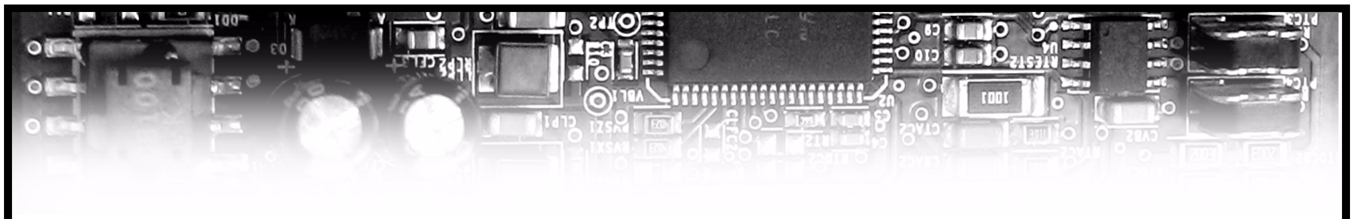


Line Test API

User's Guide

Part Number:	**VE880 and VE890 Series**
Version:	**18**
Issue Date:	**June 2010**

ZARLINK SEMICONDUCTOR

FEATURING Legerity® VOICE SOLUTIONS

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

1.1 OVERVIEW

This document describes Zarlink's Line Test Application Programming Interface (LT-API).

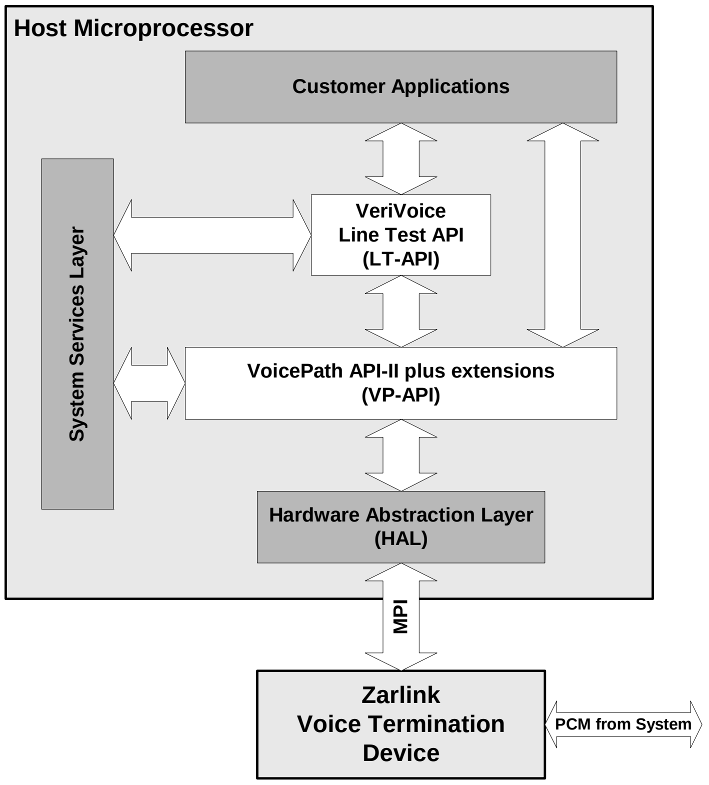
The LT-API is a common programming interface to perform line testing using Zarlink's Voice Termination Devices (VTD). It is divided into two major components:

1. Extensions to the VP-API-II as required for line testing (not available in standard VP-API-II or VP-API-II Lite distributions).
2. Application layer code that calls the VP-API-II Line Testing functions and performs the necessary computations and conversions.

1.2 SYSTEM ARCHITECTURE

The LT-API is a C-language source code module that relies on the VP-API-II. The LT-API is typically compiled into the user's application along with the VP-API-II, all of which runs on a host microprocessor controlling one or more Zarlink VTDs. Figure 1–1, *System Block Diagram*, on page 1 illustrates a basic system architecture using the VoicePath API-II and Line Test API.

Figure 1–1 System Block Diagram


Host Microprocessor
Customer Applications
System Services Layer
VeriVoice Line Test API (LT-API)
VoicePath API-II plus extensions (VP-API)
Hardware Abstraction Layer (HAL)
MPI
Zarlink Voice Termination Device
PCM from System

As shown in Figure 1–1, the customer application accesses the line testing resources through the LT-API. It also directly accesses the VP-API-II to implement call control and other management functions.

The VP-API-II relies on the System Services Layer (SSL) and Hardware Abstraction Layer (HAL) to interface with the host microprocessor, operating system, and VTD. These modules must be implemented by the customer and are described in the *VoicePath API-II User's Guide*.

The LT-API relies on additional SSL functions to interface to the OS and to the PCM Highway (PCM function is optional since LT-API PCM data can be retrieved via the MPI BUS). These functions are platform-specific and need to be implemented by the customer. (See Chapter 3, on page 49 for more information.)

The application starts the test and relays VP-API-II events to LT-API until the test is completed or until the test fails. This process is detailed in Chapter 2, on page 11.

1.3 PACKAGE CONTENTS

The LT-API package source files are organized as follows:

`lt_api\includes`
- `lt_api.h`: This file is the main header file for this package and declares the external interface to the LT-API. This is the header file that the customer must include in their application to use the LT-API. It is not necessary to include any other files.
- `lt_api_pkg_sel.h`: This file has conditional compile definitions that control how the line test API is built. Please review this file and configure it per the line test package.
- `lt_api_units.h`: This file defines the basic units used in the line test API. Users may want to review this file.
- `lt_api_temp_data.h`: This file contains the definitions for temporary variable storage for the Line Test API.
- `lt_api_test_*.h`: These files define various data types associated with individual tests. These include test inputs, test criteria, test results etc. Users of the LT-API may want to review these files.

`lt_api\common`
- This directory implements functions that are shared among various device specific Line Test libraries. The file `lt_api.c` implements the main functions of the LT-API **`LtStartTest()`** and **`LtEventHandler()`**. These main functions call the appropriate device specific Line Test implementation functions.

`lt_api\vp(880 or 890)_lt`
- This directory contains source files that are necessary to implement line testing for the VoicePort VE(880 or 890) series device.

`api_lib\common`
- This directory contains common source files that need to be added to the VP-API II.

`api_lib\vp(880 or 890)_api\testline`
- This directory contains source files that need to be added to the VP-API II and are necessary to implement low level measurement routines for the VoicePort VE(880 or 890) series device.

1.4 BUILD-TIME REQUIREMENTS

The Line Test API source code only requires access to a few external header files in order to compile. All of the required header files for the LT-API exist in the **api_lib\includes** and **lt_api\includes** folders.

1.4.1 Code Size Management

The LT-API is designed to allow all tests for all devices, as well as all debug to reside in the same application. With all possible devices, tests and debug enable, the compile code tends to be larger than necessary for a specific application.

The total LT-API Code size can be managed by compiling in or out sections of code. All compile time options shown in () can be found in the lt_api\includes\lt_api_pkg.h file. These options can and should be set to optimize the code size for a given application.

Table 1–1 Compile Time Options (Code Size Management)

Macro	Description	Default
Common		
LT_DEBUG	Includes LT-API debug code (see *LT-API DEBUG*, on page 57)	#define
Applicable to VeriVoice Auditor and Professional Packages		
LT_LINE_V	Includes the Line Voltage Test	#define
LT_ROH	Includes the Receiver Off-Hook Test	#define
LT_RINGERS	Includes the Ringers test	#define
LT_RES_FLT	Includes the Resistive Fault	#define
LT_ALL_GR_909	Includes the All GR-909 test: LT_LINE_V LT_ROH, LT_RINGERS LT_RES_FLT must all be defined in order to define this test	#define
Applicable to VeriVoice Professional Packages Only		
LT_PRE_LINE_V	Includes the Pre Line Voltage Test	#define
LT_MSOCKET	Includes the Master Socket test	#define
LT_XCONNECT	Includes the Cross Connect test	#define
LT_CAP	Includes the Capacitance test	#define
LT_LOOPBACK	Includes the Loopback test	#define
LT_DC_FEED_ST	Includes the DC Feed Self Test	#define
LT_RD_LOOP_COND	Includes the Read Loop Condition Test	#define
LT_DC_VOLTAGE	Includes the DC VOLTAGE Test	#define
LT_RINGING_ST	Includes the Ringing Self Test	#define
LT_ON_OFF_HOOK_ST	Includes the On/Off hook Self Test	#define
LT_RD_BAT_COND	Includes the Read Battery Conditions Test	#define
LT_FLT_DSCRM	Includes the Fault Discrimination Test	#define

Note:

The Compile time option LT_DEBUG has a significant affect on code size and is enabled by default. The code size related to LT_DEBUG can be adjusted by including/excluding different combinations of macros (see Table 4–1*).*

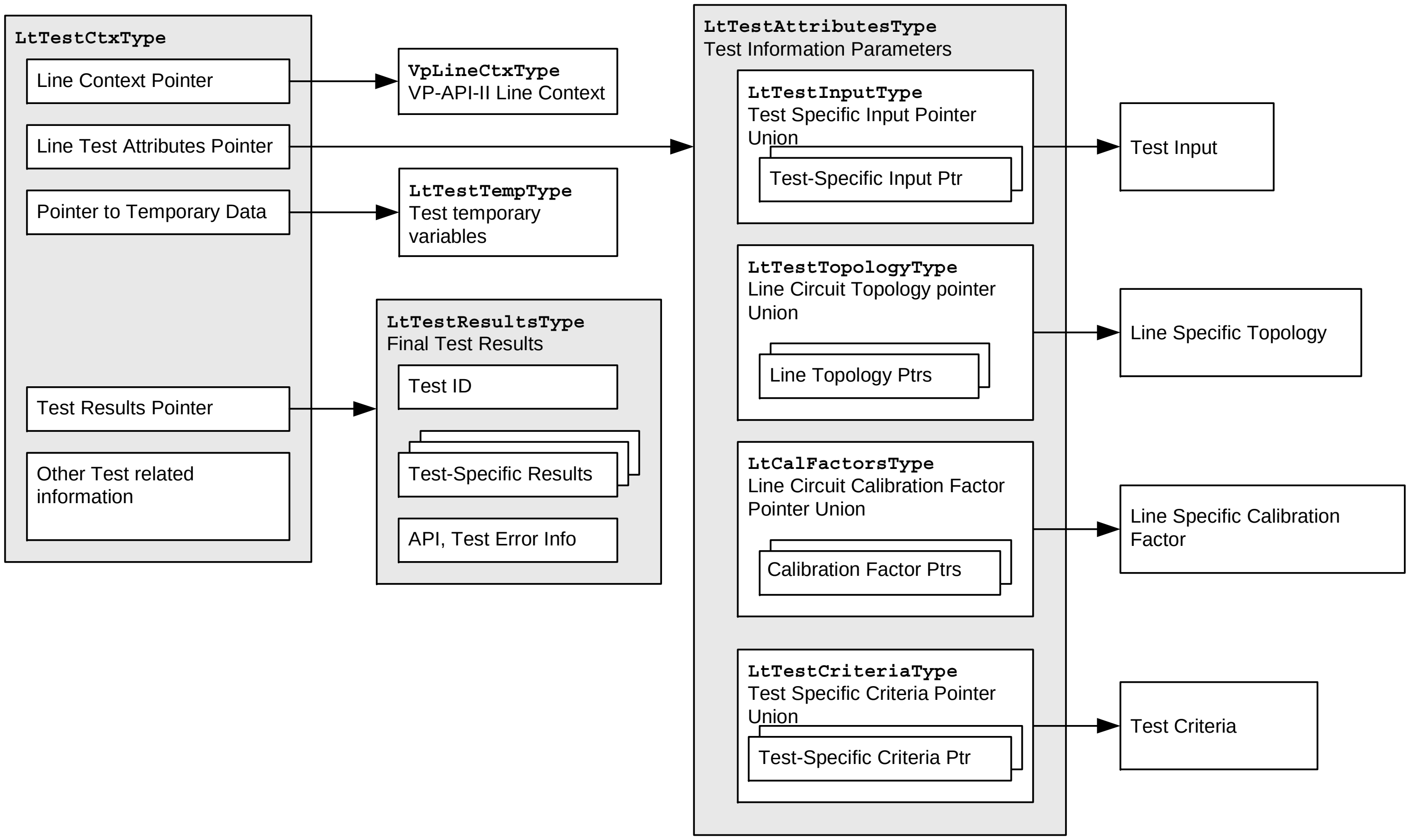
1.5 RUN-TIME REQUIREMENTS

In order to execute the tests provided by the LT-API, there are a few services required from the customer's system software. These requirements are described in the following sections.

1.5.1 Memory Allocation

The LT-API consists of a few functions, with each function performing a variety of tests based on the run-time arguments and operate on many different data structure types. All of the test-related variables are contained within or linked to the test context. The structure of a test context is illustrated in Figure 1–2.

Figure 1–2 Test Context Structure



The test context type (`LtTestCtxType`) and all of its component types are declared in `lt_api.h`. The content of `lt_api.h` may be somewhat confusing at first because of many definitions in this file and those that are included from other files. However, the user should focus primarily on the definitions that he will be interfacing with (like test inputs, test results, etc.); all other data structures are managed by the LT-API. Definitions relevant to the user are described in subsequent chapters.

To facilitate LT-API function reentrancy in a multi-tasking environment, the LT-API itself does not allocate any static/global memory for dynamic variables.[1] The LT-API requires static (not on the stack) storage for test results, test state, and other data that must be retained between calls to the test event handler. Therefore, the application is required to allocate storage for these objects before starting a test. This storage must remain allocated throughout the entire test procedure (i.e., from the time **`LtStartTest()`** is called until the time **`LtEventHandler()`** indicates the test is either completed or an error has been detected). Also, the application should not modify the content(s) of any of these objects while the objects are being used by the LT-API.

The "union" C-language construct is used to simplify the process of allocating enough storage for the test procedures' results, inputs and other structures. For example, Test A uses a results structure named `TestAResultType`, and Test B uses a different result structure named `TestBResultType`. Both `TestAResultType` and `TestBResultType` are consolidated into a union of all test result types, named `LtTestResultType`. The application must allocate storage for one instance of `LtTestResultType` before starting Test A or Test B. This scheme is also used for the LT-API's many other structures.

Prior to executing any LT-API test, the application must allocate one instance of each of the following data types:

- `LtTestAttributesType`
  This structure contains several pointers to the various attributes required to run a test, including pointers to the test inputs, test criteria, line topology and calibration factors. In general, all values must be filled in by the application. In some cases however, uninitialized (i.e., NULL) pointers will take on default values. **The pointers passed to this structure must retain their validity until the test terminates**.
- `LtTestTempType`
  This structure contains intermediate test results and internal state information that must be maintained between LT-API function calls. The application must allocate enough storage for one instance of this type and must not modify this structure during the test.
- `LtTestResultType`
  This structure contains the results of a completed test. The test-specific result types are collected in a union within this structure. The ID of completed test is also included so that the application can easily determine the type of results in the structure.
- `LtTestCtxType`
  The test context links all of the above types together along with some other test state information, as shown in Figure 1–2. The application allocates storage for one instance of this type and passes a pointer to this instance to **`LtStartTest()`**, which takes care of initializing the test context. Subsequent calls to **`LtEventHandler()`** take the test context as an argument.

1.5.2 VP-API-II Event Handling

The Line Test API must assume that other software modules in the system may be interested in events coming from the VoicePath API-II. That is, the LT-API may not assume exclusive use of the VP-API-II event facilities. Therefore, the LT-API cannot directly call **`VpGetEvent()`** and **`VpGetResult()`** to retrieve test-related events and results, as it might accidentally de-queue and discard events or results unrelated to line testing.

Even the most basic applications using the VP-API-II will probably have some sort of common VP-API-II event handler. This *Event Dispatcher* acts as a central point for retrieving events from the VP-API-II and distributes those events to relevant software modules. Figure 1–3 illustrates the concept of the Event Dispatcher.

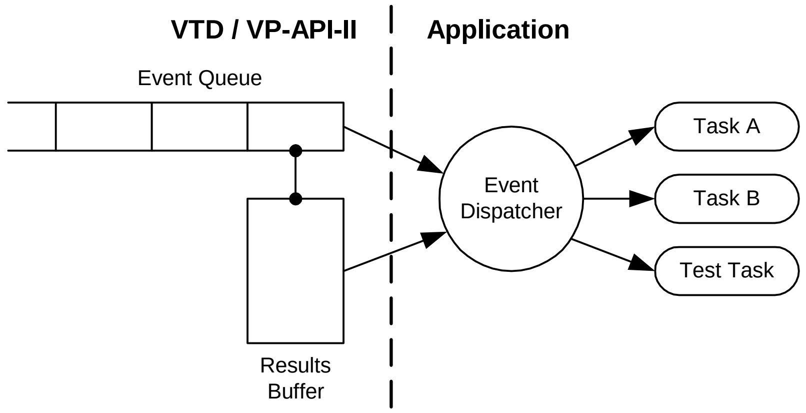
1. The LT-API may allocate storage for various look-up tables, but they are declared constant and therefore pose no problem for reentrancy.

As shown in Figure 1–3, the hypothetical Event Dispatcher retrieves the results associated with an event at the same time it retrieves the event itself. It is generally a good idea to free the Results Buffer as quickly as possible; leaving data in the Results Buffer can potentially block future events.

The Event Dispatcher can be implemented in three ways:

1. In the VTD interrupt service routine,
2. As part of "the big main loop" in a single-threaded system, or
3. As a separate task in a multi-threaded system.

Figure 1–3 Event Dispatcher



The LT-API just assumes the presence of some sort of Event Dispatcher in the application.

1.5.3 LT-API Event Reporting

The LT-API defines an event type (`LtEventType`) that includes a pointer to a VoicePath event and a pointer to the event's results (if any). Upon retrieving an event (**`VpGetEvent()`**) for the line under test, the application should:

1. Determine if **`VpGetResults()`** should be called (indicated by the event structure member *hasResults* == *TRUE*) and call it with an appropriate structure.

   Note:

   *When the VP-API-II event category is test (indicated by event structure `eventCategory = VP_EVCAT_TEST`), the application must call **`VpGetResults()`** with an instance of results structure type `VpTestResultType`.*

2. Create an instance of `LtEventType` and set the appropriate pointers in the `LtEventType` instance to the event and results.
3. Call **`LtEventHandler()`** so that the LT-API can respond to the event.

The application may pass all events and results for all lines rather than just the line under test, in which case the **`LtEventHandler()`** will return an error code for events not related to the line under test and for events not related to the test being performed.

Note:

In the case where the VP-API-II detects a thermal fault, the event returned will be `VP_EVCAT_TEST:VP_LINE_EVID_ABORT`. The LT-API will abort the currently running test when this event is passed, and while the test is running the line will not be (immediately) set to `VP_LINE_DISCONNECT` (even if the VP-API-II option `VP_OPTION_ID_CRITICAL_FLT` for thermal faults is set to auto-disconnect). Instead, the VP-API-II will delay it's user specified operation until the currently running test terminates.

The application does not need to perform any specific masking/unmasking for line tests. The LT-API will mask/unmask all events as needed during line test and restore the event mask per the application settings when complete.

1.5.4 Role of Call Control While Testing

No other application (or thread) should be accessing the line under test except for **`VpGetEvent()`** and **`VpGetResults()`** as described above.

1.6 TYPICAL USAGE

This section describes a typical LT-API usage scenario. The example shown in Figure 1–4 covers a simple case where no unusual conditions or errors occur. This is a general description of the test procedure and does not include every minute detail involved in running a test.

Figure 1–4 Typical Usage Scenario

Application / Event Handler	Line Test API	VP-API-II / VTD
Application allocates storage for test data structures (test context, test attributes, temporary variables, results, etc). Configure the Test Attributes as necessary Call **`LtStartTest()`**		
	Initialize test context and state. Prepare the VP-API-II for testing. Return `LT_STATUS_RUNNING`.	
Event dispatcher waits for the VP-API-II event.		Check resource availability. Call **`VpSysTestHeapAquire()`** (See note 1). Raise `VP_LINE_EVID_TEST_CMP` event.
Event dispatcher retrieves the event. Call **`VpGetEvent()`**		
		Retrieve event from VP-API-II.
Event dispatcher reads the results associated with the event if necessary. Call **`VpGetResults()`**		
		Retrieve results from VP-API-II.
Event dispatcher dispaches the event to the application. The application calls **`LtEventHandler()`** upon receiving LT-API event from the event dispatcher.		
	Error-check event. Go to next test state. Perform first test phase. Request the VP-API-II to start a test Return `LT_STATUS_RUNNING`.	

Application / Event Handler	Line Test API	VP-API-II / VTD
Event dispatcher waits for VP-API-II event.		Perform requested line test. Call **`VpSysPcmCollectAndProcess()`** (See Note 1 below) Raise `VP_LINE_EVID_TEST_CMP` event.
• • • • •	• • (possibly more test phases here) • •	• • • • •
Event dispatcher retrieves the event. Call **`VpGetEvent()`**		
		Retrieve event from VP-API-II.
Event dispatcher retrieves the results. Call **`VpGetResults()`**		
		Retrieve results from VP-API-II.
Event dispatcher dispatches the event to the application. The application calls **`LtEventHandler()`**		
	Error-check event. Last phase of test complete, calculate results. Conclude the VP-API-II test. Return `LT_STATUS_RUNNING.`	
Event dispatcher waits for the VP-API-II event.		Restore line state. Call **`VpSysTestHeapRelease()`** (see note 1). Raise `VP_LINE_EVID_TEST_CMP` event.
Event dispatcher retrieves the event. Call **`VpGetEvent()`**		
		Retrieve event from VP-API-II.
Event dispatcher retrieves the results. Call **`VpGetResults()`**		
		Retrieve results from VP-API-II.

Application / Event Handler	Line Test API	VP-API-II / VTD
Event dispatcher dispatches the event to the application. The application calls **`LtEventHandler()`**		
	Error-check event. Test is done. Returns `LT_STATUS_DONE`.	
Test Results structure contains valid data. The application makes use of the test results data and de-allocates all test resources (like memory, any other hardware/software resource) as necessary.		

Note:

1. *These functions are necessary only when the Line Test API is used along with a VoicePort™ VE(880 or 890) series devices.*

CHAPTER

2 APPLICATION LAYER I/F

2.1 OVERVIEW

This chapter describes the application's interface in to the Line Test API. The Line Test API is a common API that can support Line Testing for more than one device family simultaneously. Although the interface to the LT-API is the same, the capabilities and accuracies vary from one device to another, and within one device family, from one package to another. The following tables capture the LT-API support for various device families.

Table 2–1 LT-API Supported Devices

Device, VP-API-II name	Supported Line Termination Types	Available Packages
VoicePort VE880 Series (Le88231, Le88241, Le88266, Le88276 and Le88286 devices, VE8820 and VE8830 Chipsets), `VP_DEV_880_SERIES`	`VP_TERM_FXS_GENERIC`, `VP_TERM_FXS_ISOLATE`, `VP_TERM_FXS_ISOLATE_LP`, `VP_TERM_FXS_SPLITTER`, `VP_TERM_FXS_SPLITTER_LP`, `VP_TERM_FXS_LOW_PWR`	Le880SLVV, Le880SLVVP
VoicePort VE890 Series (VE8910, VE8911, VE8910-HV, VE8911-HV Chipsets), `VP_DEV_890_SERIES`	`VP_TERM_FXS_GENERIC`, `VP_TERM_FXS_ISOLATE`, `VP_TERM_FXS_ISOLATE_LP`, `VP_TERM_FXS_SPLITTER`, `VP_TERM_FXS_SPLITTER_LP`, `VP_TERM_FXS_LOW_PWR`	Le890SLVV, Le890SLVVP

Table 2–2 LT-API Supported Tests

Test Procedures	Test Packages		Page #
	VeriVoice™ Auditor (See note 1)	VeriVoice™ Professional	
Pre Line Voltage Test		●	21
Line Voltage Test	●	●	22
Receiver Off-Hook Test	●	●	24
Ringer Equivalence Number Test (Test type: Regular phone)	●	●	25
Ringer Equivalence Number Test (Test type: Electronic phone)		●	25
Resistive Fault Test	●	●	28
Capacitance Test (note 3)		●	
All GR-909 Tests	●	●	32
Loopback Test		●	34
DC Feed Self-Test (note 2)		●	36
DC Voltage Test		●	38
Ringing Self-Test (note 2)		●	39
On/Off Hook Self-Test (note 2)		●	42
Read Battery Conditions		●	43
Read Loop Conditions		●	44
Master Socket Detection Test		●	43
Cross Connect Detection Test		●	45

Note:

1. *The VeriVoice™ Auditor software package ignores input details like test topology and test criteria; It provides only test pass/fail information. Test results contain meaningful information only in the* `fltMask` *and* `measStatus` *fields of the respective test's result structure.*
2. *These tests require an external test load controlled by an internal test load switch, or Internal Test Termination capability. Test load switches are available on the Le88276, Le88286, Le88231 and Le88241 devices. Internal Test Termination is supported on Le88266, Le88276, Le88286, VE8820, VE8830 and VE890 series FXS devices.*
3. *These tests do not support the following devices: Le88231, Le88241.*

2.2 UNITS USED IN LT-API

The Line Test API defines the following basic data types for the basic physical units. These are defined in the **`lt_api_units.h`** file. The units defined in the following table are used for both the inputs and the outputs of the LT-API unless explicitly stated otherwise.

Table 2–3 LT-API Physical Units

LT-API Typedef Name	VP-API-II Equivalent Type	Physical Units
`LtImpedanceType`	int32	1/10 Ω
`LtVoltageType`	int32	mV
`LtCurrentType`	int32	µA
`LtTimeType`	int32	µs
`LtCapacitanceType`	int32	pF
`LtRenType`	int32	milli REN (Ringer Equivalence Number)
`LtPercentType`	int32	milli percent
`LtFreqType`	int32	milli Hz

2.2.1 Special Constants

The following special constants have been defined for the purposes of describing special physical conditions like open circuit, short circuit, parameter not measured, etc. Applications will receive these special constants as outputs when a given physical parameter is being measured. Similarly, these constants could also be used to specify special input conditions for test inputs. Applications should be written to handle these special constants when dealing with the LT-API.

Table 2–4 Special Constants

Physical Parameter Type	Special Constants	Meaning
LtImpedanceType	LT_IMPEDANCE_OPEN_CKT = VP_INT32_MAX = 2147483648	This is an unique value for indicating open circuit and is not a measured value. For the purposes of the LT-API and its measuring capabilities, this means that the impedance is high enough to be treated as open circuit.
	LT_IMPEDANCE_SHORT_CKT = 0	This is an unique value for indicating short circuit and is not a measured value. For the purposes of the LT-API and its measuring capabilities, this means that the impedance is low enough to be treated as short circuit.
	LT_IMPEDANCE_NOT_MEASURED = VP_INT32_MIN = -2147483648	This is an unique value for indicating the physical parameter is not measured.
LtCapacitanceType	LT_CAPACITANCE_NOT_MEASURED = VP_INT32_MIN = -2147483648	This is an unique value for indicating the physical parameter is not measured.
	LT_MAX_CAPACITANCE = VP_INT32_MAX = 2147483648	This is an unique value for indicating the physical parameter exceeded the device measurement capability.
LtVoltageType	LT_VOLTAGE_NOT_MEASURED = VP_INT32_MIN = -2147483648	This is an unique value for indicating the physical parameter is not measured.
LtCurrentType	LT_CURRENT_NOT_MEASURED = VP_INT32_MIN = -2147483648	This is an unique value for indicating the physical parameter is not measured.
	LT_MAX_CURRENT = VP_INT32_MAX = 2147483648	This is an unique value for indicating the physical parameter exceeded the device measurement capability.
LtRenType	LT_REN_NOT_MEASURED= VP_INT32_MIN = -2147483648	This is an unique value for indicating the physical parameter is not measured.
LtFreqType	LT_FREQ_NOT_MEASURED= VP_INT32_MIN = -2147483648	This is an unique value for indicating the frequency is not measured.

2.2.2 LT-API SOURCE VERSION NUMBER

Due to feature (device, termination, or function) additions and bug fixes, an application may at runtime want to determine the current version of the LT-API release. This can be found in the header file "lt_api.h" from the macro:

#define **LT_API_VERSION_TAG** (0x010600)

Note that the specific value in the source may be different than shown above.

The example shown is for Major release 01, Minor release 06 (6), Revision 00.

- Major Number: Indicates a complete interface change such that the application is not likely to compile, and will not work. An application detecting a Major release number change should immediately stop.

- Minor Number: Indicates a functional change or addition. Support for a new device, termination type, or added function would justify a new Minor number.
- Revision Number: Used for bug fixes only. Functional compatibility is guaranteed, except (of course) the bug fix itself.

2.3 LINE TEST API FUNCTIONS

There are only three functions exported by the LT-API:

- **`LtStartTest()`** starts a line test sequence.
- **`LtEventHandler()`** handles events that occur during a test sequence.
- **`LtAbortTest()`** Aborts an on-going test.

These functions are described in detail in the following sections.

2.3.1 LtStartTest()

PROTOTYPE

```
LtTestStatusType
LtStartTest(
    VpLineCtxType *pLineCtx,
    LtTestIdType testId,
    uint16 handle,
    LtTestAttributesType *pAttributes,
    LtTestTempType *pTemp,
    LtTestResultType *pResult,
    LtTestCtxType *pTestCtx);
```

DESCRIPTION

This function starts a line test sequence. It checks the inputs for errors, initializes the caller's test context, and starts the test procedure by issuing the Test Prepare command (or any other necessary first test step) to the VTD/VP-API-II.

INPUT

This function takes the following input arguments:

- `pLineCtx` points to a VP-API-II line context, which determines the line to be tested. Refer to the *VoicePath API-II User's Guide* for more information on line contexts.
- `testId` determines which line test is to be performed and can take any of the values listed in Section 2.4
- `handle` contains the application handle value that is passed through the VP-API-II and returned in the related test events. Refer to the *VoicePath API-II User's Guide* for more information on the handle parameter.
- `pAttribues` is a pointer to an instance of a structure of type `LtTestAttributesType`. This structure contains various member elements that contain or specify pointers to various types of information necessary to run the test. All members of this structure need to be filled by the application appropriately before calling this function. The pointer that is passed to this argument and any other pointer(s) passed within this structure must retain its validity until the test is complete. If `VP_NULL` is passed to this argument, all parameters that are contained in this structure are presumed to be set to their default values (if any), and the test would be carried out using those default values. The members of this structure are described below.

```
typedef struct {
    LtTestInputType inputs;
    LtTestTopologyType topology;
    LtCalFactorsType calFactors;
    LtTestCriteriaType criteria;
    uint32 ltDbgFlag;
} LtTestAttributesType;
```

– `LtTestInputType inputs:`
The individual tests might need input arguments to specify the test algorithm to be used or the test voltage/current to be applied etc. Such input arguments are specified in this member. The `LtTestInputType` data type is a 'C' language union of pointers to all the necessary test specific input structures. The test specific inputs are discussed in the respective tests themselves, and are explained later in the chapter. If the respective member corresponding to a test is set to `VP_NULL` then the default test inputs (if any) are used for that test.

– `LtTestTopologyType topology:`
The VP-API-II defines the term "Line Termination Type" to abstract the line configuration. However, beyond this abstraction of the line there might be differences in physical component values used to realize the line circuit. Such component variations if not accounted for, will result in poor test accuracies. The `LtTestTopologyType` data type is 'C' language union of pointers to all possible line topologies. The application could specify default topology by setting any member of the `LtTestTopologyType` structure to `VP_NULL`. Presently, the only topology that is defined is for the CSLAC-(880 or 890) line testing. Its structure is shown below.

```
typedef struct {
   LtImpedanceType rTestLoad;
   LtImpedanceType rSenseA;
   LtImpedanceType rSenseB;
   LtImpedanceType rLeakageA;
   LtImpedanceType rLeakageB;
} LtVp880TestTopologyType;
```

The `rTestLoad` member variable indicates the value of the test load resistor used in the line circuit. By default (if no topology profile is provided) this value is assumed to be 1000 Ω.

The `rSenseA` member variable indicates the value of the sense resistor used in the tip sense path. By default (if no topology profile is provided) this value is assumed to be 402 kΩ.

The `rSenseB` member variable indicates the value of the sense resistor used in the ring sense path. By default (if no topology profile is provided) this value is assumed to be 402 kΩ.

The `rLeakageA` member variable indicates any leakage resistance that may be present in the tip sense path. By default (if no topology profile is provided) this value is assumed to be 900 MΩ.

The `rLeakageB` member variable indicates any leakage resistance that may be present in the ring sense path. By default (if no topology profile is provided) this value is assumed to be 900 MΩ.

If application intends to modify any of these values, it needs to make an instance of the `LtVp880TestTopologyType` and then fill in all values and should assign the pointer of the object's pointer to `topology.pVp880Topology`.

– `LtCalFactorsType calFactors:`
The test accuracies can be improved by calibrating the line circuit against a known calibration circuit. This calibration circuit could contain known high precision physical components. Once calibration has been performed, the LT-API will know the correction factors to be applied while measuring unknown physical parameters. At the moment, the LT-API does not support external calibration circuit. Therefore, the pointer (any member of the union) inside this member should be set to `VP_NULL`. The value `VP_NULL` utilizes the default calibration coefficients.

– `LtTestCriteriaType criteria:`
The individual line tests may need input specifications that determine the end outcome of the test. The `LtTestCriteriaType` data type is 'C' language union of pointers to test criteria inputs. The application code needs to fill in the appropriate member of this union before calling the test. The applicable criteria for various tests are described with in the tests themselves, and are explained later in the chapter. If the respective member variable corresponding to a test is set to `VP_NULL`, then the default criteria (if any) is used for the test.

`uint32 ltDbgFlag:`
All LT-API tests have debug statements incorporated into the algorithms. At run time, this argument can be used to enable/disable debug output and should be the bitwise OR of one or more debug output selectors. See *LT-API DEBUG*, on page 57 for more info on LT-API debug.

- `pTemp` points to an instance of the universal LT-API temporary variables type `LtTestTempType`. This structure contains intermediate test results and internal state information that must be maintained between the LT-API function calls while running a particular test. The application need not be concerned about the contents of this structure.
- `pResult` points to an instance of the universal LT-API result type `LtTestResultType`. This structure contains the results of a completed test within a union of all test-specific result types. The test-specific result types are defined with the individual tests later in this chapter. This structure also includes the `testId` (see above) in the result, which provides the context for the application to interpret the result data. Please see below for more details on how and when to interpret contents in this structure.
- `pTestCtx` points to an instance of the test context type described in *Memory Allocation*, on page 4. The application must allocate storage for this structure but does not need to initialize it. Subsequent calls to **`LtEventHandler()`** take the test context as an argument. Applications should not modify the contents of this structure.

RETURN

The following LT-API result codes are returned by this function:

- `LT_STATUS_RUNNING` indicates that the requested test sequence is started and running without any error. The application should not interpret any of the results.
- `LT_STATUS_ERROR_INVALID_ARG` indicates that one or more input arguments are invalid. Results structure does not contain any valid results.
- `LT_STATUS_ERROR_VP_GENERAL` indicates that some VP-API-II function returned an error code. The exact error returned by the VP-API-II is copied to the LT-API test result member `vpGeneralErrorCode` variable of the `LtTestResultType` results structure. Please refer to the *VoicePath API-II User's Guide* for more information on the handle parameter.
- `LT_STATUS_TEST_NOT_SUPPORTED` indicates that the requested test is not supported by the device or the line termination type associated with the `pLineCtx`.
- `LT_STATUS_ERROR_RESOURCE_NA` indicates that some shared physical resource (like test-in or test-out bus) required for testing is currently not available.
- `LT_STATUS_DONE` indicates that the test sequence is complete and the test results are valid. This status value is returned by tests that do not involve multiple stages to carry out a test. Presently there are no such tests, but application developers are encouraged to develop applications that will allow this feature to be supported. If this status code is returned, the line state is restored to its pre-test condition. The application can then safely delete all test library objects created before the start of the test, including the test input object, test results object, test context object, etc.

2.3.2 LtEventHandler()

PROTOTYPE

```
LtTestStatusType
LtEventHandler(
    LtTestCtxType *pTestCtx,
    LtEventType *pEvent)
```

DESCRIPTION

This function responds to the events that occur during a test sequence. The application should call this function for all the events that occur on a line under test. Note that this function can safely be called for all events on all lines; it ignores those events that do not apply to a line under test. This function determines whether an event is related to a test by inspecting the passed test context and the event itself.

If the event (`pEvent`) is indeed related to the line test (`pTestCtx`), this function checks the event for errors and performs the next action in the test sequence, which typically includes one or more calls to the VP-API-II. If any error is detected in the test sequence, the test sequence is automatically aborted.

INPUT

This function takes the following input arguments:

- `pTestCtx` points to an instance of the test context type described in *Memory Allocation*, on page 4. The test context passed to this function must have been initialized by **`LtStartTest()`**. Recall that the test context contains several pointers to different types of objects, including test inputs, test outputs, a line context, etc. These links must be valid for the entire duration of a test sequence. That is, the objects referenced by these pointers must not be modified or deleted during the test.
- `pEvent` points to an instance of the `LtEventType` structure, which contains a pointer to a VP-API-II event and a pointer to a VP-API-II result. Before calling **`LtEventHandler()`**, the application must:
  - Allocate an instance of `LtEventType`.
  - Set the event pointer (`pVpEvent`) in this instance to the address of the VP-API-II event that just occurred.
  - If this event has associated results, set the result pointer (`pResult`) in this instance to the address of the VP-API-II results. Refer to section *LT-API Event Reporting*, on page 6 the *VoicePath API-II User's Guide* for details on VP-API-II events and results.

RETURN

The following LT-API result codes are returned by this function:

- `LT_STATUS_RUNNING` indicates that the test sequence is still in progress. **`LtEventHandler()`** returns this result if the test event was handled but the test sequence is not complete. **`LtEventHandler()`** also returns this result if the event was not related to the test and was ignored. If this return value is returned by the **`LtEventHandler()`** function the application could safely delete the LT-API event, VP event, and VP result object.
- `LT_STATUS_DONE` indicates that the test sequence is complete and the test results are valid. No more VP-API-II events related to this test sequence are expected. The line state is restored to its pre-test condition. The application can safely delete all test library objects created before the start of the test, including the test input object, test results object, test context object, etc.
- `LT_STATUS_ABORTED` indicates that the test sequence has been aborted due to a critical fault detected during the test by the VP-API or at the request of the application after calling **`LtAbortTest()`**. Please see *LtAbortTest()*, on page 19 for more details. The application can safely delete all test library objects created before the start of the test, including the test input object, test results object, test context object, etc. All data in the test's results structure should be ignored if `LT_STATUS_ABORTED` is returned.
- `LT_STATUS_ERROR_INVALID_ARG` indicates that some function call argument is invalid.
- `LT_STATUS_ERROR_UNEXPECTED_EVENT` indicates that the **`LtEventHandler()`** received an event for the line under test that was not expected.
- `LT_STATUS_ERROR_VP_GENERAL` indicates that some VP-API-II function returned an error code. The exact error returned by the VP-API-II is copied to the test result `vpGeneralErrorCode` variable.
- `LT_STATUS_ERROR_VP_TEST` indicates that the VP-API-II reported an error. The error code returned by the VP-API-II is copied to the test result `vpTestErrorCode` variable.
- `LT_STATUS_TEST_ERROR_UNKNOWN` indicates that some unexpected error occurred in the LT-API, possibly due to a software bug. Such errors should never occur and should be reported to Zarlink if observed.
- `LT_STATUS_TEST_ERROR_MATH` indicates that the LT-API caught a math error such as divide-by-zero before attempting the illegal operation.

2.3.3 LtAbortTest()

PROTOTYPE

```
LtTestStatusType
LtAbortTest(
    LtTestCtxType *pTestCtx)
```

DESCRIPTION

This function could be called by the application to abort an on-going test.

A given test might involve many small steps that are executed in a sequence. Recall that a given test is sequenced through its states using the **`LtEventHandler()`** function. This function could be called while the given test is with in its test sequence.

Calling this function **may or may not** immediately abort a test. If this function was able to immediately abort an on-going test this function returns `LT_STATUS_ABORTED` and no further test specific events are to be expected. If the process of aborting a test involves some cleanup test sequences, this function will not be able to immediately abort a test. Under such circumstances this function will initiate the sequence to abort the test and this function will return `LT_STATUS_RUNNING.` The host application could expect to receive at least one test related event under this scenario. The application is expected to call **`LtEventHandler()`** function and pass the described events until **`LtEventHandler()`** function returns `LT_STATUS_ABORTED`. The application developers are encouraged to write applications to handle both the scenarios described here.

The **`LtAbortTest()`** function should be called at the same priority level as that of the **`LtEventHandler()`** function call to prevent potential race conditions. Also, the **`LtEventHandler()`** function should not be interrupted to call the **`LtAbortTest()`** function.

INPUT

This function takes the following input arguments:

- `pTestCtx` points to an instance of the test context type described in *Memory Allocation*, on page 4. The test context passed to this function must have been initialized by **`LtStartTest()`**. Recall that the test context contains several pointers to different types of objects, including test inputs, test outputs, a line context, etc. These links must be valid for the entire duration of a test sequence. That is, the objects referenced by these pointers must not be modified or deleted during the test.

RETURN

The following LT-API result codes are returned by this function:

- `LT_STATUS_RUNNING` indicates that aborting the test involves at least one test related event. Applications should continue to call the **`LtEventHandler()`** function until that function returns a value indicating either test is aborted or completed or indicates an error.
- `LT_STATUS_ABORTED` indicates that the test has been successfully aborted and no further test specific events are expected. The test results are invalid and should be ignored.
- `LT_STATUS_DONE` indicates that test is already complete and there is no need to abort.
- `LT_STATUS_ERROR_INVALID_ARG` indicates that some function call argument is invalid.
- `LT_STATUS_ERROR_VP_GENERAL` indicates that some VP-API-II function returned an error code. The exact error returned by the VP-API-II is copied to the test result `vpGeneralErrorCode` variable.
- `LT_STATUS_TEST_ERROR_UNKNOWN` indicates that some unexpected error occurred in the LT-API, possibly due to a software bug. Such errors should never occur and should be reported to Zarlink if observed.
- `LT_STATUS_TEST_ERROR_MATH` indicates that the LT-API caught a math error such as divide-by-zero before attempting the illegal operation.

2.4 LINE TEST DESCRIPTIONS

This section describes the line test sequences supported by the LT-API.

2.4.1 Reporting Measurement Errors

The tests could be thought of as a measurement instrument. The measurement technique used in the test could experience errors while performing measurements. Such measurement errors are reported in a similar way for all the tests through the `measStatus` member variable inside each test result structure. This variable contains `LT_MSRMNT_STATUS_PASSED` or a combination (boolean 'or') of other measurement status enumeration constants defined below.

Enumeration Data Type: **`LtMsrmntStatusBitType`**

```
LT_MSRMNT_STATUS_PASSED           /* No measurement errors */
LT_MSRMNT_STATUS_EXCESSIVE_ILG    /* Excessive common mode current */
LT_MSRMNT_STATUS_DEGRADED_ACCURACY/* Degraded accuracy */
```

2.4.2 Pre Line Voltage Test

TEST ID
`LT_TID_PRE_LINE_V`

DESCRIPTION
This procedure quickly tests for the presence of a foreign voltage on the tip lead, ring lead and between the tip, ring leads. This test was designed to be run as quickly as possible in order to determine if the much longer Line Voltage Test needs to be run.

INPUT
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pPreLineVInp` is `VP_NULL` then the default inputs are used to run the test. At the moment this test does not need any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

CRITERIA
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.`pPreLineVCrt is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT
The preLineV member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
   LtPreLineVMaskType fltMask;
   LtMsrmntStatusMaskType measStatus;
} LtPreLineVResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  `Enumeration Data Type:` **`LtPreLineVBitType`**

  – `LT_PLVM_TEST_PASSED` (or `LT_TEST_PASSED`) - The measured voltages are below HEMF and FEMF test failure limits.

  – `LT_PLVM_FAILED` - A foreign voltage was found on one or more of the leads. At this point the Line Voltage Test should be run to determine where the foreign voltage is located.

  – `LT_PLVM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

2.4.3 Line Voltage Test

TEST ID
`LT_TID_LINE_V`

DESCRIPTION
This procedure tests for the hazardous electromotive force (HEMF) and foreign electromotive force (FEMF) on the tip lead, ring lead and between the tip, ring leads.

INPUT
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pLineVInp` is `VP_NULL` then the default inputs are used to run the test.
At the moment this test does not need any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

CRITERIA
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pLineVCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.

To specify a test outcome criteria, application has to make an instance of `LtLineVCriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to `pAttribues->criteria.pLineVCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtVoltageType dcHemf;
    LtVoltageType acHemf;
    LtVoltageType dcFemf;
    LtVoltageType acFemf;
} LtLineVCriteriaType;
```

- The `dcHemf` member variable indicates the foreign DC voltage above which DC HEMF fault needs to be declared. This value needs to be specified as an absolute value. Please refer *Units Used in LT-API, on page 12* for units applicable to this field.
- The `acHemf` member variable indicates the foreign AC voltage above which AC HEMF fault needs to be declared. This value needs to be specified as a RMS quantity.
- The `dcFemf` member variable indicates the foreign DC voltage above which DC FEMF voltage fault needs to be declared. This value needs to be specified as an absolute value.
- The `acFemf` member variable indicates the foreign AC voltage above which AC FEMF voltage fault needs to be declared. This value needs to be specified as a RMS quantity.

The default test criteria is indicated below. This criteria is applied if the test criteria is not specified.

– Default Foreign DC HEMF limit = 135V.
– Default Foreign AC HEMF limit = 50Vrms.
– Default Foreign DC EMF limit = 6V.
– Default Foreign AC EMF limit = 10Vrms.

OUTPUT
The `lineV` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
   LtLineVMaskType fltMask;
   LtMsrmntStatusMaskType measStatus;
   LtVoltageType vAcTip;
   LtVoltageType vAcRing;
   LtVoltageType vAcDiff;
   LtVoltageType vDcTip;
   LtVoltageType vDcRing;
   LtVoltageType vDcDiff;
} LtLineVResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  `Enumeration Data Type:` **`LtLineVBitType`**

  – `LT_LVM_TEST_PASSED` (or `LT_TEST_PASSED`) - The measured voltages are below HEMF and FEMF test failure limits.

  - `LT_LVM_HEMF_DC_TIP` - The magnitude of DC voltage detected on the tip lead exceeds the specified HEMF limits.
  - `LT_LVM_HEMF_DC_RING` - The magnitude of DC voltage detected on the ring lead exceeds the specified HEMF limits.
  - `LT_LVM_HEMF_DC_DIFF` - The magnitude of DC voltage detected across the tip and ring leads exceeds the specified HEMF limits.
  - `LT_LVM_HEMF_AC_TIP` - The AC voltage detected on the tip lead exceeds the specified HEMF limits.
  - `LT_LVM_HEMF_AC_RING` -The AC voltage detected on the ring lead exceeds the specified HEMF limits.
  - `LT_LVM_HEMF_AC_DIFF` - The AC voltage detected across the tip and ring leads exceeds the specified HEMF limits.
  - `LT_LVM_FEMF_DC_TIP` - The magnitude of DC voltage detected on the tip lead exceeds the specified FEMF limits.
  - `LT_LVM_FEMF_DC_RING` - The magnitude of DC voltage detected on the ring lead exceeds the specified FEMF limits.
  - `LT_LVM_FEMF_DC_DIFF` - The magnitude of DC voltage detected across the tip and ring leads exceeds the specified FEMF limits.
  - `LT_LVM_FEMF_AC_TIP` - The AC voltage detected on the tip lead exceeds the specified FEMF limits.
  - `LT_LVM_FEMF_AC_RING` - The AC voltage detected on the ring lead exceeds the specified FEMF limits.
  - `LT_LVM_FEMF_AC_DIFF` - The AC voltage detected across the tip and ring leads exceeds the specified FEMF limits.
  - `LT_LVM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `vAcTip` member variable indicates the measured foreign AC voltage on the tip lead. The measured voltage is specified as a RMS quantity (Note 1).
- The `vAcRing` member variable indicates the measured foreign AC voltage on the ring lead. The measured voltage is specified as a RMS quantity (Note 1).
- The `vAcDiff` member variable indicates the measured foreign AC differential voltage between tip and ring leads. The measured voltage is specified as a RMS quantity (Note 1).
- The `vDcTip` member variable indicates the measured foreign DC voltage on the tip lead (Note 1).
- The `vDcRing` member variable indicates the measured foreign DC voltage on the ring lead (Note 1).
- The `vDcDiff` member variable indicates the measured foreign DC differential voltage between tip and ring leads (Note 1).

NOTES:

1 Voltages measured in this test are expressed in mV, mVrms or by one of the special constants described in section *Special Constants*, on page 13. Applications are encouraged to be written to handle those special constants.

2 To help protect the device from possible damage, the Line V test will change the state of the line to VP_LINE_DISCONNECT if the test fails.

3 To help protect the device from possible damage, the Line V test will set the state of line relays to `VP_RELAY_RESET` for termination types that support relays.

2.4.4 Receiver Off-Hook Test

TEST ID	`LT_TID_ROH`
DESCRIPTION	This procedure can identify an on-hook phone or a off-hook phone or a resistive loop condition. An on-hook phone is considered to be passing.
INPUT	If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pRohInp` is `VP_NULL` then the default inputs are used to run the test. At the moment this test does not utilize any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.
CRITERIA	If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pRohCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test. At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT

The `roh` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtRohMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtImpedanceType rLoop1;
    LtImpedanceType rLoop2;
} LtRohResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enumeration Data Type:` **`LtRohBitType`**

  – `LT_ROHM_TEST_PASSED` (or `LT_TEST_PASSED`) - An on-hook phone was detected.

  – `LT_ROHM_RES_LOOP` - A differential resistive loop was detected.

  – `LT_ROHM_OUT_OF_RANGE_LOOP` - The device indicates the presence of an off-hook but the differential loop is beyond the measurement capabilities of the device.

  – `LT_ROHM_OFF_HOOK` - An off-hook phone was detected.

  – `LT_ROHM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `rLoop1`, `rLoop2` member variables indicate the measured loop resistances during the test (Note 1).

NOTES:

1 Resistances measured in this test are expressed in multiples of tenth of Ohms or by one of the special constants described in section *Special Constants*, on page 13. Applications are encouraged to be written to handle those special constants.

2.4.5 Ringers Equivalence Number Test

TEST ID `LT_TID_RINGERS`

DESCRIPTION This procedure measures ringer equivalence number (REN; or telephone ringer load) across the tip and ring leads.

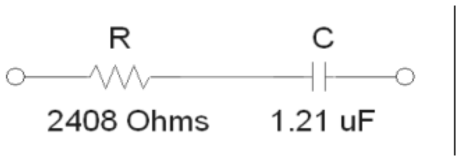
INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pRingersInp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, application has to make an instance of `LtRingerInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pRingersInp`. The contents of the test input structure are explained below.

```
Enumeration Data Type: LtRingerTestType
    LT_RINGER_REGULAR_PHNE_TEST           /* Regular phone REN */
    LT_RINGER_ELECTRONIC_PHNE_TEST        /* Electronic phone REN */
    LT_RINGER_REGULAR_PHNE_TEST_3_ELE     /* 3 Element regular phone REN */
    LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE /* 3 Element electronic phone REN */

typedef struct {
    LtRingerTestType ringerTestType;
    LtRingerTestType ringerTestType;
    LtFreqType freq;
    int32 renFactor;
} LtRingerInputType;
```

FCC part 68 section (h) of paragraph 312, for class A, B or C ringers indicates that 1 REN has an impedance of 7000 Ω at 20 Hz and 5000 Ω at 30 Hz. Zarlink has solved this requirement by creating an RC circuit that is comprised of 2408 Ω in series with 1.21 µF to represent 1 REN.

Figure 2–1 FCC Part 68 REN Model



- The `ringerTestType` member variable indicates the type of REN test to be performed. The possible options for this field are `LT_RINGER_REGULAR_PHNE_TEST`, `LT_RINGER_ELECTRONIC_PHNE_TEST`, `LT_RINGER_REGULAR_PHNE_TEST_3_ELE` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE` where the latter two are only available with the VeriVoice Professional package.
- The `vRingerTest` member variable indicates the maximum peak test signal amplitude to be applied while measuring the REN load.
- The `freq` member variable indicates the frequency of the ringing signal to be applied when `ringerTestType` is either `LT_RINGER_REGULAR_PHNE_TEST` or `LT_RINGER_REGULAR_PHNE_TEST_3_ELE`. The value of this member is not used when `ringerTestType` is either `LT_RINGER_ELECTRONIC_PHNE_TEST` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE`
- The `renFactor` member variable value depends on which `ringerTestType` has been selected.
  - If the ringerTestType is either `LT_RINGER_REGULAR_PHNE_TEST` or `LT_RINGER_REGULAR_PHNE_TEST_3_ELE` the renFactor indicates the AC impedance of a series RC at the frequency indicated by `freq` used to represent 1 REN. This value should be of type `LtImpedanceType`. The default value for this member is 7000 Ω based on the information found in Figure 2–1.
  - If the ringerTestType is either `LT_RINGER_ELECTRONIC_PHNE_TEST` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE` the `renFactor` indicates the just Capacitive portion of the load used to represent 1 REN and should be of type LtCapacitanceType. The nominal value for this member when `ringerTestType` is `LT_RINGER_ELECTRONIC_PHNE_TEST` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE` is 1,210,000 pF.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

- Default ringer test type = `LT_RINGER_REGULAR_PHNE_TEST`
- Default ringer test voltage = 16.9 Vpeak (Note 1)
- Default ringer freq = 20 Hz
- Default renFactor = 7000 Ω

CRITERIA

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pRingersCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
To specify a test outcome criteria, application has to make an instance of `LtRingersCriteriaType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->criteria.pRingerCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtRenType renHigh;
    LtRenType renLow;
} LtRingersCriteriaType;
```

- The `renHigh` member variable specifies the upper limit for the measured REN value above which test failure should be declared.
- The `renLow` member variable specifies the lower limit for the measured REN value below which test failure should be declared.

The default test criteria is indicated below. This criteria is applied if the test criteria is not specified.

- Default lower REN limit = 175 milli REN.
- Default upper REN limit = 5000 milli REN.

OUTPUT

The `ringers` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
   LtRingersMaskType fltMask;
   LtMsrmntStatusMaskType measStatus;
   LtRenType ren;
   LtRenType rentg;
   LtRenType renrg;
   LtRingerTestType ringerTestType;
} LtRingersResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enumeration Data Type:` **`LtRingersBitType`**

  - `LT_RNGM_TEST_PASSED` (or `LT_TEST_PASSED`) - The Measured REN value on the line is within the test criteria.
  - `LT_RNGM_REN_HIGH` - The Measured REN value is greater than the specified criteria.
  - `LT_RNGM_REN_LOW` - The Measured REN value is less than the specified criteria.
  - `LT_RNGM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).
- The `ren` member variable indicates the measured tip to ring REN value (Note 2).
- The `rentg` member variable indicates the measured REN value from tip to ground. The value of this member variable is equal to `LT_REN_NOT_MEASURE` by default and is changed (Note 2) when the input member variable ringerTestType is set to `LT_RINGER_REGULAR_PHNE_TEST_3_ELE` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE`. The Criteria member variables are not applied to the measured value.

- The `renrg` member variable indicates the measured REN value from tip to ground. The value of this member variable is equal to `LT_REN_NOT_MEASURE` by default and is changed (Note 2) when the input member variable ringerTestType is set to `LT_RINGER_REGULAR_PHNE_TEST_3_ELE` or `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE`. The Criteria member variables are not applied to the measured value.

NOTES

1 The maximum acceptable test voltage for the CSLAC-(880 or 890) devices is 75 Vpeak.

2 REN measured in this test is expressed in milli REN or by one of the special constants described in section *Special Constants*, on page 13. Applications are encouraged to be written to handle those special constants.

2.4.6 Resistive Fault Test

TEST ID
`LT_TID_RES_FLT`

DESCRIPTION
This procedure tests for resistive faults on the tip lead, ring lead and between tip and ring leads.

INPUT
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pResFltInp` is `VP_NULL` then the default inputs are used to run the test. At the moment this test does not need any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

CRITERIA
If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pResFltCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
To specify a test outcome criteria, application has to make an instance of `LtResFltCriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to this structure to `pAttribues->criteria.pResFltCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtImpedanceType resFltLowLimit;
} LtResFltCriteriaType;
```

- The `resFltLowLimit` member variable specifies the lower limit for the measured resistance below which resistive fault should be declared.

The default test criteria is indicated below. This criteria is applied if the test criteria is not specified.

– Default lower limit to declare resistive fault = 150 KΩ.

OUTPUT
The `resFlt` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
   LtResFltMaskType fltMask;
   LtMsrmntStatusMaskType measStatus;
   LtImpedanceType rtg;
   LtImpedanceType rrg;
   LtImpedanceType rtr;
   LtImpedanceType rGnd;
} LtResFltResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  `Enemuration Data Type:` **`LtResFltBitType`**

  – `LT_RESFM_TEST_PASSED` (or `LT_TEST_PASSED`) - Test passed. Measured leakage resistances are above the test limit.

  – `LT_RESFM_TIP` - The DC resistance measured from the tip lead to ground is less than the specified limit.

  – `LT_RESFM_RING` - The DC resistance measured from the ring lead to ground is less than the specified limit.

  – `LT_RESFM_DIFF` - The DC resistance measured across the tip and ring leads is less than the specified limit.

  – `LT_RESFM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `rtg` member variable indicates the measured resistance between the tip lead and ground (Note 1).

- The `rrg` member variable indicates the measured resistance between the ring lead and ground (Note 1).

- The `rtr` member variable indicates the measured resistance between the tip and ring leads (Note 1).

- The `rGnd` member variable indicates the measured resistance between the tip/ring lead to ground. This resistance is measured only if a fault (or very low resistance) exists between the tip and ring leads (Note 2).

NOTES

1 Resistances measured in this test are expressed in multiples of tenth of Ohms or by one of the special constants described in section *Special Constants*, on page 13. Applications are encouraged to be written to handle those special constants.

2 The `rGnd` output member is not computed for VE880 and VE890 devices

3 If one or more of rtg, rrg or rtr measurements are less than the `resFltLowLimit` only the worst case measurement is returned and the remaining values will be set to `LT_IMPEDANCE_NOT_MEASURED`.

4 If the measured result of rtg, rrg and rtr are all LT_IMPEDANCE_SHORT_CKT, this is an indication that a hard short to ground was found and the test was unable to determine which lead the fault was on.

5 If the measured result of rtg or rrg is LT_IMPEDANCE_SHORT_CKT, this is an indication that a short to ground was found on the corresponding lead and that the measured value was less than the measurable range of the device.

2.4.7 Capacitance Test

TEST ID `LT_TID_CAP`

DESCRIPTION This procedure measures capacitance on the tip lead to ground, ring lead to ground and between tip and ring leads.

INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pCapCondInp` is `VP_NULL` then the default inputs are used to run the test.
To specify the test inputs, the application has to make an instance of LtCapInputType and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pCapInp`. The contents of the test input structure are explained below.

```
Enumeration Data Type: LtCapTestFreqType
    LT_CAP_2666_HZ
    LT_CAP_1333_HZ
    LT_CAP_666_HZ
    LT_CAP_333_HZ
    LT_CAP_280_HZ

typedef struct {
    LtVoltageType       testAmp;          /* Test signal amplitude (mV RMS) */
    LtCapTestFreqType   testFreq;         /* Test signal frequency */
    LtCapacitanceType   tipCapCalValue;   /* Tip Cap Calibration Value */
    LtCapacitanceType   ringCapCalValue;  /* Ring Cap Calibration Value */
} LtCapInputType;
```

- The `testAmp` member variable indicates the RMS AC signal amplitude applied between tip and ring during the test (mV).
- The `testFreq` member variable indicates the frequency of the test signal. Where the value of the variable must be one of the `LtCapTestFreqType` enumeration values.
- The `ringCapCalValue` member variable indicates a capacitive calibration value that will be removed from the calculated tip to ground value.
- The `ringCapCalValue` member variable indicates a capacitive calibration value that will be removed from the calculated tip to ground value.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

– Default test signal amplitude = 1000 mVrms
– Default test signal frequency = `LT_CAP_DFLT_TEST_FREQ = LT_CAP_280_HZ`
– Default test tip calibration capacitance = `LT_CAP_DFLT_TIP_CAL`
– Default test ring calibration capacitance = `LT_CAP_DFLT_RING_CAL`

– The minimum signal test voltage is 100 mVrms.
– The maximum signal test voltage is 2000 mVrms.
– The minimum tip or ring calibration capacitance is 0 pF
– The maximum tip or ring calibration capacitance is 1073741824 pF
– `LT_CAP_DFLT_RING_CAL` and `LT_CAP_DFLT_TIP_CAL` indicate the use of the calibration capacitance values in the VP-API II internal device object. If no internal capacitance values are present in the device object then no correction is made.

CRITERIA If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pCapCondCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT

The cap member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
   LtCapMaskType fltMask;
   LtMsrmntStatusMaskType measStatus;
   LtCapacitanceType ctg;
   LtCapacitanceType crg;
   LtCapacitanceType ctr;
} LtCapResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  `Enumuration Data Type:` **`LtCapBitType`**

  – `LT_CAP_TEST_PASSED` (or `LT_TEST_PASSED`) - Capacitance test passed.

  – `LT_CAP_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).
- The `ctg` member variable indicates the measured capacitance between the tip lead and ground (Note 1).
- The `crg` member variable indicates the measured capacitance between the ring lead and ground (Note 1).
- The `ctr` member variable indicates the measured capacitance between the tip and ring leads (Note 1).

NOTES

1 Capacitances measured in this test are expressed in multiples of 1 pF or by one of the special constants described in section *Special Constants*, on page 13. Applications are encouraged to be written to handle those special constants.

2.4.8 All GR-909 Tests

TEST ID `LT_TID_ALL_GR_909`

DESCRIPTION This procedure runs the Line Voltage, Receiver Off-Hook, Resistive Fault and Ringer tests. However, this test is considered "complete" once a failure is detected, so not all tests will run under all fault conditions.

INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pAllGr909Inp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, application has to make an instance of `LtAllGr909InputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pAllGr909Inp`. The contents of the test input structure are explained below.

```
typedef struct {
    /* Inputs for test LT_TID_LINE_V */
    LtLineVInputType lineVInp;
    /* Inputs for test LT_TID_ROH */
    LtRohInputType rohInp;
    /* Inputs for test LT_TID_RINGERS */
    LtRingerInputType ringersInp;
    /* Inputs for test LT_TID_RES_FLT */
    LtResFltInputType resFltInp;
} LtAllGr909InputType;
```

The individual members of the above structure are explained in the respective test. The respective test also describes the default test inputs used for the test.

CRITERIA If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pAllGr909Crt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
To specify a test outcome criteria, application has to make an instance of `LtAllGr909CriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to this structure to `pAttribues->criteria.pAllGr909Crt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    /* Test criteria for LT_TID_LINE_V */
    LtLineVCriteriaType lintVCrt;

    /* Test criteria for LT_TID_ROH */
    LtRohCriteriaType rohCrt;

    /* Test criteria for LT_TID_RINGERS */
    LtRingersCriteriaType ringersCrt;

    /* Test criteria for LT_TID_RES_FLT */
    LtResFltCriteriaType resFltCrt;

} LtAllGr909CriteriaType;
```

The individual members of the above structure are explained in the respective test. The respective test also describes the default criteria used for the test.

OUTPUT The `allGr909` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtAllGr909MaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtAllGr909SubTestMaskType subTestMask;

    /* Test results for LT_TID_LINE_V */
    LtLineVResultType  lineV;

    /* Test results for LT_TID_ROH */
    LtRohResultType roh;

    /* Test results for LT_TID_RINGERS */
    LtRingersResultType ringers;

    /* Test results for LT_TID_RES_FLT */
    LtResFltResultType resFlt;

} LtAllGr909ResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtAllGr909BitType`**

  – `LT_A909TM_TEST_PASSED` (or `LT_TEST_PASSED`) - All the sub-tests passed, and are within specified test limits.

  – `LT_A909TM_LINE_V_HEMF_FAILED` - Measured line voltage exceeds the specified hazardous voltage test limits.

  – `LT_A909TM_LINE_V_FEMF_FAILED` - Measured line voltage exceeds the specified foreign voltage test limits.

  – `LT_A909TM_ROH_FAILED` - An on-hook phone was not detected and hence the test completed.

  – `LT_A909TM_RINGERS_FAILED` - Measured REN load was outside the specified test pass/fail limits.

  – `LT_A909TM_RES_FLT_FAILED` - Measured three element resistance was below the specified limit.

  – `LT_A909TM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `subTestMask` (is of type uint16) member variable indicates which sub tests were successfully initiated before this test completed. This variable will be set to `LT_NO_PF_TESTED` or a combination (boolean 'or') of the other values listed below.

  `Enemuration Data Type:` **`LtAllGr909SubTestBitType`**

  – `LT_A909STM_NO_PF_TESTED` - None of the sub-tests were completed before this test completed.

  – `LT_A909STM_LINE_V_HEMF` - The hazardous voltage test completed successfully.

  – `LT_A909STM_LINE_V_FEMF` -The foreign voltage test completed successfully.

  – `LT_A909STM_ROH` - The receiver off-hook test completed successfully.

  – `LT_A909STM_RINGERS` - The ringers tests completed successfully.

  – `LT_A909STM_RES_FLT` - The resistive faults test completed successfully.

The remaining member variables in the result structure are to be interpreted as per the definition provided in the respective tests.

2.4.9 Loopback Test

TEST ID `LT_TID_LOOPBACK`

DESCRIPTION This test could be used for production tests to verify that the digital and analog aspects of the line circuit are behaving as expected.

In case of the CSLAC devices, this test does the necessary groundwork to start the test and configure the channel to perform the loopback. It is up to the host processor to create a test signal, send it to the device's PCM highway and collect the loopback signal from the device (in response to **`the`** *VpSysPcmCollectAndProcess(), on page 52* function call by the VP-API-II). The host processor should also perform the necessary calculations to determine the outcome of the test.

It is typical that this test is used to verify the AC transmission path. In such a scenario the host processor would synthesize a signal and transmit it in PCM downstream and would collect the samples in PCM upstream and either perform RMS or FFT computation to verify the level of the test signal reflected back. It is up to the host processor to determine the necessary sample rate that is appropriate for the test signal.

To facilitate the above, the VP-API-II calls the **`VpSysPcmCollectAndProcess()`** function indicating `VP_PCM_OPERATION_APP_SPECIFIC` for the operation type. The implementation has to determine what it should do based on the identifiers passed to that function and the test that was called. If the implementation chooses to, it could pass a pointer that points to the results of the operation in the **`VpTestLineCallback()`** callback function. This pointer is then passed as typeless (void) pointer in the results of this test and would not be referenced in any way by either the LT-API or the VP-API-II.

INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pLoopbackInp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, application has to make an instance of `LtLoopbackInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pLoopbackInp`. The contents of the test input structure are explained below.

`Enemuration Data Type:` **`LtLoopbakTestType`**

– `LT_LOOPBACK_TIMESLOT` - This type of test performs a bit for bit loop back. The loop back happens at the PCM interface of the device just prior to the CODEC. Note, one can expect to see the exact same bit pattern in the loopback signal. The test will not send the applied signal to the line.

– `LT_LOOPBACK_BFILTER` - This type of test performs a half digital loop. The loop back happens in the device's B-Filter block prior to the Digital to Analog converter. When making use of this loopback mode, various filter coefficients in the device are configured such that the signal has nominally unity gain through the loopback. Note, one **can not** expect to see the same bit pattern in the loopback signal. The test will not send the applied signal to the line.

– `LT_LOOPBACK_CODEC` - This type of test performs a full digital loop back. The loop back happens at the output of CODEC just prior to 4 wire to 2 wire conversion. When making use of this loopback mode, various filter coefficients in the device are configured such that the signal has nominally unity gain through the loopback. Note, one **can not** expect to see the same bit pattern in the loopback signal. The test will also send the applied signal to the line.

– `LT_LOOPBACK_ANALOG` - This type of test performs an analog loop back test with the impedance connected at the tip and ring interface. If the tip and ring leads are in open circuit condition one could expect to see all of the transmitted power reflect back. If a test load is connected then the amount of power reflected back is determined by the source impedance ($\mathbf{Z_S}$) of the loaded AC profile and the load impedance that is present at the tip and ring interface. It is possible connect the test load across the tip and ring leads using the **`VpSetRelayState()`** function. See the *VoicePath API-II User's Guide* for more details.

```
typedef struct {
    LtLoopbakTestType loopbackType;
    LtTimeType waitTime;
    LtTimeType loopbackTime;
} LtLoopbackInputType;
```

- The `waitTime` member variable indicates the time before starting the test. This value is passed as the "settling time" to the **`VpSysPcmCollectAndProcess()`** function. The maximum acceptable value for this field is `8191875`.
- The `loopbackTime` member variable indicates the measurement time for the test. This value is passed as the "operation time" to the **`VpSysPcmCollectAndProcess()`** function. The maximum acceptable value for this field is `8191875`.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

- Default test type = `LT_LOOPBACK_CODEC`
- Default wait time = 0
- Default loopback time = 1024 * 125 micro seconds

CRITERIA

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pLoopbackCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT

The `loopback` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtLoopbackMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    void *pApplicationInfo;
} LtLoopbackResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtLoopbackBitType`**

  - `LT_LOOPBACK_TEST_PASSED` (or `LT_TEST_PASSED`) - Loopback test passed.
  - `LT_LOOPBACK_TEST_FAILED` - Loopback test failed.
  - `LT_LOOPBACK_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `pApplicationInfo` member variable has meaningful information only when the LT-API is used along with the CSLAC device family. This member variable contains a pointer that was passed to the Line Test API by the system service layers.

NOTES

1 For all `loopbackType` enumerations except `LT_LOOPBACK_TIMESLOT` the VP-API II line state of the channel under test must be one the following:

VP_LINE_ACTIVE, VP_LINE_ACTIVE_POLREV, VP_LINE_OHT, VP_LINE_OHT_POLREV, VP_LINE_TALK, VP_LINE_TALK_POLREV.

If any other line state is detected the test will return a status of LT_STATUS_TEST_NOT_SUPPORTED.

2 Hook transitions detected during the test will cause the test to abort and return a status of LT_STATUS_ERROR_VP_TEST with a vpTestErrorCode of VP_TEST_STATUS_ABORTED.

2.4.10 DC Feed Self-Test

TEST ID `LT_TID_DC_FEED_ST`

DESCRIPTION This test is used to verify the DC feed is working as expected. This test measures the voltage and current across the known test load resistor using the DC feed profile that has been programmed. It then uses the measured parameters to compute the test load resistance. The test outcome is derived by comparing the known test load resistance against the computed test load resistance.

INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pDcFeedSTInp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, the application has to make an instance of `LtDcFeedSTInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pDcFeedSTInp`. The contents of the test input structure are explained below.

```
typedef struct {
    VpLineStateType lineState;
} LtDcFeedSTInputType;
```

- The `lineState` member variable indicates the VP-API-II line state in which this test should be conducted. Meaningful values for this field would include feed states (like for example: `VP_LINE_ACTIVE`, `VP_LINE_ACTIVE_POLREV` etc). It is also possible to run this test without specifying changing the line state (i.e, by making use of the line state of the line prior to calling this test) by assigning `LT_DC_FEED_ST_DFLTI_LINE_STATE` to this field.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

– Default test voltage = `LT_DC_FEED_ST_DFLTI_LINE_STATE`

CRITERIA If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pDcFeedSTCrt` is `VP_NULL` then the default test criteria is used to determine the outcome of the test.
To specify a test outcome criteria, application has to make an instance of `LtDcFeedSTCriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to this structure to `pAttribues->criteria.pDcFeedSTCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtPercentType rLoadErr;
} LtDcFeedSTCriteriaType;
```

- The `rLoadErr` member variable indicates the measurement tolerance to be applied for the measured test load resistance value when an external test load resistor is used.
  – If the `rLoadErr` test criteria is not specified a default criteria of 15% is applied.
- When using the Internal Test Termination, a fixed criteria of 10% is applied, based on comparing the measured iTestLoad with the ILA current limit specified in the DC Profile. The internal test termination applies an effective short circuit across tip and ring and therefore the rLoadErr criteria is ignored.

OUTPUT The `dcFeedST` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtDcFeedSTMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtImpedanceType rTestLoad;
    LtVoltageType vTestLoad;
    LtCurrentType iTestLoad;
} LtDcFeedSTResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtDcFeedSTBitType`**

  – `LT_DC_FEED_ST_TEST_PASSED` (or `LT_TEST_PASSED`) - Indicates DC feed self-test passed.

  – `LT_DC_FEED_ST_TEST_FAILED` - Indicates DC feed self-test failed.

  - `LT_DC_FEED_ST_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `rTestLoad` member variable indicates the value of the external test load resistor if present.
- The `vTestLoad` member variable indicates the measured voltage across the test load resistor or internal test termination using the loaded DC feed profile. This value could be used by the application to verify the DC feed profile given the test load resistance.
- The `iTestLoad` member variable indicates the measured current through the test load resistor or internal test termination using the loaded DC feed profile. This value could be used by the application to verify the DC feed profile given the external test load resistance. When the internal test termination is used, this result is compared with the DC feed profile to determine pass / fail status.

2.4.11 DC Voltage Test

TEST ID `LT_TID_DC_VOLTAGE`

DESCRIPTION This test is used to verify the line circuit has the ability to drive the voltage ranges required for the normal operations of the line circuit like ringing the phone, providing DC feed etc. This test applies the specified voltage with normal polarity and reverse polarity using a slow ramp such that the customer phone does not ring. It then measures the applied voltage and compares it against the expected value to determine the test outcome.

INPUT If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pDcVoltageInp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, application has to make an instance of `LtDcVoltageInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pDcVoltageInp`. The contents of the test input structure are explained below.

```
typedef struct {
    LtVoltageType testVoltage;
} LtDcVoltageInputType;
```

- The `testVoltage` member variable indicates the test voltage to be applied (Note 1).

The default values for test inputs are indicated below. These are used if test inputs are not specified.

– Default test voltage = `56V`

CRITERIA If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pDcVoltageCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test. To specify a test outcome criteria, application has to make an instance of `LtDcVoltageCriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to this structure to `pAttribues->criteria.pDcVoltageCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtPercentType voltageErr;
} LtDcVoltageCriteriaType;
```

The default test criteria is indicated below. This criteria is applied if the test criteria is not specified.

– Default voltage measurement tolerance = 10%.

OUTPUT The `dcVoltage` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtDcVoltageMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtVoltageType measuredVoltage1;
    LtVoltageType measuredVoltage2;
} LtDcVoltageResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtDcVoltageBitType`**

  – `LT_DC_VOLTAGE_TEST_PASSED` (or `LT_TEST_PASSED`) - DC voltage test passed.

  – `LT_DC_VOLTAGE_TEST_FAILED` - DC voltage test failed.

  – `LT_DC_VOLTAGE_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `measuredVoltage1` member variable indicates the measured voltage across the tip and ring leads when the specified test voltage is applied across the tip and ring leads.

- The `measuredVoltage2` member variable indicates the measured voltage across the tip and ring leads when the specified test voltage with sign inversion is applied across the tip and ring leads.

NOTES

1 The maximum acceptable test voltage for the CSLAC-880/890 devices is 154V.

2.4.12 Ringing Self-Test

TEST ID

`LT_TID_RINGING_ST`

DESCRIPTION

This test verifies various aspects related to ringing. These include the following:

Ringing signal generation

The test generates ringing signal at the specified amplitude and frequency and measures the amplitude and frequency of the generated signal. The measurement results are compared against the expected results.

Ringing signal drive capability

The test generates ringing signal at the specified amplitude and frequency and connects a test load resistor across the tip and ring leads and measures the AC impedance of the test load by measuring the AC current through the test load. Such a measurement verifies the system's ability to drive the ringing signal.

Ring-trip verification

This is verified by first generating ringing signal as per the ringing signal definition contained in the ringing profile provided through the test input. The test then creates an off-hook condition using the test load resistor and the test expects to see the VTD/VP-API-II report ring trip.

The test outcome is determined based on the measurement results of the steps defined above.

For the line circuits that do not have the provision to disconnect the line circuit from the loop, the customer has to make sure either they communicate to the customer that the phone is going to ring while performing the test, or take necessary precautions to prevent ringing the subscriber phone by specifying low enough ringing signal through the test input parameters.

INPUT

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pRingingSTInp` is `VP_NULL` then the default inputs are used to run the test. To specify the test inputs, application has to make an instance of `LtRingingSTInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pRingingSTInp`. The contents of the test input structure are explained below.

```
typedef struct {
    LtVoltageType vRinging;
    LtFreqType freq;
    VpProfilePtrType pRingProfileForTest;
    VpProfilePtrType pDcProfileForTest;
} LtRingingSTInputType;
```

- The `vRinging` member variable indicates the RMS AC ringing signal amplitude (Note 1) to be applied during the ringing signal amplitude, frequency and AC test load measurement phase of the test.
- The `freq` member variable indicates the frequency of the ringing signal to be applied during the ringing signal amplitude, frequency and AC test load measurement phase of the test.
- The `pRingProfileForTest`, `pDcProfileForTest` indicates the ringing and dc profiles to be used for the ring trip phase of the test. If either profile is `VP_PTABLE_NULL` the test will use a matched set of default profiles to perform the ring trip test. The default ringing profile consists of a 20 Hz, 1 Vpk sine wave with a -25 VDC bias. The DC profile uses a DC Ring Trip algorithm with a ring trip threshold of 13 mA.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

– Default ringing signal amplitude = `12 Vrms`

– Default ringing signal frequency = `20 Hz`

– Default DC profile = DC Ring Trip algorithm with a ring trip threshold of 13 mA.

– Default Ring Profile = 20 Hz, 1.0 Vpk, -25.0 V offset

CRITERIA

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pRingingSTCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test. To specify a test outcome criteria, application has to make an instance of `LtRingingSTCriteriaType` and fill in the appropriate values in that object. Then it needs to assign the object's pointer to this structure to `pAttribues->criteria.pRingingSTCrt`. The contents of the test criteria structure are explained below.

```
typedef struct {
    LtPercentType openVoltageErr;
    LtPercentType freqErr;
    LtPercentType rLoadErr;
} LtRingingSTCriteriaType;
```

- The `openVoltageErr` member variable indicates the tolerance to be applied for the open circuit ringing signal amplitude measurement result while determining the test outcome.
- The `freqErr` member variable indicates the tolerance to be applied for the ringing signal frequency measurement result while determining the test outcome.
- The `rLoadErr` member variable indicates the tolerance to be applied for the AC impedance measurement result of the external test load resistor in determining the test outcome. This criteria is ignored when using the internal test termination.

The default test criteria is indicated below. These criteria are applied if the test criteria is not specified.

- Default open circuit ringing signal voltage measurement tolerance = 10 %.
- Default ringing signal frequency measurement tolerance = 10 %.
- Default external test load AC impedance measurement tolerance = 15 %.
- Default internal test termination ringing current measurement tolerance = 10 %
  Expected current is based on the input voltage and resulting current in the presence of a short circuit.

OUTPUT

The `ringingST` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtRingingSTMaskType fltMask;
    LtVoltageType openCktRingVol;
    LtFreqType freq;
    LtImpedanceType acRload;
    LtCurrentType iRLoad;
    LtMsrmntStatusMaskType measStatus;
} LtRingingSTResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtRingingSTBitType`**

  - `LT_RINGING_STM_TEST_PASSED` (or `LT_TEST_PASSED`) - Ringing self-test passed.
  - `LT_RINGING_STM_OPENV_OC` - Ringing self-test failed because the measured ringing open circuit voltage was outside the specified criteria.
  - `LT_RINGING_STM_FREQ_OC` - Ringing self-test failed because the measured ringing frequency was outside the specified criteria.
  - `LT_RINGING_STM_ACRLOAD_OC` - Ringing self-test failed because the measured AC test load impedance was outside the specified criteria (Applies to External Test Load only).
  - `LT_RINGING_STM_IRLOAD_OC` - Ringing self-test failed because the measured AC test load current was outside the specified criteria (Applies to Internal Test Termination only).
  - `LT_RINGING_STM_OFF_HOOK` - Ringing self-test failed to make any measurements because an off-hook phone was detected.
  - `LT_RINGING_STM_NO_RINGTRIP` - Ringing self-test failed because no ring trip was detected.
  - `LT_RINGING_STM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `openCktRingVol` member variable indicates the measured open circuit ringing voltage across the tip and ring leads. This voltage is expressed as an RMS quantity.
- The `freq` member variable indicates the measured ringing signal frequency.
- The `acRload` member variable indicates the measured AC external test load resistance or apparent impedance when the internal test termination is applied, which can be influenced by the ringing current limit.

- The `iRload` member variable indicates the measured RMS AC current through the test load. This current is measured when the specified ringing voltage (through `vRinging` member of the test input) is applied across the external test load resistor or internal test termination.

NOTES

1 The maximum acceptable test voltage for the CSLAC-880/890 devices is 53 Vrms.

2 The user application MUST restore the line under test’s RING and DC profiles after the test completes using the VP-API II functions VpConfigLine() or VpInitLine().

2.4.13 On/Off Hook Self-Test

TEST ID

`LT_TID_ON_OFF_HOOK_ST`

DESCRIPTION

This test is used to verify that the on-hook and off-hook events are detected by the VP-API-II and the VTD. This test creates on-hook and off-hook conditions on the line using the external test load resistor or internal test termination and verifies that the expected hook conditions are reported by the VP-API-II. The test outcome is based on the comparison between the expected and reported hook status.

INPUT

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pOnOffHookSTInp` is `VP_NULL` then the default inputs are used to run the test. The following structure has been defined to provide the test inputs.

```
typedef struct {
    bool overrideOffHook;
} LtOnOffHookSTInputType;
```

- The `overrideOffHook` member variable is used to specify the test behaviour if the test is called under off-hook condition. The line termination types that can disconnect the subscriber loop from the line circuit makes use of this member variable, and either continue to run the test if this member variable is set to `TRUE`, or aborts the test if this member is set to `FALSE`. For the line termination types that do not support disconnecting the loop, this member variable must be set to `FALSE` so that test can be aborted if off-hook is detected before starting the test.

The default values for test inputs are indicated below. These are used if test inputs are not specified.

– Default value for the override flag = `FALSE`

CRITERIA

If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pOnOffHookSTCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The application must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT

The `onOffHookST` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtOnOffHookSTMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
} LtOnOffHookSTResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtOnOffHookSTBitType`**

  – `LT_ON_OFF_HOOK_STM_TEST_PASSED` (or `LT_TEST_PASSED`) - On/off hook self-test passed.

  – `LT_ON_OFF_HOOK_STM_TEST_ABORTED` - Indicates that the test was aborted due to a an off-hook condition and the test input was not set to override the off-hook condition.

  – `LT_ON_OFF_HOOK_STM_TEST_HW_FAULT` - Indicates a hardware fault was detected, and the hook status reporting mechanisms did not report the expected hook status during the test.

  – `LT_ON_OFF_HOOK_STM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

2.4.14 Read Battery Conditions

TEST ID	`LT_TID_RD_BAT_COND`
DESCRIPTION	This test is used to read the battery voltages connected to the line circuit.
INPUT	If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->inputs.pReadBatCondInp` is `VP_NULL` then the default inputs are used to run the test. At the moment this test does not need any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.
CRITERIA	If either the `pAttribues` argument in the **`LtStartTest()`** function or the `pAttribues->criteria.pReadBatCondCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test. At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT

The `readBatCond` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtReadBatCondMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtVoltageType bat1;
    LtVoltageType bat2;
    LtVoltageType bat3;
} LtReadBatCondResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtReadBatCondBitType`**

  – `LT_RD_BAT_CONDM_TEST_PASSED` (or `LT_TEST_PASSED`) - Read battery voltages test passed.

  – `LT_RD_BAT_CONDM_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `bat1, bat2` and `bat3` member variables indicate the measured battery voltages. The interpretation of the battery names depends on the type of VTD that is being used. Please refer *VoicePath API-II User's Guide* for information on the mapping of the battery names to the actual batteries.

2.4.15 Read Loop Conditions

TEST ID `LT_TID_RD_LOOP_COND`

DESCRIPTION This test is used to read the loop conditions of the line given the current line state without disturbing the T/R feed conditions. It would generally be used to measure loop length (for possible gain adjustment) or as a quick self-test of the feed conditions while either on-hook (e.g., vab) or off-hook (e.g., imt).

INPUT This test does not need any test input parameters.

CRITERIA No test criteria has been defined for this test.

OUTPUT The `readBatCond` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtReadLoopCondMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
    LtImpedanceType rloop;     /* A-B Loop Resistance */
    LtCurrentType imt;         /* A-B Loop Current */
    LtCurrentType ilg;         /* (A+B)/2 Current */
    LtVoltageType vab;         /* A-B Loop Voltage */
    LtVoltageType vag;         /* A-Gnd Voltage */
    LtVoltageType vbg;         /* B-Gnd Voltage */
    LtVoltageType vbat1;
    LtVoltageType vbat2;
    LtVoltageType vbat3;
} LtReadLoopCondResultType;
```

- The `fltMask` member variable is of type `uint16` and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of other enum values shown below.

  `Enemuration Data Type:` **`LtRdLoopCondBitType`**

  – `LT_RD_LOOP_COND_TEST_PASSED` (or `LT_TEST_PASSED`) - Read loop conditions test passed.

  – `LT_RD_LOOP_COND_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.).

- The `vbat1, vbat2` and `vbat3` member variables indicate the measured battery voltages. The interpretation of the battery names depends on the type of VTD that is being used. Please refer *VoicePath API-II User's Guide* for information on the mapping of the battery names to the actual batteries.

2.4.16 Master Socket Detection Test

TEST ID `LT_TID_MSOCKET`

DESCRIPTION This procedure detects the presence of a master socket:

Type 1: 470kΩ resistor and a diode in series.

Type 2: 470kΩ resistor and a 1.8µF capacitor in series.

INPUT To specify the test inputs, the application has to make an instance of `LtMSocketInputType` and fill in the appropriate values in that object. Then it needs to assign this object's pointer to `pAttribues->inputs.pMSocketInp`. The contents of the test input structure are explained below (Note 1).

Enumeration Data Type: `LtMSocketType`

`LT_MSOCKET_TYPE_1`

`LT_MSOCKET_TYPE_2`

```
typedef struct {
    LtMSocketType   mSocket;  /* Master socket type */
} LtMSocketInputType;
```

- The `mSocket` member variable indicates the type of master socket expected on the line

CRITERIA If either the pAttribues argument in the `LtStartTest()` function or the `pAttribues->criteria.pMSocketCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT The `mSocket` member of the `LtTestResultType.result` union is defined as follows:

```
typedef struct {
    LtMSocketMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
} LtMSocketResultType;
```

- The `fltMask` member variable is of type uint16 and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  Enumuration Data Type: `LtMSocketBitType`

  – `LT_MSKT_TEST_PASSED` (or `LT_TEST_PASSED`) - Test Passed. Master socket is present.

  – `LT_MSKT_NOT_PRESENT` - Master socket not present

  – `LT_MSKT_TWO_PARALLEL` - Two parallel master sockets

  – `LT_MSKT_TWO_OPPOSITE` - Two opposing master sockets

  – `LT_MSKT_TWO_REVERSE` - Two reversed master sockets

  – `LT_MSKT_REVERSE` - Reversed Master Socket

  – `LT_MSKT_FAULT` - Line Fault

  – `LT_MSKT_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the measStatus field in the results structure (for bit definition of the `measStatus` field See *Reporting Measurement Errors*, on page 20.

NOTES

1 There is no default value for the `mSocket` input. Therefore the `LtTestInputType` and `LtTestAttributesType` cannot be NULL. when running this test.

2 The test will only return `LT_MSOCKET_PRESENT` or `LT_MSOCKET_NOT_PRESENT` if the `LT_MSOCKET_TYPE_2` input is specified.

3 If the test returns `LT_MSOCKET_FAULT` (as well as `LT_TEST_MEASUREMENT_ERROR`) it may due to an excessive leakage. It is recommended to run the Resistive Faults test in this case.

2.4.17 Cross Connect Detection Test

TEST ID `LT_TID_XCONNECT`

DESCRIPTION This procedure detects the presence of a cross connect on an FXS line.

INPUT If either the pAttribues argument in the `LtStartTest()` function or the `pAttribues->inputs.pXConnectInp` is `VP_NULL` then the default inputs are used to run the test.

At the moment this test does not need any test input parameters. The applications must specify default inputs for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

CRITERIA If either the pAttribues argument in the `LtStartTest()` function or the `pAttribues->criteria.pXConnectCrt` is `VP_NULL` then the default criteria is used to determine the outcome of the test.
At the moment no test criteria has been defined for this test. The applications must specify default criteria for this test using the mechanisms defined above to ensure compatibility to future versions of the LT-API.

OUTPUT The `xConnect` member of the `LtTestResultType`.result union is defined as follows:

```
typedef struct {
    LtXConnectMaskType fltMask;
    LtMsrmntStatusMaskType measStatus;
} LtXConnectResultType;
```

- The `fltMask` member variable is of type uint16 and indicates faults that may have occurred during the test. This variable can contain `LT_TEST_PASSED` or a combination (boolean 'or') of the other enum values shown below.

  Enumeration Data Type: `LtXConnectBitType`

  – `LT_XCONNECT_TEST_PASSED` (or `LT_TEST_PASSED`) – Test Passed. No cross connect detected.

  – `LT_XCON_NORMAL_POLARITY` – Cross connect (A->A / B->B)

  – `LT_XCON_REVERSE_POLARITY` – Cross connect (A->B / B->A).

  – `LT_XCON_FAULT` – Line Fault

  – `LT_XCONNECT_MSRMNT_STATUS` (or `LT_TEST_MEASUREMENT_ERROR`) - Indicates that there was a measurement error. The exact measurement error that was experienced is indicated by the `measStatus` field in the results structure (for bit definition of the measStatus field See *Reporting Measurement Errors*, on page 20.

CHAPTER

3 SYSTEM SERVICES

3.1 OVERVIEW

This chapter describes the three system service layer functions used by the VoicePath API to support line testing in case of VoicePort VE(880 or 890) series device line testing. These functions are not necessary when implementing line testing for other devices.

These functions are in addition to the system service layer functions listed in the *VoicePath API-II User's Guide*. The new functions are as follows:

`VpSysTestHeapAquire()` - Required

`VpSysTestHeapRelease()` - Required

`VpSysPcmCollectAndProcess()` - Optional

3.2 SYSTEM SERVICE FUNCTIONS

3.2.1 VpSysTestHeapAcquire()

PROTOTYPE

```
void *
VpSysTestHeapAcquire(
    uint8 *pHeapId);
```

DESCRIPTION

This system services function is required by the VP-API and needs to be implemented by the customer. This function is accessed by the VP-API while performing line testing.

The VP-API calls this function when it requires storage space for storing intermediate test related information. The size of the buffer necessary can be obtained by performing **`sizeof(VpTestHeapType)`**. The `VpTestHeapType` data type is defined in the VP-API.

This function is called every time an application starts a test. The buffer is released at the end of the test (or upon test failure). Please see *Typical Usage*, on page 8 for more information on when this function is called.

The test related intermediate information is not stored in the VP-API device object because of the following reasons:

- The intermediate test related information is quite extensive and hence increases the size of each VP-API device object.
- Testing is usually not an ongoing operation.
- Generally, only one or two lines in a system may actually be performing line testing at any given time.

The number of possible devices that can run a line test at any given time is decided by the number of `VpTestHeapType` buffers the customer's application is able to maintain.

Each time the **`VpSysTestHeapAcquire()`** function is called, it must return a valid buffer pointer or `VP_NULL` when no buffer is available. It must also return a buffer ID to the location pointed by `pHeapId` argument upon successful acquisition of the buffer.

The allocated buffer must not be reused by any other application until the VP-API releases the buffer by calling the **`VpSysTestHeapRelease()`** function.

INPUT

This function takes the following input arguments:

– `pHeapId` - Pointer to the location where the identifier for the buffer should be stored.

OUTPUT

This function must modify the value pointed to by `pHeapId` with an ID that the system service layer will recognize when the VP-API releases the test buffer resource.

RETURN

This function returns a pointer to a memory segment with the size of `VpTestHeapType`. If the function was unable to allocate test heap the function must return `VP_NULL`.

DEVICES

CSLAC-(880 or 890)

3.2.2 VpSysTestHeapRelease()

PROTOTYPE

```
bool
VpSysTestHeapRelease(
   uint8 heapId);
```

DESCRIPTION

This is a system services function required by the VP-API that needs to be implemented by the customer. This function is accessed by the VP-API while performing line testing. Please see *Typical Usage*, on page 8 for more information on when this function is called.

The VP-API makes use of this function when it has completed a line test. The purpose of the function is to return the ownership of the test heap, which was acquired using **`VpSysTestHeapAcquire()`**, back to the system.

INPUT

This function takes the following input arguments:

- `heapId` indicates which of the test heap buffers the VP-API no longer needs to be able to access.

OUTPUT

NONE.

RETURN

The following indications are returned by this function:

- `TRUE` - the function was successful
- `FALSE` - the function was unsuccessful

DEVICES

CSLAC-(880 or 890)

3.2.3 VpSysPcmCollectAndProcess()

PROTOTYPE

```
void
VpSysPcmCollectAndProcess(
   void *pLineCtx,
   VpDeviceIdType deviceId,
   uint8 chanId,
   uint8 startTimeslot,
   uint16 operationTime,
   uint16 settlingTime,
   uint16 operationMask);
```

DESCRIPTION This optional system service function is used to collect and process PCM samples from a CSLAC-(880 or 890). This function only needs to be implemented when collecting and processing PCM samples via MPI commands (EZ mode) from the CSLAC-(880 or 890) device is not feasible.

The VP-API configures the VTD such that the physical parameter that needs to be measured is routed to the Analog to Digital Converter (ADC). The routed analog signal could include physical parameters such as the DC/AC voltage or DC/AC current etc. The CODEC digitizes the analog signal at an 8kHz (narrowband) or 16kHz (wideband) rate and then transmits the data to an internal PCM buffer and the transmit PCM interface block of the device.

Line test accuracies are only guaranteed when the digital samples are collected at a minimum sampling rate of 500 Hz or higher.

PCM samples must be collected using one of the two schemes discussed below. When deciding which scheme to use, the implementation takes into account system design considerations.
The two schemes for collecting PCM samples are as follows:

- **EZ Mode PCM sample collection**

  By default the VP-AP II is capable of collecting and processing PCM samples from the internal PCM buffer via an MPI command. This implementation know as EZ mode is enabled by default in the vp_api_cfg.h file for 880 and 890 devices with the VP880_EZ_MPI_PCM_COLLECT and VP890_EZ_MPI_PCM_COLLECT defines respectively.

  It is recommended to use EZ mode unless the application is unable to meet the requirements below.

  Requirements:

  In order to use EZ mode and still meet the required 500 Hz sampling rate the VP-API II VpApiTick() function must be called at a minimum of 10 ms intervals in narrowband or 5ms intervals in wideband.

  When compiled for EZ mode, the VP-API will increase MPI traffic to/from the device by 17 bytes every tick for the duration of a test.

- **PCM sample collection from PCM transmit Interface**

  This function supplements the device capabilities and should be implemented if the EZ mode requirements cannot be met. Collecting PCM samples via the PCM interface should only be considered if the host processor (or some other peripheral device) has access to the PCM interface.

  Requirements:

  The implementation is expected to collect and process 16 bit linear samples from the PCM transmit interface from two consecutive timeslots specified by the `startTimeslot` input argument, at a sampling rate between 500Hz and 8 kHz in either narrowband or wideband modes of operation.

  Implementation of this function must be non-blocking in nature. In other words, this function should not wait to collect the samples. Instead, it should perform the necessary setup and configuration for such a collection and data post processing. These configurations might involve configuring for data settling time, data integrate time, and the type(s) of data post-processing operations. Once the necessary initialization are done, this function should return to allow the API to continue regular operation.

  Once the indicated data has been collected and all mathematical operation(s), specified by the `operationMask` input argument, have been performed. The customer's source code must then call the **`VpTestLineCallback()`** function to communicate the results (or errors) of the measurements back to the LT-API.

INPUT

This function takes the following input arguments:

- `pLineCtx` argument is a pointer that points to the location of the current VP-API line context. This argument value must be maintained (with no modifications to either this value or to any content pointed by this argument) by the customer implementation and passed back into the VP-API via the **`VpTestLineCallback()`** function once all PCM data operations are complete.
- Once the VP-API calls the **`VpSysPcmCollectAndProcess()`** function, the customer application must respond by calling the **`VpTestLineCallback()`** function within a certain time frame. If the callback function is not called within the required time frame, the VP-API will report a timeout to the LT-API. The LT-API in turn will report the error indicated by the API with an error `LT_STATUS_ERROR_VP_TEST`. The time frame with in which the customer application must call the **`VpTestLineCallback()`** is calculated as indicated below:

  ```
  operationTime + settlingTime + VP_PCM_CALCULATION_TIME (default 1000ms)
  ```
  NOTE: The last constant in the above equation is defined in `vp_api_cfg.h`.

- `deviceId` indicates the device that is requesting to use this system service function. In systems with more than one VTD attached to the host microprocessor, the application can use this information to identify the VTD that is needing this service. In most implementations the `deviceId` argument can be ignored.
- `chanId` indicates which line is requesting the use of this system service function. In most implementations, the `chanId` argument can be ignored.
- `startTimeslot` - During all VP-API tests, the VoicePort device is placed into 16-bit (linear) mode. The start time slot communicated in this function is the timeslot that was set by the application using **`VpSetOption(VP_OPTION_ID_TIMESLOT...).`** In this mode, the device requires two consecutive timeslots on the PCM highway (The assigned timeslots are used in the MPI data collection mode as well). The customer's application is responsible for making sure that the device can be placed into this mode without causing PCM data from other channels to overlap. The VP-API is expecting the customer application to collect and operate on PCM samples in linear mode starting with `startTimeslot` where the data contained in the `startTimeslot` is the msb. The PCM data is represented in 16 bit 2's complement mode in linear CODEC mode.

- `operationTime` indicates how long to collect and operate on PCM samples. The value of this variable is in 125-µs increments, and it is up to the customer to convert this number into the actual number of PCM samples at the systems PCM collection rate. For example, if the customer is able to collect PCM samples at a 4-kHz rate and the `operationTime` value is 1600 (125 µs * 1600 = 0.2 s), then the customer must collect and operate on 0.2 s * 4 kHz = 800 samples. Round the calculated number of samples to nearest integer.
  The PCM highway always transmits data at an 8-kHz rate (and the MPI PCM data register is updated at 8 -kHz rate). A lower data sampling rate could be obtained by skipping samples. Line testing accuracies for VoicePort devices are guaranteed at sampling frequencies of 500Hz or higher.
- `settlingTime` indicates how long to delay before collecting and operating on PCM samples. The value of this variable is in 125-µs increments and it is up to the customer to convert this number into the actual number of PCM samples at the systems PCM collection rate. For example, if the customer is able to collect PCM samples at a rate of 4 kHz and the `settlingTime` value is 800 (125 µs * 800 = 0.1 s), then the customer must skip 0.1 s * 4 kHz = 400 samples. Round the calculated number of samples to nearest integer.

  `operationMask` (is of type `uint16`) indicates what operation(s) to perform on the PCM data. These operations are indicated by bitwise 'or' of the enumeration data type members shown below. The **`VpSysPcmCollectAndOperate()`** function must be able to collect and operate on the data in multiple ways at the same time. A simple mask scheme is used to indicate which operation(s) are requested by a particular test. The **`VpTestLineCallback()`** function must be able to provide the results pertaining to all the requested data operation types indicated by this argument. Currently, the following operations must be supported by the system service layer:

`Enumeration Data Type:` **`VpPcmOperationBitType`**

`VP_PCM_OPERATION_AVERAGE`

This mask indicates that the `int16` PCM samples collected by the system service layer must be added up and divided by the number of actual samples that were collected during the `operationTime` period. The implementation should ensure that there is no overflow while performing the average function.

`VP_PCM_OPERATION_RANGE`

This mask indicates that the system service layer must return the maximum PCM sample value minus the minimum PCM sample value found during the `operationTime` period.

`VP_PCM_OPERATION_RMS`

This mask indicates that the system service layer must return the measured Root Mean Square (RMS) value of the PCM samples collected during the operationTime period. The implementation should ensure that there is no overflow while performing the RMS function. The RMS can be computed using the following relation.

$$\mathbf{rms = (\{ (x_1^2 + x_2^2 \ldots + x_N^2) / N \} - \{ (x_1 + x_2 \ldots + x_N) / N\}^2)^{0.5}}$$

Where x_1, x_2, x_N are PCM samples and N is the number of samples. Note that the above equation removes the DC (average) component before computing the RMS.

`VP_PCM_OPERATION_MIN`

This mask indicates that the system service layer must return the minimum value that is found in the PCM samples during the `operationTime` period.

`VP_PCM_OPERATION_MAX`

This mask indicates that the system service layer must return the maximum value that is found in the PCM samples during the `operationTime` period.

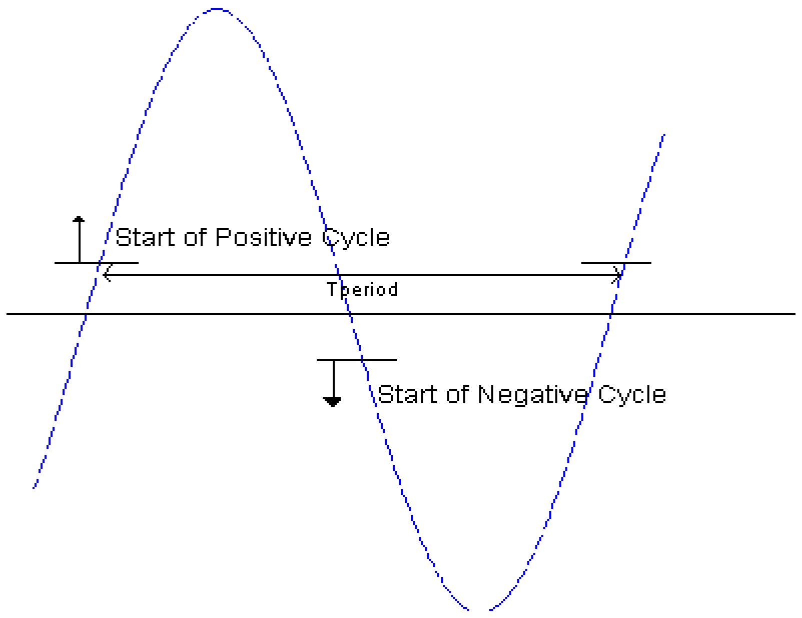
`VP_PCM_OPERATION_APP_SPECIFIC`

When this operation type is passed the application could perform any calculations (or none) it chooses to do based on the identifiers it is passed and given the high level test it is running.

`VP_PCM_OPERATION_FREQ`

This mask indicates that system service layer must compute the average frequency of signal present in the PCM samples. This operation is requested while performing the ringing self test. During this test, the tip and ring lead differential voltage is routed on to the PCM highway. The test would make sure no DC component is added to the ringing signal. Further, the signal level is high enough so that noise contribution to the test can be ignored. Also, for the purposes of this measurement it is safe to assume that there is only one sinusoidal signal is present in the collected PCM samples. To measure the frequency the implementation would measure the average period. A measurement approach is illustrated in the following figure.

Figure 3–1 Frequency Measurement



To measure accurate results the implementation should be able to detect transition from the negative cycle to the positive cycle and vice versa. It might also want to incorporate a hysteresis to the transition points. The suggested hysteresis is 512. Based on this hysteresis an implementation could consider it has detected a transition to positive cycle if it sees two consecutive PCM samples such that x(n-1) < 512 and x(n) >= 512. The same concept should be extended to detecting negative cycles as well. The implementation should now measure the period (or count the number of samples and then translate it to time based on sampling rate) between similar transitions (while making sure it sees an opposite transition in between). To improve the accuracy the implementation should compute the average period. The average frequency can be computed by measuring the ratio of the average period. The reported frequency is in milli hertz resolution.

OUTPUT

This function is not responsible for returning any data directly to the calling function. However, the application that starts a PCM collection routine is responsible for the calling the VP-API **`VpTestLineCallback()`** function. The prototype of that function and the format of the return data are described below.

- **`VpTestLineCallback()`** `prototype:`

```
void
VpTestLineCallback(
    VpLineCtxType *pLineCtx,
    VpPcmOperationResultsType *pResults);
```

`pLineCtx` - pointer to the line context of the line under test. This was passed to the **`VpSysPcmCollectAndProcess()`** function.

`pResults` - pointer to a structure containing the results of all the requested operations on the collected PCM data. The following is the definition of the structure.

```
typedef struct {
    /* Specified operationMask == VP_PCM_OPERATION_AVERAGE */
    int16 average;/* Average value of all collected PCM samples. */

    /* Specified operationMask == VP_PCM_OPERATION_RANGE */
    uint16 range;/* Maximum - minimum PCM value during operation time */

    /* Specified operationMask == VP_PCM_OPERATION_RMS */
    int16 rms;    /* RMS value of PCM samples during operation time */

    /* Specified operationMask == VP_PCM_OPERATION_MIN */
    int16 min;    /* Minimum value of PCM samples during operation time */

    /* Specified operationMask == VP_PCM_OPERATION_MAX */
    int16 max;    /* Maximum value of PCM samples during operation time */

    /* Specified operationMask == VP_PCM_OPERATION_APP_SPECIFIC */
    void *pApplicationInfo;/* Any (or no)results that the implementation
                            * chooses to pass. This data is not interpreted
                            * neither by the VP-API nor the LP-API. The
                            * LP-API passes this pointer back to the
                            * application as part of the test result. */

    /* Specified operationMask == VP_PCM_OPERATION_FREQ */
    int32 freq;     /* Measured average frequency */

    bool error;   /* Indication of error in PCM data collection or
                   * result computation(TRUE = failure) */
} VpPcmOperationResultsType;
```

RETURN NONE

DEVICES CSLAC-(880 or 890)

CHAPTER

4 LT-API DEBUG

4.1 OVERVIEW

The LT-API source code includes many different types of debug output which customers can use to isolate problems with their application and/or system. Each type of output can be include/excluded at compile time. If included, the output can be enabled/disabled at run time using the `ltDbgFlag` member of the `LtTestAttributesType` structure that is described in *LtStartTest()*, on page 15.

4.1.1 TYPES OF DEBUG OUTPUT

The following types of debug output are supported:

Table 4–1 Debug Flags

Macro	Description	Purpose
`LT_DBG_ERRR`	Error messages	Most LT-API Functions display error messages when they return a value other than LT_STATUS_SUCCESS
`LT_DBG_WARN`	Warning messages	Reports behavior that is probably not what was intended
`LT_DBG_INFO`	Info messages	Gives status information during normal operation of complicated functions.
`LT_DBG_FUNC`	Function entry/ exit messages	Displays a message each time a LT-API function is called, and another message when the function returns.

The above macros are defined as bitmasks which can be ORed together to select any combination of debug output types. The macro `LT_DBG_ALL` is provided for selecting all debug output types.

4.1.2 DEBUG OUTPUT SELECTION AT COMPILE TIME

Debug output strings, and the code necessary for displaying them, can occupy a non-negligible amount of memory. Therefore, the LT-API provides the ability to exclude unwanted types of debug output at compile time.

Customers may want to compile in all types of debug during initial development, but include less debug output in the final application. Or, if separate “production” and "debug" builds are maintained, a different selection of debug output may be specified in each.

The selection of debug output types at compile time is specified in the header files in the lt_api/ includes directory:

The LT_DEBUG option (in lt_api_pkg.h) applies to all types of debug output.

- If LT_DEBUG is undefined, then all debug output is compiled out.
- If LT_DEBUG is defined, then the LT_DBG_DFLT (in lt_debug.h) macro selects which types of debug output will be included.

Any debug output that is excluded at compile time cannot be enabled at runtime.

4.1.3 DEBUG OUTPUT SELECTION AT RUN TIME

At run time, the `ltDbgFlag` member of the `LtTestAttributesType` structure that is described in *LtStartTest()*, on page 15 can be used to enable/disable debug output.

The `ltDbgFlag` argument should be a uint32 number. The value should be the bitwise OR of one or more debug output selectors (Table 4–1). Other debug output types will be disabled.

CHAPTER

5 REVISION HISTORY

REVISION A3

- Added new LT-API function to support aborting the test.
- Added a new test for performing on/off hook self test.
- Added a new special constant to indicate current not measured.
- Added test inputs to REN measurement test.
- Added test inputs to GR909 all test.
- Removed SLAC calibrate test since its SLAC calibration is now performed in the individual tests themselves.
- Added test criteria for the DC feed self test.
- Modified the ringing self-test to support ringing frequency measurement. Also modified the PCM collection and processing function to support frequency measurement.

REVISION A4

- Changed the default constant for the PCM calculations time (`VP_PCM_CALCULATION_TIME`) to 1000ms.
- Changed the specification of test voltage for the Ringers equivalence number test from an RMS quantity to peak voltage. This field is now applicable for electronic phone test type as well. Also, changed the default value for this field to 16.9Vpeak.
- Changed the default test voltage for the Ringing self-test to 12Vrms.
- Added maximum test voltage to Ringer equivalence number test, DC voltage test.
- Modified the Ringing self-test fault masks and also removed the boolean flag to indicate ring trip detected (now part of the fault mask).
- Renamed the `LtHzType` to `LtFreqType`.
- Added test inputs to the DC feed self test to specify the line state in which the test should be conducted.

REVISION A5

- Changed the default ringing self test AC load impedance measurement tolerance from 10% to 15%.

REVISION A6

- Modified the `LtVp880TestTopologyType` struct to include 4 new member variables.
- Modified the `LtRingerTestType` enum to include two new Ringer algorithms `LT_RINGER_REGULAR_PHNE_TEST_3_ELE` and `LT_RINGER_ELECTRONIC_PHNE_TEST_3_ELE`. Modified the `LtRingersResultType` struct to include the new member variables `rentg`, `renrg` and `ringerTestType`.

REVISION A7

- Modified the LtRingerInputType struct to include two new members. This change will break and previously existing code that makes use of the LtRingerInputType struct.
- Modified the All GR-909 Test sub test order and abort conditions.

REVISION A8

- Corrected the Physical Units description of `LtPercentType` from tenth of a percent to milli percent.

REVISION A9

- Replaced the `pRingingProfileAfterTest` with `pDcProfileForTest` in the `LtRingingSTInputType` struct. This change will break any previously existing code that makes use of the `pRingingProfileAfterTest` member of the `LtRingingSTInputType` structure.
- Added support for the `VP_TERM_FXS_SPLITTER` termination type.
- Resistive Faults test now goes into disconnect prior to running algorithm.

REVISION A10

- Added a better description of the Special Constance in Table 2–4
- Added #define **LT_API_VERSION_TAG** to LT-API and description to document.

REVISION A11

- Added 890 Chipset.

REVISION A12

- Added a new Receiver Off-Hook enumeration to the LtRohBitType enum.

REVISION A13

- Added a new Get Loop Conditions test
- Improved some of the formatting and figures (to replace "Legerity" with "Zarlink" where appropriate).

REVISION A14

- Added a new termination types `VP_TERM_FXS_SPLITTER_LP` and `VP_TERM_FXS_ISOLATE` to 890 chipset series.
- Added comments regarding test abort due to thermal fault detected in the VP-API-II.

VERSION 15

- Added a new termination types `VP_TERM_FXS_ISOLATE_LP` to 880 chipset series.
- Added comments regarding the DC and Ringing Profiles used during Ringing Self Test.

VERSION 16

- Added support for Internal Test Termination to *DC Feed Self-Test*, on page 36, *Ringing Self-Test*, on page 39 and *On/Off Hook Self-Test*, on page 42. This enables more devices to support these self tests. Use of the internal test termination does modify return values and pass/fail criteria.
- Added 3 Element *Capacitance Test*, on page 30.
- Added degraded accuracy return type for Resistive Faults in the presence of large capacitance and capacitance test with large capacitances in *Reporting Measurement Errors*, on page 20.
- Added information pertaining to CSLAC-(880 or 890) series line test support of the VP-API II VP_OPTION_WIDEBAND CODEC mode in *VpSysPcmCollectAndProcess()*, on page 52.
- Added info pertaining to the EZ mode PCM collection option in *VpSysPcmCollectAndProcess()*, on page 52.

VERSION 17

- Internal Release

VERSION 18

- Added *Code Size Management*, on page 2
- Added *Master Socket Detection Test*, on page 45.
- Added *Cross Connect Detection Test*, on page 47.
- Added description of new `LT_LOOPBACK_BFILTER and LT_LOOPBACK_TIMESLOT` to the `LtLoopbakTestType` enumeration of the *Loopback Test*, on page 34
- Added `LT_MAX_CAPACITANCE` to *Special Constants*, on page 14.
- Added `ltDbgFlag` to the `LtTestAttributesType` discussed in *LtStartTest()*, on page 15.
- Added Chapter *LT-API DEBUG*, on page 57 to the document.
- Added notes to *Line Voltage Test*, on page 22 pertaining to the affects the test has on the VP-API II.
- Added notes to *Resistive Fault Test*, on page 28 pertaining to the returned results.
- Added description of new `LT_RINGING_STM_OFF_HOOK` to the `LtRingingSTResultType` enumeration of the *Ringing Self-Test*, on page 39.

For more information about all Zarlink products visit our Web Site at:

www.zarlink.com