze happened, it seems nothing is running in the mobile. The mobile does not respond to anything like keypad etc. LCD does not change.
Further description: how often? Do we have any understanding on what happens before the mobile freeze? Is it possible to re-start the mobile using ON/OFF key? Or is it necessary to remove the battery?

- **Reset Category**

Description of the problem: the mobile seems to be re-started abnormally.
Further description: how often? Do we have any understanding on what happens before the mobile reset? Was the mobile in 'freeze' state before resetting?

- **One function is systematically not working**

Description of the problem: One function of the phone is not working, systematically (that means that when a function is activated, it never works).
Further description: which function does not work? How the function has been activated? What was the situation when the function has been activated? What happens when the function is activated?

- **One function is erratically not working**

Description of the problem: One function of the phone is not working, erratically (that means that when a function is activated, sometimes it works, sometimes it does not).
Further description: which function does not work erratically? How the function has been activated? What was the situation when the function has been activated and failed? What happens when the function is activated?

Note: Most of the time, **Network connectivity issue** can be seen as a 'One function is erratically not working'. It could be described as:

> Description of the problem: the mobile seems to be run normally, except that it is impossible to make or receive a call.
> Further description: how often? Do we have any understanding on what happened before the mobile stop behaving properly? How can we restart properly the phone: using ON/OFF? Battery switch?

2.4 Exceptions

Exceptions are generated by internal and external sources to cause the processor to handle an event, such as an externally generated interrupt or an attempt to execute an undefined instruction. When certain exceptions are raised by the system, it may indicate that the system is in a pretty bad shape…. For that reason, the DAR SW entity is handling such exception, and store information in the diagnose buffer to indicate which exception has been raised before the system crashed.

- **Undefined instruction**

If the ARM processor executes a coprocessor instruction, it waits for any external coprocessor to acknowledge that it can execute the instruction. If no coprocessor responds, an Undefined Instruction exception occurs.
If an attempt is made to execute an instruction is Undefined, an Undefined Instruction exception occurs.
The Undefined Instruction exception can be used for software emulation of a coprocessor in a system that does not have the physical coprocessor (hardware), or for general-purpose instruction set extension by software emulation.

- **Prefetch Abort (Instruction fetch memory abort)**

A memory abort is signaled by the memory system. Activating an abort in response to an instruction fetch marks the fetch instruction as invalid. A Prefetch Abort exception is generated if the processor tries to execute the invalid instruction. If the instruction if not executed (for example, because a branch occurs while it is in the pipeline), no Prefetch Abort occurs.

- **Data Abort (Data access memory abort)**

A memory abort is signaled by the memory system. Activating an abort in response to a data access (load or store) marks the data as invalid. A Data Abort exception occurs before any following instruction or exceptions have altered the state of the CPU.

2.5 Examples

Example of Reset Category issue:
The MMI is using a Watchdog timer: every X seconds, MMI should update the watchdog otherwise the system is reset. For example, if MMI code is blocked in an infinite loop, MMI will no longer update the watchdog and then the mobile is reset. Perception from the user will be that mobile is abnormally re-started.

The same symptom may come from a completely different reason. For example, if a pointer is badly initialized, an exception may be raised and reset the system.

2.6 Recovery

The purpose of the recovery is to re-start the system in the cleanest way as possible.

The recovery can be activated in 2 manners:

- automatically : in this case, the DAR entity will take the responsibility to activate the recovery procedure. This procedure is usually initiated when an exception is raised and cannot be handled properly.
- on demand (using a normal function call): in this case, a SW entity request the DAR entity to activate the recovery procedure

The recovery procedure gathers the following steps:

- ➔ Recovery initiation (automatic or on demand)
- ➔ Upgrade information in the circular buffer accordingly the status when recovery has been activated
- ➔ Call custom functions, in order to allow a SW entity to store very critical information that should be retrieved at re-start.
- ➔ Reset of the system

When the system is re-started, recovery can retrieve the critical information and provide information on the status of the last system stop. Diagnose information are then available in a file and can be downloaded from target for further analysis.

2.7 Level of Information and DAR dynamic configuration (filtering)

Level of information

Some pre-defined levels of information are used in DAR:

- **DAR_ERROR** (information level = 0x80)

This level of information is related to an ERROR message.
The SW entity has detected that something abnormal is happening and it reveals that the system is not working correctly or may go to an unknown state.

Example: The Software entity fails to get memory, and so it may raise an ERROR.
These Error messages are managed by the *dar_diagnose_generate_emergency* function.

- **DAR_WARNING** (information level = 0x40)

This level of information indicates that the diagnose information is related to a WARNING information. The SW entity has detected that something abnormal is happening but this problem can be handled by the SWE.

Example: A DAR function is called with a wrong parameter value. The function has detected the wrong parameter value and so, it sends a Warning to the DAR entity.
These Warning messages are managed by the *dar_diagnose_write* function.

- **DAR_DEBUG** (information level = 0x01)

This level of information indicates that the diagnose information is related to a DEBUG information. The purpose of the Debug information is to provide information on the behavior of the SWE (but no problem has been identified).

Example: Dar_debug can be used to trace messages providing internal information on the SW entity and that may be used to check a correct behavior of the SW entity
This Debug information is managed by the *dar_diagnose_write* function.

- **DAR_NO_DIAGNOSE** (information level = 0x00)

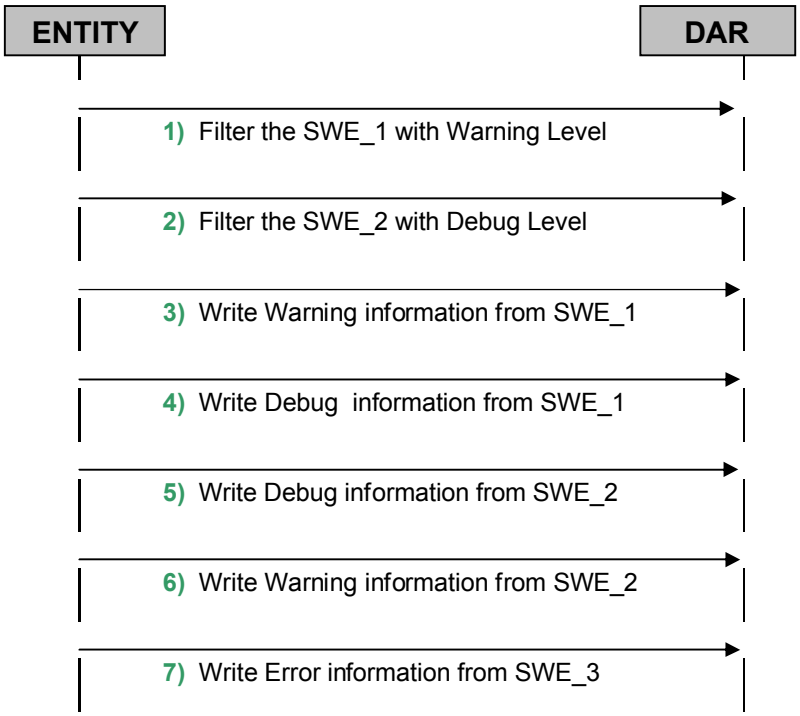
This level of information indicates that a SW entity doesn't want the diagnose information about another SW entity.

Example: A user doesn't want the Bluetooth diagnose information. Thanks to a function defined in the API (*dar_diagnose_filter* function) and the DAR_NO_DIAGNOSE level of information, the Bluetooth messages aren't stored in the RAM.

2.7.1 DAR dynamic configuration

The level information and the group of SW entity that can access to the diagnose buffer are dynamically configurable.

Example of the dynamic configuration:



- First, the SW entity that wants to use the Diagnose services must record itself by giving filtering information. It must specify if it wants Warning, Debug or none level of information.

Note: The Error messages are automatically stored in a file even if the SW entity gives no filtering information.

1) The SWE_1 wants to use the diagnose services to store Warning messages.
2) The SWE_1 wants to use the diagnose services to store Debug messages.
   Note: The Debug messages are less importance than the Warning messages. So, when the Debug level is chosen, the Dar Entity stores Debug **AND** Warning messages.

- When the SW entity gives its filtering information, it can write data in the RAM buffer.

3) Thanks to 1), the Warning information from SWE_1are stored.
4) As the Debug Level hasn't be chosen by the SWE_1, these information are not stored.
5) Thanks to 2), the Debug information from SWE_2are stored.
6) Thanks to 2) and as the Debug messages are less importance than the Warning messages, the Warning information from SWE_2 are stored.

- The Error data are always stored in the RAM buffer, even if the SW entity isn't prerecorded (no filtering information).

7) The Error message from SWE_3 is stored.

2.8 Flash file

2.8.1 Flash file description

When an error occurs, some diagnose information are stored in a permanent file (using the Flash File System) and can be downloaded from the mobile to a PC and further analyzed.

This permanent file is a Flash file and is divided in three parts:

- The circular buffer that contains diagnose information such as Warning, error, exception information, traces information. This buffer size is 3 Kbytes but this size can be increased.

- The Xdump buffer that contains the general mode registers, the Link Register, the Program Counter, the Current Processor Status Register, exception information and the bottom 20words of user mode stack. This buffer size is 152 bytes.

- The Debug Unit that is saved into the Flash file if a Prefetch abort exception or a data abort exception is generated.
  The Debug Unit is a hardware resource intended to provide additional support to a software abort-handler. The Debug Unit provided 64 stages deep history table of the last memory accesses prior entering the abort mode, then permitting analysis of previous bus transaction's. When the abort mode is entered, the automatic-write is locked and the Debug-memory content is frozen. In this state, the debug unit acts as a standard static RAM block where the data previously collected is available to the software abort-handler for working out the abort cause.
  This debug unit offers a sixty-four 32-bit word deep FIFO register file.

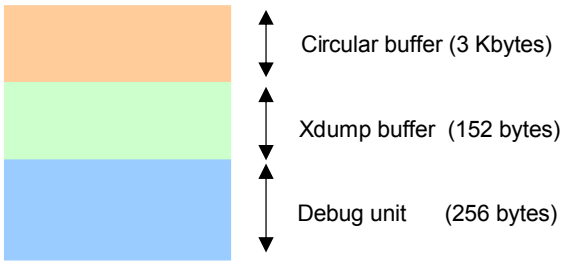


Fig 3: Flash file content

2.8.2 Flash file location

The DAR data are saved into the flash, in a debug data directory defined in the file hierarchy standard for TI GSM/BT/Wireless solutions (the PSW241 specification.) This debug data directory is called '/var/dbg'.
Before starting the DAR entity, it is necessary to format the flash using the pctm tool. This tool is in the Riviera release in the "tools/pctm" directory and "*mkfs –f*" is the command used to format the flash.

3 Diagnose and Recovery Module Use Interface (DAR API)

3.1 Recovery related API

3.1.1 dar_recovery_get_status

*T_RV_RET dar_recovery_get_status(T_DAR_RECOVERY_STATUS * status);*

Description

This function can be called by the MMI at the beginning of the mobile start procedure, in order to get the status of the last reset of the system.
Actually, system can have been reset for several reasons:

- A watchdog reset
- A scuttling reset when a recovery module has decided to activate the reset
- An emergency scuttling reset when a recovery module has detected an error and activates the reset in emergency mode
- Power ON/OFF: normal power ON/OFF sequence activated by the mobile use.

Parameters

- **T_DAR_RECOVERY_STATUS**
  Type: `typedef INT8 T_DAR_RECOVERY_STATUS;`

The possible values are:

Id	Definition
DAR_WATCHDOG	Watchdog reset
DAR_NORMAL_SCUTTLING	Recovery module has decided to activate the reset
DAR_EMERGENCY_ SCUTTLING	Emergency detection → Recovery module has decided to activate the reset in emergency mode
DAR_POWER _ON_OFF	Power on/off

Immediate Return

- **T_RV_RET**

The immediate value returned is defined as:

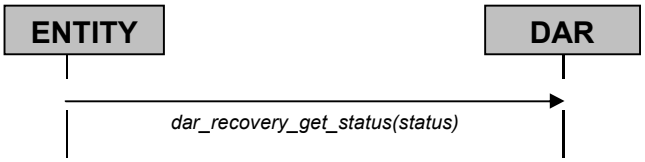
Type: `typedef INT8 T_RV_RET;`

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_ERROR	An error was occurred during the execution of this function

RV_NOT_READY	The Software entity hasn't been started
RV_INVALID_PARAMETER	The Software entity has an invalid parameter

Process flow



3.1.2 dar_recovery_config

```
T_RV_RET dar_recovery_config( T_RV_RET  (*dar_store_recovery_data)
                                        ( T_DAR_BUFFER   buffer_p,
                                          UINT16         max_ length ))
```

Description

This function can be used to store a callback function that will be called by the recovery system when a recovery procedure has been initiated. This callback should be used to store information in a buffer (pointed by buffer_p).

Parameters

- **DAR_CALLBACK_FUNC**
  *T_RV_RET (*dar_store_recovery_data) (T_DAR_BUFFER buffer_p,*
  UINT16 max_length))
  Where T_DAR_BUFFER is a `typedef UINT8 *T_DAR_BUFFER.`

  This callback function is called by the DAR entity, to allow the user to save "`max_length`" data in the buffer (allocated by the DAR entity). This buffer is used to store data before a reset.

Immediate Return

- **T_RV_RET**

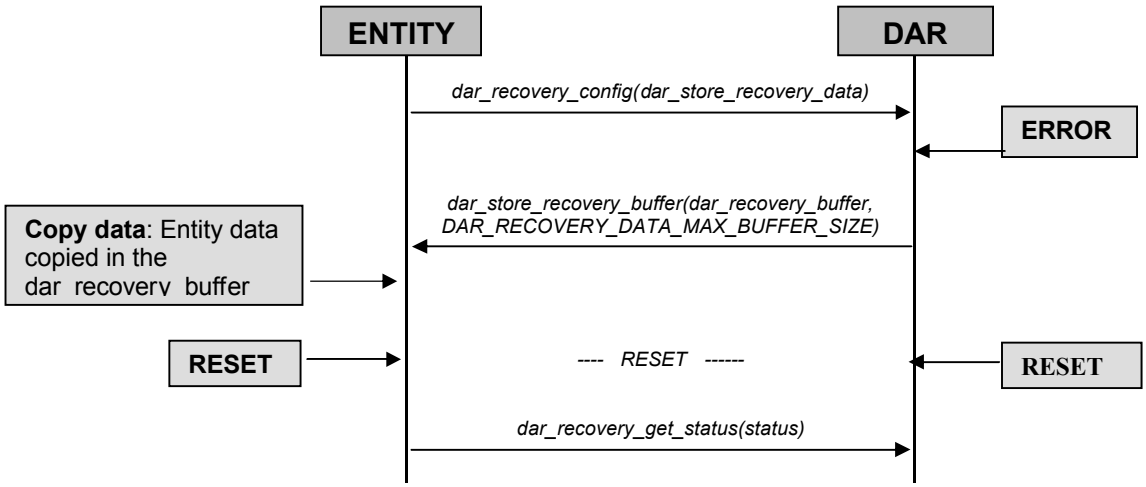
The immediate returned value is defined as:

```
typedef INT8 T_RV_RET;
```

The possible values are:

Id	Definition
RV_OK	The API function was successfully executed.
RV_ERROR	An error was occurred during the execution of this function
RV_NOT_SUPPORTED	The callback function was already given

Process flow


ENTITY
DAR
dar_recovery_config(dar_store_recovery_data)
ERROR
dar_store_recovery_buffer(dar_recovery_buffer,
DAR_RECOVERY_DATA_MAX_BUFFER_SIZE)
Copy data: Entity data copied in the dar recovery buffer
RESET
---- RESET ------
RESET
dar_recovery_get_status(status)

3.1.3 dar_get_recovery_data

T_RV_RET dar_get_recovery_data(T_DAR_BUFFER buffer_p, UINT16 length))

Description

This function is used to retrieve data that have been stored in the buffer just before a reset.

It must be called if the status returned by dar_recovery_get_status( ) is abnormal (different from POWER_ON_OFF).

Parameters

- **T_DAR_BUFFER buffer_p**

  `typedef UINT8 *T_DAR_BUFFER.`
  This buffer is allocated by the user entity (entity that uses DAR recovery services) in order for DAR to copy in it the recovered data (see previous section).

- **UINT16 length**

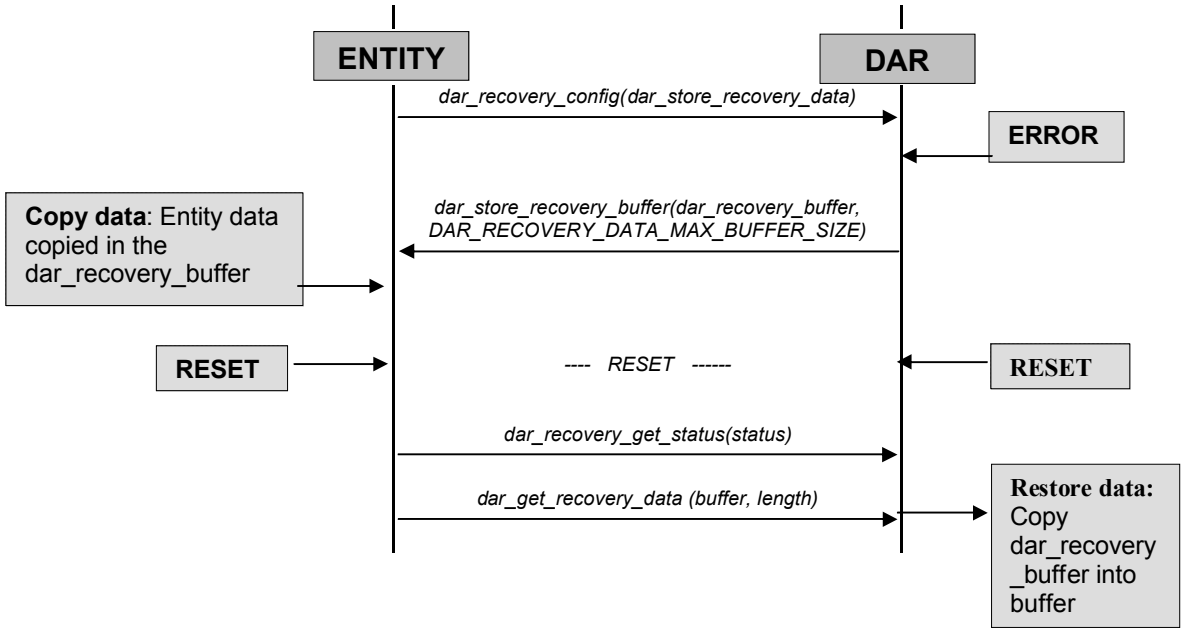
  This parameter corresponds to the buffer length used to store the data.

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.1.4 dar_start_watchdog_timer

T_RV_RET dar_start_ watchdog_timer(UINT16 time_expiration_value)

Description

This function uses the timer as a general-purpose timer instead of watchdog. It loads the timer and then starts it.

Parameters

- **UINT16 timer_expiration_value**

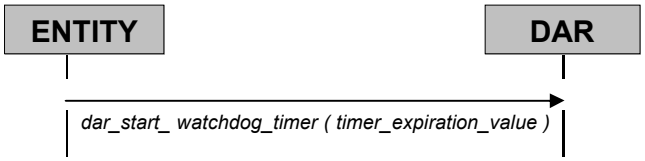
This parameter specifies the time's interval (in milliseconds) before the timer expires.
Note that the maximum value that can be used as a parameter is approximately **18085** (in decimal). If this number is passed more than, the timer value can be wrapped round to zero.

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.1.5 dar_reload_watchdog_timer

T_RV_RET dar_reload_ watchdog_timer (void)

Description

This function is used to maintain the previous timer in reloading it periodically before it expires.

If a problem occurs (infinite loop for example), the timer won't be maintained and therefore when the timer expires, it will produce an interrupt.

Parameters

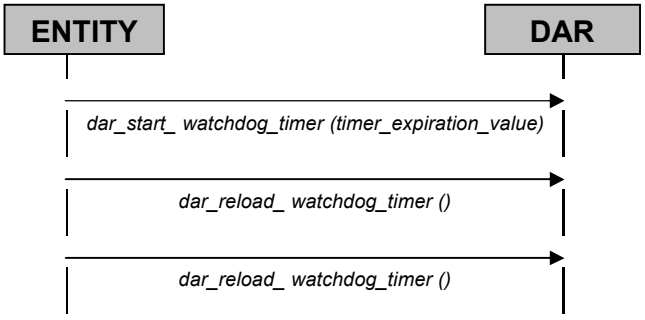
None

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.1.6 dar_stop_watchdog_timer

T_RV_RET dar_stop_ watchdog_timer(void)

Description

This function stops the timer used as a general-purpose timer instead of watchdog.

Parameters

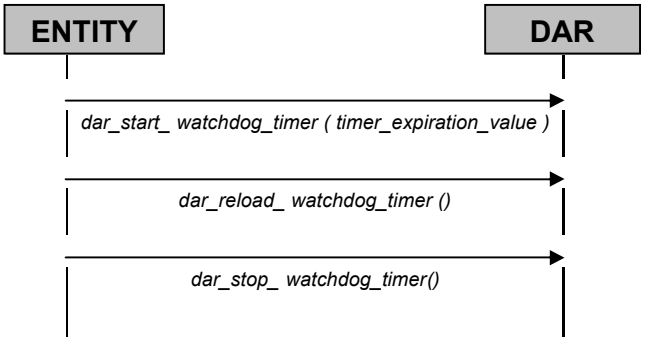
None

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.1.7 dar_reset_system

```
T_RV_RET  dar_reset_system(void);
```

Description

This function can be used to reset the system voluntarily.
It can be called if a recovery module has decided to activate the reset.
This reset is not due to an emergency or an error detection, but it's a "normal" reset.

Parameters

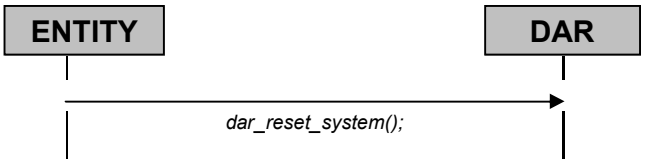
None

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.2 Diagnose Related API

3.2.1 dar_diagnose_swe_filter

```
T_RV_RET dar_diagnose_swe_filter( T_RVM_USE_ID      dar_use_id,
                                  T_DAR_LEVEL       dar_level     )
```

Description

This function is called to configure the Diagnose filtering.
It allows to determine what Software Entity (*dar_use_id*) wants to use the Diagnose and what is the diagnose level for this use_id.
The Error messages of the *dar_use_id* are automatically stored in a file, when the Debug and Warning messages are only stored if the dar_level is activated.

Parameters

- **T_RVM_USE_ID:**
  Type: `typedef UINT32 T_RVM_USE_ID;`
  This type is used to identify the DAR use that wants to use diagnose filtering.
  Every SW entity has a unique ID, called here use_id.

- **T_DAR_LEVEL:**

  Type: `typedef UINT8 T_DAR_LEVEL;`

  This parameter indicates the level of the diagnose messages.
  The level of information is coded on a byte with the following meanings:

Error	Warning	Debug	Not used	Not used	Not used	Not used	Not used
7	6	5	0	0	0	0	0

  - **Error**: It indicates that the diagnose information is related to an ERROR information. An Error should be raised when a problem has been identified. The Error messages of the *dar_use_id* are **automatically** stored in a file.
  - **Warning**: When set, indicates that the diagnose information is related to a WARNING information and that Error and Warning messages are stored in a file. A warning should be raised when a problem has been identified and can be handled by the SWE.
  - **Debug**: When set, indicates that the diagnose information is related to a DEBUG information and that Error, Warning and Debug messages are stored in a file. The purpose of the Debug information is to provide information on the behavior of the SWE (but no problem has been identified).
  - **No diagnose:** When set, indicates that the diagnose information is related to a NO_DIAGNOSE information. In fact, when an entity is using the DAR services, and doesn't want to use it anymore, it uses this NO_DIAGNOSE level. (the NO_DIAGNOSE level means that the 8 bits are equals to 0).
  - **Not used:** The 5 others bits are not supported for the moment.

For example, the possible defined values can be:

Id	Definition
DAR_WARNING	Error and Warning messages to diagnose
DAR_DEBUG	Error, Warning and Debug messages to diagnose.

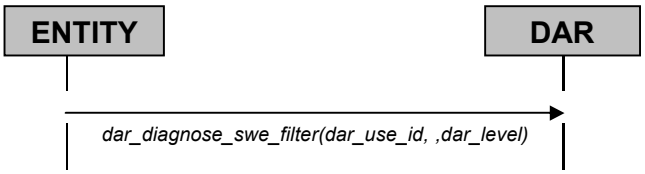
DAR_NO_DIAGNOSE	No messages to diagnose

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.2.2 dar_diagnose_write

```
T_RV_RET dar_diagnose_write(    T_DAR_INFO          *buffer_p,
                                T_DAR_FORMAT         format,
                          T_DAR_LEVEL           diagnose_info_level,
                                T_RVM_USE_ID              dar_use_id )
```

Description

This function makes a filtering depending on the use_id and the diagnose level values. According to the filtering results, the use_id, the dar_level and the string pointed by buffer_p can be stored in the diagnose RAM buffer.

For information: The ASCII file format

In order to separate the different strings, the data are stored as follows:

OxFF	**Use ID 1**	**Dar level 1**	**String 1**	**OxFF**	**Use ID 2**		

OxFF: used to separate 2 dar_diagnose_write uses
Use_Name: used to know which Software Entity has stored data into RAM.
Dar_level contains the DAR messages level (Warning or Debug)
String contains the data to diagnose

Parameters

- **T_DAR_INFO**

  Type: `typedef char T_DAR_INFO;`
  b*uffer_p* is a pointer to the message to store (this message is a string)

- **T_DAR_FORMAT**

  The data can be ASCII or binary formatted.
  The advantage of the binary format is the higher stream of information that can be carried.
  The disadvantage is the need to have a tool to unformatted incoming data.
  The message format is one byte, and its meaning is 0 for ASCII (DAR_ASCII_FORMAT) and 1 for binary (DAR_BINARY_FORMAT).

  Type : `typedef INT8 T_DAR_FORMAT;`

  The possible values are:

Id	Definition
DAR_ASCII_FORMAT	The data are ASCII formatted.
DAR_BINARY_FORMAT	The data are binary formatted. (Not supported)

- **T_DAR_LEVEL:**
  Type: `typedef UINT8 T_DAR_LEVEL;`
  This parameter indicates the diagnose level.
  This level of information is coded on a byte with the following meanings:

Error	Warning	Debug	Not used	Not used	Not used	Not used	Not used
7	6	5	0	0	0	0	0

- **Error**: It indicates that the diagnose information is related to an ERROR information. An Error should be raised when a problem has been identified. This parameter is automatically set.
- **Warning**: When set, indicates that the diagnose information is related to a WARNING information. A warning should be raised when a problem has been identified and can be handled by the SWE.
- **Debug**: When set, indicates that the diagnose information is related to a DEBUG information. The purpose of the Debug information is to provide information on the behavior of the SWE (but no problem has been identified).
- **Not used:** The 5 others bits are not supported for the moment.

For example, the possible defined values can be:

Id	Definition
DAR_WARNING	Warning messages to diagnose
DAR_DEBUG	Debug messages to diagnose

- **T_RVM_USE_ID:**

  Type: `typedef UINT32 T_RVM_USE_ID;`

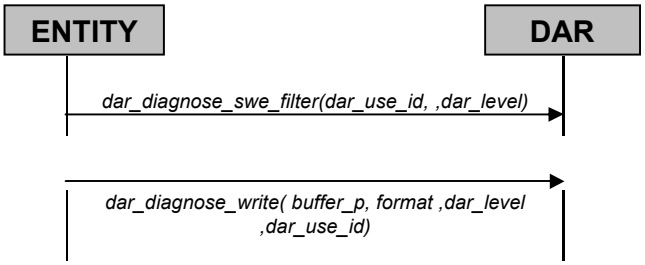
  This type is used to identify the DAR use that wants to use diagnose filtering.
  Every SW entity has a unique ID, called here Use_id.

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.2.2 dar_diagnose_generate_emergency

T_RV_RET dar_diagnose_generate_emergency (	*T_DAR_INFO* *T_DAR_FORMAT* *T_RVM_USE_ID*	**buffer_p,* *format,* *dar_use_id)*

Description

This function is called to store diagnose data in RAM buffer when an emergency has been detected and goes to emergency (automatic reset) with EMERGENCY_ SCUTTLING status.

For information: The ASCII file format

In order to separate the different string, the data are stored as follows:

OxFF	**Use ID**	**1**	**Dar Error**	**String**	**1**	**OxFF**	**Use ID 2**

OxFF: used to separate 2 dar_diagnose_write uses
Use_Name: used to know which Software Entity has stored data into RAM.
Dar_ERROR used to indicate an emergency error
String contains the data to diagnose

Parameters

- **T_DAR_INFO**

  Type: `typedef char T_DAR_INFO;`
  b*uffer_p* is a pointer to the message to store (this message is a string)

- **T_DAR_FORMAT**

  The data can be ASCII or binary formatted.
  The advantage of the binary format is the higher stream of information that can be carried.
  The disadvantage is the need to have a tool to unformatted incoming data.
  The message format is one byte, and its meaning is 0 for ASCII (DAR_ASCII_FORMAT) and 1 for binary (DAR_BINARY_FORMAT).

  Type : `typedef INT8 T_DAR_FORMAT;`

  The possible values are:

Id	Definition
DAR_ASCII_FORMAT	The data are ASCII formatted.
DAR_BINARY_FORMAT	The data are binary formatted (Not supported)

- **T_RVM_USE_ID:**

  Type: `typedef UINT32 T_RVM_USE_ID;`

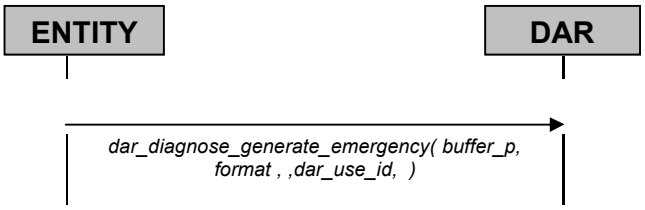
  This type is used to identify the DAR use that wants to use diagnose filtering.
  Every SW entity has a unique ID, called here Use_id.

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



3.2.3 dar_diagnose_write_emergency

```
T_RV_RET dar_diagnose_write_emergency(
                                    T_DAR_INFO      *buffer_p,
                                    T_DAR_FORMAT    format,
                                    T_RVM_USE_ID    dar_use_id,
                                    UINT32          flags)
```

Description

This function is called to store diagnose data in RAM buffer when an emergency has been detected and goes to emergency (automatic reset) with EMERGENCY_ SCUTTLING status.
This function works similar to dar_diagnose_generate_emergency but via the additional parameter 'flags' it can be controlled if :

- the data to be stored shall be appended to the last entry or if a new entry will be created.
- a reset is done or the data is only written into the dar_write_buffer

Parameters

- **T_DAR_INFO**

  Type: `typedef char T_DAR_INFO;`
  b*uffer_p* is a pointer to the message to store (this message is a string)

- **T_DAR_FORMAT**

  The data can be ASCII or binary formatted.
  The advantage of the binary format is the higher stream of information that can be carried.
  The disadvantage is the need to have a tool to unformatted incoming data.
  The message format is one byte, and its meaning is 0 for ASCII (DAR_ASCII_FORMAT) and 1 for binary (DAR_BINARY_FORMAT).

  Type : `typedef INT8 T_DAR_FORMAT;`

  The possible values are:

Id	Definition
DAR_ASCII_FORMAT	The data are ASCII formatted.
DAR_BINARY_FORMAT	The data are binary formatted (Not supported)

- **T_RVM_USE_ID:**

  Type: `typedef UINT32 T_RVM_USE_ID;`

  This type is used to identify the DAR use that wants to use diagnose filtering.
  Every SW entity has a unique ID, called here Use_id.

- **UINT32 flags**

  The possible values are:

Id	Definition
DAR_EMERGENCY_RESET	Causes a reset

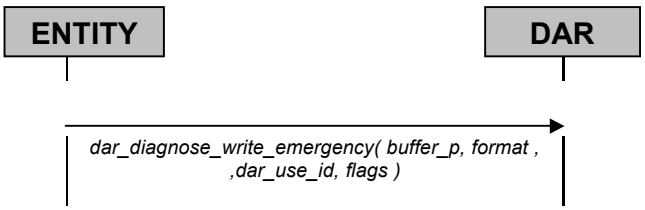
DAR_NEW_ENTRY	New data is appended to last entry

Immediate Return

- **T_RV_RET**

C.f. API function *dar_recovery_get_status*

Process flow



Appendices

A. Acronyms

API	Application Program Interface: list of functions or messages used to access to a SW entity service
DAR	Diagnose and Recovery
RAM	Random Access Memory
RTT	Real Time Trace
RVF	Riviera Frame
RVT	Riviera Trace
SW	Software
SWE	Software Entity

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)

Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>