Generic Open Link-Layer API for Unified Media Access—GOLLUM

D3.3
Validation & Performance Report

Contractual date of delivery to the CEC:	**31st Aug 2006**
Actual date of delivery to the CEC:	**31st Aug 2006 (v1.0)**
	21st Dec 2006 (v2.0 including the activities during the project extension period)
Editor:	**Mahesh Sooriyabandara, Costas Efthymiou (TREL)**
Authors:	**Tim Farnham, Costas Efthymiou (TREL), Mahesh Sooriyabandara (TREL),**
	Arthur van Rooijen (MA), Diego Melpignano (STM), Jose Antonio Galache, Veronica Gutierrez (UC), Matthias Wellens (RWTH), Alain Gefflaut (EMIC)
Participants:	**STM, RWTH, EMIC, UC, TID, TREL, MA**
Workpackage	**WP3**
Security:	**Public**
Nature:	**R**
Version:	**2.0**
Total number of pages:	**128**

Abstract:

The report presents the results of ULLA validation tests, which assess the correct operation of interface functions. The API validation tests were performed on prototype API implementations, to ensure that the API accommodate for all the requirements defined in the ULLA requirements document. Further to this, results of the performance analysis of ULLA implementations, on different target platforms with different operating systems and communication interface combinations are described. Alternative implementations have been tested and evaluated with the view of finding optimal solution/s and/or implementation options in terms of key requirements such as scalability, extensibility and responsiveness as described in the requirements document.

Keyword list: Unified Link Layer API, Validation, Performance Analysis

Document Revision History

Version	Date	Author	Summary of main changes
0.01	14/05/06	TREL	Proposed table of contents
0.02	22/05/06	TREL	Updated table of contents
	20/06/06	UC	Added section 2
0.03	22/08/06	RWTH	Added description of validation of low-level API
0.04	24/08/2006	EMIC	Added EMIC performance evaluation section.
0.05	28/08/06	UC	Added contribution to section 5.4.2
1.0	30/8/06	TREL	Finalised document
1.01	11/12/06	TREL	Extended the conformance test and benchmarking sections
1.02	14/12/06	RWTH	Minor corrections
1.9	18/12/06	TREL	Updated document with partner contributions
2.0	19/12/06	TREL	Finalised updated document

Contents

1. Introduction

The ULLA validation and performance report describes the evaluation process and results of ULLA API framework validation and performance analysis. The aim of this exercise is to assess, whether the design and implementations of the ULLA conform to the ULLA requirements identified during the design process. Testing aims to validate the functionality of the ULLA and evaluate performance on a diverse range of platforms supporting different OSes and networking interfaces. The details of ULLA requirements could be found in [3].

1.1. Purpose

This purpose of this document is to

- present a common evaluation framework to evaluate the independent ULLA implementations
- define validation test cases
- present the results and outcome of validation tests
- present the results and outcome of performance analysis of ULLA on a range of platforms

1.2. Scope

This document provides the details of validation and performance testing of the ULLA to validate the final API design against the original requirements identified during initial stage of the project. The aim is to validate and evaluate both functional and non-functional (e.g. performance, scalability etc.) aspects using prototype implementations. The validation tests are intended to provide, full coverage of requirements defined in the ULLA requirements document (D2.2). However, defining test cases for some non-functional requirements is not trivial. In those cases, the method used for validation will be described separately. In addition, scope of some test cases will be limited to a particular version of ULLA implementation. The validation will only test the ULLA requirements and testing of Link User and device driver requirements will be beyond the scope of this activity

Section 2 of this document presents an overview of ULLA design. It describes the main functional blocks in the ULLA framework as well as Link Provider and Link User interfaces. Section 3 provide and overview of the testing procedure, test environments and requirement mapping to test cases. Further details on the testing platforms are in the Appendix A and test specification could be found in Appendix B. Section 4 describes the description of the outcome of the validation tests. Section 6 presents the results and the outcome of the performance evaluation tests. Section 7 concludes by summarising overall findings of the tests. Additional details on implementation specific results and recommendations could be found in ULLA API Guide Book.

1.3. Definitions

Command

Commands are issued from LUs to the ULLA and forwarded to the corresponding LPs or Links. They support for configuring and controlling the behaviour or state of the addressed entity. Commands and attributes are organized in separate inheritance hierarchies.

Most relevant commands on LPs support the possibility to discover new Links. Links generally might not have so many commands, but for instance connect and disconnect are mandatory. Connectionless links might not do anything there except returning OK.

Link

LLAs can register links for pre-registered LPs to ULLA. For ULLA a link is exhaustively defined by a link class description.

With respect to the ISO/OSI or TCI/IP model a link is intended to be an interconnecting communication channel in layer-2. Pair of layer-2 addresses, which usually identify the two respective peers, uniquely identify each link. In this context, peers are the terminal installations, which implement the layer-2 communication protocol For instance a single network interface card (NIC) might be able to maintain multiple links with different peers.

Link Layer Adapter (LLA)

Link layer adapters are not visible to ULLA or Link Users.

A link layer adapter is a piece of software, which registers Link Providers and links to ULLA by the means of the defined interface. A link layer adapter is typically tailored for a specific device driver or device driver tool, respectively.

Link Provider (LP)

LLAs can register Link Provider with ULLA, defined by their Link Provider class description. A LP itself hosts links with the same link class description.

A LP is an abstraction of a network interface card (NIC). For instance a LLA for a multi-mode NIC supporting GPRS and WLAN might register two LPs with ULLA. Each of them is defined by the means of a specific class description.

Link User (LU)

A Link User is any application registering with ULLA as a Link User.

Here, the term application is understood in a software architectural sense. It does not limit the potential Link Users to "layer-7" applications with reference to a communication model, but also includes communication middleware, transport entities or routing agents.

Notification

A notification is send from ULLA to the LU when a pre-registered UQL condition on LP or Link attributes is met. Additionally, they can indicate a change in LP or Link state, such as new LP registered or Link deregistered. The UQL language is also used at registration time to specify the associated information coming alongside with the notification.

This ULLA feature is very powerful for LUs that are interested in link adaptation. Most applications might reside on highly portable attribute conditions for many links, such as browsers, while others make use of the full flexibility of the UQL, such as connection managers or multimedia clients.

Query

A query is send from LU to ULLA in order to retrieve information about LP or Link attributes by the means of a UQL statement. When used with requestInfo() the result is returned as fast as possible and may be empty. When used with requestNotification() the functional call blocks until the condition is met and the result is non-empty.

A query specifies a list of interesting attributes for the LU, such as bandwidth, and optionally a condition that all in the result listed LPs or Links have to meet. The scope of a query can be scaled-down to a particular class, such as Cellular-Link-class or IEEE 802.11-LinkProvider-class. In sub-classes more attributes are available, for instance cellular-type of links specific attributes. If an attribute is requested from a class, which does not support it, the query fails. A validity parameter indicates the requirements on the freshness of the addressed attributes.

ULLA Core (UC)

ULLA Core is the module that implements all the functionalities of ULLA including the management of the interfaces towards Link Users and Link Providers. ULLA Core comprised of three functional components that deals with command processing, query processing and event processing.

ULLA Storage

ULLA keeps persistent information in the ULLA storage. This can rely on a legacy database or can be a custom realization tuned for a given platform.

The ULLA storage system will typically contain link related information and parameters which will be accessed by the Link User by means of the ULLA Query Language. UQL queries are translated in specific ULLA storage system language by the ULLA core.

Unified Link Layer API (ULLA)

ULLA is an abstraction of an operating-system independent implementation of the defined standard interfaces, which Link Users and link layer adapters can use.

1.4. Acronyms

API Application Programming Interface

CMP Connection Management Platform

HED High End Device

LD Limited Device

LLA Link Layer Adaptor

LP Link Provider

LU Link User

RMP Realtime Multimedia Platform

SMP Streaming Multimedia Platform

SNP Sensor Network Platform

TC Test Case

TG Test Group

UC ULLA Core

UCP ULLA Command Processing

UCR ULLA Command Requirements

UEP ULLA Event Processing

UER ULLA Event Requirements

UGR ULLA General Requirements

ULLA Unified Link Layer API

UNR ULLA Notification Requirements

UQP ULLA Query Processing

UQR ULLA Query Requirements

URD ULLA Requirements Document

VLD Very Limited Device

2. ULLA Design & Requirements Overview

A general overview of ULLA architecture is presented in this section. The main components that constitute it and their functionality as well as interrelations between each of them are described. Firstly, a summary of the requirements to be fulfilled by the ULLA are presented. Then the main modules of ULLA are described using the appropriate terminology.

2.1. ULLA requirements

Considering the ULLA functionality, the following high level requirements have been identified:

- **Provide link information and configure links**

  Applications may require specific information before selecting and trying to configure certain links. The access to these links is achieved through ULLA as follows. The link information is obtained via queries sent through the ULLA to the corresponding link provider that returns to the user the values of the corresponding link attributes. On the other hand, the links could be configured changing some of the attributes that characterize them. The commands used to configure these links need to be processed to determine which link provider the command corresponds to.

- **Process link events**

  ULLA should implement handlers processing the events sent by the link providers to the ULLA. These events could be forwarded to the link user or analyzed within the ULLA Core (UC) in order to store their results and use them in the future t do statistics.

  From all links discovered by the link provider, only some of them are interesting from the user point of view. These events should be processed to form the notifications that are sent to the Link Users (LU). The rest of them provide information used for building statistics of the main network attributes.

- **Notify of appropriate link changes and statistics**

  LUs are only notified of those events that they are interested. These notifications are received with the suitable time space and granularity, parameters previously defined by the user when required the notification. The information provided could be used by the LU to generate statistics of interesting communication parameters (e.g. packet error rate, RSSI). These notifications could also warn the LU when some attribute fulfils the specified condition (e.g. send an alarm when battery level is under a certain threshold).

Apart from providing a uniform API, the ULLA architecture has been designed to fulfil the following requirements.

I. Compatibility and Extensibility: The proposed architecture should work well with old implementations and should be able to integrate new link layer technologies, extending the existing classes or creating new ones.
II. Scalability and Portability: The architecture design should work in a wide range of devices. It should be light enough to be used on limited platforms as sensor devices as well as more capacity devices like PDAs or laptops.

III. Interoperability and Security: LU should be able to work with different ULLA implementations, independently of the platform and Operating System, where they are running on. As the information provided by ULLA is quite important for communications, the registration process of a LU with ULLA should be secure enough.

IV. CPU usage limitation, memory occupation and battery life: ULLA should not be very resource demanding (in terms of CPU usage and memory) and be battery efficient. These concepts could be determinant for the performance of low battery and memory constraint devices, such as sensor nodes.

V. Link inheritance and link locking: The proposed architecture lets some link instances to inherit the characteristics of another link instance and then, expand it adding specific functionality (e.g. 802.11a and 802.11b links inherit 802.11 link parameters). On the other hand, when some links are using the same technology, communication channel or frequency could not be able to work simultaneously. There should also be a method to provide mutual exclusion between the different links.

2.2. *ULLA architecture*

The aim of ULLA API is to provide an abstract view of the link layers to the Link Users (LU). In this sense, LU will access in a uniform way to link parameters, modifying or getting their values, without knowing the subjacent technology as well as internal drivers. The generic ULLA architecture is shown in the next figure:

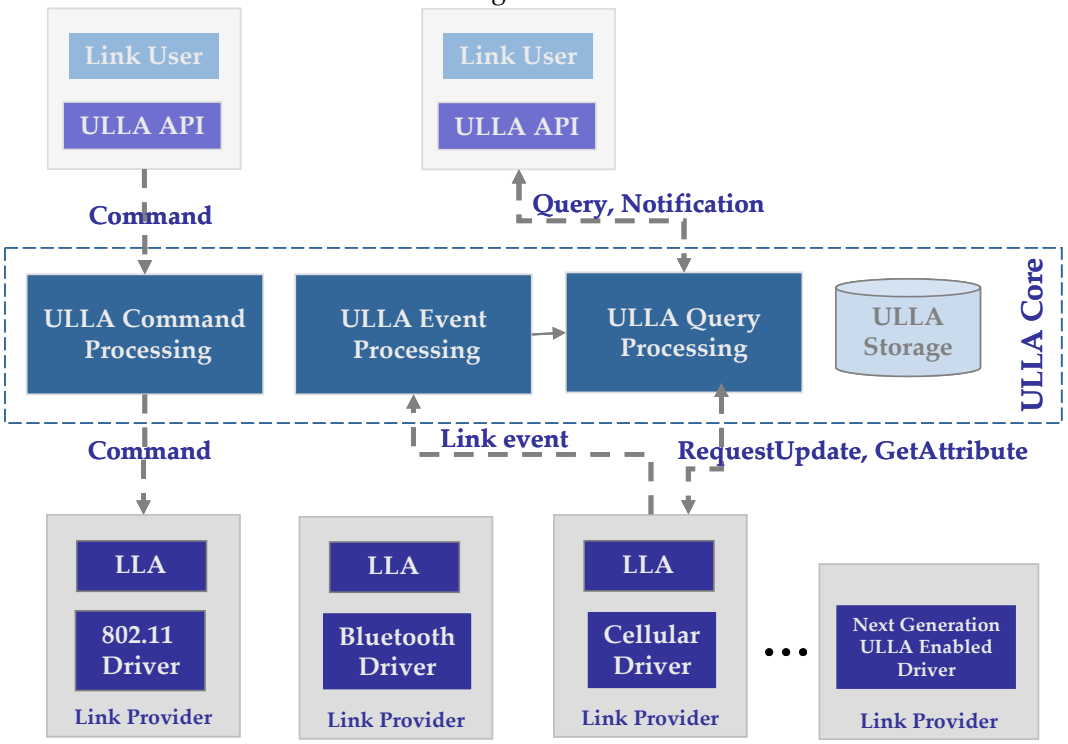

Figure 2-1 ULLA Architecture

As it can be seen in Figure 2-1, the ULLA architecture composes of three main parts: LUs, ULLA Core (UC), and Link Layer Adapters (LLA). UC interacts with LLAs and LUs, in order to achieve the required functionality. In a typical implementation, LU will use ULLA user

application library to communicate with the UC, which processes this information, parses it and forward it to the corresponding LLA.

There are four functional entities within the ULLA architecture that form the UC, namely ULLA Query Processing, ULLA Event Processing and ULLA Command Processing and ULLA Storage. The first three entities are focused on handling and processing of commands, queries, notifications and events sent to and from the UC. The last one is intended to store selected attribute values to be retrieved when queried by the LU or remain stored in order to perform statistics.

The next paragraphs describe the details about ULLA entities, their functionality, interactions and interfaces provided by them:

- **ULLA Query Processing (UQP)**: It implements a direct interface with LU and handles all requests coming from the LU. Two types of requests can be handled by the UQP functionality: processes queries (information requests) and notification requests. The first ones are ULLA Query Language (UQL) sentences sent to the UQP requesting for an attribute value/s. These values are taken from the ULLA Storage and returned back to the corresponding the LU. The second type specifies conditions on one or several attributes. These conditions will be evaluated and when fulfilled, the corresponding notification is sent back to the LU.
- **ULLA Command Processing (UCP)**: It is used to handle command requests arriving from LU. The main purpose of the UCP is to map command strings, received as command requests, to function calls and to forward this information to the appropriate underlying LP. The commands could be synchronous or asynchronous, being used in this last case the UQP to set a notification request.
- **ULLA Event Processing (UEP)**: It handles all events reported by the link providers. Three types of events could be defined:
    - Attribute events: Sent by link providers either on a regular basis or upon reaching a predefined threshold.
    - Link events: Sent periodically from the link providers to the UEP in order to report the list of currently detected links.
    - Command completion events: sent by link providers upon completion of a command executed in an asynchronous mode.
- **ULLA Storage**: This module is in charge of storing class definitions (representing link definitions) as well as instances of the classes (e.g representing discovered links). When a LU queries for a link attribute, it will be retrieved from the ULLA storage instead of having to access to the underlying driver.
- **Link Layer Adapters (LLA)**: ULLA envisages Link Providers to implement an interface and built certain types of functionality into them to realise ULLA services. As today's drivers don't support the ULLA, they require some form of adaptation, which is provided by a software layer called LLA.

3. Test Procedure

This section describes the testing procedure used for validation of the ULLA. The primary purpose of validation and performance tests is to uncover the limitations of the design and measure its full capabilities. Therefore, the testing process aimed at validating the ULLA design against the requirements and evaluating the performance of different ULLA implementations. Four different prototype implementations developed independently by four project partners; namely, TREL, STM, EMIC and RWTH covering a range of usage scenarios was used as testing platforms. This approach enables full requirements coverage.

The following sub sections describe a number of different tests used for the validation. They include API conformance testing, functional testing, performance testing, etc. In addition to ULLA design validation; these tests are aimed at finding optimal solution/s in terms of key requirements such as scalability, extensibility and responsiveness with respect to different platforms and implementation options. The test platforms and ULLA implementations as well as an overview of test cases are presented in this section. The detailed test case description can be found in the Appendix B.

3.1. Testing Environments

This section describes the test beds used for the validation. Details of different hardware, software platforms and type of ULLA implementation are summarised here. A detailed description on each of test platforms could be found in Appendix A.

3.1.1. Test Platforms

The following table summaries specification of each test platform. Additional details on each of these platforms could be found in the Appendix A.

No	Test Env	Hardware Platform									Operating System
		Partner	Processor	RAM	ROM	GPRS/GSM		Wifi	BT	Sensor	
1	MSP	TREL	XscalePXA255	64MB				PCMCIA D-Link 802.11b			winCE4.2 and arm-Linux
2	RMP	STM	Nomadik 8810 (ARM9 + 2xDSP)	64MB	32 MB NAND + 32 MB NOR Flash			STM STLC4370 802.11g			arm-Linux 2.4.19
3	CMP	EMIC	HTC Wizard Pocket PC. OMAP850 195MHZ	64MB	64 MB flash	yes		802.11g	yes		Windows Mobile 5.0
4	SNP	UC	ATMEL 7.37 MHz ATMega128L, low- power, 8-bit micro-	4KB SRAM + 128KB Program flash	4 KB EEPROM + 512KB External flash					XBOW MIB510 Programming board XBOW Micaz/Mica2	TinyOS 2.0 compatible with TinyOS 1.x

			controller	memory	memory					processor/radio modules XBOW MTS310/MTS420 sensor boards	

Table 3-1 Overview of Test Platforms

3.1.2. ULLA Core Implementations

The following table summaries the details of different platforms tested during the validation and performance evaluation study.

No	Prototype	System memory occupancy	Data storage	Implementation details	Usage
1	MSP	User Space	Postgres 8.0.2 & Polyhedra™ 6.2	Shared library	Dynamic linking
2	RTP	User space	SQLite-3	Daemon	Sys V IPC through shared library
3	CMP	User space	Proprietary	Stream device driver	Ioctl to driver.
4	SNP	User Space	Static allocation of a chunk of memory	Modules included statically through a configuration file	Static memory allocation

Table 3-2 Details of ULLA Core implementations used in each test platform

3.2. Traceability of Requirements

The following table maps various requirements to the ULLA core implementations used in each test platform. It also shows relationship between the test cases and the requirements. The complete test specification (i.e. details of test cases) could be found in Appendix B.

URID	Requirement	Test Platform				Test Case Identifier
		MSP	CMP	RMP	SNP	
UQR1	Synchronous Query	X	X	X	X	TC 2
UQR2	Asynchronous Query	X	X	X	X	TC 3
UQR3	ULLA Query Language	X	X	X	X	TC 2,3

UQR4	Link Statistics Query	X	X	X	X	TC 11
UQR5	Mandatory Statistics Functions	X	X	X	X	TC 11
UQR6	Optional Statistical functions	X		X		NT
UQR7	Operation (Statistical or otherwise) Extensibility	X		X		NT
UQR8	Qualifier for attributes	X	X	X		TC 2 (62-68)
UQR9	Minimal qualifier set	X	X	X		TC 2 (62-68)
UQR10	Link Information Access	X	X	X	X	TC 2
UQR11	Query processing fairness	X	X	X		NT
UQR12	Minimal Number of query per seconds.	X	X	X	X	TC 2.10
UQR13	Query latency	X	X	X	X	TC 2.10
UCR 1	Command Capability	X	X	X	X	TC 4
UCR 2	Command List	X	X	X	X	TC 9
UCR 3	Command Extensibility	X	X	X	X	NT
UCR 4	Command Conflict Handling	X	X	X		TC 8
UCR5	Synchronous Command Execution	X	X	X	X	TC 4
UCR6	Asynchronous Command Execution	X	X	X	X	TC 5

Table 3-3 Requirements mapping to test cases and test platforms

3.3. Test Procedure

A test suite is used to validate the API against the ULLA specification described in D3.1. The test cases described in Appendix B are incorporated in the test suite. Additional test cases described in Appendix C provide the test coverage for rest of the ULLA requirements including all non-functional aspects.

The following table summaries the approaches used to test different aspects of the ULLA.

Feature	**Test Method**	**Description**
API Conformance	Test suite	Appendix B
Functionality	Test suite	Appendix B
Security	Test suite	Appendix B

Stress & Volume	NOT TESTED	Appendix C
Performance	Test Suite	Appendix B
Recovery	NOT TESTED	Appendix C
Scalability	Test suite & Demonstration	Appendix B & Appendix C
Extensibility	Demonstration	Appendix C
Interoperability	Demonstration	Appendix C
Portability	Demonstration	Appendix C

Table 3-4 Testing method used for testing of different aspects of ULLA

3.4. Testing procedure for validation of the low level API

Both Wireless Sensor Networks (WSN) implementations are based on the TinyOS operating system and the nesC programming language. As the standard GOLLUM test suite is based on the full profile and is implemented in C it cannot be used to evaluate and validate the WSN versions.

Instead separate functional validation tests were performed using dummy Link Users that test diverse ULLA features. Additionally, the performance of the overall system was evaluated using specific test programs repeating certain usage scenarios numerous times and averaging the investigated performance metrics over these results. Respective results are presented in section 5.4.

4. Results from Validation Testing

A validation test program that implements test cases 1-6 and 13 was executed on laptop and PDA test platforms. Tests were not performed on the sensor prototype. All tests were targeted at validating the conformance of ULLA APIs used in each implementation against the final ULLA specification described in D3.1. In addition, test cases 2.11 and 13.1 have been used for performance benchmarking. The complete conformance test specification can be found in Appendix C, together with the complete set of results in Appendix D.

4.1. API Conformance Test Results

Platform	Hardware	OS	ULLA Core	Code footprint	ULLA storage	Code footprint
Triton – Local GDB Linux	PXA270, CPU 520MHz, 128MB RAM	Linux 2.6.17	User space shared Library	189k	Postgres	2.6M (backend) 112k (frontend)
Tecra M3 – Local GDB Linux	Pentium M, CPU 1.8GHz, 1GB RAM	Linux 2.4.x	User space shared Library	249k	Postgres	16M (backend) 134k (frontend)
HTC Wizard Windows Mobile 5.0	OMAP 850 195 MHz	Windows CE 5.0	Stream interface driver	72k	Parser based proprietary storage	Part of the ULLA Core

Table 4-1 ULLA implementations used for conformance tests

Table 4-1 summarises the details of the ULLA implementation, the hardware and the operating system of the test platforms used to run the conformance test suite. The test platforms include a kernel space ULLA implementation running on a Windows CE based smart-phone device, a user space Linux implementation on a laptop, and finally a user space Linux implementation on embedded device. The table shown in Appendix D summarises the results from the test runs on the three test platforms.

The pass/fail status and the time taken to complete each test case have been recorded in the results table shown in the Appendix D. The conformance criteria used by the test program only considers the returned error code for assessing the conformance. Therefore, a pass indicates that the returned error code for each API call tested is in conformance with the error codes defined in the ULLA API specification.

The results show that kernel space ULLA implementation on Windows CE based PDA device passed majority of the tests, indicating that it is almost fully conformed to the ULLA specification. The failures have resulted mainly because of a slight semantic difference in some of the helper functions used to retrieve information from an UllaResult structure. For example, the Windows CE implementation does not return an error when calling ullaQueryValueType without calling first ullaResultNextTuple.

The user space ULLA implementation tested on laptop and embedded (Triton) platforms essentially used the same code. As expected, similar pass/fail results were observed. The

results for the user space implementation on two test platforms passed most of the test cases but higher number of failed test cases was observed. When analysing the results for the user space ULLA implementations, it was found that most of the test case failures resulted from nothing other than mismatched error code syntax. This was expected, given that the partners have been developing the prototype in parallel to the development of API specification. Given that the Gollum API specification was only finalised at the end of the project period, partners did not have enough time to synchronise the header files they used with the final version.

Latency results for test group 1, indicates that link user registration and unregistration process for Linux based user space implementation is much higher that the kernel based ULLA implementation. The higher delay for Windows CE implementation is explained by the fact that each time the unregister Lu function is called, the Ulla core is actually stopped and the LLAs are unloaded.

Command latency for kernel space ULLA implementations on Windows CE platform was insignificant in comparison to the performance of user space Linux implementation. There are two main reasons for that. First the clock resolution on Windows CE does not go under 1 millisecond so the 0 value simply means that the time is less than 1 millisecond. Second as the ULLA core runs actually in the context of the device manager on Windows CE, there is no costly context switch from user to kernel space to pay. As the test was just calling a dummy command, the measurement represents simply the time to call a function in a LLA and come back.

The latency for Information requests and request notifications was much less on the Windows CE based test platform when compared with the user space Linux implementations. A detailed analysis of this could be found in the next subsection.

Overall the validation testing results were promising and in line with the expectations of consortium. The reason for failure of some of the test cases is due to different interpretations of the ULLA specification and also, in some cases to incomplete implementation of all of the result codes. However, the different interpretations were limited to minor aspects such as the detailed meaning of error codes etc. This highlights the need for conformance testing to ensure different implementations of the specification will produce the same behaviours. For instance, the TREL ULLA core implementation makes the following assumptions:

1) The error code returned is not as important as the implementation of the ULLA API functionality therefore some error codes are not implemented and the generic ULLA_ERROR_FAILED is returned instead of specific codes, for instance ULLA_ERROR_VERSION_MISMATCH in test case 1.1 and ULLA_ERROR_UNSUPPORTED_PROFILE in test case 1.4.

2) It is assumed that all UQL related errors are reported as ULLA_ERROR_SYNTAX_ERROR, as the database performs the UQL parsing and mapping error codes between database error codes and ULLA error codes is necessary and has not been fully completed.

3) The result values provided by the ULLA core are converted to the requested format by the ULLA core. This permits, for instance an integer type to be returned as a string if requested. Therefore, an error is not raised in these cases. This is not part of the ULLA specification, but is felt to be a useful feature.

4) The ULLA result field index 0 always refers to the row (or tuple) ID value. In the specification the requested field index (field numbers) starts at 1, so this is consistent with the ULLA specification, but means that this additional (primary key) field is always returned in

all results and so it is not an invalid field number. However, valid field number checking is not currently performed in the implementation.

5) The specification requires calling of the ullaResultNextTuple before any other result access function and so calling ullaResultNumFieldValues or any other result access function first will result in a ULLA_ERROR_NO_CURRENT_TUPLE error even though the result may be invalid. The test suite expects to still get an ULLA_ERROR_INVALID_RESULT even though the ullaResultNextTuple has not been called first.

6) The ULLA core does not currently check that the minimum periodic notification interval has been violated. Therefore, it accepts the request for 1ms periodic notifications even though this is not possible to achieve.

7) Command validation is assumed to be performed by the link provider. Therefore, all commands are forwarded on to the relevant link providers and it is the responsibility of the link providers to check the validity of the requested command.

8) Error string retrieval is not fully supported yet by the ULLA core as this is felt to be a less important aspect of the API.

9) The ullaPrepareCmd functionality is not fully supported in the current implementation, as this will be handled within the Link Manager implementation and plug-in link managers are supported by the ULLA core. Therefore, it is assumed to be up to the link manager implementers to support the ullaPrepareCmd result codes. In the absence of a link manager the ULLA core currently always returns success.

4.2. Performance Benchmarking Results using Validation Tool

The performance results discussed in this section are complementary to the more detailed performance analyses conducted on each implementation platform and discussed in full in section 5. Whereas the performance details in section 5 were obtained by each project partner using their own methods and performance tools, the results discussed in this section were obtained using the cross-platform validation testing suite as described earlier in this report. Thus it is possible to offer a direct performance comparison between the various implementation platforms shown in Table 4-2 below.

Test cases 2.11 and 13.1 are two benchmarks that were conducted with ULLA implementations on different device and OS platforms. The first involves a repeated query (information request) operation that selects elements from a test link object that is registered by the link provider upon execution. The second is a repeated set attribute and notification trigger operation. This is achieved by the link user registering a notification request on a test link attribute, then repeatedly updating the attribute itself, and waiting for the notification trigger before updating again.

Platform	**Hardware**	**OS**	**ULLA Core**	**Code footprint**	**ULLA storage**	**Code footprint**
VIPER-Remote GDB Linux	PXA255 / CPU 400MHz, 32MB RAM	Linux 2.4.26	User space shared library	189k	Postgres	112k
iPAQ-	PXA255/	Linux	User space	189k	Postgres	112k

Remote GDB- Linux	CPU400MHz, 32MB RAM	2.4.19	shared Library			
Triton – Local GDB Linux	PXA270, CPU 520MHz, 128MB RAM	Linux 2.6.17	User space shared Library	189k	Postgres	2.6M (backend) 112k (frontend)
Triton – Local RTDB Linux	PXA270, CPU 520MHz, 128MB RAM	Linux 2.6.17	User space shared Library	183K	Polyhedra	2.57M (backend) 281k (frontend)
Tecra M3 – Local GDB Linux	Pentium M, CPU 1.8GHz, 1GB RAM	Linux 2.4.x	User space shared Library	249k	Postgres	16M (backend) 134k (frontend)
Tecra M3 – Local RTDB Linux	Pentium M, CPU 1.8GHz, 1GB RAM	Linux 2.4.x	User space shared Library	183k	Polyhedra	2.1M (backend) 370k (frontend)
HTC Wizard Windows Mobile 5.0	OMAP 850 195 MHz	Window CE 5.0	Stream interface driver	72k	Parser based proprietary storage	

Table 4-2 ULLA implementations used for testing purposes

Table 4-2 summarises the hardware and OS platforms and software architectures of each ULLA implementation used for the benchmarking. The implementation options include kernel and user space design and the use of database-based or parser-based data storage at the backend. The variety of listed implemented options shows that the ULLA framework only mandates the software specification of the APIs. In other words, it does not mandate a specific software architecture for the ULLA core because such design details depend on the type of device and the application scenarios and as such are best left to be decided by the ULLA core implementers.

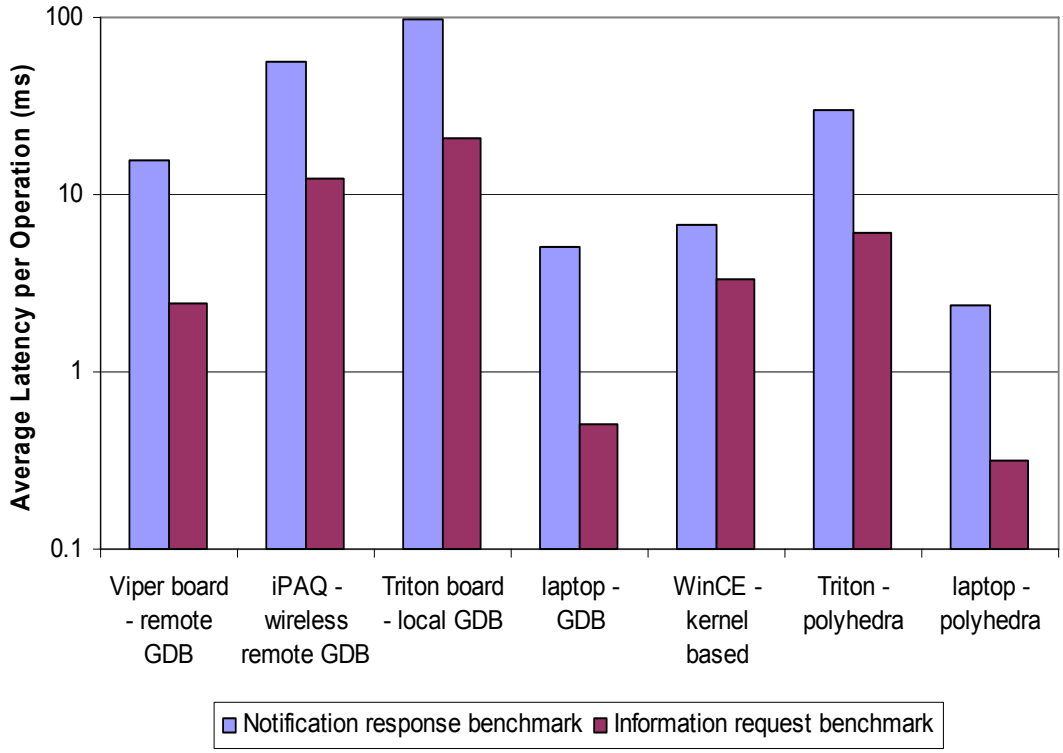

Figure 4-1 Comparison of Information request Latency and Notification Requests on a selected set of hardware platforms

Figure 4-1 compares the latency for information requests (or query latency) of each of the test platforms shown in the Table 4-2. The information request latency is the average time taken for a test link user to receive a reply for a query request it sends towards the ULLA core (using a test length of fifty repeated requests). For embedded devices, the best case latency for information requests was observed with an ULLA implementation utilising a remote data storage connected via Ethernet. This is clearly an unrealistic solution to deploy in a real system and so the embedded kernel resident ULLA (within Windows CE) provides the next best performance. Performance clearly improves when devices have greater processing capability and more memory capacity and so the laptop platforms exhibit at least an order of magnitude better performance than the embedded platforms. It is also observed that when ULLA is using a local database at the backend, a difference in performance is observable between the general purpose database solution (using Postgres) and the real-time database solution (using Polyhedra). The real-time database performs much better in the repeated query test when the database is locally resident on the device and is only outperformed by the kernel resident implementation and when the database is on a remote server PC.

Our results revealed that while the database solutions add value to ULLA implementations by providing additional functionality (e.g. full SQL compliant support, trigger mechanisms, historical storage and statistical capability), they also add significant overhead to simple query latency because they are not specifically designed for the ULLA scenarios, which generally have small table sizes and simple query operations. We observed two to five times better performance for information requests when a kernel resident parser based ULLA is used. Overall, the results show that information request latency is mainly dependent on the processing and memory capability of the device.

Figure 4-1 also compares the average set attribute notification response latency on the different platforms listed in the table 1. The average delay is measured over 2000 successive set attribute and notification response (trigger) cycles. The delay measurement incorporates the *ullaSetAttribute* command latency in addition to the notification response latency. Our test results show that for embedded devices, in similar way to the information request tests, the kernel resident parser based approach outperforms the database solution approaches by at least a factor two. Clearly the kernel resident solution is much more efficient at handling the interaction between link user and test link provider than with indirection though a separate database process required when there is a database solution utilised by the ULLA core. However, it is interesting to see that the real-time database solution is much faster (a factor of three better) at handling the notification request triggers than the general purpose database and so it is an attractive solution for embedded ULLA implementation.

In summary, the results demonstrate that both information request and notification performance on resource constrained embedded devices could fulfill the real-time requirements of handover and dynamic optimisation type of scenarios. Also, the framework provides sufficient flexibility for the implementers to adapt suitable techniques depending on the specification of hardware and software platforms and application requirements. The average ULLA core code footprint is around 200 – 300 Kbytes. The size of the storage could vary from couple of hundreds of Kbytes for a parser based approach up to few Mbytes for a database approach. The benchmarking also showed that by selecting a suitable software architecture, the overhead added by ULLA can be kept to a minimum so that it does not adversely impact the performance of the overall system.

5. Results from Performance Analysis

Performance tests were conducted to ensure that the ULLA's response times meet expectations of a particular LU and do not exceed the specified performance criteria. Performance test cases will be defined with respect to different usage scenarios (or platform type) and during these tests, response times were measured under heavy stress and/or volume. Some of the performance metrics are: query latency, notification latency, Minimal Number of queries per second etc.

5.1. Multimedia Streaming Platform (MSP)

The following sections describes the results from the performance tests on TREL ULLA implementation that targets high-end devices hence implements the ULLA high-end profile. This implementation is used in the multimedia streaming platform, which employs adaptation algorithms as performance optimization technique. The multimedia streaming platform comprises of an Arcom Viper single board computer running Linux Kernel 2.4.26-vrs1-pxa1. A D-Link WLAN adapter, DCF-660W, is also used in the platform with the hostap driver to provide IEEE 802.11g wireless LAN connectivity. The hostap driver has been augmented with a WLAN LLA to facilitate interoperability with the ULLA Core.

5.1.1. Command Latency

The latency distribution for performing commands was evaluated by using a test LU that measures the latency for performing each of a succession of repeated command (doCmd). The repetition frequency was set to 10 per second (an interval of 100ms) and the latency was measured using the real time clock on the Toshiba Tecra M3 laptop running Linux 2.6.11. The command selected to be executed has a dummy implementation in the corresponding LP which simply returns a success code and nothing else. The LU passes five arguments within the command, four integers and one string of length 30 bytes. The latency distribution is measured for each command operation using the real time clock. The results are shown in Figure 5-1 which indicates that the worst case latency is 280 μs and 80% of the time the latency is less than 110 μs.

5.1.2. Request Notification Latency

The tests performed to determine the latency of the asynchronous notification is using a combined LP / LU module. This is possible because the TREL ULLA implementation is within user space and so the same application can register both as an LU and an LP. The time taken between an update on the link information and the notification being received is then accurately measured using the real time clock. Two different tests were conducted; the first uses the notification request:

```
SELECT * from ieee802_11channel;
```

The second uses:

```
SELECT frequency from ieee802_11channel;
```

The ieee802_11channel class has 34 fields per row, the majority being of integer type. The results in compares the latency distributions obtained for notifications occurring at an interval (between each operation) of 50ms for the PXA255 single board computer. The worst

case observed latency is 100ms for the asynchronous notification of an update occurring to the table, but the vast majority of observed latencies are less than 60ms. The latency does not change substantially as the number of notifications per second is changed, but the less capable PXA255 platform exhibits significantly higher latencies compared to the Tecra M3 platform, and is in fact over an order of magnitude greater.

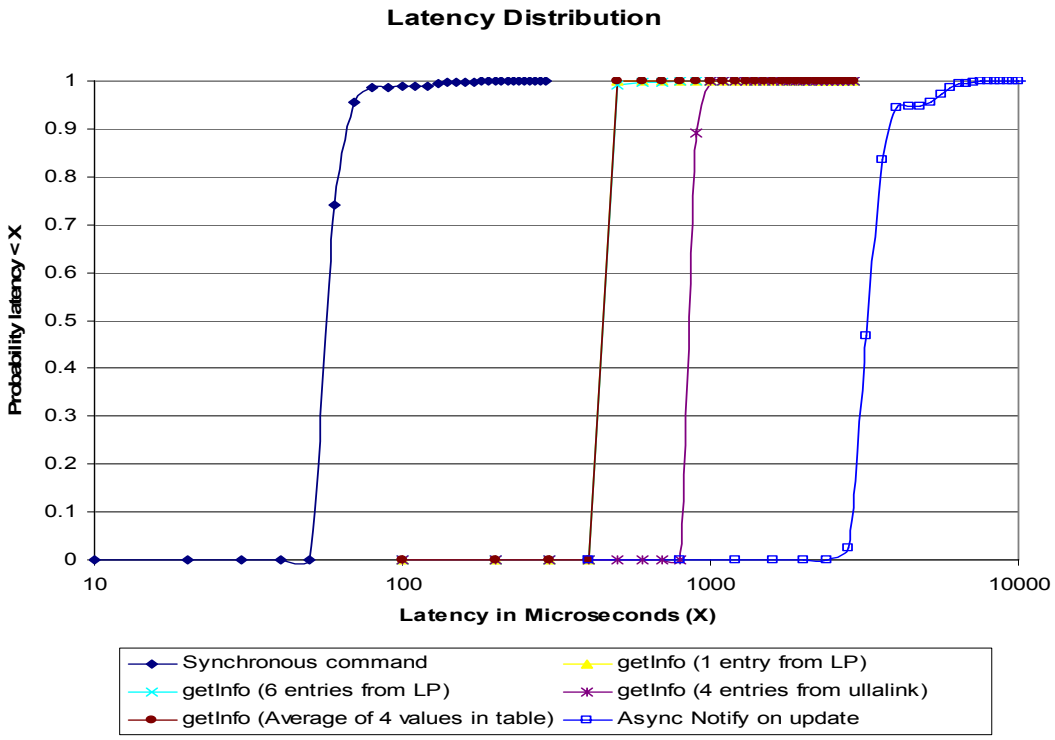

Figure 5-1 Latency Distribution Results for TREL ULLA Core Tests

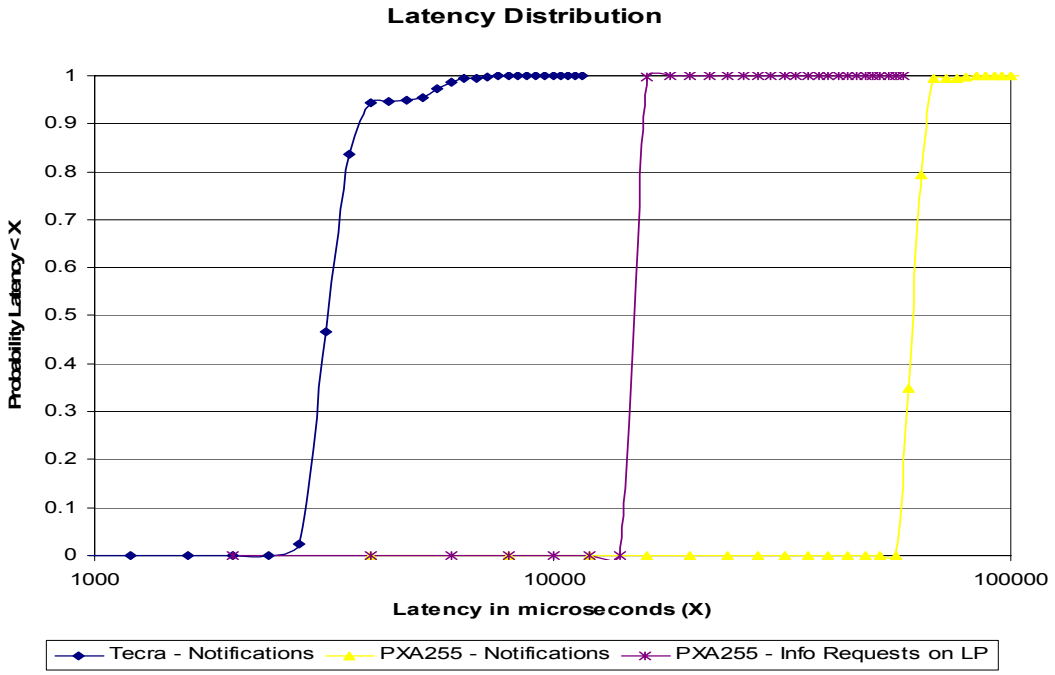

Figure 5-2 Comparison of Latency Distribution on Different Platforms

5.1.3. Request Info Latency

The evaluation of latency distribution for performing queries uses the same technique as for commands. The queries used for this purpose are:

```
SELECT * from ullalink;
SELECT * from linkprovider;
```

There are five entries in the ullalink tables and either one or two entries in the LP table for this test. The results in Figure 5-1 show that the distribution is not dependent on the amount of data returned but rather on the table size. When either one or six entries are returned from the LP table the worst case latency is in the order of 400µs, regardless of whether one or six rows are returned, but with 2 entries from the larger ullalink table the latency is 900 µs. The example results received from the requests are shown in Figure 5-3, in which the linkProvider table has 10 fields per row with the majority being of string type. On the other had, the ullalink table has 30 fields per row and the majority are of integer type.

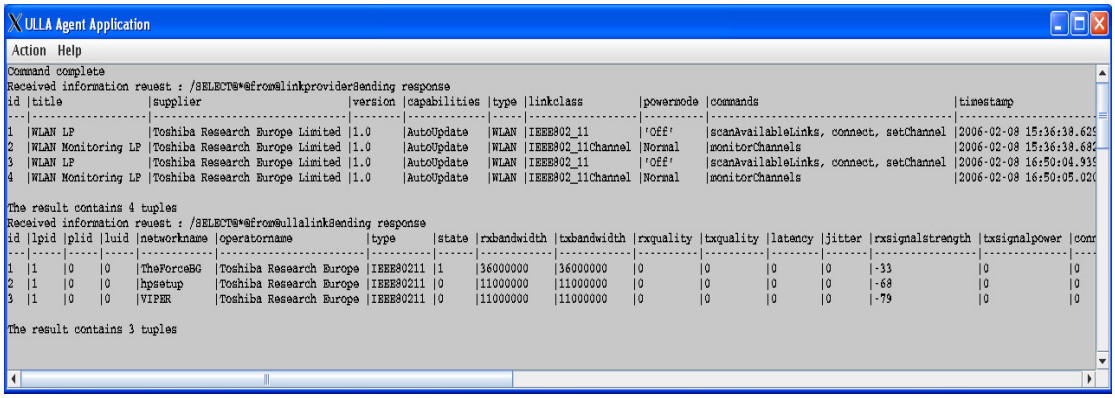

Figure 5-3: Example results from Information Requests

Another query is used to determine latency for performing a simple statistical operation using:

```
SELECT AVG(noiselevel) from IEEE802_11;
```

This type of simple statistic does not incur much overhead as the main overhead is observed as coming from the accessing the database and so performing this averaging operation is negligible in this case.

5.1.4. Memory occupancy

The following table summarises the code footprint of the ULLA implementation. The memory required for dynamic data and stack are not included in this summary as this depends largely on the types of query being performed and the amount of data stored in the database.

Module	Code Footprint (kBytes)
ULLA User Library	46k - LU and LP interface 107k – postgres database lib 52k – System V IPC libs (Postgres backend 2.6Mbytes)
ULLA Core	
IEEE 802.11 LLA	51k lla & 25k wireless extensions lib
TOTAL	281k plus database (2.9MBytes)

Table 5-1 Memory usage summary (code footprint)

5.1.5. Power Consumption

The CPU utilisation was measured for the above tests to indicate power consumption and are summarised in Table 5-2. The results indicate that the Tecra M3 platform is much more efficient at performing the information request operations. This is perhaps because of larger caches and other optimisations performed by the database implementation on this platform. The PXA255 platform is slightly more efficient in terms of power consumed for the notification operations. However, these results do not take into account the memory and bus access power consumption, only the CPU power consumption and also there could be accuracy limitations when computing CPU utilisation at the low levels observed on the Tecra.

Test (50 ms interval)	CPU Utilisation % (estimated energy consumption mJ)			
	Tecra M3 *		PXA255	
	%	mJ per op.	%	mJ per op.
Notifications	0.413	6	3.988	5
Information request	0.006	0.1	2.133	1

Note: * from White paper "Critical Success Factors For Next-Generation Mobile Computing" available at http://cdgenp01.csd.toshiba.com/content/pr/download/Toshiba%20Centrino%20WhitePaper_Mar2005.pdf

Table 5-2 : CPU Utilisation and Power Consumption

5.2. *Connection Management Platform (CMP)*

This section presents performance results obtaining when running the EMIC ULLA implementation on an HTC Himalaya Pocket PC Phone Edition 2003 device. The device uses an Intel XScale CPU (PXA263) clocked at 400MHz, 128 MB of RAM and 32 MB of flash. For communications, the device also comes with an onboard triband GSM/GPRS modem and Bluetooth support. Because the device does not natively support 802.11, we use an external SDIO wireless LAN card from Socket Communications.

We don't present here results for the Windows XP implementation since Windows CE represents the critical case as the devices used are less powerful.

5.2.1. Command Latency

The evaluation of command latency heavily depends on the type of the command. For example, a connect command asking a LP to connect to a discovered link might take several seconds while changing a LP configuration might be almost instantaneous. To deal with this we actually measure here the time to execute a dummy command that simply returns without doing any specific work. Such a value should provide a best case scenario for a command execution. In order to deal with the clock resolution of the devices and average the returned value, we measure the time to execute 100000 times the dummy command. The measurements done report a time of 86 ms to execute the 100000 commands which translates to an average command latency of 0.86 us.

5.2.2. Request Notification Latency

The ULLA core evaluates notifications registered by LUs each time new measurement values are available. In order to examine the latency added by this evaluation we measured the latency between the point in time when ULLA knows about the change of an attribute and the point in time when the application is notified by the ULLA. The latter one is the same as the time when the notification handler in the application is called.

Three queries of increasing complexity were used during these tests:

```
SELECT id, networkName FROM ULLALink WHERE id==1;

SELECT id, networkName FROM ULLALink WHERE ((id==1) AND
(rxBitRate>10000));

SELECT id, networkName FROM ULLALink WHERE ((id==1 AND
rxBitRate>10000) OR (id==1 AND rxSignalStrength>10));
```

Figure 5-4 shows the measurement results for the three queries of increasing complexity and 500 trials each taken on the Windows CE implementation using the ULLA as a driver. We can see that approximately 90% of the queries are finished in less than 1.5 ms and only less than 2% take more than 6 ms. As the smallest unit used by the Windows CE timing service is a millisecond, more detailed results could not be obtained. The difference caused by the complexity of the query is nearly negligible showing that the dominating part of the latency is caused by the mechanisms used to deliver the notification to the application.

However, nearly all latencies are below 1 ms as measured with the Windows CE prototype. The major part of the latency on the Windows CE platform is caused by overhead such as context switches between kernel and user space. Since all measurement results are near to the measurement accuracy we can conclude that the latency added by the evaluation of the notification query and thus the overhead added by ULLA is acceptable even for real-time systems. Additionally, it has to be considered that the flexibility offered by the described notifications is much higher than offered by existing solutions without adding considerable latency.

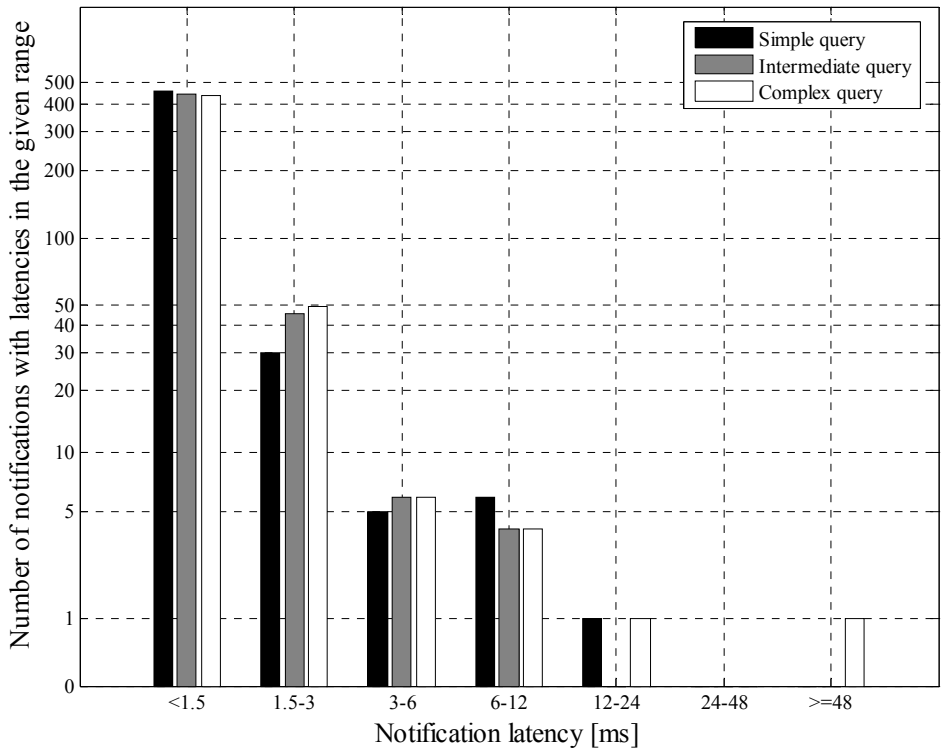

Figure 5-4: Distribution of notification latencies measured with queries of different complexity

5.2.3. Request Notification Rate

In the request for notifications API, applications are able to specify the rate of notification arrival (this is called the notification period). Even if the ULLA has not been specified to respect any kind of real-time requirement, another important aspect of the ULLA that has to be verified is whether the time between 2 consecutive notifications is as expected by the application that requested the notification. Figure 5-5 shows the measured time between notifications against the requested inter notification time as requested by the LU application. To measure this time, we use a notification that always fires and we save the time at which the notification handler is called at the beginning of the handler. The set of time differences between consecutives handlers is used to average inter notification time. We used the ULLA version implemented as a driver because it is the one we recommend for general use. Several different requested minimum times were used and each test was repeated 100 times. For shorter periods such as 1 and 3~ms the measured time is a bit higher than the requested one because the overhead created by the delivery mechanism of the notification is almost as big as the notification period. Overall it looks like the delivery rate of notifications respects pretty much the period required by the LU application.

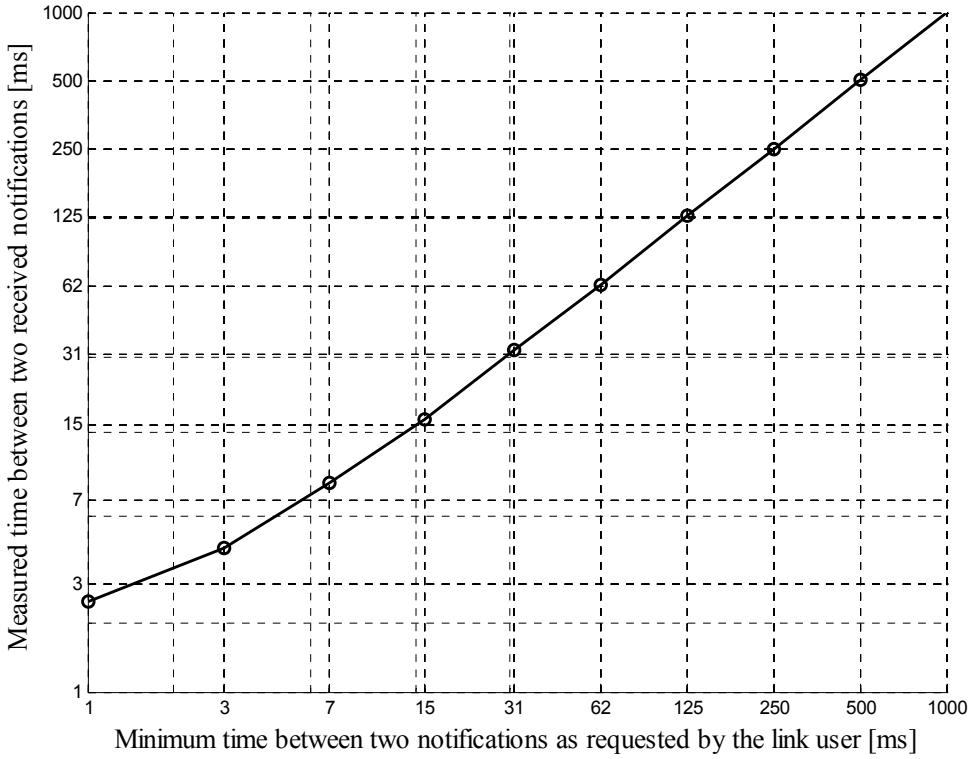

Figure 5-5: Measured and requested inter notification time

5.2.4. Request Info Latency

This section presents a set of measurements in order to estimate the time necessary to perform a standard information request (requestInfo() call). To measure this value, we run the same request 2000 times and averaged the time to get the average time for a single requestInfo call. This experiment is repeated 10 times in order to compute the standard deviation of the calculated average measured durations.

Several factors influence the duration of an information request. To take into account the influence of the number of requested attributes on the average request time, we vary the number of attributes from two to eight. To consider the effect of the number of available links, we also made the same measures in the presence of either 2 or five links. Finally, the tests are also realized with different attribute validity in the storage. The attribute validity defines the expected correctness of an attribute by requiring the ULLA core to fetch the value from the LLA if the attribute values have expired. Obviously, shorter attribute validities will imply more requests to the underlying LLAs, hence an increase in the measured average request time.

The queries used during this experiment are the following:

```
SELECT linkId,operatorName FROM ULLALink;

SELECT linkId, networkName, operatorName, type FROM ULLALink;"

SELECT linkId, networkName, operatorName, type,
rxSignalStrength, localL2address FROM ULLALink;
```

```
SELECT linkId, networkName, operatorName, type,
rxSignalStrength, localL2address, rxBitRate, remoteL2address
FROM ULLALink;
```

Figure 5-6 and Figure 5-7 present the result obtained for two different implementations of the ULLA on Windows CE. As previously mentioned the ULLA core can be compiled as a driver or directly linked with Link User applications. The driver version corresponds to a version where the ULLA core is implemented as a stream driver. In this case, the ULLA user library uses IOControl functions to communicate with the ULLA core. The second version of the ULLA core corresponds to a version where the ULLA core is directly linked with the ULLA user library. In this later case the ULLA core will run in the same address space as the application, while in the driver version of the ULLA, the ULLA core actually runs in the device manager process. The non driver version is mostly used for debugging purposes. Because access to the ULLA is performed through direct function calls (no system calls), this version represents a best case scenario from a performance point of view. In order to offer ULLA as an operating system service to all running applications ULLA should be implemented as a driver.

Figure 5-6 shows the results obtained when the ULLA core is linked to the test application while Figure 5-7 presents the same results when the ULLA core is implemented as an independent driver. The overhead created by the use of a driver increases slightly with the number of attributes involved in the queries. This is an expected behavior since more attributes will result in more data being copied from the driver to the ULLA core.

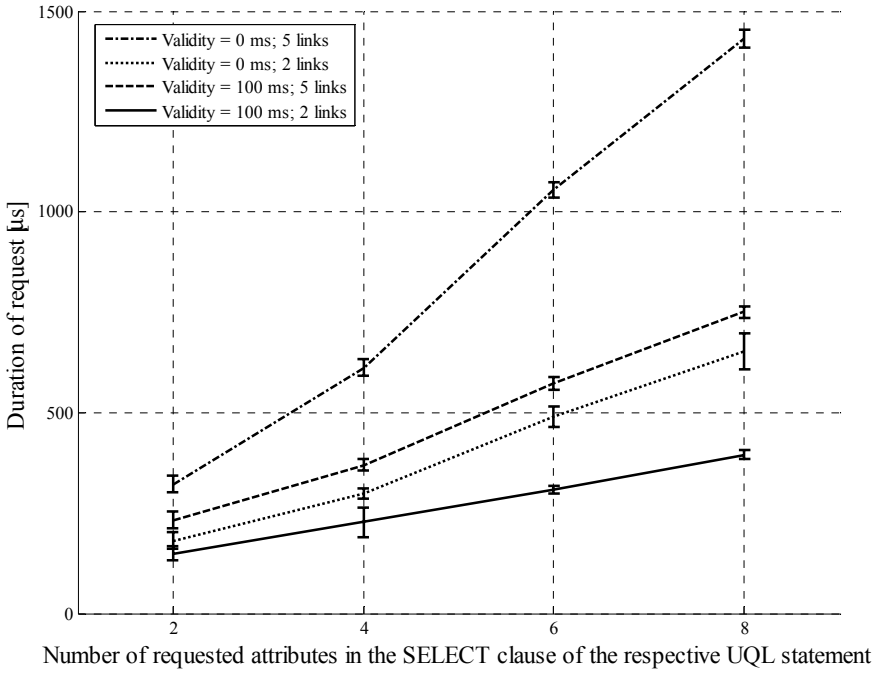

Figure 5-6: Duration of different queries using the requestInfo()-call measured with two or five WLAN-links present and the ULLA linked directly to the application

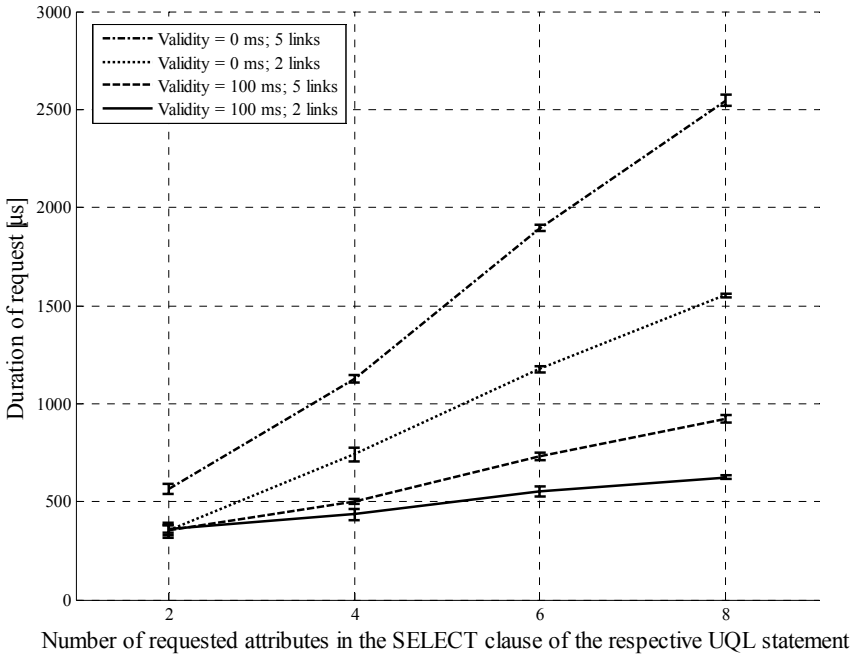

Figure 5-7: Duration of different queries using the requestInfo()-call measured with two or five WLAN-links present and the ULLA implemented as a driver

The performance is about half compared to the ULLA which was directly linked to the test application but still clearly in the acceptable range. Other than that, the query duration more or less linearly increases with the number of requested attributes and also the presence of more links leads to higher duration results. Both are expected effects because more results have to be copied to user-space and made available to the LU. With 5 links and 8 attributes (representing a total of 40 attributes to return) a query takes approximately 2.5 ms. This is not bad considering the speed of the device.

As expected, the effect of shorter attribute validity is an increase of the request duration since attributes have to be retrieved from the LLA more often. It is interesting to note that even with a 0 attribute validity (equivalent to having no storage), the results are still acceptable.

5.2.5. Memory occupancy

A major goal of the implementation on Windows CE was to prove that the concept of a ULLA is sustainable on mobile devices with limited memory (including phones). Table 5-3 lists the memory footprint for all components implemented in the Windows CE and the Windows XP implementations of the ULLA.

ULLA component	Memory footprint Windows CE	Memory Footprint Windows XP
User library	19.5 kB	328 kB
ULLA core	72 kB (driver)	76 kB (COM service)
802.11 LLA	43.5 kB	72 kB

GPRS LLA	30 kB	N/A
Bluetooth LLA	34 kB	N/A
Ethernet	N/A	26 kB
Total	199 kB	503kB

Table 5-3: Microsoft Windows implementation memory footprint

The presented results just show the current memory footprint without considering any specific optimization that could even reduce the memory usage. This type of memory footprint is certainly compatible with almost any current phone or smartphone that exists on the market. The dynamic memory occupation (allocated memory) is not presented here because it is highly dependent on the chosen implementation. In particular the implementation of the storage will dictate the dynamic memory occupancy. Smart implementations where only requested attributes are stored would provide minimal occupancy.

Although memory space is probably less of an issue for a Windows XP based device, the memory footprint observed is similar to Windows CE.

5.2.6. Power Consumption

One of the most important requirements of the ULLA is to make sure that the ULLA does not become a major source of battery consumption. This is especially true for mobile devices such as phones or PDAs.

During this experiment, we measure the battery level while running in 3 different configurations. In the first configuration, we measure the battery level while having an 802.11 wireless card on and the ULLA evaluating notifications every 10 ms. This level of activity is already beyond most expected application scenarios for ULLA. In the second configuration, the wireless card is switched on but the ULLA is not activated. Finally, in the third configuration we simply measure the battery level in the absence of wireless card and ULLA. This last configuration provides a base for the minimal battery consumption. For all experiments, the battery was fully recharged between each experiment, and the readings were recorded every three minutes during half an hour period.

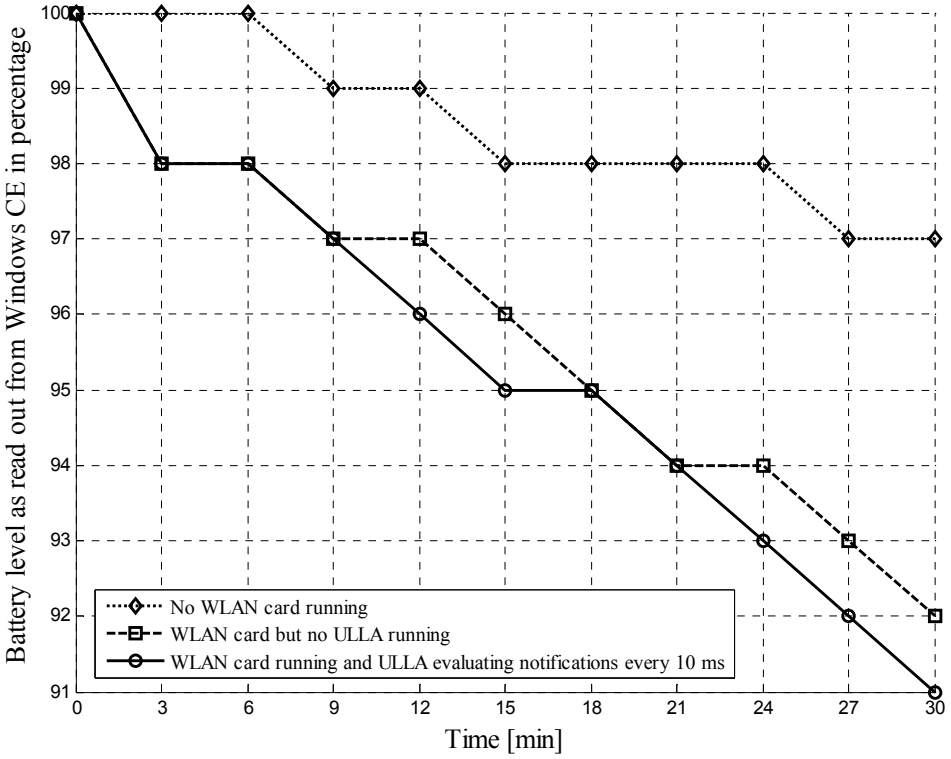

Figure 5-8: Battery usage according to time for different configurations

The results clearly show that the difference between having the ULLA on or off is negligible. The battery level is almost the same in both cases. As can be seen from Figure 5-8, the use of ULLA even with very frequent notifications employed in the test, has negligible impact on the device general battery level. Clearly the wireless interface itself dominates the power consumed, and the additional processing and IO activity resulting from the continuous stream of notifications cause no significant increase in battery depletion rate. This result indicates that adding ULLA does not considerably change the power consumption making the deployment of ULLA also in mobile devices feasible.

5.3. Real-time Multimedia Platform (RMP)

This section illustrates some performance figures of STM ULLA implementation. The tests have been conducted on a Nomadik Development Kit board NDK-10 hosting the Nomadik STn8810 system on chip. The main processor is an ARM926-EJ running at 200MHz. System RAM is 64MB. The main board is plugged in to an 802.11g "Phaser" adapter powered by the STLC4370 device, a low power 802.11g/b chipset targeted for mobile applications such as cellular phones. The wireless LAN chip is connected to the Nomadik through a hi-speed synchronous serial port clocked at 48MHz.

The software platform is based on ARM Linux kernel 2.4.21 with DirectFB based user interface.

5.3.1. Command Latency

The latency introduced by the ULLA framework to deliver commands to the wireless LAN adapter is presented. A dedicated ULLA Link User issues a *scanAvailableLinks* command using the synchronous *doCmd()* call directed to the Phaser link provider and the time elapsed between the function call and its return is measured. The function is called 200 times and the average time is taken for 10 different runs.

Figure 5-9 represents graphically the obtained values.

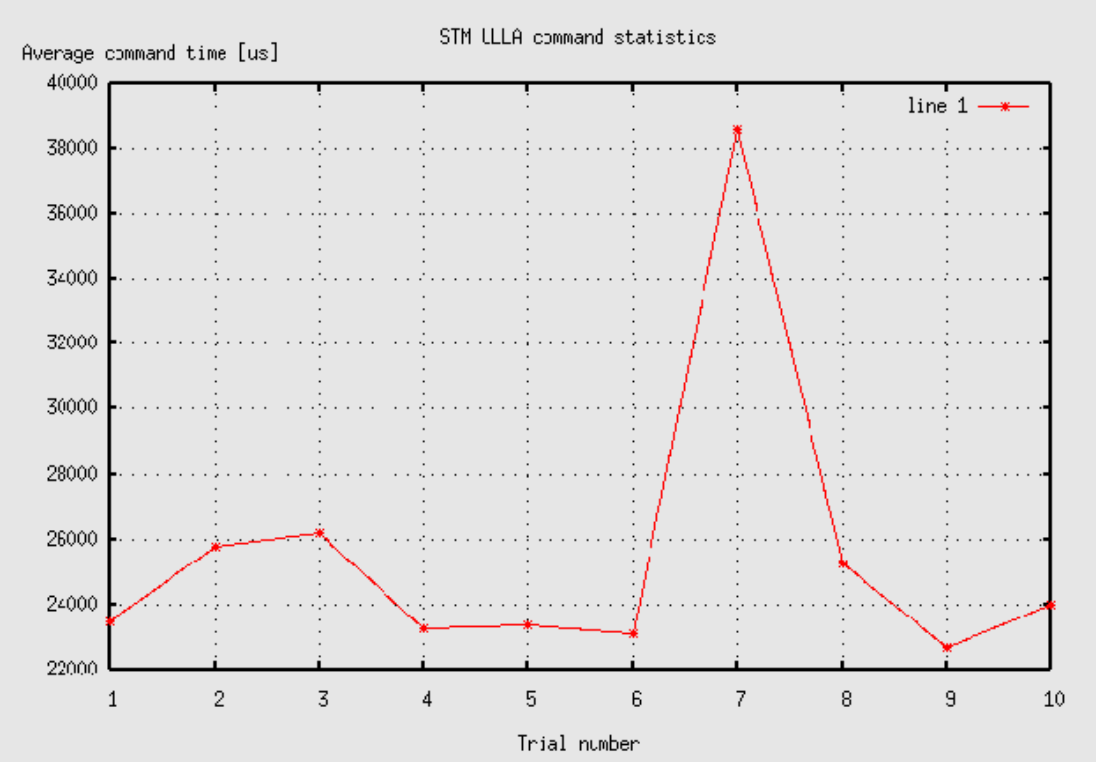

Figure 5-9: Command latency for *scanAvailableLink* command

We notice that command execution is in the order of about 25 milliseconds, which is a rather high figure if compared to a direct IO control call to the adapter's driver which takes about 250 microseconds average.

This is mainly due to the interprocess communication overhead introduced by Sys-V message queues needed to convey information between the Link User and the ULLA core, which live in two different address spaces. More efficient IPC mechanisms such as shared memory may mitigate this performance issue and some optimizations could boost the performances. However, it should be noticed that two or more context switches are still needed to exchange information between the Link User and ULLA Core processes and hence some penalty is introduced.

5.3.2. Request Info Latency

This test session aims at characterising the performance of the ullaRequestInfo() entry point, used to query the ULLA database using the UQL syntax.

The following queries have been run 3000 times each and the average and standard deviation of the time for the function to complete have been evaluated:

```
1. "select id from ullaLink";
2. "select id, lpId from ullaLink";
3. "select id, lpId, type from ullaLink";
4. "select id, lpId, type, linkSignature from ullaLink";
5. "select id, lpId, type, linkSignature, remoteL2Address from
   ullaLink";
6. "select id, lpId, type, linkSignature, remoteL2Address,
   rxSignalStrength from ullaLink";
7. "select id, lpId, type, linkSignature, remoteL2Address,
   rxSignalStrength, networkName from ullaLink";
8. "select id, lpId, type, linkSignature, remoteL2Address,
   rxSignalStrength, networkName, quality from ullaLink";
```

The mean query time is plotted in Figure 5-10 for the eight different cases.

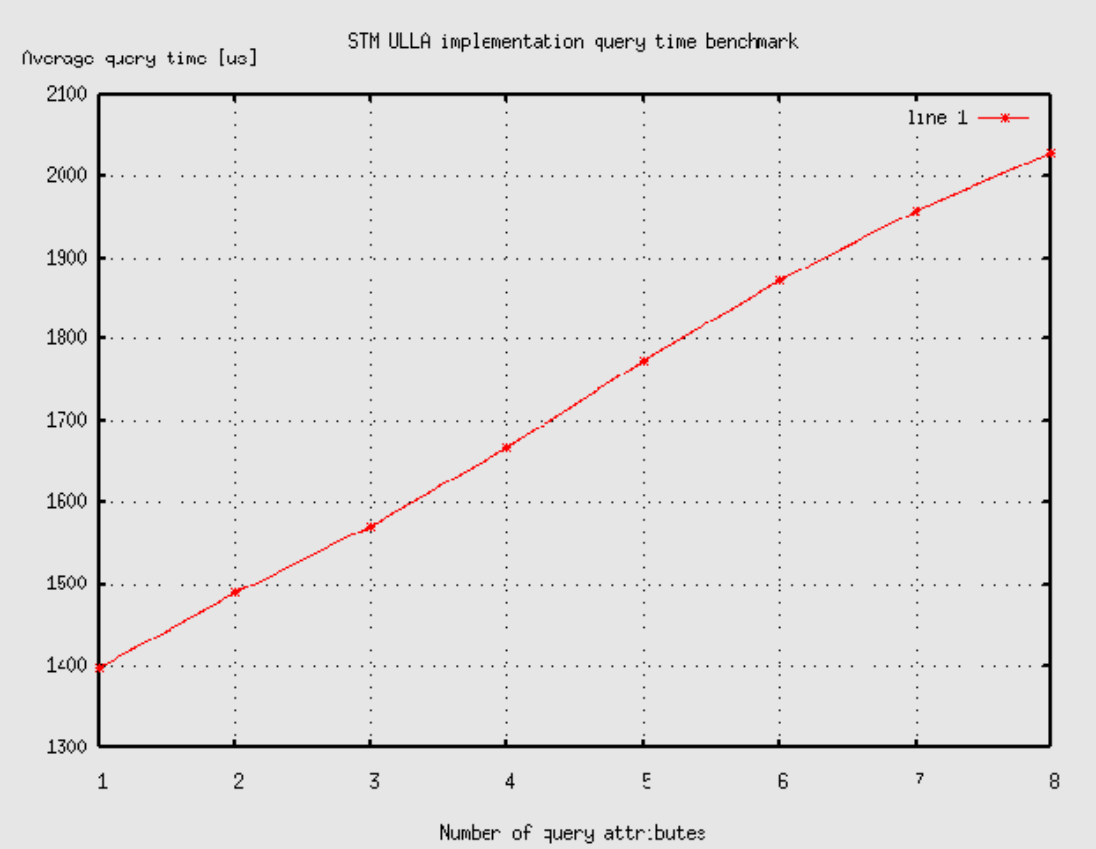

Figure 5-10: Query duration

As expected, it can be noticed that the average mean time increases linearly with the number of attributes that are requested. The standard deviation of the average query time, instead, does not show any relation with the number of attributes and its range is very limited (from 7 to 16 microseconds).

The results show that the performances of the ULLA Query Processing module are outstanding with respect to the ULLA Command Processing ones. The reason is that in former case the Link User does not need to communicate with the ULLA core to retrieve the results but interacts directly with the underlying database via the ULLA library remaining in

the same address space. This approach eliminates both IPC overhead and delays introduced by context switches leading to much better performances.

5.3.3. Memory occupancy

Memory occupancy figures have been collected in order to evaluate the impact of a complete ULLA installation both in terms of disk usage and memory usage.

STM ULLA installation consists of several software components:

- libulla.so, libullacommon.so: these shared libraries are used by all applications (Link Users) willing to use ULLA functionalities.
- libullasqlitedb.so: ULLA Query Processing functionalities map on top of the SQLite database system. Such functionalities are collected in this shared library.
- liblpphaser.so: a Link Layer Adapter has been developed for the 802.11b/g STM Phaser chipset. It is implemented in this shared object which is loaded at run time during the ULLA daemon initialisation phase.
- liblpcommon.so: common LLA functionalities are grouped together in this to avoid code duplication across different LLAs.
- ullad: ULLA core functionalities are implemented in a system wide daemon which accepts and serves Link User requests. The ULLA Core daemon is launched at system startup as every other system service.
- ludemo: a simple graphical Link User demo showing ULLA functionalities has been developed.

Table 5-4 illustrates the footprint of the separate components that make up the STM ULLA implementation, including a graphical Link User which gathers statistics from the Phaser Link Provider. The disk storage footprint represents the amount of disk space required to store the components itself, whereas the RAM footprint is the sum of the text, data and BSS regions once loaded in the process address space.

ULLA component	Disk storage footprint	RAM footprint
libulla.so	26 KB	60 KB
libullacommon.so	19 KB	52 KB
Libullasqlitedb	404 KB	436 KB
Liblpphaser.so	17 KB	48 KB
liblpcommon.so	23 KB	56 KB
Ullad	42 KB	76 KB
Reviewdemo	12 KB	44 KB
Total	543 KB	772 KB

Table 5-4 STM implementation memory footprint

5.3.4. Power Consumption

Power consumption figures could not have been collected on the target platform due to the fact that development boards are not battery powered and no test points are available to get this kind of information.

However, it is reasonable to argue that the ULLA impact on power consumption is negligible if no polling-like schemes are used to retrieve information from the lower drivers.

5.4. Wireless Sensor Networks Platform (WSNP)

As described in section 3.4 the standard performance evaluation test suite could not be run on the sensors because programming language and operating system are too different to systems running the ULLA full profile.

Instead proprietary test tools were developed that evaluate a subset of the measured items as described below.

5.4.1. RWTH WSN prototype

At first the footprint of the RWTH WSN ULLA was evaluated. It has very small memory demands because 1.6 KB of RAM and 4.6 KB of program memory are enough for all three major modules, namely the ULLA core, the ULLA storage and the LLA for CC2420 radio chip based motes. Taking the example of a Telos B mote this corresponds to about 9.5 % of the available ROM and about 16 % of the available RAM.

In order to evaluate the duration used by a single query requested by a Local LU, we performed a set of measurements using a standard ullaRequestInfo() call. The duration of a single call and its standard deviation were calculated by averaging the time over 10 tries where each try is done by submitting a request 2000 times. Comparing these results to latency values measured for, e.g., the B-MAC protocol shows that the latency induced by ULLA is acceptable even for algorithms taking decisions on a per packet level. Figure 5-11 shows the results for the duration of different requests using five present 802.15.4 links and two up to eight requested attributes with the validity period of 200 and 500 ms. The validity period defines the period of time during which an attribute saved in the ULLA storage is still valid. If it is invalid, it has to be retrieved directly from the LLA.

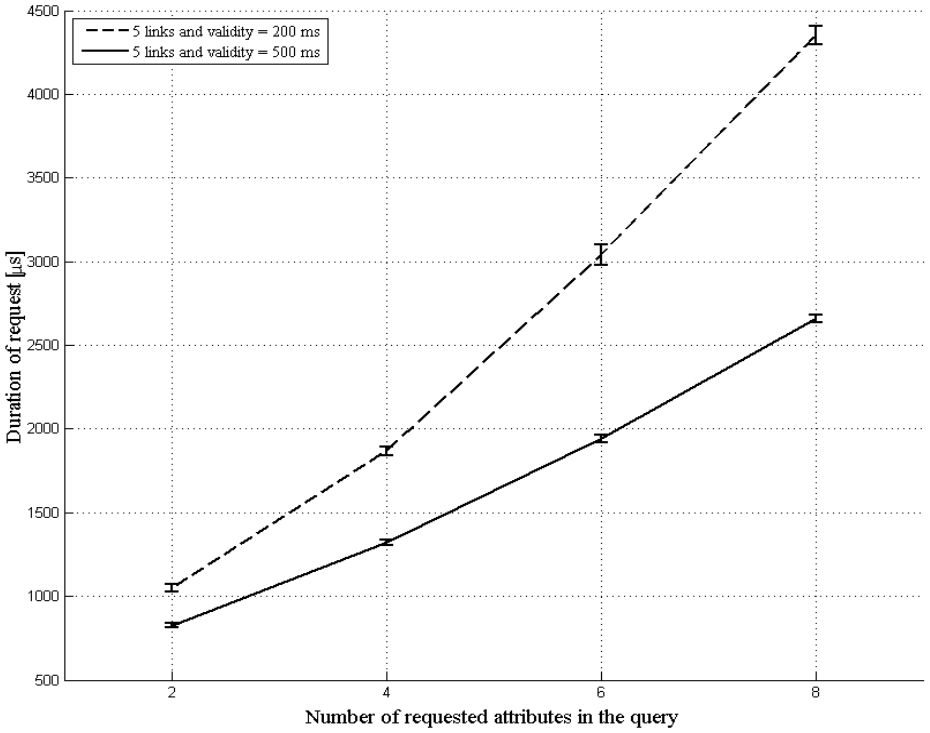

Figure 5-11: Query duration measured with the RWTH WSN prototype and five links present

The impact of the validity is clearly visible from the resulting graph, Figure 5-11. The LLA used performs probing each time new attribute values are requested. Thus, if the validity requirement is set to 200 ms, the 2000 queries in each measurement run lead to at least 10 probing actions. In case validity is 500 ms less probing actions are required leading to a lower average value for the query duration. Additionally, the number of queried attributes is clearly visible. The low standard deviations show that the behaviour is quite stable and all 10 measurement runs gave similar results.

The measurements show that the query duration for a Local LU and rather simple queries with two attributes are in the order of 1 ms, which is definitely fast enough for any link-aware application running on wireless sensor devices. Additionally, a carefully chosen validity value can further improve the resulting performance.

5.4.2. UC WSN prototype

Before presenting some of the measurements carried out by this prototype, it must be indicated the footprint of the ULLA implementation. In terms of RAM the size is 1.92 KB whereas it is 9.85 KB for ROM memory. This size comprises the Link User (Routing Agent), ULLA Core, ULLA Storage and Link Provider. Rests of modules, provided by the tinyos package and used by ULLA components for packet transmission and reception, driver access or multihop routing are not included in this size.

Query Latency on the Remote LU

In order to assess the duration of a single query, requested by a remote LU, LU running in the base station node (mote attached to the programming board and communicated with the PC through the serial port), and its dependency on the particular sensor network, UC has performed a thorough set of measurements using ullaResquestInfo() call.

To carry out the measurements, a local LU together with a routing agent has been installed within the different mote devices. This routing agent implements the *mintroute* routing protocol, which allows different nodes deployed in the network to communicate with the base station, where the Remote LU is running. *Mintroute* is an "any to base" routing protocol, and thus it provides routes from the wireless nodes to the base, but not in the other way. As base station has to send query packets to the corresponding node, it has been added a new functionality, so that the Remote LU can send queries, notifications or commands to the different nodes of the network.

The remote LU is in charge of parsing the requested information and sending it to the corresponding mote device. Sometimes, due to the length, a single query must be codified in several packets. In this case, once the destination sensor device receives the corresponding packets, it reassembles the query and then, the information is requested in a standard manner.

The first scenario where measurements are carried out consists of a remoteLU, which makes a single ullaRequestInfo() call, in which only one attribute is requested to a specific node. The same query is sent 200 times to each node, and we account for the overall duration of the experiment (measuring from the moment when the query packet is sent by the mote attached to the gateway until the response is received) and we average this value. This measurement is performed for nodes 1, 2, 3 and 4 hops away.

Figure 5-12 analyzes the impact of accessing the mote driver for retrieving the requested information, comparing it against the situation in which the information is directly taken from the ULLA Storage. In this case, we are taking fixed validity values ensuring either the data is never valid (the node always has to access to the LLA) or the data is always valid (the info is taken from the ULLA Storage). The finality of these measurements is trying to show the impact of accessing directly to the values provided by the ULLA (stored in the ULLA Storage), instead of having to access to the mote driver to get them. For this test, it is being measured the battery value which means that, in order to get the value from the LLA, it is not needed to send a probe message (e.g. *rxQuality*), but it is directly obtained from the driver. As it can be seen in the figure, there is just some relevance for one hop distance, where the access to the LLA takes around 0.2 ms more than getting it from the ULLA Storage. This is due to the fact that the mote uses a task for retrieving the information from the ADC and when it is completed, a signal is sent and handled by an event within the LLA. Logically, as previously mentioned, if it was necessary the transmission of probe packets to measure the corresponding attribute value, the time overhead due to the LLA access would be quite higher. After that, a call is launched to the ULLA Core (ULLA Query Processing module) and then to the local LU. Once this process has ended, the reply packet containing the requested information is sent to the Remote LU, which forwards the received information to the serial port in order to be displayed and analyzed. Once this packet is received in the remote LU the query duration will be calculated. On the other hand, taking into account the variability of the measurements, the overhead time due to the access to the driver can be considered as negligible with multiple hops.

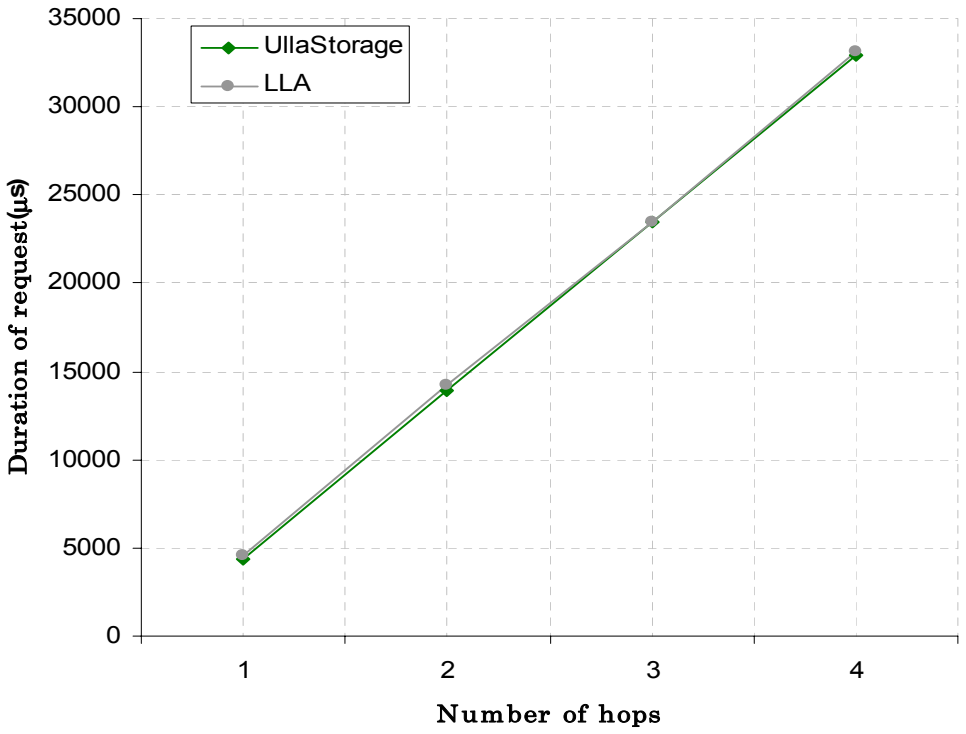

Figure 5-12: 1 Attribute, Micaz motes (ULLA Storage vs LLA access)

Second scenario of measurements is similar to first one but, in this case, it is included a condition that should be fulfilled in order to retrieve the corresponding value. In order to compare the results with those obtained for the first scenario, the condition is always fulfilled and the reply packet is sent. It is important to highlight that the condition is implemented in an extra packet with the corresponding time overhead, as instead of one packet two must be sent. The query is sent 200 times to the same node and the average of the query duration is calculated as previously indicated for the first scenario.

As mentioned before, in this case the RemoteLU parses the query into two different packets. The first one includes the requested attribute and the second one the appropriate condition. This measurement is performed for nodes which are 1, 2, 3 and 4 hop away from the base.

Figure 5-13 analyzes the impact of including a condition in the query sentence or not, that is, to send one packet or two. When sending a single query in one hop distance, the query duration is approximately halved, when compared with the case of including the condition (2 packets). For multiple hops cases, it can be inferred that the query duration follows a lineal tendency, but the fact of sending two packets is not translated into doubling the time, due to the overload added by the medium access and routing protocols.

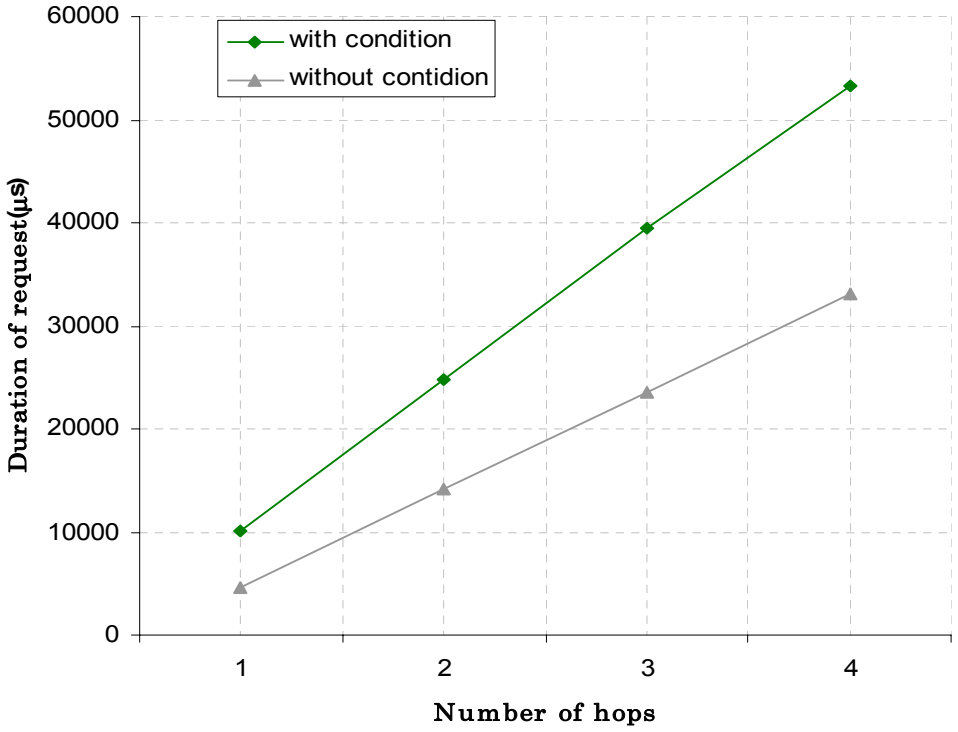

Figure 5-13: 1 Attribute, 1 condition, Micaz motes (ULLA Storage vs LLA access)

Packet error rate

As WSNs are usually dense networks - sometimes organized in a grid structure - the number of potential neighbours which might become sources of interference when sending data to the base station can be quite large. This way, a query sent to the network in order to request some attribute may derive in a traffic peak, since all the sensor nodes would be retrieving their values at the same time (if they are placed at a similar distance). The ULLA receives the request and processes it simultaneously within all nodes.

The following figure shows the packet error rate which was observed during the transmission of a query response packet, against the the number of neighbours. The test comprises the transmission of a query message from the base station, requesting300 samples of the same attribute each time, repeating this process several times in order to get an averaged value.

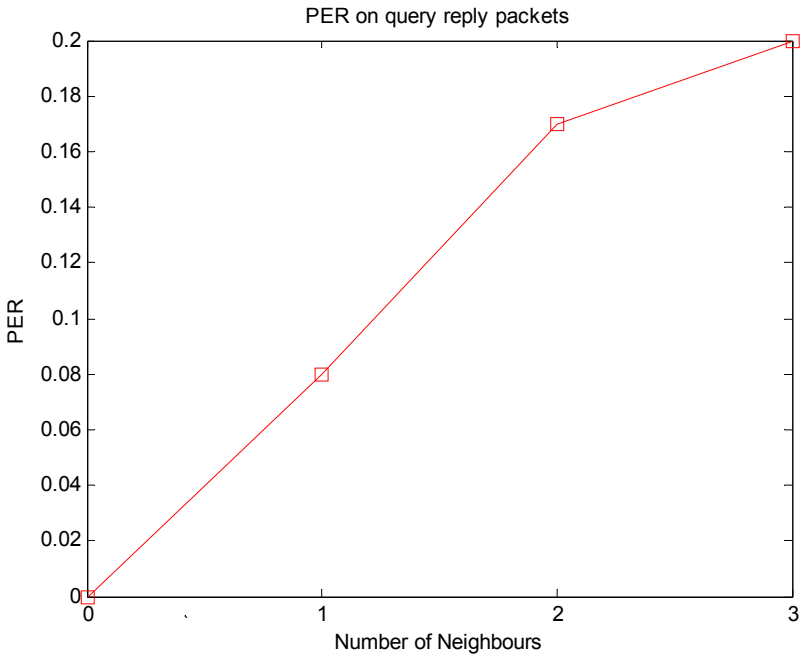

Figure 5-14 Packet error rate in the query reply packets sent from the remote LU

As can be derived from Figure 5-14, the PER increases with the number of neighbours. Obviously, if the node is isolated, all packets are correctly received (the number of reply packets and the number of samples requested is the same); however, when the number of neighbours increases, some packets are lost while other ones are received duplicated. It must be taken into account that in addition to these reply packets, the nodes are continuously sending routing messages, thus contributing to the traffic load increase within the network.

These measurements lead to the use of ULLA in order to retrieve information about the PER associated to one fixed link, being able to store this information in the ULLA Storage. This translates into the extension of ULLA functionality, not only behaving as a transparent interface for the user to access a fixed attribute value, but also managing the transmission of reply packets, in order to avoid network flooding with an excessive traffic overload. Furthermore, in the base station side, the ULLA handles the received packets in order not to send duplicated packets to the smart home monitoring application runing on the server side.

As has been already discussed, it is worth highlighting that this type of problem will be more relevant on WSN due to both their large node density and low transmission data rate.

6. Conclusions

During the validation testing performed so far, two out of the four different ULLA implementations independently developed by TREL, EMIC, STM and RWTH using the same API specification were tested using the conformance test suite independently developed by another project partner, Materna. Initial results of these tests are satisfactory and indicate that the implementations are highly conformant to the API specification. An important point to note here is that the implementations were each tested with the same conformance test code, proving that there is a high degree of interoperability and portability between the implementations. Further, the platform independence of ULLA has been demonstrated by successful implementation of four prototypes targeting systems differing both in hardware and software. ULLA has been implemented in Linux, both in the user and the kernel space, Windows CE, and a reduced version in TinyOS for WSNs. The ability to implement the same API functionality on hardware platforms such as notebooks, PDAs, smartphones, embedded platforms, and devices with very limited resources such as wireless sensors with different resource capabilities demonstrates scalability of ULLA in terms of size of the implementation.

The initial performance tests carried out on each platform have shown that ULLA can be integrated in current wireless systems while adding only a small amount of additional overhead in terms of memory footprint, CPU usage or battery consumption, which demonstrates that ULLA is useful and feasible even for mobile devices with CPU and memory constraints. The measurements also showed that ULLA core developers can tailor their implementation to the specific needs of the desired platform without losing conformance with the ULLA specifications. Various implementation options were evaluated by the consortium and led to the main trade-off between flexibility and simplicity of integration and implementation on the one hand and efficiency and performance on the other hand. The former advantages can be realized when choosing a database backend based solution implemented in user-space. The latter ones can be achieved by a kernel resident solution using a specifically designed ULLA storage. Additionally, the development effort for the same final set of supported optional API-features is expected to be higher for the hand-tailored and optimized solution.

The testing so far has proven that the specification is in a suitably mature stage for a public release. Any feedback and experience from development and usage outside the GOLLUM consortium would provide the essential scenario based testing that is required for the evolution of ULLA further.

The full set of conformance test specifications can be found in Appendix B.

References

[1] Gollum D2.4, “Final Architecture and API”

[2] Gollum D3.1, “ULLA API Definition”

[3] Gollum D3.2 “Validation Plan”

[4] Gollum D3.4, “API Guide Book”

Appendix A Test Platforms

A.1 Conformance Test Validation Environment

Operating Systems

The test applications were developed for Windows CE and Linux. They used the ULLA core implementations of the partners therefore using the same OS as the ULLA cores are developed for.

Networks

The test applications used the LPs that were developed by the partners.

Hardware

The development hardware for Windows CE was based on pocketPC PDA of type MDA III or MDA IV running Windows CE version 4.2 or higher. The Linux platform was based on one of the platforms supplied by one of the partners.

Software

The applications were developed in a rapid prototyping way. This enabled the partners to quickly use the applications to test their implementation. This lead to multiple versions during the project lifecycle. During the development process different test cases were developed in order to test as many requirements as possible. A complete conformance test suite to test ULLA core and LP implementations is now available. Although the applications run on two different platforms, the conformance environment reuses as much code as is possible. GUI modules are excluded.

Tools

For Windows CE all development and debugging was performed on a PC running Windows XP and using Visual Studio 2005. Because the EMIC implementation was used for testing, the same development tools were used.

A.2 Multimedia Streaming Validation Environment

Operating Systems

ULLA was developed and tested on Linux and WinCE operating systems. ARM Embedded Linux 2.4.26-vrs1-pxa for the Arcom boards (see below), 2.6.17-karo-mmx for the Triton boards (see below) and WinCE version 4.2 were used for testing as well as demonstration.

Networks

Wireless LAN technology was tested in the Linux environment. Removable DLink DCF-660W WirelessLAN Compact Flash card connected to the embedded platform were used to test the 802.11 link layer adaptors.

Hardware

The development and testing was carried out on a range of platforms including an ARCOM Viper Single Board Computer with Intel XScale PXA255 Processor with serial, USB and 10/100

Mbps Ethernet ports and removable DLink DCF-660W WirelessLAN Compact Flash card and a Triton board with an Intel PXA270 processor.

Software

- ULLA core implementation used SQL language for queries and included the PostgreSQL 8.0.2 server (with some enhancements) at the ULLA backend. We also utilised the Polyhedra™ embedded database server for performance comparison purposes. However, the implementation makes the choice of database technology transparent to both LU and LPs.
- Link Layer Adaptors were developed on top of hostap-driver 0.3.7 used by WLAN technology and also for the WinCE OS driver environment.
- A multimedia LU was developed using VideoLAN vlc-0.8.2 software to demonstrate the real-time video adaptations. This performed video bitrate adjustment and automatic configuration of the WLAN link layer in response to performance notifications (containing performance statistics) received from the ULLA core.

Tools

Two simple test agents that emulated the behaviour of LU and LP were used to test the ULLA core functionality during unit development and testing. Performance benchmarks were developed and used to evaluate the performance of the ULLA core implementations in terms of latency of different operations and system loading (i.e. processing requirements). The multimedia streaming LU was developed and used to evaluate the performance enhancement capabilities of using adaptation exploiting the ULLA approach compared with the case of either no adaptation, or adaptation without link level interactions.

A.3 Connection Manager Validation Environment

Operating Systems

ULLA was implemented and validated on a Windows operating system focusing on Windows CE 5.0.

Networks

The Connection Manager environment included WLAN, BT and GPRS/UMTS LPs.

Hardware

The development hardware was based on HTC pocket PC Wizard running Windows CE 5.0. These PDAs are actually also GSM phones and support GPRS for data communications.

The characteristics of these devices are summarized in the following table.

	Wizard
Manufacturer	HTC
Processor	TI OMAP850 195 Mhz
RAM	64 MB
ROM	128 MB

GSM	850 / 900 / 1800 / 1900
GPRS	EGPRS Class B (Multi-slot Class 10)
Bluetooth	Yes
Wifi	Internal 802.11 b/g card

Table A.1: Characteristics of the Devices used by Connection Manager Platform

Software

ULLA was used to develop a simple application reporting availability of surrounding networks and allowing users to choose to connect to these networks. This application however has no real "intelligence" in the sense that it won't take any decision for the user.

We also investigated the implementation of a service that automatically connects the device to the "best available network" around the device. This application is called a Connection Manager (CM) on Pocket PC devices. The connection manager is existing software delivered on pocket PC platforms. The CM is used to synchronize and automate the establishment of network connections. The current implementation of the connection manager is limited because it does not really provide a way to dynamically discover networks and compare them.

Tools

All development and debugging was performed on a PC running Windows XP and using Visual Studio 2005. The majority of the development involved unmanaged code using C or C++ as development languages. In order to enable access to the Connection Manager from managed code (.Net), we also provided a wrapper for .Net. This wrapper was written in C#.

A.4 Real-time Audio Validation Environment

Operating Systems

ULLA was implemented and tested on a Linux operating system, version 2.4.19 for ARM processors. Specifically the version that was utilized is Linux-arm-2.4.19.-rmk2-ds6, which is a customization of the original Linux-arm-2.4.19 with patches publicly available from Russell King and further adapted to the Nomadik ™ processor by STM.

No tests with other operating systems were conducted.

Networks

ULLA was tested with Wireless LAN (IEEE802.11b/g) and Bluetooth 1.2 (dial-up and PAN profiles) LPs.

For WLAN, a commercial access point was used to provide connectivity; for Bluetooth, a PC with Bluetooth interface was used as a BT access point.

Hardware

As mentioned before, ULLA was prototyped and tested on a Nomadik™ board (NDK-10) based on ST chipset STn8810. The board has a QVGA LCD display, keyboard, 4 serial lines, GPIO, analog audio, USB On-The-Go (OTG) and Ethernet interfaces.

A proprietary low-power WLAN board prototype is used which is connected to the host processor through a synchronous serial interface running at 12 MHz. The STLC4370 chipset is used, which complies with IEEE802.11g standard and also features a Bluetooth coexistence HW interface to coordinate access to the ISM band with the colocated STLC2500 Bluetooth 1.2 chipset.

Audio acceleration is natively supported by STn8810 through a dedicated programmable DSP named Smart Audio Accelerator (SAA), which may be directly interfaced with the analog audio front-end. The SAA HW block is controlled by the ARM processor by means of dedicated Linux drivers. This is relevant in the scope of GOLLUM since the intended prototype showed how audio coding can be adapted to varying wireless link conditions, as a result of specific ULLA notifications.

Software

On top of the existing Linux OS port (which also includes all the drivers necessary to exploit STn8810 features) and the ULLA framework, an open source Voice-over-IP application was used.

In particular, LinPhone was the candidate. Modifications to this application were made in order to link the VoIP application with an existing, STM proprietary RTP library, which adds Forward Error Correction support, according to RFC2733. This increases the robustness of the VoIP flow when the link quality is scarce, be reducing the packet loss rate. Further modifications to LinPhone were obviously necessary to support ULLA.

Another application what was evaluated is "miniSIP" which also adds real-time video support. Therefore in this case, the application was ULLA-driven adaptive IP video telephony over WLAN.

As far as wireless drivers are concerned, the following Linux packages were used:

- WLAN: Proprietary "Phaser" driver for STLC4370.
- Bluetooth: BlueZ stack.

Tools

For unit testing, the tools developed in the GOLLUM project were used. These tools were also used for performance testing, especially of the ULLA core.

A.5 Sensor Network Validation Environment

Operating Systems

The sensor network validation environment was implemented and tested under three different operating systems:

- Mote Processor /Radio Modules of each wireless sensor node: TinyOS 1.x
- Laptop: Windows XP/Cygwin
- Stargate: ARM Embedded Linux 2.4.19-rmk7-pxa2-star (patched for arm, Xscale and Motes interface)

Networks

ULLA was tested under the wireless sensor networks. Two different radio stacks were used to provide compatibility with existing radio technologies, namely IEEE 802.15.4 and Chipcon CC1000 radio. Moreover, the following types of network topologies were tested:

- Mote-to-Mote communication
- Mote-to-Base station communication

Hardware

The wireless sensor network prototype was implemented and tested on two different hardware platforms:

Wireless sensor node

- Telos revision B (TI MSP430) from Moteiv and MicaZ (Atmel MPR2400CA) from Crossbow. Both are 2.4GHz IEEE 802.15.4 radio compliant.
- Mica2 (Atmel MPR400CB) from Crossbow. It is Chipcon CC1000 radio compliant.

Sensor board compatible with MicaZ and Mica2

- MTS310 Multi-sensor module from Crossbow (light, temperature, microphone, sounder, tone detection circuit, 2-axis accelerometer and 2-axis magnetometer).
- Multi function Data Adquisition Board with humidity and temperature sensors (CrossBow MDA310).

Gateway Laptop or Gateway Stargate

Communication between the WSN and the outer world is carried out by a gateway connected to a laptop through the serial port or Ethernet.

An alternative to this combination is the use of an embedded platform, the Stargate from CrossBow. This platform is a single board computer with enhanced communications and sensor signal processing capabilities. XScale PXA255 Processor with serial, USB, 10/100 Mbps Ethernet port, PCMCIA slot, MicaZ or Mica2 interface and removable Ambicom WL 1100C Wireless LAN Compact Flash card.

Software

A reduced version of the ULLA core architecture which has a small size and lightweight concurrency was ported to every sensor node (running TinyOS/NesC) in the network with applications running on top of the ULLA. As mentioned before, two types of network topologies were tested. Therefore, the WSN applications were divided into two groups to handle all the possible situations. Two of the example applications were:

Mote-to-Mote Communication

- Routing Agent: responsible for delivering the data across the network according to the needs of the application. Using the ULLA over this platform allowed us to control the power saving mode of the radio module and take care of the battery consumption (to be developed in NesC).
- The lightweight ULLA core was running on each sensor node. Besides, the LU and the LP were both running in the sensor nodes.

Mote-to-Base station Communication

- Smart Home Monitoring: responsible for the decision making inside the sensor nodes and for sending messages to the controller of the smart home (this is presumably a powerful computer with Internet connection).
- The ULLA architecture is separated into two parts: ULLA in the mote and ULLA in the base station. A part of the ULLA core and the LU was running in the base station laptop. In each sensor node, the lightweight version of ULLA core and the LP were running.

Tools

Mote-to-Mote Communication

- In order to test the lightweight ULLA core running in each sensor node which is written in NesC programming language, TOSSIM (a simulator for TinyOS networks) was used. TOSSIM was used to simulate a mote application developed in NesC (Routing agent) and to emulate the behaviour of the LU and the LP.

Mote-to-Base station Communication

- No standard tools were used to test the lightweight ULLA in each sensor node. Instead, a simple test application was used in the base station laptop to emulate the behaviour of the LU in order to evaluate the functionally and to verify the design integrity of the ULLA core. Smart home monitoring was used for testing as a LU.

Finally, user-like applications were implemented in order to validate a proof-of-concept of the complete system.

Appendix B Test cases for API validation & Performance testing

The following test cases tested all separate functions with several good and bad parameters. The time for every call was measured. Also test cases defining a use case were defined to test the complete system. Tests were performed on systems that have at least a one LLA (e.g. test Link Provider) with an active link (e.g. test Link).

This Link Provider test specification consists of the definitions of testLinkProvider and testLink classes and optionally the testChannel class. Each of these classes includes attributes and methods to provide full test coverage for ULLA type definitions, commands and notifications. Specific test cases to test the ULLA API function calls at the link provider level are described and this testLinkProvider was used to perform the conformance testing described in the remainder of this Appendix.

testLinkProvider Definition

Attributes

Attribute	Value	Type	Notes
id	x	public: *Id_t*	Unique link provider identifier.
title	"Test Link Provider"	public: *ULLA_TYPE_STRING*	LP name
supplier	"Gollum Project"	public:*ULLA_TYPE_STRING*	Supplier of the software implementation. Null-terminated string.
version	"1.0"	public:*ULLA_TYPE_STRING*	Null-terminated string describing the version.
type	ULLA_MEDIATYPE_UNKNOWN	public:*MediaType_t*	Type of communication system, e.g IEEE 802.11. As LPs might support multiple technologies this attribute is a multi-value attribute.
powerMode	ULLA_PM_ACTIVE	public:*PowerModes_t*	Which power mode is currently used, e.g. power saving, sleep etc.
networkScanPeriod	0	public:*ULLA_TYPE_INT*	Period of network scan used during link discovery given in milliseconds.
minPowerConsumption	0	public:*ULLA_TYPE_INT*	Minimum power consumption in stand-by mode with no active communication given in microwatts. This attribute should usually be constant but this depends on the technology. INFORMATIONAL.
maxPowerConsumpt	0	public:*ULLA_TYPE*	Maximum power consumption while continuously blasting at

ion		*_INT*	full power given in microwatts. This attribute should usually be constant but this depends on technology. INFORMATIONAL.
excludes		public:*ULLA_TYPE_INT*	This is a list of LPs (i.e. IDs) that are absolutely excluded from use when this LP is in use.
coexistsWith		public:*ULLA_TYPE_INT*	This is a list of LPs (i.e. IDs) that are allowed to coexist with this LP, i.e. they can operate in parallel with no problems.
simultaneous		public:*ULLA_TYPE_INT*	This denotes the number of LPs that can be used in parallel from the set of this link and those listed in the coexistsWith attribute.
dependentOn		public:*ULLA_TYPE_INT*	This is a list of LPs (i.e. IDs) that this link has some sort of dependency on. This attribute simply provides an indication that performance problems can arise when using this LP with others the list. It does not attempt to quantify the possible interference levels.
testCmdTimeDelay	- not set by default -	public:*ULLA_TYPE_INT*	This is specifically for testing the command that requires an attribute to be set before execution is possible. The test command simple delays for this length of time before returning.
testCmdState	0	public:*ULLA_TYPE_INT*	This indicates the state of the test command. 0 is not run yet 1 is running and 2 is completed

Methods

Method	Type	Notes
scanAvailableLinks ()	public: *ULLA_TYPE_INT*	Start the scanning procedure for any available links in the surrounding. Used to enable a forced scan for available networks. For testing link provider a test link is created and ULLA_OK is returned
createLink ()	public: *ULLA_TYPE_INT*	The LP should create a new aggregate link.
reset()	public: *ULLA_TYPE_INT*	The test link provider is reset and all links and channels associated with this test LP are deregistered

duplicate()	public: *ULLA_TYPE_INT*	This command causes another test LP to be registered. Should create another test LP and return ULLA_OK.
duplicateLink()	public: *ULLA_TYPE_INT*	This command creates a further test link and adds it to the ullaLink table. ULLA_OK should be returned.
Shutdown()	public: *ULLA_TYPE_INT*	This command causes the corresponding test LP to be deregistered. It should return ULLA_OK, however, if it is the first test LP (the original one) it is not and any other value is returned
registerChannel()	public: *ULLA_TYPE_INT*	Instructs the test LP to registers a test channel associated with the test link Returns ULLA_OK if the test LP supports channels and any other value if not
testCmd()	public: *ULLA_TYPE_INT*	A test command that requires the testCmdTimeDelay Attribute to be set. It simply sets its state attribute to 1 (running) delays for the length of time (in ms) given by testCmdTimeDelay and then sets the state to 2 (completed) and returns. If the command is terminated part way through its state is set to 0 again (not running).

testLink Definition

Attributes

Attribute	Value	Type	Notes
id	x	public: *Id_t*	Unique link identifier
signature	x	public : *ULLA_TYPE_STRING*	Signature computed with hash function (16 bytes). Null terminated character array.
lpId	x	public : *Id_t*	Unique identifier of the Link Provider offering this link.
plId		public : *Id_t*	Parent Link ID. If the link is an aggregate link the plId will be zero, otherwise the identifier of the respective aggregate link is given.
networkName	"Test Network"	public : *ULLA_TYPE_STRING*	Examples: ESSID for WLANs based on IEEE 802.11, mobile network code/country node
operatorName	"Test Operator"	public : *ULLA_TYPE_STRING*	Examples: T-Mobile, Telefonica, etc.
type	ULLA_MEDIATYPE_UNKNOWN	public : *MediaType_t*	Type of communication technology used, Examples: Bluetooth, IEEE 802.11, etc.

state		public : *MediaState_t*	State of the link (read-only).
remoteL2-address	"01:02:03:04:05:06\0"	public : *ULLA_TYPE_STRING*	Remote link layer address (L2 address): Unicast link: L2 address of the remote peer Broadcast link: L2 broadcast address Multicast link: L2 address of the multicast group Examples:_"01:02:03:04:05:06\0", IMEI = 16 bytes, etc. The maximum size of the null-terminated character array is 64 bytes.
localL2-address	"01:02:03:04:05:07\0"	public : *ULLA_TYPE_STRING*	Local link layer address
txBitRate	0	public : *ULLA_TYPE_INT*	Uplink bitrate in bits per second
rxBitRate	0	public : *ULLA_TYPE_INT*	Downlink bitrate in bits per second
rxQuality	0	public Range:1 to 100: *ULLA_TYPE_INT*	Quality of the link in the receive path given in percentage [0..100]. The algorithm how to calculate this value is LLA-implementation dependent. Possible approaches might incorporate the received signal strength, error rates, etc.
txQuality	0	public Range:1 to 100: *ULLA_TYPE_INT*	Quality of the link in the transmit path given in percentage [0..100]. The algorithm how to calculate this value is LLA-implementation dependent. Possible approaches might incorporate the transmit power, error rates, etc.
rxSignalStrength	0	public : *ULLA_TYPE_INT*	Signal strength of the received signal given in dBm.
rxNoise	0	public : *ULLA_TYPE_INT*	Received interference and noise. Usual devices cannot differentiate between noise and interference so that this attribute covers both values.
txSignalPower	0	public : *ULLA_TYPE_INT*	Power used for transmitted signals given in dBm.
rxJitter	0	public : *ULLA_TYPE_INT*	Link jitter in the receive path given in microseconds.
txJitter	0	public : *ULLA_TYPE_INT*	Link jitter in the transmit path given in microseconds.
rxLatency	0	public : *ULLA_TYPE_INT*	Link latency in the receive path
txLatency	0	public : *ULLA_TYPE*	Link latency in the transmit path

		_INT	
connectedTime	0	public : *ULLA_TYPE_INT*	Connection time in milliseconds describing how long the connection exists without interruption. The connection time will be reset if a connection break occurs. Initial Value: 0;
costPerUnit	0	public : *ULLA_TYPE_INT*	Cost per defined unit
costUnit	0	public : *CostUnit_t*	Cost unit used. Examples: kByte, MByte, sec, hour, flat, etc.
packetsReceived	0	public : *ULLA_TYPE_INT*	Number of packets received. The attribute is only valid in state CONNECTED.
packetsSent	0	public : *ULLA_TYPE_INT*	Number of packets sent. The attribute is only valid in state CONNECTED.
bytesReceived	0	public : *ULLA_TYPE_INT*	Number of bytes received. The attribute is only valid in state CONNECTED.
bytesSend	0	public : *ULLA_TYPE_INT*	Number of bytes sent. The attribute is only valid in state CONNECTED.
frameReceiveErrors	0	public : *ULLA_TYPE_INT*	Number of received frames that contained errors. The attribute is only valid in state CONNECTED.
frameSendErrors	0	public : *ULLA_TYPE_INT*	Number of sent frames that could not be transmitted successfully because of errors. The attribute is only valid in state CONNECTED.
communicationMode	ULLA_MODE_UNKNOWN	public : *CommunicationMode_t*	Communication mode used, examples: TxSIMPLEX, RxSIMPLEX, HALF-DUPLEX, FULL-DUPLEX
txMTU	0	public : *ULLA_TYPE_INT*	Uplink Maximum Transfer Unit in bytes
rxEncryption	0	public : *ULLA_TYPE_INT*	Indicate whether downlink encryption is supported. To be interpreted as a boolean.
txEncryption	0	public : *ULLA_TYPE_INT*	Indicate whether uplink encryption is supported. To be interpreted as a boolean.
degradingLink	0	public : *ULLA_TYPE_INT*	This flag is set by the LLA using some technology-specific mechanism to indicate when the link is constantly degrading, e.g. in the case of moving further and further away from a cellular base station. To be interpreted as a boolean.

excludes		public : *ULLA_TYPE _INT* []	This is a list of ullalinks (i.e. IDs) that are absolutely excluded from use when this link is in use.
coexistsWith		public : *ULLA_TYPE _INT*	This is a list of ullalinks (i.e. IDs) that are allowed to coexist with this link, i.e. they can operate in parallel with no problems.
simultaneous		public : *ULLA_TYPE _INT*	This denotes the number of ullalinks that can be used in parallel from the set of this link and those listed in the coexistsWith attribute.
dependentOn		public : *ULLA_TYPE _INT*	This is a list of ullalinks (i.e. IDs) that this link has some sort of dependency on. This attribute simply provides an indication that performance problems can arise when using this link with others the list. It does not attempt to quantify the possible interference levels.
testInt	1	*ULLA_TYPE _INT*	A test integer used for testing purposes
testMVInt	1,2,3,4,5,6, 7,8	*ULLA_TYPE _INT[]*	A multi-valued test integer used for testing purposes
testString	"a1"	*ULLA_TYPE _STRING*	A test string used for testing purposes
testMVString	"a1", "b2", "c3", "d4", "e5", "f6", "g7", "h8"	*ULLA_TYPE _STRING[]*	A multi-valued test string used for testing purposes
testDouble	1.1	*ULLA_TYPE _DOUBLE*	A test double used for testing purposes
testMVDouble	1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8	*ULLA_TYPE _DOUBLE[]*	A multi-valued test double used for testing purposes
testRawData	"a"	*ULLA_TYPE _RAWDATA*	A test byte array used for testing purposes
testMVRawDat a	"a", "b", "c", "d", "e", "f", "g", "h"	*ULLA_TYPE _RAWDATA []*	A multi-valued test byte array used for testing purposes
testInt_mean	2	*ULLA_TYPE _INT*	A test integer used for testing purposes – this is the mean value computed over the given window and interval
testInt_min	1	*ULLA_TYPE _INT*	A test integer used for testing purposes – this is the min value computed over the given window and interval
testInt_max	3	*ULLA_TYPE _INT*	A test integer used for testing purposes – this is the max value computed over the given window and interval

Methods

Method	Type	Notes
connect ()	public: *ULLA_TYPE_INT*	Connect the link given in the link id. Should return ULLA_OK if it is possible to hypothetically connect the test Link (i.e. if the link is a valid testLink and in the DISCONNECTED state)
disconnect()	public: *ULLA_TYPE_INT*	Disconnect the link given in the link id. Should return ULLA_OK if it is possible to hypothetically disconnect the test Link (i.e. if the link is a valid testLink and is in CONNECT state)
listen ()	public: *ULLA_TYPE_INT*	Switch a link to the LISTENING state. Should return ULLA_OK if it is possible to put test link into listen state (i.e. if the link is a valid testLink and it is DISCONNECTED)
close ()	public: *ULLA_TYPE_INT*	Switch a link back to the DISCONNECTED state when it was listening before. Should return ULLA_OK if it is possible to close the test Link (i.e. if the link is a valid testLink and it is in the LISTENING state)
accept ()	public: *ULLA_TYPE_INT*	When a connection request was received the link state was changed to PendingAuthentication. This command accepts the connection request and the link state is set to CONNECTED. Should return ULLA_OK if it is possible to put test link into CONNECTED state (i.e. if the link is a valid testLink and it is in the PENDING_AUTHENTICATION state)
reject ()	Public: *ULLA_TYPE_INT*	When a connection request was received the link state was changed to PendingAuthentication. This command rejects the connection request and the link state is set to DISCONNECTING. Should return ULLA_OK if it is possible to put test link into DISCONNECTED state (i.e. if the link is a valid testLink and it is in the PENDING_AUTHENTICATION state)
deleteLink ()	Public: *ULLA_TYPE_INT*	The link specified is deleted. This should usually only be used with aggregate links that were created before but might also be applied to other links. Should return ULLA_OK and delete the corresponding test link entry.

Optional testChannel Definition

Attributes

Attribute	Value	Type	Notes
technology	ULLA_MEDIATYPE_UNKNOWN, ULLA_MEDIATYPE_80211, ULLA_MEDIATYPE_BLUETOOTH, ULLA_MEDIATYPE_CSD,	*MediaType[]*	The type of link technology the channel is used for - could be multiple technologies (limited to 8)

	ULLA_MEDIATYPE_GPRS, ULLA_MEDIATYPE_ZIGBEE, ULLA_MEDIATYPE_UWB, ULLA_MEDIATYPE_UMTS		
type	ULLA_NONE, ULLA_TIME, ULLA_FREQUENCY, ULLA_CODE, ULLA_SPATIAL	*ChannelType[]*	The type of channel this object corresponds to - could be multiple types for hybrid channel concepts (limited to 8)
bandwidth	0	*ULLA_TYPE_INT*	The bandwidth of the channel in kHz
frequency	0	*ULLA_TYPE_INT*	The frequency of the channel in MHz
channelNumber	1, 2, 3, 4, 5, 6, ……,64	*ULLA_TYPE_INT []*	The unique technology specific channel number for this channel - could be multiple numbers for aggregate channels (limited to 64)
activityLevel	0	*ULLA_TYPE_INT*	The activity level observed on the channel over the monitoring time as average number of packets / frames in monitoring window.
noiseLevel	0	*ULLA_TYPE_INT*	The noiselevel observed in dBm for transmissions detected on the channel as average noise level detected in monitoring window. Noise being the background (i.e. thermal, and receiver noise) detected when no packet transmissions are being detected.
monitorDuration	0	*ULLA_TYPE_INT*	Duration of a single monitoring operation (in microseconds)
signalLevel	0	*ULLA_TYPE_INT*	The average signal level in dBm of the detected packets that are observed over the monitoring window.
monitorMethod	ULLA_PASSIVE	*MonitorType*	The type of monitoring that is being performed on this channel. This can be passive (observation of all transmissions on the channel), active (observation of specific packets transmitted for monitoring purposes) or selective (only observing certain packet types).
id	x	*ULLA_TYPE_INT*	Identifier of the channel (primary key)

extraInfo	1	*ULLA_TYPE_INT*	Flag to indicate whether additional information is available (valid values are : 1 for yes 0 for no)
antennaConfig	Increasing byte values of 0,1,2,3,4,5,6 …. 255	*ULLA_TYPE_RAW DATA [256]*	The antenna configuration specified as a byte array (limited to 256 bytes)
codes	Increasing byte values of 0,1,2,3,4,5,6 …. 255	*ULLA_TYPE_RAW DATA [256]*	The codes used to specify the channel configuration (for code division multiplexing or channel coding schemes) - specified as a byte array (limited to 256 bytes).
testDouble	Initial value 0	*ULLA_TYPE_ DOUBLE*	A test double used for testing purposes. One subsequent setattribute operations the value is updated and so is the mean, min and max values according to the measurement window and interval settings
testDouble_mean	Initial value 0	*ULLA_TYPE_ DOUBLE*	A test double used for testing purposes – this is the mean value computed over the given window and interval
testDouble_min	Initial value 0	*ULLA_TYPE_DOU BLE*	A test double used for testing purposes - this is the min value computed over the given window and interval
testDouble_max	Initial value 0	*ULLA_TYPE_DOU BLE*	A test double used for testing purposes - this is the max value computed over the given window and interval

ullaMeasureCap definition

To test measurement capabilities of the test link provider requires that the appropriate measurement capability and configuration information is provided by the test link provider.

Attributes

Attribute	Value1	Value2	Type	Notes
id	*x*	*y*	*ULLA_TYPE_INT*	Identifier of the measurement capability
param	*"testInt"*	*"testDoub le"*	*ULLA_TYPE_STR ING*	Name of the attribute that is associated with this measurement capability
units	*ULLA_C OUNTER*	*ULLA_D BM*	*MeasurementUnits*	These are the units used for these measurements.
Type	ULLA_LI NK	ULLA_C HANNE L	MeasurementTyp e	The type of object that this measurement corresponds to (currently either Link or Channel)
className	*testLink*	*testChann*	*ULLA_TYPE_STR*	The className of the class that this

		el	*ING*	measurement attribute corresponds to
time	*0*	*0*	*ULLA_TYPE_INT*	Time taken to perform a single measurement of this attribute in nanoseconds
power	*0*	*0*	*ULLA_TYPE_INT*	Energy consumed to perform a single measurement of this attribute in nJ
accuracy	*0*	*0*	*ULLA_TYPE_INT*	The absolute accuracy of a single measurement specified in the units associated with this measurement
precision	*0*	*0*	*ULLA_TYPE_INT*	The precision (relative error) between measurements specified in the units associated with this attribute measurement
windowMax	*10*	*10*	*ULLA_TYPE_INT*	The maximum window over which averaging (and other optional statistical operations) are performed for this attribute measurement. The window is specified in number of measurement samples of the attribute taken at intervals (determined by the interval value). A value of 0 in the windowMin indicates that the window is a smoothing function using the formula x(n+1) = a*x(n) + (1-a)*s [where x(n) represents the previous stored smoothed values, s is the measurement and a is the smoothing factor]. In which case the window contains the % value of a (i.e. 95%). Otherwise: +ve values indicate a periodic window with statistics computed on each non-overlapping window -ve values indicate a sliding window with statistics computed on each overlapping window that slides once per sample interval
windowMin	*-10*	*-10*	*ULLA_TYPE_INT*	The minimum window over which averaging (and other optional statistical operations) are performed for this attribute measurement. The window is specified in number of measurement samples of the attribute taken at intervals (determined by the interval value). A value of 0 in the windowMin indicates that the window is a smoothing function using the formula x(n+1) = a*x(n) + (1-a)*s [where x(n) represents the previous stored smoothed value, s is the measurement and a is the smoothing factor]. In which case the window contains the % value of a (i.e. 95%). Otherwise:

				+ve values indicate a periodic window with statistics computed on each non-overlapping window -ve values indicate a sliding window with statistics computed on each overlapping window that slides once per sample interval
intervalMin	*1*	*1*	*ULLA_TYPE_INT*	The minimum interval at which single measurements can be taken for this attribute measurement. The interval (i) is specified in terms of successive values of the test attribute taken on every ith value
intervalMax	*10*	*10*	*ULLA_TYPE_INT*	The maximum interval at which single measurements can be taken for this attribute measurement. The interval (i) is specified in terms of successive values of the test attribute taken on every ith value
interval	*1*	*1*	*ULLA_TYPE_INT*	The actual interval between successive updates of the test attribute on which to perform measurement statistics.
window	*10*	*10*	*ULLA_TYPE_INT*	The actual window size in number of samples. A value of 0 in the windowMin and windowMax indicates that the window is a smoothing function using the formula x(n+1) = a*x(n) + (1-a)*s [where x(n) represents the previous stored smoothed value, s is the measurement and a is the smoothing factor]. In which case the window contains the % value of a (i.e. 95%). Otherwise: +ve values indicate a periodic window with statistics computed on each non-overlapping window -ve values indicate a sliding window with statistics computed on each overlapping window that slides once per sample interval

LU API Function Call Test Cases

Test Case 1: Registration

ullaRegisterLu

Function call

ullaResultCode ullaRegisterLu(IN LuDescr_t* luDescr, IN LuRole_t luRole);

Case	Test with wrong version			
1.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	(see above) luDescr.Version="5.0"	ULLA_ERROR_VERSION_ MISMATCH		□
Case	Test with illegal Role			
1.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	(see above) luRole=ULLA_LINK_MANAGER	ULLA_ERROR_ROLE_ DENIED		□
Case	Test with unsupported Role			
1.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	all like case 1.1 except: luRole= ULLA_ROLE_MAX	ULLA_ERROR_ UNSUPPORTED_ROLE		□
Case	Test with unsupported profile			
1.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	all like case 1.1 except: luDescr.profile= ULLA_PROFILE_MAX	ULLA_ERROR_ UNSUPPORTED_PROFILE		□
Case	Test with luDescr =NULL			
1.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	luDescr = NULL	ULLA_ERROR_INVALID_ PARAMETER		□
Case	Test with correct parameters			
1.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	luDescr.name="LU conformance" luDescr.description="Conformance Test" luDescr.apiVersion="1.0" luDescr.profile=ULLA_PROFILE_UNSPECIFI ED luDescr. ullaExceptionHandler=excptionHandler luRole=ULLA_STANDARD_LINK_USER	ULLA_OK		□
Case	Register Twice			
1.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	luDescr.name="LU conformance" luDescr.description="Conformance Test" luDescr.apiVersion="1.0" luDescr.profile=ULLA_PROFILE_UNSPECIFI	ULLA_ERROR_ALREADY_ REGISTERED		□

ED

luDescr.
ullaExceptionHandler=excptionHandler

luRole=ULLA_STANDARD_LINK_USER

ullaUnregisterLu

Function call

ullaResultCode ullaUnregisterLu();

Case Test with correct parameters

1.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
		ULLA_OK		□

Case Link User has not registered yet

1.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
		ULLA_ERROR_NOTREGISTERED		□

Test Case 2: Synchronous Query

ullaRequestInfo

Function call

ULLA_API ullaResultCode ullaRequestInfo(IN ULLA_ STRING_t query, OUT ullaResult_t *result);

Case Link User has not registered yet

2.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = “SELECT * FROM testLink”	ULLA_ERROR_NOTREGISTERED		□

Case Test with correct parameters

2.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = “SELECT * FROM testLink”	ULLA_OK		□

Case Test with wrong Class

2.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = “SELECT * FROM gollum”	ULLA_ERROR_INVALID_CLASS		□

Case Test with wrong attribute

2.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = “SELECT gollum FROM testLink”	ULLA_ERROR_INVALID_ATTRIBUTE		□

Case Test with syntax error

2.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = “SELECT id FOM testLink”	ULLA_ERROR_SYNTAX_ERROR		□

Case Test with NULL pointer for ullaResult_t

2.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	result = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with WHERE =			
2.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT rxQuality FROM testLink WHERE id=1"	ULLA_OK		□
Case	Test with WHERE >			
2.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT id FROM testLink WHERE rxQuality>30"	ULLA_OK		□
Case	Test with WHERE AND			
2.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT id FROM testLink WHERE rxQuality>30 AND rxBitrate>100"	ULLA_OK		□
Case	Test with WHERE OR			
2.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT id FROM testLink WHERE rxQuality>30 OR rxBitrate>100"	ULLA_OK		□
Case	Call query 50 time to test minimal query per seconds and query latency			
2.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT id,networkName,type,rxSignalStrength FROM testLink"	ULLA_OK		□

ullaResultFree

Function call

ullaResultCode ullaResultFree(IN ullaResult_t res);

Case	Link User has not registered yet			
2.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with correct parameter			
2.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK		□
Case	Test with illegal result ID			
2.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□

ullaResultNumTuples

Function call

ULLA_API ullaResultCode ullaResultNumTuples(IN ullaResult_t res, OUT ULLA_ INT_t *num);

NOTE: As query "SELECT id,networkName,type,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case Link User has not registered yet

2.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with correct parameters

2.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct result parameter	ULLA_OK		□

Case Test with illegal result ID

2.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□

Case Test with NULL pointer for num

2.18	*Parameters*	*Expected Result*	*T(ms)*	*P*
	num = NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaResultTupleNumFields

Function call

ULLA_API ullaResultCode ullaResultNumFields(IN ullaResult_t res, OUT ULLA_ INT_t *num);

NOTE: As query "SELECT id,networkName,type,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case Link User has not registered yet

2.19	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with correct parameters

2.20	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct result parameter	ULLA_OK		□

Case Test with illegal result ID

2.21	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□

Case Test with NULL pointer for num

2.22	*Parameters*	*Expected Result*	*T(ms)*	*P*
	num = NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaResultNextTuple

Function call

ullaResultCode ullaResultNextTuple(IN ullaResult_t res);

NOTE: As query "SELECT id,networkName,type,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case Link User has not registered yet

2.23	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with correct parameters

2.24	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct result parameter	ULLA_OK		□

Case Test with illegal result ID

2.25	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□

Case Repeat ullaResultNextTuple beyond last tuple

2.26	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_MORE_TUPLES		□

ullaResultFieldName

Function call

ULLA_API ullaResultCode ullaResultFieldName(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_STRING_t name, INOUT ULLA_INT_t *size);

NOTE: As query "SELECT id,networkName,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case Link User has not registered yet

2.27	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with size=0

2.28	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo =2,size=0	ULLA_ERROR_BUFFER_TOO_SMALL		□

Case Test with size smaller as needed

2.29	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo =2,size < size returned in case 2.28	ULLA_ERROR_BUFFER_TOO_SMALL		□

Case Test with correct parameters

2.30	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK		□

	fieldNo =1, size =size returned in case 2.28 + 1	Value = "id"		
Case	Test with illegal result ID			
2.31	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.32	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.33	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=3	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for name			
2.34	*Parameters*	*Expected Result*	*T(ms)*	*P*
	name = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first **(Removed: Obsolete)**			
2.35	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
	fieldNo =2, size = 50			
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES **(Removed: Obsolete)**			
2.36	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
	fieldNo =2, size = 50			

ullaResultFieldNumber

Function call

ULLA_API ullaResultCode ullaResultFieldNumber(IN ullaResult_t res, IN ULLA_STRING_t fieldName, OUT ULLA_INT_t *num);

NOTE: As query "SELECT id,networkName,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.37	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with correct parameters			
2.38	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK		□
	fieldName="networkName"	Value = 2		

Case	Test with illegal result ID			
2.39	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field name			
2.40	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldName = "networkNam"	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for field name			
2.41	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldName = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for field			
2.42	*Parameters*	*Expected Result*	*T(ms)*	*P*
	num = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first **(Removed: Obsolete)**			
2.43	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldName="id"	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES **(Removed: Obsolete)**			
2.44	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldName="id"	ULLA_ERROR_NO_CURRENT_TUPLE		□

ullaResultNumFieldValues

Function call

ULLA_API ullaResultCode ullaResultNumFieldValues(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_INT_t *num);

NOTE: As query "SELECT id,networkName,testMVInt,testMVString,testMVRawData,testMVDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.45	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test first field			
2.46	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_OK, num=1		□
Case	Test with illegal result ID			

2.47	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.48	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.49	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=50	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for number			
2.50	*Parameters*	*Expected Result*	*T(ms)*	*P*
	num = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.51	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.52	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test testMVInt Field			
2.53	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=3	ULLA_OK, num=8		□
Case	Test testMVString Field			
2.54	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=4	ULLA_OK, num=8		□
Case	Test testMVRawData Field			
2.55	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=5	ULLA_OK, num=8		□
Case	Test testMVDoubleField			
2.56	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=6	ULLA_OK, num=8		□

ullaResultValueLength

Function call

ULLA_API ullaResultCode ullaResultValueLength(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_INT_t *size);

NOTE: As query "SELECT id,networkName,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.57	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with integer			
2.58	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, size=4		□
	fieldNo=3			
Case	Test with double			
2.59	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, size=8		□
	fieldNo=6			
Case	Test with testString			
2.60	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, size=2		□
	fieldNo=4			
Case	Test with illegal result ID			
2.61	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.62	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.63	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=50	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for size			
2.64	*Parameters*	*Expected Result*	*T(ms)*	*P*
	size = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.65	*Parameters*	*Expected Result*	*T(ms)*	*P*

	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.66	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test with testRawData			
2.67	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, size=1		□
	fieldNo=5			

ullaResultValueType

Function call

ULLA_API ullaResultCode ullaResultValueType(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT BaseType_t *type);

NOTE: As query "SELECT networkName,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.68	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with testInt			
2.69	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, type= ULLA_TYPE_INT		□
	fieldNo=2			
Case	Test with testDouble			
2.70	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, type= ULLA_TYPE_DOUBLE		□
	fieldNo=5			
Case	Test with testString			
2.71	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_OK, type= ULLA_TYPE_STRING		□
	fieldNo=3			
Case	Test with illegal result ID			
2.72	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.73	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□

Case	Test with illegal field number			
2.74	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=7	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for type			
2.75	*Parameters*	*Expected Result*	*T(ms)*	*P*
	type = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.76	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.77	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test with testRawData			
2.78	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=4	ULLA_OK, type = ULLA_TYPE_RAWDATA		□

ullaResultValueQualifier

Function call

ULLA_API ullaResultCode ullaResultValueQualifier(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT AttrQual_t* qualifier);

NOTE: As query "SELECT networkName,testInt,testString,testRawData,testDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.79	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with networkName			
2.88	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=1	ULLA_OK One of AttrQual_t values		□
Case	Test with testInt			
2.81	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=2	ULLA_OK One of AttrQual_t values		□

Case	Test with illegal result ID			
2.82	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.83	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.84	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=3	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for qualifier			
2.85	*Parameters*	*Expected Result*	*T(ms)*	*P*
	qualifier = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.86	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.87	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□

ullaResultIntValue

Function call

ULLA_API ullaResultCode ullaResultIntValue(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_INT_t *value);

NOTE: As query "SELECT networkName,testInt,testString,testRawData,testDouble,testMVInt,testMVString,testMVRawData,testMVDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.88	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with testInt			
2.89	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=2	ULLA_OK, subsequent call returns ULLA_ERROR_NO_MORE_VALUES Value = 1		□
Case	Test with testDouble			

2.90	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=5	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with testString			
2.91	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=3	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with testRawData			
2.92	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 4	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with illegal result ID			
2.93	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.94	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number = 50			
2.95	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=50	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for value			
2.96	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.97	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.98	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□

Case Test with multiple value testMVInt

	Parameters	*Expected Result*	*T(ms)*	*P*
2.99	Correct parameters fieldNo = 6	ULLA_OK (8 times) ULLA_ERROR_NO_MORE_VALUES once Values are 1,2,3,4,5,6,7,8		□

ullaResultDoubleValue

Function call

ULLA_API ullaResultCode ullaResultDoubleValue(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_DOUBLE_t *value);

NOTE: As query "SELECT networkName,testInt,testString,testRawData,testDouble,testMVInt,testMVString,testMVRawData,testMVDouble FROM testLink;" will be used

Case Link User has not registered yet

	Parameters	*Expected Result*	*T(ms)*	*P*
2.100	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with testDouble

	Parameters	*Expected Result*	*T(ms)*	*P*
2.101	Correct parameters fieldNo=5	ULLA_OK, subsequent call returns ULLA_ERROR_NO_MORE_VALUES Value = 1.1		□

Case Test with testInt

	Parameters	*Expected Result*	*T(ms)*	*P*
2.102	Correct parameters fieldNo=2	ULLA_ERROR_TYPE_MISMATCH		□

Case Test with testString

	Parameters	*Expected Result*	*T(ms)*	*P*
2.103	Correct parameters fieldNo=3	ULLA_ERROR_TYPE_MISMATCH		□

Case Test with multiple value testRawData

	Parameters	*Expected Result*	*T(ms)*	*P*
2.104	Correct parameters fieldNo = 4	ULLA_ERROR_TYPE_MISMATCH		□

Case	Test with illegal result ID			
2.105	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.106	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.107	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=10	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for value			
2.108	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.109	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.110	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test with multiple value testMVDouble			
2.111	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 9	ULLA_OK (8 times) ULLA_ERROR_NO_MORE_VALUES once Values are 1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8		□

ullaResultStringValue

Function call

ULLA_API ullaResultCode ullaResultStringValue(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_STRING_t str, INOUT ULLA_INT_t* size);

NOTE: As query "SELECT networkName,testInt,testString,testRawData,testDouble,testMVInt,testMVString,testMVRawData,testMVDouble FROM testLink;" will be used

Case	Link User has not registered yet			
2.112	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Test with testString and size=0			
2.113	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=3, size=0	ULLA_ERROR_BUFFER_TOO_SMALL Size =2		□
Case	Test with a testString and size to small			
2.114	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=3, size=2 (not including '\0' terminator	ULLA_ERROR_BUFFER_TOO_SMALL Size =2		□
Case	Test with testString			
2.115	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=3	ULLA_OK, Value = "a1" subsequent call returns ULLA_ERROR_NO_MORE_VALUES		□
Case	Test with testDouble			
2.116	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=5	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with testInt			
2.117	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=2	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with testRawData			
2.118	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 4	ULLA_ERROR_TYPE_MISMATCH		□
Case	Test with illegal result ID			
2.119	*Parameters*	*Expected Result*	*T(ms)*	*P*

	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□
Case	Test with illegal field number=0			
2.120	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.121	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=10	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for value			
2.122	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.123	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.124	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test with multiple value testMVString			
2.125	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 7	ULLA_ERROR_TYPE_MISMATCH Subsequent call to ullaResultStringValue returns ULLA_OK (8 times) ULLA_ERROR_NO_MORE_VALUES once Values = “a1”,”b2”,”c3”,”d4”,”e5”,”f6”, ”g7”,”h8”		□

ullaResultRawDataValue

Function call

ULLA_API ullaResultCode ullaResultRawDataValue(IN ullaResult_t res, IN ULLA_INT_t fieldNo, OUT ULLA_RAWDATA_t str, INOUT ULLA_INT_t* size);

NOTE: As query “SELECT

networkName,testInt,testString,testRawData,testDouble,testMVInt,testMVString,testMVRawData,testMVDouble FROM testLink;" will be used

Case Link User has not registered yet

2.126	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Test with a RawData and size=0

2.127	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=4, size=0	ULLA_ERROR_NOT_ENOUGH_SPACE Size = 1		□

Case Test with testRawData

2.128	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=4	ULLA_OK, Value = "a" subsequent call returns ULLA_ERROR_NO_MORE_VALUES		□

Case Test with testDouble

2.129	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=5	ULLA_ERROR_TYPE_MISMATCH		□

Case Test with testInt

2.130	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo=2	ULLA_ERROR_TYPE_MISMATCH		□

Case Test with testString

2.131	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 3	ULLA_ERROR_TYPE_MISMATCH		□

Case Test with illegal result ID

2.132	*Parameters*	*Expected Result*	*T(ms)*	*P*
	res = -1	ULLA_ERROR_INVALID_ULLARESULT		□

Case Test with illegal field number=0

2.133	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=0	ULLA_ERROR_INVALID_FIELD		□
Case	Test with illegal field number			
2.134	*Parameters*	*Expected Result*	*T(ms)*	*P*
	fieldNo=10	ULLA_ERROR_INVALID_FIELD		□
Case	Test with NULL pointer for value			
2.135	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test without calling ullaResultNextTuple first			
2.136	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test after ullaResultNextTuple returned ULLA_ERROR_NO_MORE_TUPLES			
2.137	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_CURRENT_TUPLE		□
Case	Test with multiple value testMVRawData			
2.138	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters fieldNo = 8	ULLA_ERROR_TYPE_MISMATCH subsequent call to ullaResultRawDataValue returns ULLA_OK (8 times) ULLA_ERROR_NO_MORE_VALUES once Values = “a”,”b”,”c”,”d”,”e”,”f”,”g”,”h”		□

Test Case 3: Asynchronous Query

ullaRequestNotification

Function call

ullaResultCode ullaRequestNotification(IN RnDescr_t* rndescr, IN handleNotification_t handler, OUT RnId_t* rnId);

Case	Link User has not registered yet			
3.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Single shot notification with correct query and RnDescr_t			
3.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.count=1;	ULLA_OK		□
	rndescr. period=0; rndescr.privdata=NULL;	Hander called with value x (unless it is cancelled)		
	rndescr.query = "SELECT id FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';"; handler =notificationHandler			
Case	Test with wrong Class			
3.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.query = "SELECT id FROM gollum WHERE rxBitRate > 2000000"	ULLA_ERROR_INVALID_CLASS		□
Case	Test with wrong attribute			
3.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.query = "SELECT xxx FROM testLink WHERE rxBitRate > 2000000"	ULLA_ERROR_INVALID_ATTRIBUTE		□
Case	Test with syntax error			
3.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.query = "SELECT id FOM testLink WHERE rxBitRate > 2000000"	ULLA_ERROR_SYNTAX_ERROR		□
Case	Test with NULL pointer for rnId			
3.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rnID = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for query			
3.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.query = NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for rndescr			
3.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr = NULL	ULLA_ERROR_INVALID_PARAMETER		□

Case Repeating notification with correct parameters (period=50ms)

3.9	*Parameters*	*Expected Result*	*T(ms)*	*P*

	rndescr.query = "SELECT rxQuality FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';;"
	rndescr.count=10;
	rndescr. period=50;
	rndescr.privdata=NULL;
	handler =notificationHandler;
		ULLA_OK,
		Should stop after 10 notifications. The period is also going to be checked
				□

Case Repeating notification with correct parameters (period=100ms)

3.10	*Parameters*	*Expected Result*	*T(ms)*	*P*

	rndescr.query = "SELECT rxQuality FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';;"
	rndescr.count=10;
	rndescr. period=100;
	rndescr.privdata=NULL;
	handler =notificationHandler;
		ULLA_OK,
		Should stop after 10 notifications. The period is also going to be checked
				□

Case Repeating notification with correct parameters (period=1000ms)

3.11	*Parameters*	*Expected Result*	*T(ms)*	*P*

	rndescr.query = "SELECT rxQuality FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';;"
	rndescr.count=10;
	rndescr. period=1000;
	rndescr.privdata=NULL;
	handler =notificationHandler;
		ULLA_OK,
		Should stop after 10 notifications. The period is also going to be checked
				□

Case Repeating notification with correct parameters where period is too short

3.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rndescr.query = "SELECT rxQuality FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';"	ULLA_ERROR_PERIOD_TOO_SHORT		□

	rndescr.count=10;
	rndescr. period=1;
	rndescr.privdata=NULL;
	handler =notificationHandler;			

Case	Illegal notification handler			
3.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	handler=NULL	ULLA_ERROR_INVALID_HANDLER		□
Case	Single shot notification with correct query and RnDescr_t with passing privdata			
3.14	*Parameters*	*Expected Result*	*T(ms)*	*P*

	Query = "SELECT id FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';"
	rndescr.count=1;
	rndescr. period=0;
	rndescr.privdata=data;
	handler =notificationHandler;
		ULLA_OK,
		Data should be accessible in the handler.
				□

ullaCancelNotification

Function call

ullaResultCode ullaCancelNotification(IN RnId_t rnId);

Case	Link User has not registered yet			
3.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Cancel repetitive notification			
3.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct rnId	ULLA_OK		□
Case	Cancel single shot notification that has not been fired			
3.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct rnId	ULLA_OK		□
Case	Cancel single shot notification that already has been fired			
3.18	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rnId of notification that has been fired	ULLA_OK		□
Case	Cancel of wrong rnId			

3.19	*Parameters*	*Expected Result*	*T(ms)*	*P*
	rnId = -1	ULLA_ERROR_INVALID_NOTIFICATION		□

Test Case 4: Synchronous Command Execution

ullaPrepareCmd

Function call

ullaResultCode ullaPrepareCmd (IN CmdDescr_t* cmddescr);

NOTE: with the query "SELECT id from testLinkProvider;" the first Link Provider will be selected

Case Link User has not registered yet

4.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Lock link for "scanAvailableLinks" command

4.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks";	ULLA_OK		□

Case Test with unknown id

4.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = -1; cmddescr.class=" testLinkProvider"; cmddescr.cmd=" scanAvailableLinks";	ULLA_ERROR_UNKNOWN_ID		□

Case Test with illegal command "something"

4.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd=" something";	ULLA_ERROR_CMD_NOT_VALID		□

Case Test with illegal class "gollum"

4.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="gollum"; cmddescr.cmd=" scanAvailableLinks";	ULLA_ERROR_INVALID_CLASS		□

Case Test with NULL pointer for cmddescr

4.6	*Parameters*	*Expected Result*	*T(ms)*	*P*

	cmddescr = NULL;	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for cmddescr.cmd			
4.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=NULL;	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for cmddescr.class			
4.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=NULL; cmddescr.cmd=“scanAvailableLinks”;	ULLA_ERROR_INVALID_PARAMETER		□

ullaDoCmd

Function call

ULLA_API ullaResultCode ullaDoCmd(IN CmdDescr_t* cmddescr, IN ULLA_ INT_t timeoutValue);

NOTE: with the query “SELECT id from testLinkProvider;” the first Link Provider will be selected

Case	Link User has not registered yet			
4.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Execute “scanAvailableLinks” command			
4.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“scanAvailableLinks”; timeoutValue=5000;	ULLA_OK		□
Case	Test with unknown id			
4.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = -1; cmddescr.class=” testLinkProvider”; cmddescr.cmd=“ scanAvailableLinks”;	ULLA_ERROR_UNKNOWN_ID		□
Case	Test with illegal command “something”			
4.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“ something”;	ULLA_ERROR_CMD_NOT_VALID		□

Case	Test with illegal class "gollum"			
4.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="gollum"; cmddescr.cmd="scanAvailableLinks";	ULLA_ERROR_INVALID_CLASS		□
Case	Test with NULL pointer for cmddescr			
4.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr = NULL;	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for cmddescr.cmd			
4.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd=NULL;	ULLA_ERROR_INVALID_PARAMETER		□
Case	Test with NULL pointer for cmddescr.class			
4.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=NULL; cmddescr.cmd="scanAvailableLinks";	ULLA_ERROR_INVALID_PARAMETER		□
Case	Execute command with a very short timeout			
4.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks"; timeoutValue=5;	ULLA_OK or ULLA_ERROR_TIMEOUT		□

Test Case 5: Asynchronous Command Execution

ullaRequestCmd

Function call

ULLA_API ullaResultCode ullaRequestCmd(IN CmdDescr_t* cmddescr, IN handleAsyncCmd_t handler, OUT CmdId_t *cmdId);

NOTE: with the query "SELECT id from testLinkProvider;" the first Link Provider will be selected

Case	Link User has not registered yet			
5.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case	Execute "scanAvailableLinks" command asynchronously			
5.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks"; handler = commandHandler;	ULLA_OK Cmd executed and ULLA_OK in the command handler.		□
Case	Test with unknown id			
5.3	*Parameters*	*Expected Result*	*T(ms)*	*P*

	cmddescr.id = -1;
	cmddescr.class=" testLinkProvider";
	cmddescr.cmd="
	scanAvailableLinks";
	handler = commandHandler;
		ULLA_ERROR_UNKNOWN_ID		□

Case	Test with illegal command "something"			
5.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd=" something"; handler = commandHandler;	ULLA_ERROR_CMD_NOT_VALID		□
Case	Test with illegal class "gollum"			
5.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="gollum"; cmddescr.cmd=" scanAvailableLinks"; handler = commandHandler;	ULLA_ERROR_INVALID_CLASS		□
Case	Test with NULL pointer for cmddescr			
5.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Cmddescr = NULL;	ULLA_ERROR_INVALID_PARAMETER Cmd not executed and cmd handler not called		□
Case	Test with NULL pointer for cmddescr.cmd			
5.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.cmd = NULL;	ULLA_ERROR_INVALID_PARAMETER Cmd not executed and cmd handler not called		□

Case	Test with NULL pointer for cmddescr.class			
5.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.class = NULL;	ULLA_ERROR_INVALID_PARAMETER Cmd not executed and cmd handler not called		□
Case	Test with NULL pointer for handler			
5.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“scanAvailableLinks”; handler = NULL;	ULLA_ERROR_INVALID_HANDLER And cmd handler not called.		□
Case	Execute command that requires attributes to be set			
5.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“ testCmd”; timeoutValue=5000;	ULLA_ERROR_ATTRIBUTE_NOT_SET And cmd handler not called		□

ullaCancelCmd

Function call

ullaResultCode ullaCancelCmd(IN CmdId_t cmdId);

Case	Link User has not registered yet			
5.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Cancel command that has not finished yet			
5.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct cmdId	ULLA_OK		□
Case	Cancel command that has finished			
5.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct cmdId	ULLA_OK		□
Case	Cancel of wrong cmdId			
5.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmdId = -1	ULLA_ERROR_INVALID_COMMAND		□

Test Case 6: Other API functions

ullaGetErrorString

Function call

ULLA_API ullaResultCode ullaGetErrorString(OUT ULLA_STRING_t str, INOUT ULLA_INT_t *size);

Case Call when there has been an error with size = 0

6.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	size=0	ULLA_ERROR_NOT_ENOUGH_SPACE Size contains number of bytes needed		□

Case Call when there has been an error

6.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	size = size from above +1	ULLA_OK		□

Case Call when there has not been an error

6.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NO_KNOWN_ERROR		□

Case Call with str=NULL

6.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	str=NULL	ULLA_ERROR_INVALID_PARAMETER		□

Case Call with size=NULL

6.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	size=NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaRegisterLm

Function call

ullaResultCode ullaRegisterLm(IN LuDescr_t* luDescr, IN LmAuthorizationHandlers_t auth);

Case Link manager is not supported. This is the only return value that MUST be implemented

6.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Parameters are not inportant	ULLA_ERROR_LM_NOT_SUPPORTED		□

Case Link manager is supported. But null presented instead of parameters

6.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	NULL for handlers	ULLA_ERROR_INVALID_PARAMETER		□

Case Link manager is supported, but the process registering has insufficient privilege

6.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Parameters are not inportant	ULLA_AUTHORIZATION_FAILED		□

ullaGetLinkIdFromDest

Function call

ullaResultCode ullaGetLinkIdFromDest(IN layer3Address_t *dest, OUT Id_t *linkId, OUT Id_t *lpId);

Case	Link User has not registered yet			
6.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	The function is not supported. This is the only return value that MUST be implemented			
6.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest	ULLA_ERROR_UNSUPPORTED_FEATURE		□
Case	Function is supported with valid destination address			
6.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address)	ULLA_OK		□
Case	Function is supported with invalid destination address			
6.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address)	ULLA_ERROR_DESTINATION_NOT_REACHABLE		□
Case	Function is supported dest = NULL			
6.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest=NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Function is supported destination address is invalid			
6.14	*Parameters*	*Expected Result*	*T(ms)*	*P*

	dest.protocol = ULLA_L3ADDR_APPLETALK dest.address="127.0.0.1" dest.length=strlen(dest.address)	ULLA_ERROR_INVALID_PARAMETER		□
Case	Function is supported, linkdId = NULL			
6.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	linkId=NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Function is supported, lpId = NULL			
6.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId=NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaConfigureL3

Function call

ULLA_API ullaResultCode ullaConfigureL3(IN Id_t linkId, IN layer3Address_t *dest);

Case	Link User has not registered yet			
6.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	The function is not supported. This is the only return value that MUST be implemented			
6.18	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest	ULLA_ERROR_UNSUPPORTED_FEATURE		□
Case	Function is supported with valid destination address			
6.19	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address) linked= valid id	ULLA_OK		□
Case	Function is supported with invalid destination address			

6.20	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address)	ULLA_ERROR_DESTINATION_NOT_REACHABLE		□
Case	Function is supported dest = NULL			
6.21	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest=NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	Function is supported destination address is invalid			
6.22	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_APPLETALK dest.address="127.0.0.1" dest.length=strlen(dest.address)	ULLA_ERROR_INVALID_PARAMETER		□
Case	Function is supported, linkdId = invalid			
6.23	*Parameters*	*Expected Result*	*T(ms)*	*P*
	linkId=-1	ULLA_ERROR_INVALID_PARAMETER		□

ullaSetAttribute

Function call

ullaResultCode ullaSetAttribute(IN AttrDescr_t attrDescr);

Case	Link User has not registered yet			
6.24	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Setting an int (using testLink testInt assuming it is not read-only)			
6.25	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= 42 attrDescr.id=correct LinkId attrDescr.className="testLink"	ULLA_OK Subsequent ullaRequestInfo on testInt should have value 42		□

	attrDescr.attribute=”testInt” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value			
Case	Setting a string(using testLink.testString assuming it is not read-only)			
6.26	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= “Gollum” attrDescr.id=correct LinkId attrDescr.className=”testLink” attrDescr.attribute=”testString” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_STRING attrDescr.length=strlen(value) attrDescr.numValues=1 attrDescr.data=value	ULLA_OK Subsequent ullaRequestInfo on testString should have value 0		□
Case	Setting a read-only attribute (using id)			
6.27	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= 42 attrDescr.id=correct LinkId attrDescr.className=”testLink” attrDescr.attribute=”id” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_SETATTR_READ ONLY		□
Case	Using an invalid attribute			
6.28	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= 42 attrDescr.id=correct LinkId attrDescr.className=”testLink”	ULLA_ERROR_INVALID_ATTRI BUTE		□

	attrDescr.attribute=”gollum” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value			
Case	Using an invalid class			
6.29	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_KBYTE attrDescr.id=correct LinkId attrDescr.className=”gollumLink” attrDescr.attribute=”costUnit” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_INVALID_CLASS		□
Case	Using an invalid link id			
6.30	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_KBYTE attrDescr.id=-1 attrDescr.className=”testLink” attrDescr.attribute=”costUnit” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_UNKNOWN_ID		□
Case	Setting an invalid value (using testLink. costUnit assuming it is not read-only)			
6.31	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_MAX attrDescr.id=correct LinkId attrDescr.className=”testLink”	ULLA_ERROR_INVALID_VALUE		□

	attrDescr.attribute=”costUnit” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value			
Case	Using an invalid qualifier (using testLink. costUnit assuming it is not read-only)			
6.32	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_KBYTE attrDescr.id=correct LinkId attrDescr.className=”testLink” attrDescr.attribute=”costUnit” attrDescr.qualifier= ULLA_QUAL_MAX attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_INVALID_QUALIFIER		□
Case	attrDescr = NULL			
6.33	*Parameters*	*Expected Result*	*T(ms)*	*P*
	attrDescr=NULL	ULLA_ERROR_INVALID_PARAMETER		□
Case	trying to set multiple values for a single value attribute			
6.34	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value[]= {1,2,3,4}; attrDescr.id=correct LinkId attrDescr.className=”testLink” attrDescr.attribute=”costUnit” attrDescr.qualifier= ULLA_QUAL_MAX attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=4 attrDescr.data=value	ULLA_ERROR_SETATTR_NOTMULTIPLE		□
Cas	Setting multiple value integer attribute			

e

6.35	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value[]= {0,1,2,3,4,5,6,7}; attrDescr.id=correct LinkId attrDescr.className="testLink" attrDescr.attribute="testMVInt" attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=8 attrDescr.data=value	ULLA_OK A subsequent call to ullaRequestInfo will reveal that the parameters have been updated correctly		□

Case	Setting multiple value string attribute

6.36	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value[]= {"A1","B2","C3","D4","E5","F6","G7","H8"}; attrDescr.id=correct LinkId attrDescr.className="testLink" attrDescr.attribute="testMVInt" attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_STRING attrDescr.length=3; attrDescr.numValues=8 attrDescr.data=value	ULLA_OK A subsequent call to ullaRequestInfo will reveal that the parameters have been updated correctly		□

ullaGetAppInfo

Function call

ullaResultCode ullaGetAppInfo(IN LuId_t luId, OUT ULLA_STRING_t info, INOUT ULLA_INT_t *size);

NOTE calling this method as ULLA_ROLE_STD_LU

Case	Link User has not registered yet			
6.37	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Cancel command that has not finished yet			
6.38	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_AUTHORIZATION_FAILED		□

ullaGetCoreDescriptor

Function call

ullaResultCode ullaGetCoreDescriptor(INOUT CoreDescr_t *coreDescriptor);

Case Call with correct parameter

6.39	*Parameters*	*Expected Result*	*T(ms)*	*P*
	CoreDesct_t desc; coreDescriptor=&desc	ULLA_OK		□

Case coreDescriptor=NULL

6.40	*Parameters*	*Expected Result*	*T(ms)*	*P*
	coreDescriptor=NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaGetSupportedClasses

Function call

ullaResultCode ullaGetSupportedClasses(IN Id_t lpId, OUT ULLA_STRING_t classList, INOUT ULLA_INT_t *size);

Case The function is not supported. This is the only return value that MUST be implemented

6.41	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaGetClassAttributes

Function call

ullaResultCode ullaGetClassAttributes(IN Id_t lpId, IN ULLA_STRING_t className, OUT ULLA_STRING_t attributeList, INOUT ULLA_INT_t *size);

NOTE the testLink class will be used

Case The function is not supported. This is the only return value that MUST be implemented

6.42	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaGetAttributeInfo

Function call

ullaResultCode ullaGetAttributeInfo(IN Id_t lpId, IN ULLA_STRING_t className, IN ULLA_STRING_t attributeName, OUT ULLA_STRING_t attributeDescription, INOUT ULLA_INT_t *size);

NOTE the testLink class and the netwrokName attribute will be used

Case The function is not supported. This is the only return value that MUST be implemented

6.43	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaGetClassCommands

Function call

ullaResultCode ullaGetClassCommands(IN Id_t lpId, IN ULLA_STRING_t className, OUT ULLA_STRING_t commandList, ULLA_INT_t *size);

NOTE: the testLink class will be used

Case The function is not supported. This is the only return value that MUST be implemented

6.44	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaGetCommandAttributes

Function call

ullaResultCode ullaGetCommandAttributes(IN Id_t lpId, IN ULLA_STRING_t className, IN ULLA_STRING_t commandName, OUT ULLA_STRING_t attributeList, INOUT ULLA_INT_t *size);

NOTE the testLinkProvider class and the scanAvailableLinks commandwill be used

Case The function is not supported. This is the only return value that MUST be implemented

6.45	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaCreateHistoricalTable

Function call

ullaResultCode ullaCreateHistoricalTable(IN ULLA_STRING_t tableName, IN Id_t sourceId, IN ULLA_STRING_t valueName, IN ULLA_INT_t period, IN ULLA_INT_t count);

NOTE using histTable and testLink.rxSignalStrenght as parameters

Case The function is not supported. This is the only return value that MUST be implemented

6.46	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaDeleteHistoricalTable

Function call

ullaResultCode ullaDeleteHistoricalTable(IN ULLA_STRING_t tableName);

Case The function is not supported. This is the only return value that MUST be implemented

6.47	*Parameters*	*Expected Result*	*T(ms)*	*P*
	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

ullaToggleStatusHistoricalTable

Function call

ullaResultCode ullaToggleStatusHistoricalTable(IN ULLA_STRING_t tableName, OUT

ULLA_INT_t status);

Case The function is not supported. This is the only return value that MUST be implemented

	Parameters	*Expected Result*	*T(ms)*	*P*
6.48	parameters do not matter	ULLA_ERROR_UNSUPPORTED_FEATURE		□

Security Test Cases

Test Case 7: Link Manager

This test case will only be run when a Link Manager is supported. The test will start a small Link Manger that will return ULLA_AUTHORIZATION_FAILED on all but the lmRegisterLu call.

ullaRegisterLm

Function call

ullaResultCode ullaRegisterLm(IN LuDescr_t* luDescr, IN LmAuthorizationHandlers_t auth);

Case Register Link Manager

	Parameters	*Expected Result*	*T(ms)*	*P*
7.1	luDescr.name=”LU conformance” luDescr.description=”Conformance Test” luDescr.apiVersion=”1.0” luDescr.profile=ULLA_PROFILE_UNSPECIFIED luDescr. ullaExceptionHandler=excptionHandler auth= structure will all right handlers	ULLA_OK		□

Case Test with wrong version

	Parameters	*Expected Result*	*T(ms)*	*P*
7.2	(see above) luDescr.Version=”5.0”	ULLA_ERROR_VERSION_MISMATCH		□

Case Test with unsupported profile

	Parameters	*Expected Result*	*T(ms)*	*P*
7.3	all like first case but: luDescr.profile= ULLA_PROFILE_MAX	ULLA_ERROR_UNSUPPORTED_PROFILE		□

Case Test with luDescr = NULL

	Parameters	*Expected Result*	*T(ms)*	*P*
7.4	luDescr = NULL	ULLA_ERROR_INVALID_PARAMETER		□

Case Test with auth = NULL

7.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	auth = NULL	ULLA_ERROR_INVALID_PARAMETER		□

Case Test where some handlers in auth are NULL

7.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	See 7.1 but some handlers are NULL	ULLA_ERROR_INVALID_PARAMETER		□

Testing with authorization

Calling all methods that need authorization

LM will only authorize the registerLU others will return ULLA_AUTHORIZATION_FAILED

Case Register Link User

7.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	luDescr.name=”LU conformance” luDescr.description=”Conformance Test” luDescr.apiVersion=”1.0” luDescr.profile=ULLA_PROFILE_UNSPECIFIED luDescr. ullaExceptionHandler=excptionHandler luRole=ULLA_STANDARD_LINK_USER	ULLA_OK		□

Case Lock link for “scanAvailableLinks” command

7.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“scanAvailableLinks”;	ULLA_ERROR_CMD_NOT_ALLOWED		□

Case Execute “scanAvailableLinks” command

7.9	*Parameters*	*Expected Result*	*T(ms)*	*P*

	cmddescr.id = valid Id;
	cmddescr.class=”testLinkProvider”;
	cmddescr.cmd=“scanAvailableLinks”;
	timeoutValue=5000;
		ULLA_ERROR_CMD_NOT_ALLOWED		□

Case Setting an int (using testLink. costUnit assuming it is not read-only)

7.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_KBYTE attrDescr.id=correct LinkId attrDescr.className=”testLink” attrDescr.attribute=”costUnit”	ULLA_ERROR_SETATTR_NOT_ALLOWED		□

	attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value			
Case	Request Info			
7.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT id,networkName,type,rxSignalStrength FROM testLink;"	ULLA_ERROR_QUERY_NOT_ALLOWED		□
Case	Request notification			
7.12	*Parameters*	*Expected Result*	*T(ms)*	*P*

	rndescr.count=1;
	rndescr. period=0;
	rndescr.privdata=NULL;
	rndescr.query = "SELECT id FROM testLink WHERE rxBitRate > 2000000";
	handler =notificationHandler
		ULLA_ERROR_QUERY_NOT_ALLOWED		□

Case	ullaConfigureL3 (note this call can only be done if ullaConfigureL3 is supported)			
7.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address) linked= valid id	ULLA_ERROR_CMD_NOT_ALLOWED		□
Case	ullaDeregisterLu			
7.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
		ULLA_OK deregister should also be passed to the LM		□

Test Case 8: Link Locking

This test case will start a small link user application that will set a lock on the first available link and keep this lick locked by repeatable calls to UllaPrepareCmd.

Call functions while link is already locked by another Link User

Calling functions that are influenced by locked links

Case **ullaPrepareCmd** : Lock link for "testCmd" command

8.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks";	ULLA_ERROR_ALREADY_LOCKED		□

Case **ullaDoCmd** : Execute "testCmd" command

8.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks"; timeoutValue=5000;	ULLA_ERROR_ALREADY_LOCKED		□

Case **ullaRequestCmd** : Execute "testCmd" command asynchronously

8.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class="testLinkProvider"; cmddescr.cmd="scanAvailableLinks"; handler = commandHandler;	ULLA_ERROR_ALREADY_LOCKED		□

Case **ullaConfigureL3**: Function is supported with valid destination address

8.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	dest.protocol = ULLA_L3ADDR_IPV4 dest.address="…" dest.length=strlen(dest.address) linked= valid id	ULLA_ERROR_ALREADY_LOCKED		□

Case **ullaSetAttribute** : Setting an int (using testLink. testCmdTimeDelay assuming it is not read-only)

8.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	value= ULLA_CU_KBYTE attrDescr.id=correct LinkId attrDescr.className="testLink" attrDescr.attribute="costUnit" attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t);	ULLA_ERROR_ALREADY_LOCKED		□

attrDescr.numValues=1

attrDescr.data=&value

Special Test Cases

Test Case 9: Reflection interface

This test case will only be run if all reflection interface methods are supported.

ullaGetSupportedClasses

Function call

ullaResultCode ullaGetSupportedClasses(IN Id_t lpId, OUT ULLA_STRING_t classList, INOUT ULLA_INT_t *size);

Case Link User has not registered yet

9.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Call with size = 0 to get correct size

9.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId is a valid link provider	ULLA_ERROR_NOT_ENOUGH_SPACE		□
	size = 0	correct size must be returned		

Case Correct Call

9.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = valid lpId	ULLA_OK		□
	size = size return in previous test case			

Case Invalid lpId

9.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = -1	ULLA_ERROR_UNKNOWN_ID		□
	size = size return in previous test case			

Case classList =NULL

9.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	classList =NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaGetClassAttributes

Function call

ullaResultCode ullaGetClassAttributes(IN Id_t lpId, IN ULLA_STRING_t className, OUT ULLA_STRING_t attributeList, INOUT ULLA_INT_t *size);

NOTE the testLink class will be used

Case	Link User has not registered yet			
9.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Call with size = 0 to get correct size			
9.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = a valid link provider className=”testLink” size = 0	ULLA_ERROR_NOT_ENOUGH_SPACE correct size must be returned		□
Case	Correct Call			
9.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = valid lpId className=”testLink” size = size return in previous test case	ULLA_OK		□
Case	Invalid lpId			
9.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = -1 className=”testLink” size = 100	ULLA_ERROR_UNKNOWN_ID		□
Case	Invalid class			
9.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className=”gollum” size = 100	ULLA_ERROR_INVALID_CLASS		□
Case	attributeList =NULL			
9.11	*Parameters*	*Expected Result*	*T(ms)*	*P*
	classList =NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaGetAttributeInfo

Function call

ullaResultCode ullaGetAttributeInfo(IN Id_t lpId, IN ULLA_STRING_t className, IN ULLA_STRING_t attributeName, OUT ULLA_STRING_t attributeDescription, INOUT ULLA_INT_t *size);

NOTE the testLink class and the netwrokName attribute will be used

Case	Link User has not registered yet			
9.12	*Parameters*	*Expected Result*	*T(ms)*	*P*

	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Call with size = 0 to get correct size			
9.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = a valid link provider className="testLink" attributeName="networkName" size = 0	ULLA_ERROR_NOT_ENOUGH_SPACE correct size must be returned		□
Case	Correct Call			
9.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = valid lpId className="testLink" attributeName="networkName" size = size return in previous test case	ULLA_OK		□
Case	Invalid lpId			
9.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = -1 className="testLink" attributeName="networkName" size = 100	ULLA_ERROR_UNKNOWN_ID		□
Case	Invalid class			
9.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className="gollum" attributeName="networkName" size = 100	ULLA_ERROR_INVALID_CLASS		□
Case	Invalid attribute			
9.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className="testLink" attributeName="gollum" size = 100	ULLA_ERROR_INVALID_ATTRIBUTE		□
Case	attributeDescription =NULL			
9.18	*Parameters*	*Expected Result*	*T(ms)*	*P*
	classList =NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaGetClassCommands

Function call

ullaResultCode ullaGetClassCommands(IN Id_t lpId, IN ULLA_STRING_t className, OUT ULLA_STRING_t commandList, ULLA_INT_t *size);

NOTE: the testLink class will be used

Case Link User has not registered yet

9.19	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Call with size = 0 to get correct size

9.20	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = a valid link provider className=”testLink” size = 0	ULLA_ERROR_NOT_ENOUGH_SPACE correct size must be returned		□

Case Correct Call

9.21	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = valid lpId className=”testLink” size = size return in previous test case	ULLA_OK		□

Case Invalid lpId

9.22	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = -1 className=”testLink” size = 100	ULLA_ERROR_UNKNOWN_ID		□

Case Invalid class

9.23	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className=”gollum” size = 100	ULLA_ERROR_INVALID_CLASS		□

Case commandList =NULL

9.24	*Parameters*	*Expected Result*	*T(ms)*	*P*
	classList =NULL	ULLA_ERROR_INVALID_PARAMETER		□

ullaGetCommandAttributes

Function call

ullaResultCode ullaGetCommandAttributes(IN Id_t lpId, IN ULLA_STRING_t className, IN ULLA_STRING_t commandName, OUT ULLA_STRING_t attributeList, INOUT

ULLA_INT_t *size);

NOTE the testLinkProvider class and the testCmd command will be used

Case	Link User has not registered yet			
9.25	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Call with size = 0 to get correct size			
9.26	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = a valid link provider className=”testLinkProvider” commandName=”testCmd” size = 0	ULLA_ERROR_NOT_ENOUGH_SPACE correct size must be returned		□
Case	Correct Call			
9.27	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = a valid link provider className=”testLinkProvider” commandName=”testCmd” size = size returned above	ULLA_OK		□
Case	Invalid lpId			
9.28	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = -1 className=”testLinkProvider” commandName=” testCmd” size = 100	ULLA_ERROR_UNKNOWN_ID		□
Case	Invalid class			
9.29	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className=”gollum” commandName=”testCmd” size = 100	ULLA_ERROR_INVALID_CLASS		□
Case	Invalid command			
9.30	*Parameters*	*Expected Result*	*T(ms)*	*P*
	lpId = = a valid link provider className=”testLinkProvider” commandName=”gollum” size = 100	ULLA_ERROR_CMD_NOT_VALID		□

Case	attributeList =NULL			
9.31	*Parameters*	*Expected Result*	*T(ms)*	*P*
	classList =NULL	ULLA_ERROR_INVALID_PARAMETER		□

Test Case 10 Historical Table

This test case will run only if all historical table methods are supported.

ullaCreateHistoricalTable

Function call

ullaResultCode ullaCreateHistoricalTable(IN ULLA_STRING_t tableName, IN Id_t sourceId, IN ULLA_STRING_t valueName, IN ULLA_INT_t period, IN ULLA_INT_t count);

NOTE using histTable and testLink.rxSignalStrenght as parameters

Case	Link User has not registered yet			
10.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Correct call			
10.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_OK		□
	sourceId = a valid ID			
	valueName =”testLink.testInt”			
	period=1000			
	count=50			
Case	Wrong sourceId			
10.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_ERROR_UNKNOWN_ID		□
	sourceId = -1			
	valueName =”testLink.testInt”			
	period=1000			
	count=50			
Case	Wrong valueName			
10.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_ERROR_INVALID_ATTRIBUTE		□
	sourceId = -1			
	valueName =”testLink.gollum”			
	period=1000			

	count=50			
Case	tableName=NULL			
10.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=””	ULLA_ERROR_INVALID_CLASS		□
	sourceId = a valid id			
	valueName =”testLink.testInt”			
	period=1000			
	count=50			
Case	valueName=NULL			
10.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=” histTable”	ULLA_ERROR_INVALID_ ATTRIBUTE		□
	sourceId = a valid id			
	valueName =””			
	period=1000			
	count=50			

ullaToggleStatusHistoricalTable

Function call

ullaResultCode ullaDeleteHistoricalTable(IN ULLA_STRING_t tableName);

Case	Link User has not registered yet			
10.7	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□
Case	Stop updating			
10.8	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_OK		□
	status = 0;			
Case	Start updating			
10.9	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_OK		□
	status = 1;			
Case	Wrong table name			
10.10	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName=”histTable”	ULLA_ERROR_INVALID_CLASS		□
	status = 0;			
Case	Wrong status value			
10.11	*Parameters*	*Expected Result*	*T(ms)*	*P*

	tableName="histTable"	ULLA_ERROR_INVALID_PARAMETER	□
	status = -1;		

Query the historical table

Query the Table create above and use accessor functions to access the values

Case Link User has not registered yet

10.12	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Query table

10.13	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query = "SELECT * FROM histTable"	ULLA_OK		□

Case List values

10.14	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Use accessor functions to retrieve 50 values	ULLA_OK		□

ullaDeleteHistoricalTable

Function call

ullaResultCode ullaDeleteHistoricalTable(IN ULLA_STRING_t tableName);

Case Link User has not registered yet

10.15	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Correct parameters	ULLA_ERROR_NOTREGISTERED		□

Case Cancel command that has not finished yet

10.16	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName="histTable"	ULLA_OK		□

Case Wrong table name

10.17	*Parameters*	*Expected Result*	*T(ms)*	*P*
	tableName="gollum"	ULLA_ERROR_INVALID_CLASS		□

Test Case 11: Mandatory Statistics Aggregators

The following UQL aggregators are mandatory:

- MIN
- MAX
- AVG
- COUNT
- SUM

Test with mandatory statistics aggregators

Function call

uses ullaRequestInfo to test

Case MIN

11.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query: "SELECT MIN(testInt) FROM testLink;	ULLA_OK		□

Case MAX

11.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query: "SELECT MAX(testInt) FROM testLink;	ULLA_OK		□

Case AVG

11.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query: "SELECT AVG(testInt) FROM testLink;	ULLA_OK		□

Case COUNT

11.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query: "SELECT COUNT(testInt) FROM testLink;	ULLA_OK		□

Case SUM

11.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	query: "SELECT SUM(testInt) FROM testLink;	ULLA_OK		□

Test Case 12: Optional Measurement and measurement statistics

Function call

Uses ullaSetAttribute and ullaRequestInfo to test optional measurement statistics

Case Statistics over fixed window of 10 and interval 1 with integer values

12.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	10 successive setting of testInt using ullaSetAttribute. value= x attrDescr.id=*LinkId* attrDescr.className="testLink" attrDescr.attribute="testInt" attrDescr.qualifier=	ULLA_OK for each setting. After 10 settings retrieve the average using requestInfo with the query "SELECT testInt_mean, testInt_max, testInt_min from testLink where id = *LinkId*" Check that the returned value is in fact the average, min and max of		□

	ULLA_QUAL_MEASUREMENT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	the 10 values entered. Repeat for another 10 different values.		
Case	Statistics over fixed window of 10 and interval 1 with double values			
12.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	10 successive setting of testDouble using ullaSetAttribute. value= x attrDescr.id=Channel*Id* attrDescr.className="testChannel" attrDescr.attribute="testDouble" attrDescr.qualifier= ULLA_QUAL_MEASUREMENT attrDescr.type= ULLA_TYPE_DOUBLE attrDescr.length=sizeof(ULLA_DOUBLE_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_OK for each setting. After 10 settings retrieve the average using requestInfo with the query "SELECT testDouble_mean, testDouble_max, testDouble_min from testChannel where id = *LinkId*" Check that the returned value is in fact the average, min and max of the 10 values entered. Repeat for another 10 different values.		□
Case	Statistics over sliding window of -10 and interval 1 with integer values			
12.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set the ullaMeasreCap window attribute for the testLinkProvider -10 using value = -10 attrDescr.id=*LinkProviderId* attrDescr.className="ullaMeasureCap" attrDescr.attribute=" testLink.testInt.window " attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value 10 successive setting of testInt using ullaSetAttribute. value= x	ULLA_OK for each setting. After each setting of testInt retrieve the average using requestInfo with the query "SELECT testInt_mean, testInt_max, testInt_min from testLink where id = *LinkId*" Check that the returned value is in fact the average, min and max of the previous values entered. Repeat for another 10 different values checking that the sliding window average is correct.		□

attrDescr.id=*LinkId*

attrDescr.className=”testLink”

attrDescr.attribute=”testInt”

attrDescr.qualifier=
ULLA_QUAL_MEASUREMENT

attrDescr.type= ULLA_TYPE_INT

attrDescr.length=sizeof(ULLA_INT_t);

attrDescr.numValues=1

attrDescr.data=&value

Case Statistics over sliding window of -10 and interval 1 with double values

12.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set the ullaMeasreCap window attribute for the testLinkProvider -10 using value = -10 attrDescr.id=*LinkProviderId* attrDescr.className=”ullaMeasureCap” attrDescr.attribute=” testChannel.testDouble.window ” attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_OK for each setting. After each setting of testDouble retrieve the average using requestInfo with the query “SELECT testDouble_mean, testDouble_max, testDouble_min from testChannel where id = *LinkId*” Check that the returned value is in fact the average, min and max of the previous values entered. Repeat for another 10 different values checking that the sliding window average is correct.		□
	10 successive setting of testDouble using ullaSetAttribute. value= x attrDescr.id=Channel*Id* attrDescr.className=”testChannel” attrDescr.attribute=”testDouble” attrDescr.qualifier= ULLA_QUAL_MEASUREMENT attrDescr.type= ULLA_TYPE_DOUBLE attrDescr.length=sizeof(ULLA_DOUBLE_t); attrDescr.numValues=1 attrDescr.data=&value			

Case Invalid window size

12.5	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set the ullaMeasreCap window attribute for the testLinkProvider to 100 using value = 100 attrDescr.id=*LinkProviderId* attrDescr.className="ullaMeasureCap" attrDescr.attribute=" testLink.testInt.window " attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_ INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_INVALID_VALUE		□

Case Invalid interval size

12.6	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set the ullaMeasreCap interval attribute for the testLinkProvider to 100 using value = 100 attrDescr.id=*LinkProviderId* attrDescr.className="ullaMeasureCap" attrDescr.attribute=" testLink.testInt.interval " attrDescr.qualifier= ULLA_QUAL_EXACT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_ERROR_INVALID_VALUE		□

Test Cases 13: Performance Evaluation

Request Notification Latency

Function call

Sets a notification request for changes to the testLink table and then calls a command to change an attribute in the table. The time is measured from this point to when a notification is received by the link user regarding this change.

Case Notification latency

13.1	*Parameters*	*Expected Result*	*T(ms)*	*P*
	2000 successive alterations of testInt using ullaSetAttribute. value= x attrDescr.id=*LinkId* attrDescr.className="testLink" attrDescr.attribute="testInt" attrDescr.qualifier= ULLA_QUAL_MEASUREMENT attrDescr.type= ULLA_TYPE_INT attrDescr.length=sizeof(ULLA_INT_t); attrDescr.numValues=1 attrDescr.data=&value	ULLA_OK for each setting. For each iteration, measure the time from attribute alteration to reception of notification with a suitable (50ms) wait in between each successive iteration. The results are stored in pdf or cdf format. Total test time is 100s. Each test is repeated for a range of registered notifications in the core (1,2,3,4,5 notifications).		□

Query Latency

Function call

Calls requestInfo() on the testLink table. duplicateLink() is called to increase the number of registered links (see description of test below).

Case Query latency

13.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	2000 successive requestInfo() calls Query="SELECT id from testlink" ... ... Query="SELECT * from testlink"	ULLA_OK for each call. For each iteration, measure the time from query call to reception of relevant data, with a suitable (50ms) wait in between each successive iteration. The results are stored in pdf or cdf format. Total test time is 100s.		□

		Each test is repeated for a range of requested columns of data, i.e. include the first x columns of data from the testLink specification, where x=(1,5,15,*). Each of these tests is then also repeated for a range of links registered in the testLink table (1,2,3,4,5 links).		

Test Cases 14: Additional test cases

Additional notification test

Case Event notification with correct query and RnDescr_t with changing attributes with ullaSetAttribute

	Parameters	*Expected Result*	*T(ms)*	*P*

14.1	Query = "SELECT id FROM testLink WHERE testInt = 1 and testDouble = 1.1 and testString = 'a1' and testRawData = 'a';"
	
	rndescr.count=10;
	
	rndescr. period=0;
	
	rndescr.privdata=data;
	
	handler =notificationHandler;
		ULLA_OK,
		
		Data should be accessible in the handler.
		
		The handler should then be triggered again by changing the trigger attributes. The handler should be invoked after every other call is made to ullaSetAttribute with argument *value* alternating between 0 and 1 (starting with 0). For example
		
		attriDescr.id = *testLinkId*;
		
		attrDescr.attribute = "testInt";
		
		attrDescr.data = &*value*;
		
		attrDescr.numValues=1
		
		ullaSetAttribute(&attrDescr);
		
		After 10 triggers of the handler the handler is no longer called when testInt is toggled
		
		The test can be repeated for the other test attributes.
				□

Additional doCmd tests

Case Execute command that requires attributes to be set

14.2	*Parameters*	*Expected Result*	*T(ms)*	*P*
	cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“ testCmd”; timeoutValue=5000;	ULLA_ERROR_ATTRIBUTE_NOT_SET		□

Case Execute command that will timeout

14.3	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set attribute testCmdTimeDelay to 1000 ms cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“ testCmd”; timeoutValue=100;	ULLA_ERROR_TIMEOUT after 100ms Now to check that the command was actually terminated properly. The state (testCmdState) attribute should have gone from 0 to 1 and now beck to 0 and this should be checked.		□

Case Execute command that will not timeout

14.4	*Parameters*	*Expected Result*	*T(ms)*	*P*
	Set attribute testCmdTimeDelay to 1000 ms cmddescr.id = id found in the query; cmddescr.class=”testLinkProvider”; cmddescr.cmd=“ testCmd”; timeoutValue=5000;	ULLA_OK after 1000ms Now to check that the command was actually completed properly. The state (testCmdState) attribute should have gone from 0 to 1 and now to 2 and this should be checked.		□

Appendix C Additional Test cases to validate other non-functional requirements

The following test cases describe the validation process for rest of the ULLA requirements not directly covered by the test suite (or test cases) described in Appendix B.

Test case	**Extensibility**
Features to be tested	Ability to support new Link/Link Provider classes though the use of new classes or extending existing classes
Testing approach	The use of testLink, testLinkProvider and optionally testChannel classes in the API conformance test suite to demonstrate that ULLA supports extensibility for new Link technologies. (Alternatively, the fact that one implementation supports more than one type of link classes (i.e. 802.11, cellular etc) also demonstrates the ability to extend support for new link technologies)
Pass/Fail criteria	Pass when one or more of the test cases using TestLink or TestLinkProvider or testChannel class pass their Pass criteria.

Test case	**Scalability**
Features to be tested	Ability to scale: 1) in terms of the number of LUs or LPs registered with the Core 2) in terms of the size of data records or data storage 3) in terms of the size of implementation (by adjusting the functional capability) to fit into different hardware platforms with varying capabilities
Testing approach	1) See performance testing, section 5. 2) See performance testing, section 5. 3) By demonstration: ULLA is implemented and tested on four different hardware platforms running different operating systems with various capabilities
Pass/Fail criteria	1) Linear increases in delay times according to query complexity? 2) Linear increases in delay times according to query complexity? 3) Pass when test suite pass on more than one of four test platforms used. Scalability is a difficult property to quantify in as much as there isn't a particular metric available to assess it in numerical form. Demonstration of ULLA on a number of platforms with heterogeneous capabilities proves that this kind of scalability is present.

Test case	**Recovery**
Features to be tested	Ability to recover after a failure of one or more components of ULLA 1) system failure 2) LU crash 3) LP crash 4) Data storage service crash 5) Data storage corrupt or inconsistent
Testing approach	Such features will not be tested at present as the correct recovery behaviour for each of the situations listed above has not been determined and the decision on suitable behavior for a particular system has been left for implementers.
Pass/Fail criteria	-

Test case	**Stress and Volume**
Features to be tested	Ability to handle 1) Large number of queries 2) Repeated LU and LP registrations/deregistrations 3) Large number of LP events and associated notifications
Testing approach	See performance testing, section 5.
Pass/Fail criteria	1) Robust handling of a large number of queries through a queuing mechanism if necessary – the Core should not crash when bombarded with such requests 2) The data store should be maintained in a clean state with such repeated registrations/deregistrations with no old and useless information being kept that could disrupt normal operation 3) A large number of stored notification requests should not adversely affect the notification latency to a large degree. This is explored more fully in the performance test section.

Test case	**Portability**
Features to be tested	Ability to use the same ULLA API header files in different device platforms
Testing approach	By demonstration & inspection: All four test implementations use the same ULLA API header files written in ANSI-C.
Pass/Fail criteria	Pass when same header files are used in each test platform

Test case	**Interoperability**
Features to be tested	Ability of a certain LU to work with different ULLA implementations independently of the hardware and OS platform
Testing approach	By demonstration: Same test program will be used on all four test environments each of which made up of different hardware and software combinations.
Pass/Fail criteria	Pass when common test program runs on each test platform.

Appendix D Results from testing suite runs

Test Case	Windows CE kernel based ULLA (Connection Management Platform)			Linux User space ULLA (based on Postgres) (Multimedia Streaming Platform)			
	PASS/ FAIL	Time (ms)	Result Code	PASS/ FAIL	Time (ms)		Result Code
					Laptop	Triton	
1.1	PASS	3	ULLA_ERROR_API_VERSION_MISMATCH	FAIL	0	2	ULLA_ERROR_FAILED
1.2	PASS	3960	ULLA_ERROR_ROLE_DENIED	FAIL	0	0	ULLA_ERROR_UNSUPPORTED_ROLE
1.3	PASS	2002	ULLA_ERROR_UNSUPPORTED_ROLE	PASS	0	0	ULLA_ERROR_UNSUPPORTED_ROLE
1.4	PASS	2003	ULLA_ERROR_UNSUPPORTED_PROFILE	FAIL	0	1	ULLA_ERROR_FAILED
1.5	PASS	0	ULLA_ERROR_INVALID_PARAMETER	FAIL	0	1	ULLA_ERROR_FAILED
1.6	PASS	2012	ULLA_OK	PASS	36	168	ULLA_OK
1.7	PASS	0	ULLA_ERROR_ALREADY_REGISTERED	PASS	0	0	ULLA_ERROR_ALREADY_REGISTERED
1.8	PASS	10839	ULLA_OK	PASS	30	127	ULLA_OK
1.9	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.1	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.2	PASS	58	ULLA_OK	PASS	3	82	ULLA_OK
2.3	PASS	36	ULLA_ERROR_INVALID_CLASS	FAIL	1	17	ULLA_ERROR_SYNTAX_ERROR
2.4	PASS	36	ULLA_ERROR_INVALID_ATTRIBUTE	FAIL	5	16	ULLA_ERROR_SYNTAX_ERROR
2.5	PASS	31	ULLA_ERROR_SYNTAX_ERROR	PASS	4	16	ULLA_ERROR_SYNTAX_ERROR
2.6	PASS	0	ULLA_ERROR_INVALID_PARAMETER	FAIL	0	0	ULLA_ERROR_NOTREGISTERED
2.7	PASS	4	ULLA_OK	PASS	3	66	ULLA_OK
2.8	PASS	2	ULLA_OK	PASS	1	31	ULLA_OK
2.9	PASS	3	ULLA_OK	PASS	1	23	ULLA_OK
2.10	PASS	3	ULLA_OK	PASS	0	22	ULLA_OK
2.11	PASS	168	Queries per seconds: 297.6	PASS	25	1059	Queries per seconds: 2000 and 47.2
2.12	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.13	PASS	1	ULLA_OK	PASS	0	0	ULLA_OK
2.14	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.15	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.16	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.17	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.18	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.19	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.20	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.21	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.22	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.23	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.24	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.25	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.26	PASS	0	ULLA_ERROR_NO_MORE_TUPLES	FAIL	0	0	ULLA_OK
2.27	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.28	PASS	0	ULLA_ERROR_BUFFER_TOO_SMALL	FAIL	0	0	ULLA_OK
2.29	PASS	0	ULLA_ERROR_BUFFER_TOO_SMALL	FAIL	0	1	ULLA_OK
2.30	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK

2.31	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.32	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.33	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.34	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.35	PASS		TEST REMOVED, obselete	PASS	(null)	(null)	TEST REMOVED, obselete
2.36	PASS		TEST REMOVED, obselete	PASS	(null)	(null)	TEST REMOVED, obselete
2.37	PASS	1	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.38	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.39	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.40	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.41	PASS	0	ULLA_ERROR_INVALID_PARAMETER	FAIL	0	0	ULLA_OK
2.42	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.43	PASS		TEST REMOVED, obselete	PASS	(null)	(null)	TEST REMOVED, obselete
2.44	PASS		TEST REMOVED, obselete	PASS	(null)	(null)	TEST REMOVED, obselete
2.46	FAIL	0	ULLA_OK	FAIL	0	1	ULLA_OK
2.47	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.48	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.49	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	3	1	ULLA_ERROR_INVALID_PARAMETER
2.50	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.51	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.52	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.53	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.54	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.55	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.56	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.57	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	1	ULLA_ERROR_NOTREGISTERED
2.58	PASS	2	ULLA_OK	FAIL	0	0	ULLA_OK
2.59	PASS	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.60	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.61	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.62	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.63	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.64	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.65	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.65	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.67	FAIL	0	ULLA_OK	PASS	0	0	ULLA_OK
2.68	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.69	PASS	0	ULLA_OK	PASS	0	1	ULLA_OK
2.70	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.71	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.72	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.73	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.74	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.75	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.76	FAIL	0	ULLA_OK	FAIL	0	0	ULLA_OK

2.77	FAIL	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.78	PASS	0	ULLA_OK	PASS	0	0	ULLA_OK
2.79	PASS	0	ULLA_ERROR_NOTREGISTERED	FAIL	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.80	PASS	0	ULLA_OK	PASS	1	17	ULLA_OK
2.81	PASS	0	ULLA_OK	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.82	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	PASS	0	0	ULLA_ERROR_INVALID_ULLARESULT
2.83	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.84	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.85	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.86	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.87	FAIL	0	ULLA_OK	FAIL	0	0	ULLA_OK
2.88	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.89	PASS	0	ULLA_ERROR_NO_MORE_VALUES	PASS	0	0	ULLA_ERROR_NO_MORE_VALUES
2.90	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	1	ULLA_OK
2.91	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.92	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.93	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.94	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.95	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	1	ULLA_ERROR_INVALID_PARAMETER
2.96	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.97	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.98	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	1	ULLA_ERROR_INVALID_PARAMETER
2.99	PASS	225	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once	PASS	0	6	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once
2.100	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.101	PASS	0	ULLA_ERROR_NO_MORE_VALUES	FAIL	0	33	ULLA_OK
2.102	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.103	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.104	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	1	ULLA_OK
2.105	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.106	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	1	ULLA_OK
2.107	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.108	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.109	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.110	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.111	PASS	233	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once	FAIL	0	7	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once
2.112	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.113	PASS	0	ULLA_ERROR_BUFFER_TOO_SMALL	FAIL	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
2.114	PASS	0	ULLA_ERROR_BUFFER_TOO_SMALL	FAIL	0	0	ULLA_OK
2.115	PASS	0	ULLA_ERROR_NO_MORE_VALUES	FAIL	0	0	ULLA_ERROR_NO_MORE_VALUES
2.116	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	1	ULLA_OK
2.117	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.118	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.119	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.120	PASS	1	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK

2.121	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	1	ULLA_ERROR_INVALID_PARAMETER
2.122	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.123	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.124	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.125	PASS	221	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once	PASS	0	5	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once
2.126	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
2.127	PASS	0	ULLA_ERROR_BUFFER_TOO_SMALL	FAIL	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
2.128	PASS	0	ULLA_ERROR_NO_MORE_VALUES	PASS	0	0	ULLA_ERROR_NO_MORE_VALUES
2.129	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
2.130	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_OK
2.131	PASS	0	ULLA_ERROR_TYPE_MISMATCH	FAIL	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
2.132	PASS	0	ULLA_ERROR_INVALID_ULLARESULT	FAIL	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.133	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_OK
2.134	PASS	0	ULLA_ERROR_INVALID_FIELD	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.135	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
2.136	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	PASS	0	0	ULLA_ERROR_NO_CURRENT_TUPLE
2.137	PASS	0	ULLA_ERROR_NO_CURRENT_TUPLE	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
2.138	PASS	228	ULLA_ERROR_NO_MORE_VALUES	PASS	0	5	ULLA_OK 8 times ULLA_ERROR_NO_MORE_VALUES once
3.1	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
3.2	PASS	3	ULLA_OK	PASS	9	144	ULLA_OK
3.3	PASS	0	ULLA_ERROR_INVALID_CLASS	FAIL	5	16	ULLA_ERROR_SYNTAX_ERROR
3.4	PASS	2	ULLA_ERROR_INVALID_ATTRIBUTE	FAIL	4	15	ULLA_ERROR_SYNTAX_ERROR
3.5	PASS	0	ULLA_ERROR_SYNTAX_ERROR	PASS	4	15	ULLA_ERROR_SYNTAX_ERROR
3.6	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
3.7	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
3.8	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
3.9	PASS	2	ULLA_OK	PASS	6	59	ULLA_OK
3.10	PASS	2	ULLA_OK	PASS	16	57	ULLA_OK
3.11	PASS	1	ULLA_OK	FAIL	9	59	ULLA_OK
3.12	PASS	0	ULLA_ERROR_PERIOD_TOO_SHORT	FAIL	11	58	ULLA_OK
3.13	PASS	0	ULLA_ERROR_INVALID_HANDLER	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
3.14	PASS	6	ULLA_OK	FAIL	7	68	ULLA_ERROR_NOTREGISTERED
3.15	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_OK
3.16	PASS	1	ULLA_OK	PASS	13	51	ULLA_OK
3.17	PASS	0	ULLA_OK	PASS	13	52	ULLA_OK
3.18	PASS	0	ULLA_OK	PASS	13	53	ULLA_OK
3.19	FAIL	0	ULLA_OK	FAIL	12	29	ULLA_ERROR_SYNTAX_ERROR
4.1	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
4.2	PASS	2	ULLA_OK	FAIL	0	0	ULLA_AUTHORIZATION_OK
4.3	PASS	0	ULLA_ERROR_UNKNOWN_ID	FAIL	0	1	ULLA_AUTHORIZATION_OK
4.4	PASS	2	ULLA_ERROR_INVALID_COMMAND	FAIL	0	1	ULLA_AUTHORIZATION_OK
4.5	PASS	0	ULLA_ERROR_INVALID_CLASS	FAIL	0	0	ULLA_AUTHORIZATION_OK
4.6	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
4.7	PASS	0	ULLA_ERROR_INVALID_PARAMETER	FAIL	0	0	ULLA_AUTHORIZATION_OK

4.8	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
4.9	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
4.10	PASS	101	ULLA_OK	PASS	31	92	ULLA_OK
4.11	PASS	0	ULLA_ERROR_UNKNOWN_ID	FAIL	7	23	ULLA_ERROR_INVALID_PARAMETER
4.12	PASS	1	ULLA_ERROR_INVALID_COMMAND	FAIL	4	22	ULLA_OK
4.13	PASS	1	ULLA_ERROR_INVALID_CLASS	FAIL	25	23	ULLA_OK
4.14	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
4.15	PASS	1	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
4.16	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
5.1	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
5.2	PASS	0	ULLA_OK	FAIL	7	48	ULLA_ERROR_INVALID_COMMAND
5.3	PASS	0	ULLA_ERROR_UNKNOWN_ID	FAIL	11	22	ULLA_ERROR_SYNTAX_ERROR
5.4	PASS	1	ULLA_ERROR_INVALID_COMMAND	PASS	8	21	ULLA_ERROR_INVALID_COMMAND
5.5	PASS	0	ULLA_ERROR_INVALID_CLASS	FAIL	7	20	ULLA_ERROR_INVALID_COMMAND
5.6	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
5.7	PASS	0	ULLA_ERROR_INVALID_PARAMETER	FAIL	11	21	ULLA_ERROR_INVALID_COMMAND
5.8	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
5.9	PASS	0	ULLA_ERROR_INVALID_HANDLER	FAIL	0	0	ULLA_ERROR_INVALID_PARAMETER
5.11	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
5.12	PASS	1	ULLA_OK	FAIL	0	0	ULLA_ERROR_INVALID_COMMAND
5.13	PASS	0	ULLA_OK	FAIL	0	0	ULLA_ERROR_INVALID_COMMAND
5.14	FAIL	0	ULLA_OK	PASS	0	0	ULLA_ERROR_INVALID_COMMAND
6.1	FAIL	0	ULLA_ERROR_UNSUPPORTED_FEATURE	PASS	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
6.2	FAIL	0	ULLA_ERROR_UNSUPPORTED_FEATURE	FAIL	0	0	ULLA_ERROR_BUFFER_TOO_SMALL
6.3	FAIL	0	ULLA_ERROR_UNSUPPORTED_FEATURE	FAIL	0	0	ULLA_OK
6.4	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
6.5	PASS	0	ULLA_ERROR_INVALID_PARAMETER	PASS	0	0	ULLA_ERROR_INVALID_PARAMETER
6.24	PASS	0	ULLA_ERROR_NOTREGISTERED	PASS	0	0	ULLA_ERROR_NOTREGISTERED
6.25	PASS	0	ULLA_OK	FAIL	5	91	ULLA_ERROR_NO_MORE_VALUES
13.1	PASS	13,503	ULLA_OK	PASS	10,072	19,615	ULLA_OK

The test results in the table below were obtained from the Real-time Audio platform using the same test application as the results above. The development of this platform was more focused on the driver/LLA integration and less so on the ULLA core itself to save work duplication and to explore other parts of the design. Therefore not all of the test cases were run; the result is an incomplete table compared to above.

Test Case	Linux embedded (Nomadik processor) (Real-time Audio Platform)		
	PASS/ FAIL	Time (ms)	Result Code
1.1	PASS	0 ms	ULLA_ERROR_API_VERSION_MISMATCH
1.2	PASS	29 ms	ULLA_ERROR_ROLE_DENIED

1.3	PASS	0 ms	ULLA_ERROR_UNSUPPORTED_ROLE
1.4	PASS	0 ms	ULLA_ERROR_UNSUPPORTED_PROFILE
1.5	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
1.6	PASS	34 ms	ULLA_OK
1.7	PASS	0 ms	ULLA_ERROR_ALREADY_REGISTERED
1.8	PASS	29 ms	ULLA_OK
1.9	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.1	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.2	PASS	1 ms	ULLA_OK
2.3	PASS	0 ms	ULLA_ERROR_INVALID_CLASS
2.4	PASS	0 ms	ULLA_ERROR_INVALID_ATTRIBUTE
2.5	PASS	1 ms	ULLA_ERROR_SYNTAX_ERROR
2.6	PASS	1 ms	ULLA_ERROR_INVALID_PARAMETER
2.7	PASS	0 ms	ULLA_OK
2.8	PASS	0 ms	ULLA_OK
2.9	PASS	0 ms	ULLA_OK
2.1	PASS	0 ms	ULLA_OK
2.11	PASS	30 ms	Queries per seconds: 1666.666667
2.12	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.13	PASS	0 ms	ULLA_OK
2.14	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.15	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.16	PASS	0 ms	ULLA_OK
2.17	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.18	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.19	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.2	PASS	0 ms	ULLA_OK
2.21	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.22	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.23	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.24	PASS	0 ms	ULLA_OK
2.25	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.26	PASS	0 ms	ULLA_ERROR_NO_MORE_TUPLES
2.27	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.28	PASS	0 ms	ULLA_ERROR_BUFFER_TOO_SMALL
2.29	PASS	0 ms	ULLA_ERROR_BUFFER_TOO_SMALL
2.3	PASS	0 ms	ULLA_OK
2.31	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.32	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.33	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.34	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.35	PASS	-	TEST REMOVED, obselete
2.36	PASS	-	TEST REMOVED, obselete
2.37	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.38	PASS	0 ms	ULLA_OK

2.39	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.4	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.41	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.42	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.43	PASS	-	TEST REMOVED, obselete
2.44	PASS	-	TEST REMOVED, obselete
2.46	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.47	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.48	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.49	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.5	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.51	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.52	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.53	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.54	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.55	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.56	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.57	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.58	FAIL	0 ms	ULLA_OK
2.59	FAIL	0 ms	ULLA_OK
2.6	PASS	0 ms	ULLA_OK
2.61	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.62	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.63	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.64	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.65	PASS	0 ms	ULLA_ERROR_NO_CURRENT_TUPLE
2.65	FAIL	0 ms	ULLA_OK
2.67	FAIL	0 ms	ULLA_OK
2.68	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.69	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.7	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.71	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.72	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.73	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.74	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.75	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.76	PASS	0 ms	ULLA_ERROR_NO_CURRENT_TUPLE
2.77	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.78	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.79	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.8	PASS	0 ms	ULLA_OK
2.81	PASS	0 ms	ULLA_OK
2.82	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.83	PASS	0 ms	ULLA_ERROR_INVALID_FIELD
2.84	PASS	0 ms	ULLA_ERROR_INVALID_FIELD

2.85	PASS	0 ms	ULLA_ERROR_INVALID_PARAMETER
2.86	PASS	0 ms	ULLA_ERROR_NO_CURRENT_TUPLE
2.87	FAIL	0 ms	ULLA_ERROR_UNSUPPORTED_FEATURE
2.88	FAIL	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.89	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.9	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.91	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.92	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.93	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.94	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.95	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.96	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.97	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.98	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.99	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.1	FAIL	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.101	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.102	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.103	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.104	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.105	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.106	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.107	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.108	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.109	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.11	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.111	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.112	FAIL	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.113	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.114	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.115	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.116	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.117	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.118	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.119	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.12	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.121	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.122	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.123	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.124	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.125	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.126	PASS	0 ms	ULLA_ERROR_NOTREGISTERED
2.127	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.128	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.129	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE

2.13	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.131	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.132	PASS	0 ms	ULLA_ERROR_INVALID_ULLARESULT
2.133	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.134	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.135	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.136	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.137	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE
2.138	FAIL	-	ULLA_ERROR_INVALID_ATTRIBUTE