LAT Flight Software

LTX User Manual

Type:	User Manual
Version:	V2-3-1
Author:	S.Maldonado
Created:	10 April 2003
Updated:	10 February 2005
Printed:	10 February 2005

Manual for the GLAST LAT Flight Software Test Executive System

Contents

Figures

0 Preface

LTX (LAT Flight Software Test Executive System) is a tool designed to create, edit, execute, and save tests written by the LAT flight software group. The tool is implemented entirely in Python and XML and utilizes some shell scripting. The syntax of the commands and various other features are based heavily on the CMX tool.

There will be many references to LAT specific components and functionality, especially in describing the hierarchy. Newbies might want to review some high-level LAT software documentation, including the software test plan document, to be prepared. Impatient readers can skip ahead to the walkthrough to get started right away. The divisions of this document will cover the following main topics:

- Brief introduction to the tool and its uses
- Sample walkthrough using a simple test
- Definition of a hierarchical test description
- Commanding, environment, and internals
- Installation

1 Introduction

LTX is designed to provide a means of recording all information associated with a software test. This tool provides a definition of a test description, and a standard for test execution and archiving. The system can be broken down into a few major subsystems:

- Test Executive
- Test Database
- Test Analyzer

Each subsystem will play a vital role in the LAT software test lifecycle and are described in more detail below.

1.0 Test Executive

1.0.0 Function

This subsystem is the workhorse of LTX. It has an editor that can create and manipulate test descriptions and store them as xml documents. The executive also generates an interface to host and embedded systems, allowing users to control multiple systems through a single "command" script.

1.0.1 What's Implemented

Currently, all of the functionality described above exists.

1.1 Test Database

The test database is a MySQL managed repository for all test instance data. It will be used heavily throughout the software test phase for archiving data, test tracking, and later for reporting purposes. Access to the database will be provided by the executive subsystem.

1.1.0 What's Implemented

MySQL database implemented with tables storing test instance information. LTX database interface via "save" command and SQL "SELECT" type query execution.

1.2 Test Analyzer

1.2.0 Function

This will be a magic box that takes all test instance data, processes it, and returns “PASS” or “FAIL”. It will generate a test summary document, post it to the web, and polish your shoes while you wait.

1.2.1 What’s Implemented

Currently not implemented.

2 GUI Walkthrough

This walkthrough should provide a quick ‘n dirty way to get started using the LTX user interface. This section includes a number of screen captures to demonstrate functionality. Start up the interface by typing the command “ltxui” in your terminal session.

2.0 Creating A New Test Description

To create a new test description, click the ‘New’ icon or select ‘New’ from the File pull down menu. You’ll be prompted to enter some test properties like test name, package name, and location. For this example we’ll use ‘testX’ and ‘PKGX’.

Users must have an active CMX environment to use LTX. Additionally, the all LTX files should be created in a directory named LTX in the package /ptd directory.

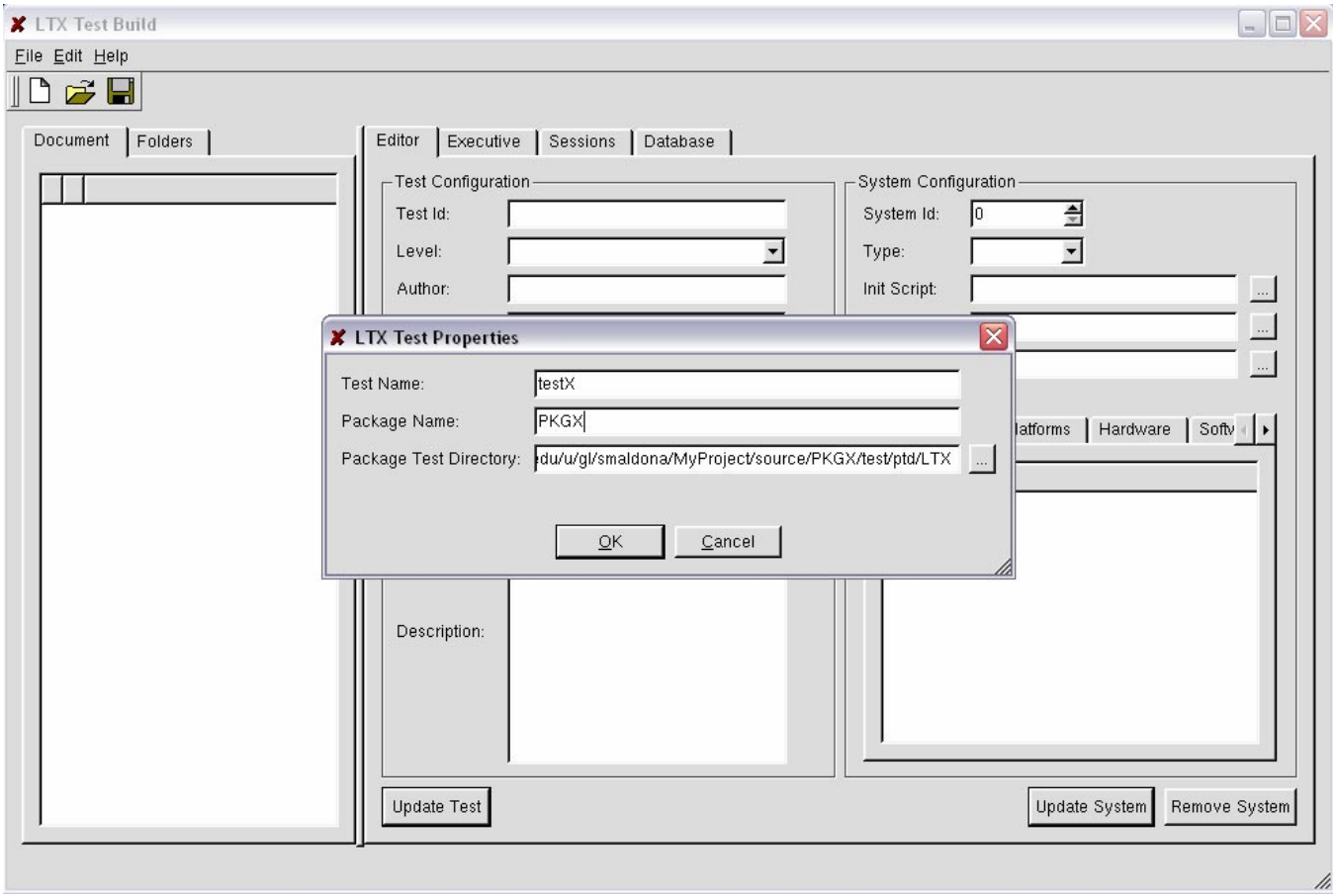

Figure 1 Creating a New Document

2.0.0 Editing Test Information

At this point you can begin editing the desired fields. Some fields are read-only that store environment and setup data. The tool buttons with labeled with '…' will open a file chooser window. After editing the test configuration, pressing the "Update Test" button will refresh the "Document" view in the leftmost tab.

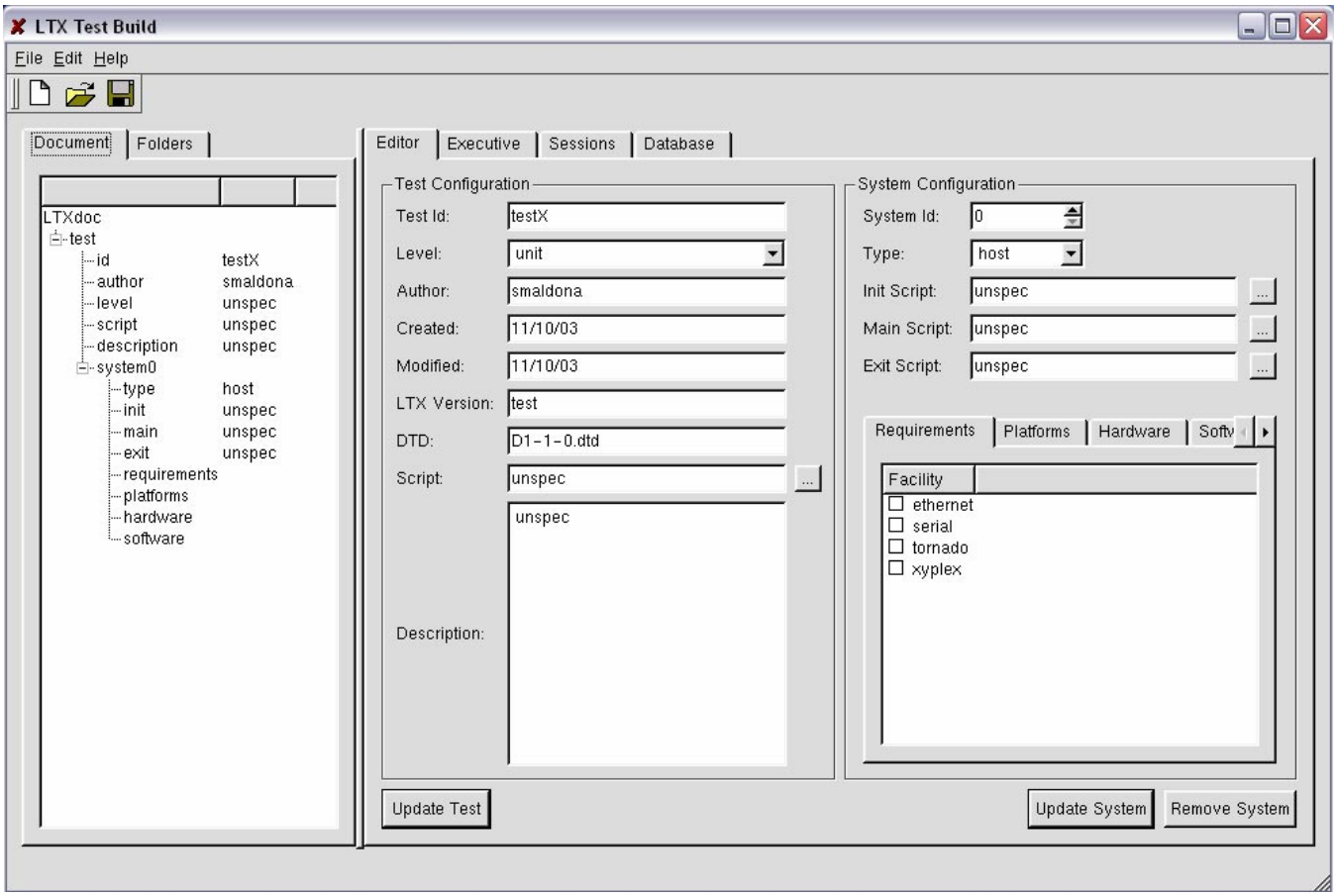

Figure 2 Editing Test Information

2.0.0.0 Editing System Information

To edit system level information, move to the “System Configuration” box. The “Id” spin box navigates through the systems. Pressing the “Update System” button will add the edited system information to the description, and will populate the Hierarchy tab accordingly. Pressing the “Remove System” button removes the currently displayed system from the hierarchy.

Note that using the update buttons will not save the test description. Press the Save icon on the menu bar or select File->Save from the menu.

2.0.0.0.1 Editing Requirements

Select the “Requirements” tab in the “Systems” box to begin editing system requirements. The system type determines what checkboxes are displayed for facilities, platforms, and peripherals. Checking boxes will update available choices according to the available systems at your site as specified in the system.db file.

2.0.0.0.2 Editing Software

Select the “Software” tab in the “Systems” box to begin editing package/constituent pairs. Move the mouse over the list box and click the right mouse button to bring up a context menu. Here you can add, remove, and order your software.

Note that the order in which the software appears in the list, is the order in which it will be loaded onto the target system.

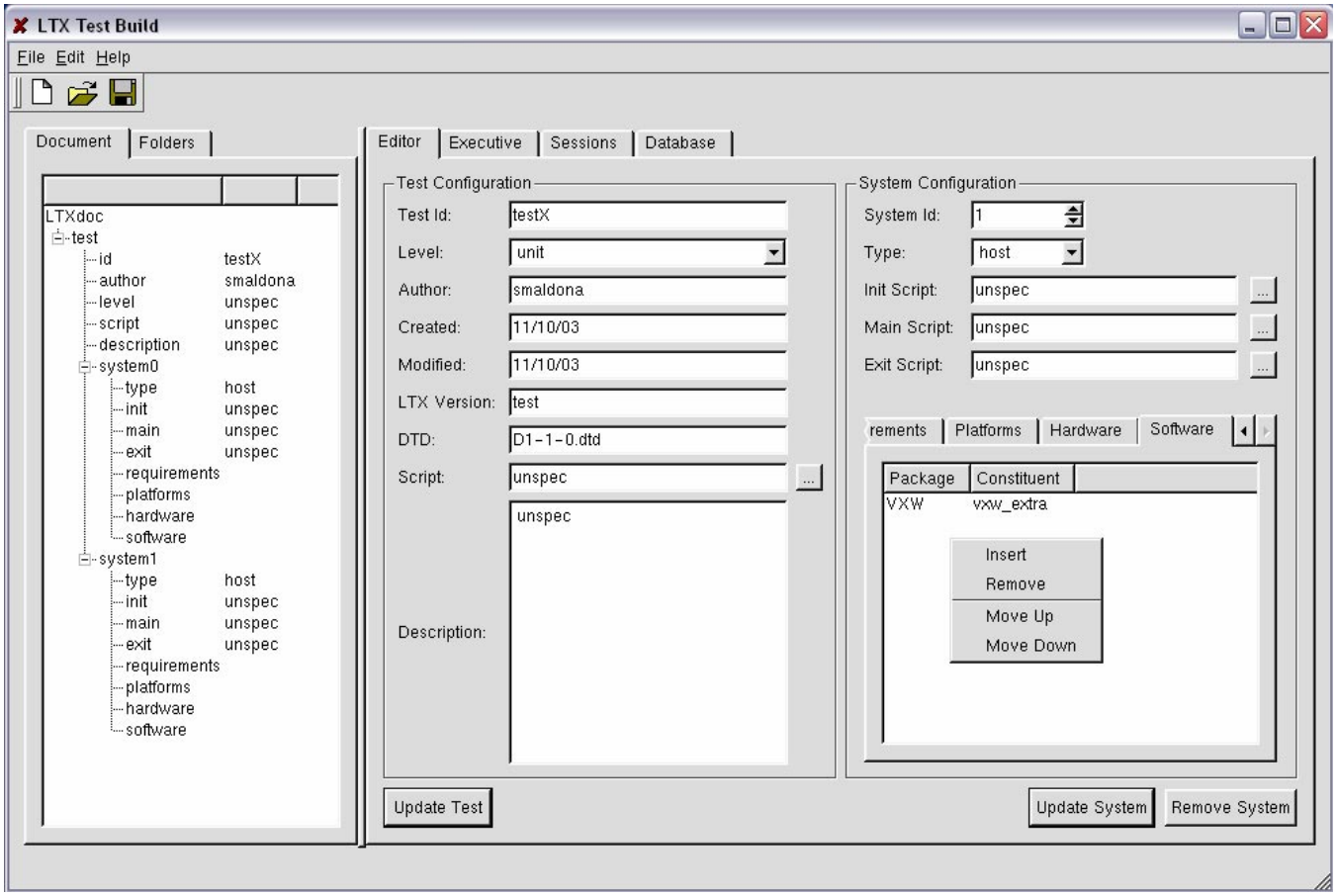

Figure 3 Editing Software

2.0.1 Saving Test Description Information

Once you have finished editing system and test information, select File->Save. The file will be saved in the location entered in the 'test properties' box when the document was created. Be sure to review section on LTX Script Interface for instructions on creating a test main script.

LTX requires that test descriptions be stored in the /ptd/LTX directory of your package. Be sure to save to that directory!

2.1 Executing Tests

2.1.0 Test Run Setup

2.1.0.0 Executive Configuration

Select the “Executive” tab from the main tab group, then the “Configuration” tab. This section sets up the options for a test run. If the test uses embedded systems, be sure to select a target communications option.

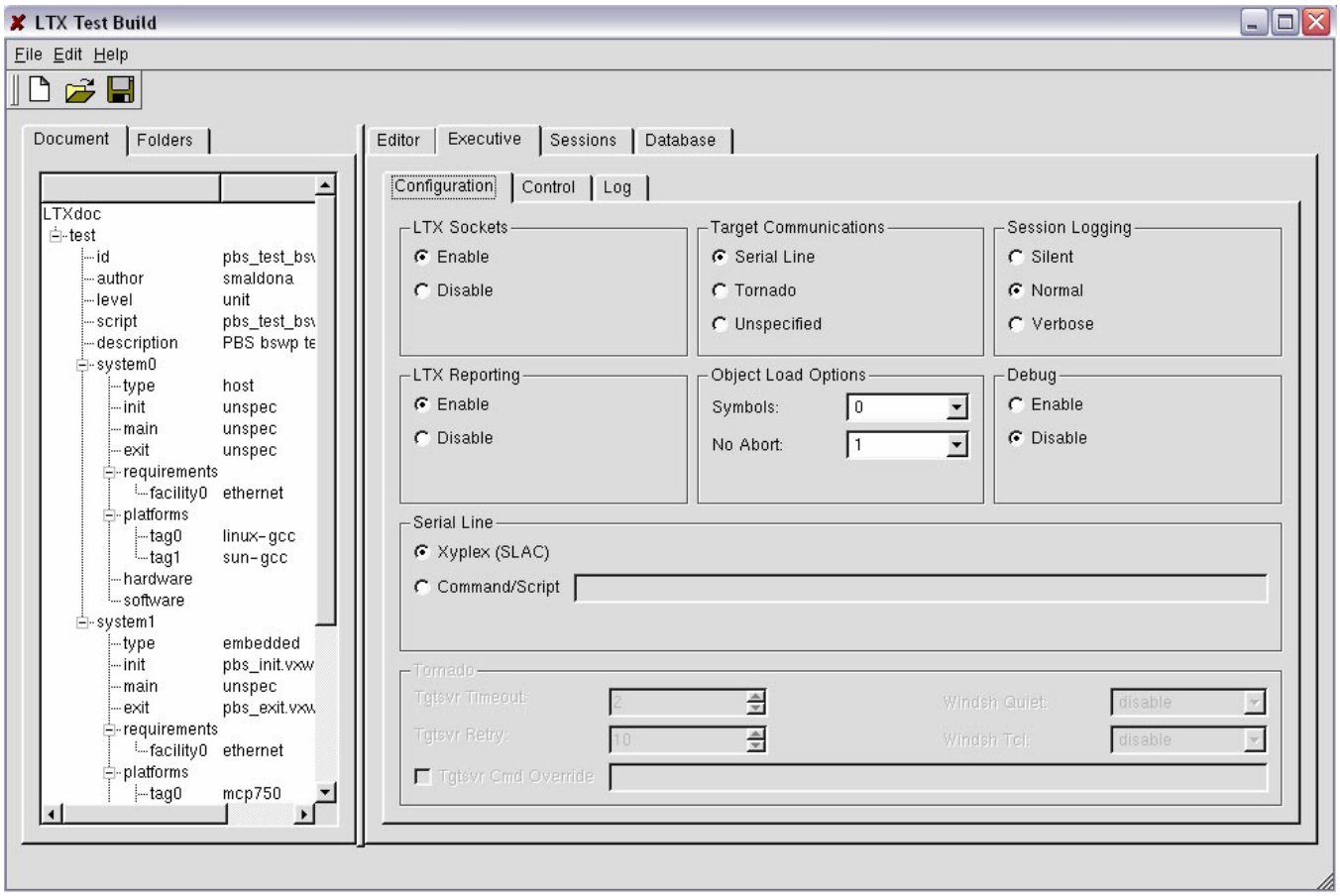

Figure 4 Executive Configuration

2.1.0.1 Platform Setup

Once the executive options have been set, select the “Control” tab. Select the desired run options for displays and scheduling. Select the desired platform setup and press “Add Setup” to queue up the test run. Right clicking over the “Batch List” box will open a context menu. Use this menu to change the run order or test, or remove a test setup.

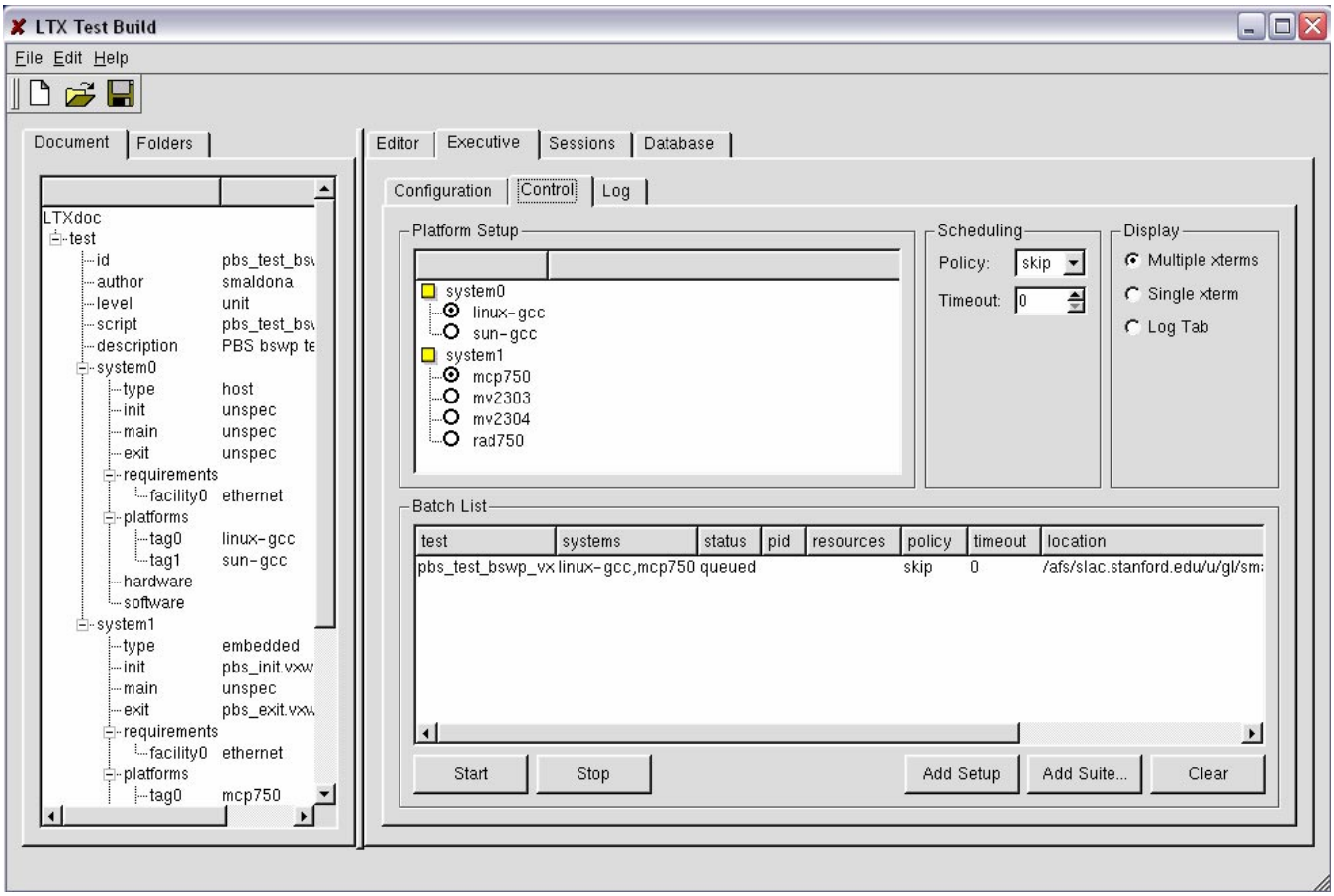

Figure 5 Run Control

2.1.0.2 Batch Suite Setup

You can load up an entire suite of tests by using the “Add Suite” button. A directory chooser window will open. Select a directory that has only LTX .xml files in it. You should be selecting the /ptd/LTX directory of your package. LTX will read each file in the directory and add all tag combinations to the “Batch List”.

2.1.1 Run Control

Once all tests setups have been added to the “Batch List” list box, press the “Start” button to begin execution. LTX executes tests in serial order, however in future releases, tests can be executed concurrently.

Once the test run has been started, you cannot alter the order of the tests, but you can queue up new tests or remove queued tests. The list can be cleared using the “Clear” button only when Run Control is inactive. Pressing the “Stop” button will halt Run Control.

Run progress and any errors will be reported in the “status” column of the Run Control list. IP names will be reported in the “resources” column reflecting systems currently in use. To kill a test, right click in the “Batch List” box, and select “kill” from the context menu.

If you are running out of your private test area, a session directory will be created in $HOME/LTX/<pkg>/<test_id>/<timestamp> containing the logs from the test run. If running a production controlled test, the session directory will be created in the official LTX repository directory residing in NFS space.

2.2 Test Instance Sessions

Once a test has completed, you can view the output and status from the test run. Select the "Sessions" tab from the main tab group. Select the "Folders" list and right click over the instance directory to view the session information. You can also press the "Open Session" button and open a file chooser. The "Sessions" tab will be populated with the test instance data recorded during the test run. Select a file in the session directory and right click to open in an editor. If the instance was a production run and the test passed, press "Save to Database" to submit the test to the test database.

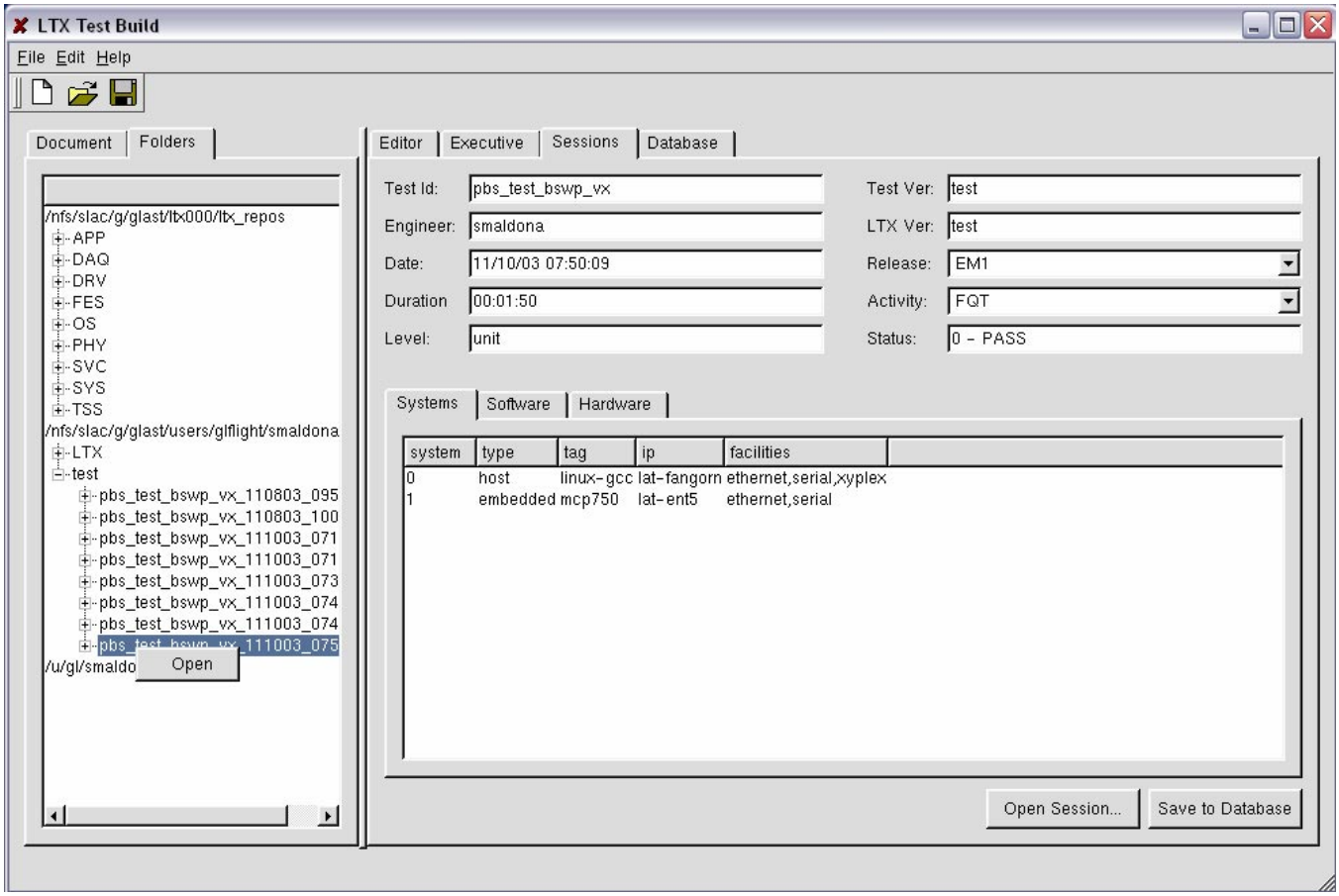

Figure 6 Session Information

2.3 Test Database

2.3.0 Connecting to the Database

To view tables or perform a query in the test database, select the "Database" tab, then the "Connection" tab and click on the "Connect" button. If you have your .my.cnf file with the proper password, you will see list of the tables in the database. Otherwise you will be prompted to enter the correct password.

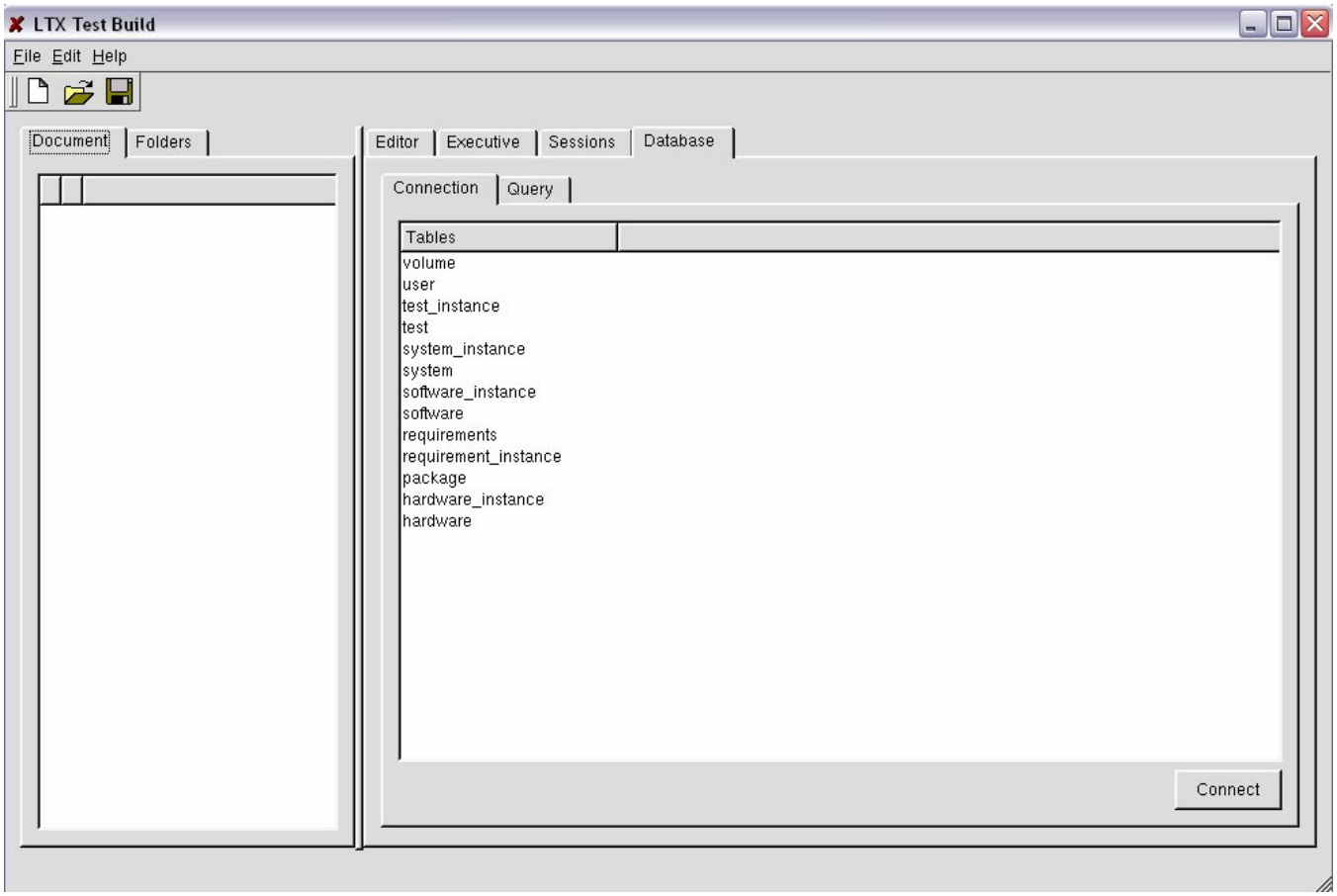

Figure 7 Database Connection

2.3.1 Querying the Database

To perform a query, select the “Query” tab and enter the SQL query string. Press the “Submit” button to view the query result set.

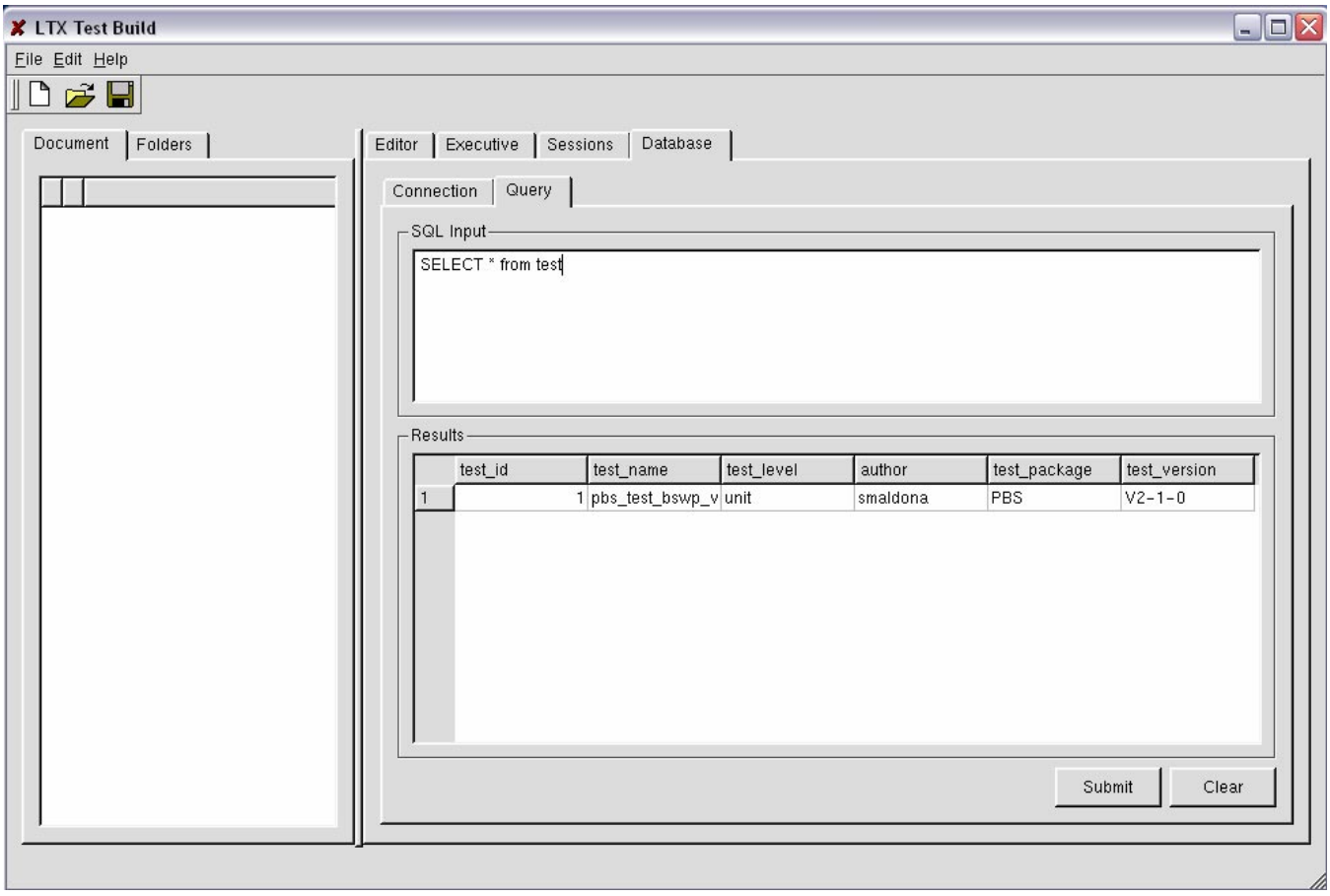

Figure 8 Performing a Database Query

3 Command Line Walkthrough

This walkthrough should provide a quick ‘n dirty way to get started using the LTX command line interface. This section includes a number of terminal session extracts shown in boxes. The format is as follows:

- Extracts are for a Linux box called `anduril`.
- The session prompt is `anduril:smaldona>`
- User input in is **`boldface`**.

Users must set their CMX environment to the appropriate LTX version

3.0 Creating A Test Description

All test descriptions are stored as xml files. LTX provides an interface to these documents via commands. The “create” command will generate a default test description containing a single default system and store it as an xml document. So, the assumption for this exercise is that the user has a local project containing a package with a /ptd directory located in it. If this structure is not in place, LTX will not allow you to create a new test description. The syntax of the “create” command is nearly identical to that of the CMX “create” command. The user must specify the package name. LTX will search the CMX environment for the package set to the ‘test’ branch. Like CMX, commands can be executed from any directory.

```
anduril:smaldona> ltx create testX PKGX
anduril:smaldona>
```

Test descriptions can only be created in a user’s local space. The user must the location of the CMX project where the package lives. A file called testX.xml should now be in the PKGX’s /ptd directory. To verify that the test was created, we can use the LTX “show” command.

```
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         platforms:
         hardware:
         software:
anduril:smaldona>
```

We see that the test description file was created, and was added to the user's current LTX environment. Any future LTX commands using testX, will reference the test description created in the /ptd directory.

The user should never, under any circumstances, hack the xml file. A hacked xml file is sure to crash LTX and cause much mayhem to the test description and/or wreak havoc during the execution phase. Additionally, this behavior is sure warrant foul looks and grimaces of disapproval.

3.1 Editing A Test Description

Now that we have created a test description, it needs to be edited. This is accomplished using the LTX "set" command. This command works as 'edit' or 'add' depending on the context of the element in the hierarchy. If it exists, it will be 'edited'. If it does not exist and is a valid element name, it will be added. A user can add/edit any of the elements contained in the test hierarchy. A detailed description of this hierarchy will be covered in later chapters. For example, let's set the "init" element of system 0.

```
anduril:smaldona> ltx set testX test/system0/init=sysinit.csh
anduril:smaldona>
```

In order to set an element, the path to that element must be specified in the command argument. The document level "LTXdoc" element does not need to be specified and cannot be altered.

- start_element/element*/terminator_element=<value | value/value | value,value… >

Now we can try editing a more complex element, like the "software" element". Software is specified as package/constituents pairs and should be comma separated.

```
anduril:smaldona> ltx set testX test/system0/software=PKGC/pkgc
anduril:smaldona>
```

This is the test description thus far.

```
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         tag: sun-gcc
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         hardware:
         software:
            pkg 0: PKGC
               con 0: pkgc
         tornado:
            common: unspec
            quiet: unspec
            retry: unspec
            symbol: unspec
            tcl: unspec
            timeout: unspec
anduril:smaldona>
```

Notice that that package “PKGC” has been given an id of 0 and constituent “pkgc” an id of 0.

Let’s try a comma separated software list:

```
anduril:smaldona> ltx set testX test/system0/software=PKGC/pkgc,PKGB/pkgb
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         platforms:
         hardware:
         software:
            pkg 0: PKGC
               con 0: pkgc
            pkg 1: PKGB
               con 0: pkgb
anduril:smaldona>
```

Here we see that the constituents were numbered in the order entered, as children of the same package node.

The order in which comma separated elements are specified on the command line is the order in which they will be stored. Consequently, for embedded systems it is also the order in which the objects will be loaded.

If we want to change the value of package 0's constituent 1 to "new_con":

```
anduril:smaldona> ltx set testX test/system0/software/pkg0/con=new_con
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         platforms:
         hardware:
         software:
            pkg 0: PKGC
               con 0: pkgc
               con 1: new_con
            pkg 1: PKGB
               con 0: pkgb
anduril:smaldona>
```

Finally, if we want to add a new constituent to “PKGX” called “PKGA”:

```
anduril:smaldona> ltx set testX test/system0/software/pkg=PKGA/pkga
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         platforms:
         hardware:
         software:
            pkg 0: PKGC
               con 0: pkgc
               con 1: new_con
            pkg 1: PKGB
               con 0: pkgb
            pkg 2: PKGA
               con 0: pkga
anduril:smaldona>
```

Notice that we did not specify a constituent number in the command line. If the element does not exist, it will be created and numbered appropriately. An attempt to set a numbered constituent that does not exist will have the same result.

The “remove” command can function as a “delete” or a “reset” depending on whether or not the element is required. Removing the <software> element will remove all <pkg> and <con> elements\, but not the <software> element itself. Removing system0 would reset system0 to the default state, but removing any other system would result in the system being deleted entirely, including the <system> element itself. Some elements cannot be removed and must be reset using the set command. See chapters on test hierarchy for required elements specifications.

```
anduril:smaldona> ltx remove testX test/system0/software
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unspec
      script: unspec
      description: unspec
      system 0:
         type: host
         init: unspec
         main: unspec
         exit: unspec
         requirements:
         platforms:
         hardware:
         software:
anduril:smaldona>
```

3.2 Executing a Test

Once a test description has been populated with valid elements, the test description is ready to be “instantiated”. LTX gathers all of the data in the test description, validates it, and proceeds to create a test instance. LTX creates a /LTX directory in the user’s home directory. Subdirectories in this directory are identified by test names. For our example, the test description was edited to look like this:

```
anduril:smaldona> ltx show testX

LTXdoc( filename=testX.xml modified=04/17/03 created=04/17/03 ):
   test:
      id: testX
      author: smaldona
      level: unit
      script: user_main_sun.py
      description: a sample test description
      system 0:
         type: host
         init: unspec
         main: my_main_script.csh
         exit: unspec
         platforms:
            tag0: sun-gcc
            tag1: linux-gcc
         requirements:
            facility 0: ethernet
         hardware:
         software:
            pkg 0: PKGC
               con 0: pkgc
            pkg 1: PKGB
               con 0: pkgb
            pkg 2: PKGA
               con 0: pkga
               con 1: link_test
anduril:smaldona>
```

A “test main” python script must be created and added to the /ptd directory and specified in the test description as part of the add/edit process. See LTX Environment and LTX Scripting Interface chapters for instructions on creating a “test main” python script. This is a sample test main script used in our example.

```
# ltx_script_template.py: A template for a user LTX script

#REQUIRED
from ltx_scriptinterface import *

# use this to sleep -- OPTIONAL
import time

def main():

    #define interface - REQUIRED
    interface = None

    #always a good idea to enclose body in try/except block
    try:
        #initialize the script interface -- REQUIRED
        interface = LTX_ScriptInterface()

        #initialize each system -- REQUIRED
        interface.start_sys(0)
        interface.start_sys(1)
        # -- or -- initialize all systems
        #interface.start_sys('all')

        #execute LTX generated initialization commands -- OPTIONAL
        interface.exec_ltx_cmds(0,'init')
        interface.exec_ltx_cmds(1,'init')

        #execute a system script
        #interface.exec_script(1,'main')

        #write a command to a host system shell
        interface.write_sys(0,'ls -l')

        #time.sleep(10)

        #read system shell output buffer
        #buf = interface.read_sys(0)

        #execute LTX generated exit commands -- OPTIONAL but recommended
        interface.exec_ltx_cmds(1,'exit')
        interface.exec_ltx_cmds(0,'exit')

        #shut down each system -- REQUIRED
        interface.stop_sys(1)
        interface.stop_sys(0)
        # -- or -- shut down all systems
        #interface.stop_sys('all')

        #close the script interface -- REQUIRED
        interface.close(0)
    except LTX_Fatal,e:
        #if a thrown exception is severe, you might want to handle it
        print 'Fatal exception in main script',e
        if interface != None:
            interface.kill_all()
```

```
        except Exception,e:
            #maybe you have an error in your script.
            print 'Fatal exception in main script',e
            if interface != None:
                interface.close()


    if __name__ == "__main__":
        main()
```

Running of the test is performed with the LTX "run" command. Executing this command will result in a test instance directory being created, where the output files, logs, and other data will be stored. A few xterm windows are launched tailing a log for each system.

```
    anduril:smaldona> ltx run testX
    anduril:smaldona>

    LTX: initializing test executive...
    LTX: using host: tersk05
    LTX: initializing host I/O...
    LTX: using host: tersk05
    LTX: executing LTX generated init cmds for system 0
    LTX: executing cmd on tersk05: cmx se br PKGC --
test=/afs/slac.stanford.edu/u/gl/smaldona/LNK/
    LTX: executing cmd on tersk05: cmx se br PKGB --
test=/afs/slac.stanford.edu/u/gl/smaldona/LNK/
    LTX: executing cmd on tersk05: cmx se br PKGA --
test=/afs/slac.stanford.edu/u/gl/smaldona/LNK/
    LTX: executing system main script: my_main_script.csh on tersk05
    LTX: executing cmd on tersk05: source
/u/gl/smaldona/MyProject/source/PKGX/test/ptd/my_main_script.csh
    LTX: executing LTX generated exit cmds for system 0
    LTX: executing cmd on tersk05: cmx stop
    LTX: executing cmd on tersk05: exit
    LTX: killing tersk05 xterm

    anduril:smaldona>
```

3.3 Summary

Basically, we have a created and executed a test that requires a single host system with Ethernet capability, uses a modest software specification, and needs no hardware. All this test does is call an executable called link_test, that could easily be called from a shell with an active and configured CMX environment. The point is that we have successfully described a test, and every execution should yield the same results every time. The test description can be recalled at any point in time, and used over and over again. We will explore more complex tests in later chapters, after a lesson on the specifics of a test description.

4 LTX Hierarchy

In order to provide for a logical description of a test and its components, LTX was designed to store test information in a hierarchical fashion. This hierarchy can be likened to a directory tree structure and thus allows for multiple layers of abstraction. The deeper one delves in the hierarchy, the more specific the description becomes. The most generic description starts with tests and systems, and can be followed by inputs, software and hardware.

4.0 Test Elements

A test is given a name or an identifier to allow for distinction between other tests. To further describe a test, we can specify the level of software testing associated with it. LTX uses unit, component, and functional levels. Additionally, we identify an author, a description blurb, and a top-level test script that serves as the launching point during execution.

Now that we can identify a test and its purpose, we need to define the object that will be used during the execution of a test. LTX classifies these objects as systems.

- ID
- Author
- Level
- Description
- Test Script
- Systems

Refer to the software test plan document for a detailed description of testing levels

4.0.0 System Elements

A system is the fundamental object that is manipulated within a test. A test can use multiple systems, and must contain at least one in order to be valid for LTX. Types of systems can be hosts, like sun or Linux boxes, or embedded systems. Systems are also characterized by their compilation tags, scripts, software, facilities and peripherals.

- System 0
    - Type
    - Initialization script
    - Main script
    - Exit script
    - Facilities
    - Platforms
    - Peripherals
    - Software
- System n …

Other items which could be included in system descriptions are input vectors and simulators.

4.0.0.0 Facilities

A system facility is a system requirement or capability that must be present in order to execute a test. A facility can be ethernet capability or availability of VxWorks/Tornado tools. Facilities are collectively named system specifications and are not required elements.

- Specifications
    - Facility 0
    - Facility n

4.0.0.1 Platforms

Platforms consist of the set of compilation tags for which the test is valid. Valid platforms can include sun-gcc, linux-gcc, mv2303, mv2304, mcp750, and rad750.

- Platforms
    - Tag 0
    - Tag n

4.0.0.2 Peripherals

Peripherals are physical system components that are required to execute a test. A LAT COMM Board, a TEM, or a GASU are all peripherals and are collectively referred to as system hardware. Hardware is not considered an LTX required element.

- Hardware
    - Peripheral 0
    - Peripheral n

4.0.0.3 Software

System software specification follows the CMX package/constituent model and is not considered required by LTX. However, a software test with no software specified will always pass, won't it?

- Software
    - Package 0
        - Constituent 0
        - Constituent n
    - Package n

All software specified will be resolved from the user's current CMX environment. If the package has not been set to a specific branch or version, LTX defaults to loading the production version.

4.1 Summary

Let's put it all together and view the hierarchy, as it exists, form top to bottom:

Test:

- ID
- Author
- Level
- Description
- Test Script
- System 0
    - Type
    - Initialization script
    - Main script
    - Exit script

- Requirements
  - Facility 0
  - Facility n
- Platforms
  - Tag 0
  - Tag n
- Hardware
  - Peripheral 0
  - Peripheral n
- Software
  - Package 0
    - Constituent 0
    - Constituent n
  - Package n

- System n…

4.2 Continuing Evolution

As the software and hardware become more mature, the test descriptions will have to evolve with them, and LTX must provide the corresponding level of functionality. The hierarchy described reflects the maturity of LTX and the software it will be immediately testing. Fortunately, this model allows for levels of specification to be as detailed or general as required, and should provide for an extremely scalable system.

5 LTX Repository

The LTX repository archives test instance data from tests executed out of production packages. The repository consists of a MySQL database and an NFS volume.

5.0 Public Directory Structure

The NFS volume provides a location to execute tests and store both temporary and permanent instance data. The directory structure of the repository is as follows:

NFS Volume

|----LTX Repository

|--------Project

|------------Package

|----------------Package Version

|--------------------Test Name

|------------------------Test Instance

When a production test is executed, the directory structure is generated and an instance directory is created labeled “MMDDYY-HHMMSS_unsaved”. All instance data will be placed in this directory, including logs and output files. When the “ltx save” command is executed using one of these directories, the “_unsaved” will be removed from the directory name and the directory and its contents will become read-only to all users. Information about the test instance is captured to the MySQL database and is described in the next section.

5.1 MySQL Database

LTX utilizes a MySQL database to track information about production tests and test instances. The database captures the following test and instance information:

- Test name
- Test level: Unit, Functional, System

- Test author
- Test package
- Test package version
- Instance date
- Instance duration
- Test engineer (tester)
- CMX Site
- LTX version
- LTX DTD version
- Test Activity: FQT, Validation
- Test Release
- Return status code
- Analysis code
- Storage volume
- Save date
- Save time

The following information is captured for each system used in a test:

- System number
- System type: host/embedded
- System tag: sun-gcc,mcp750, etc
- System ip: lat-fangorn, lat-ent5
- System requirements: ethernet, tornado support

The software setup is recorded for each system implementing the CMX asBuilt interface:

- Package
- Constituent
- Version
- Target
- Builder
- Build site
- Built date

The hardware setup is recorded for each embedded system:

- Device ID - hex code
- Chip ID

- Description
- Vendor ID - hex code
- Vendor Name

6 LTX Environment

LTX was designed to work with tests and test data that reside in a CMX package. Unit level tests will evolve with the package itself. Component and system level tests can be an empty CMX package that contains nothing more that the test and its inputs. Users can only edit tests while they are in their private working directories. Once a test has been checked into CMX, LTX can only execute it. The package must be checked out to a user's local directory in order to alter it.

6.0 Test File Location

LTX uses the /ptd/LTX directory of a package. All test files, including scripts and input vectors, are required to reside in the /ptd/LTX directory as well.

6.1 Input Files

6.1.0 XML

The primary test document used by LTX is an xml file. This xml file stores all test description information, and is validated every time it is read or edited. Users do not edit this file manually, as LTX provides the interface for this.

6.1.1 Scripts

LTX is capable of executing most types of user specified scripts, so long as it has been created as executable, and is compatible with the system it is being executed on. Scripts written in Perl, Python, and shell will be sourced in a tcsh shell. VxWorks and Tcl scripts will be directed to a Tornado windshell for execution. The user is responsible for maintaining any scripts and ensuring their validity.

6.1.1.0 The Test Main Script

This script is a user created Python source file that controls the test executive and directs the execution of a test. It is a required file and is the sole responsibility of the user once created.

For those not familiar with the Python programming language, tutorials and reference guides are freely available at the Python website.

Users can make their test main scripts as simple or advanced at they require. LTX provides a scripting interface class that allows users to call methods that will start, stop, write, and read from systems. The following is a sample test main python script that will be described in detail.

```
# ltx_script_template.py: A template for a user LTX script

#REQUIRED
from ltx_scriptinterface import *

# use this to sleep -- OPTIONAL
import time

def main():

    #define interface - REQUIRED
    interface = None

    #always a good idea to enclose body in try/except block
    try:
        #initialize the script interface -- REQUIRED
        interface = LTX_ScriptInterface()

        #initialize each system -- REQUIRED
        interface.start_sys(0)
        interface.start_sys(1)
        # -- or -- initialize all systems
        #interface.start_sys('all')

        #execute LTX generated initialization commands -- OPTIONAL
        interface.exec_ltx_cmds(0,'init')
        interface.exec_ltx_cmds(1,'init')

        #execute a system script
        #interface.exec_script(1,'main')

        #write a command to a host system shell
        interface.write_sys(0,'ls -l')

        #time.sleep(10)

        #read system shell output buffer
        #buf = interface.read_sys(0)

        #execute LTX generated exit commands -- OPTIONAL but recommended
        interface.exec_ltx_cmds(1,'exit')
        interface.exec_ltx_cmds(0,'exit')

        #shut down each system -- REQUIRED
        interface.stop_sys(1)
        interface.stop_sys(0)
        # -- or -- shut down all systems
        #interface.stop_sys('all')

        #close the script interface -- REQUIRED
        interface.close(0)
    except LTX_Fatal,e:
        #if a thrown exception is severe, you might want to handle it
        print 'Fatal exception in main script',e
        if interface != None:
            interface.kill_all()
```

```
        except Exception,e:
            #maybe you have an error in your script.
            print 'Fatal exception in main script',e
            if interface != None:
                interface.close()


    if __name__ == "__main__":
        main()
```

- # is simply a comment line. Python lines are commented with the '#' symbol.
- All test main scripts must import this LTX scripting interface class.
- The exec_ltx_cmds method instructs the executive to send LTX generated initialization commands to the specified system. These generated commands could include loading the software or performing a cmx set branch command. The first parameter denotes the system number as specified in the test description. The next parameter is a string denoting the set of commands to execute. Valid values are 'init', 'main', and exit.
- interface.exec_script() instructs the executive to run the system's main script as specified in the test description. The first parameter is the system number, the next is the script type.

For a more detailed description of the scripting interface and its methods, see chapter on the LTX Scripting Interface.

6.1.2 Input Vectors

If input vectors are used, LTX will direct a system to load them from the /ptd directory. The user is responsible for the input's compatibility with the specified system.

6.2 The Private Directory Structure

LTX creates new directories in the user's home directory to store session and instance data.

6.2.0 The LTX Directory

The /LTX directory is created the first time a user executes an LTX command. This is the top level directory that stores all relevant data. It can be deleted by the user, and will be re-created upon execution of a new LTX command. The LTX directory is created in $HOME unless the user sets the environment variable LTX_C_HOME to an alternate directory.

6.2.1 Test Instance Directories

When LTX executes a test, it creates a directory using the package name, test name, and a timestamp. All test output files and data will be kept in this directory.

6.2.1.0 Output Files

LTX creates log file for each system specified in the test description. . Host system logs are created as: ltx_host_<hostname>.log. Embedded system logs are created as: ltx_system_<system ip>.log. A session log is also created that stores all LTX output messages and error information generated during test execution. It is named ltx_session_<test_name>.log.

If a test requires the creation of output data files, LTX will attempt to place them in the instance directory as well. Since systems using ethernet generally have visibility into the file system, it is possible for user tests to allow files to be created elsewhere.

7 LTX Command

The LTX command structure was copied directly from the CMX command line syntax. Users familiar with CMX should quickly recognize the verb – parameter – qualifier structure implemented in LTX.

7.0 Syntax

LTX commands expect a verb, followed by a test identifier, plus zero or more parameters, plus zero or more qualifiers. The following are valid commands:

```
ltx run testX --nowin

ltx create testY PKGZ

ltx set testZ test/system1/tag=mv2304

ltx save testW –-status=PASS -–release=EM1

ltx query “SELECT * FROM tests WHERE date > 05/12/03”
```

The command verb is specified following the ltx command. With the exception of a few special commands, like “query”, the test id follows the command verb. The parameter(s) follow the test id, a value assignment next, and qualifiers are entered last.

```
ltx <cmd> <test name> <parameters>[=<values>] --[<qualifiers>]

ltx <cmd> <argument> --[<qualifiers>]
```

7.0.0 Parameters

Parameters can be a single value, or multiple values separated by ‘/’. Generally, the divisions represent levels in the test hierarchy. The ‘/’ means that the preceding value is the parent and the following value is a child. LTX will validate the string and report an error if the relationships do not exist in the test description hierarchy. A terminating value followed by ‘=’ indicates an assignment to that element. Some terminators do not require an assignment, such as in the remove command. Assignment values can be comma separated, and also ‘/’ separated, when specifying package/constituent pairs.

```
ltx set testZ test/system1/software=PKGA/pkga,PKGB/pkgb
```

7.0.1 Enumerated Elements

Some elements are enumerated and require a numerical id to be referenced. Systems, facilities, peripherals, packages, and constituents are all enumerated elements. They are referenced with the syntax:

- <element_name><enumerated_value>

So, to set constituent 3 of package 2 of system 0 for testX, the syntax would be:

```
ltx set testX test/system0/software/pkg3/con2=<value>
```

7.1 LTX Commands

LTX uses the CMX environment to locate the test document. Commands that alter test documents require that the test package be set to the 'test' branch. Other read-only type commands can be executed on packages in all branches.

7.1.0 ltx create

Create a new test in the /ptd directory of the specified package (which must be set to branch test). The LTX environment will now associate the test name with the created test.

Note that new tests can only be created in user private projects.

```
ltx create <test_name> <package>
```

7.1.1 ltx delete

Delete a test from the /ptd directory of the specified package. The test must be part of the current LTX environment. The xml file is deleted from the /ptd directory and the test is removed from the active environment.

```
ltx delete <test_name>
```

7.1.2 ltx help

Print a command description or command syntax. Incomplete commands (e.g. `ltx help show`) are allowed in which case the help command will print the syntax of the all commands. The help is intended to provide a quick reminder of a command's function or syntax. It is not intended to be a replacement for a manual.

```
ltx help [<command>]
```

7.1.3 ltx kill

Stop the current test run. This command will send a signal to the test process on the remote host commanding an immediate shutdown of all active systems, and end the test. Change directory to the active session directory located in $HOME/LTX/<pkg>/<session_dir>, or the appropriate production session directory, and execute the command. User must specify the --session argument to execute kill from any directory. The signal option should only be used if the kill command does not result in a proper shutdown. Sending a signal of 9 will terminate the remote

python process and could leave other child processes active. The user should kill any orphaned processes on the remote machine after sending this signal.

```
ltx kill <test_name> [--session=<session_dir>] [--signal=<signal_num>]
```

7.1.4 ltx query

Perform an SQL query on the test database. Query should be enclosed in quotes.

```
ltx query <'sql query string'>
```

7.1.5 ltx report

Prints out a report of the test instance setup. The user must cd to the test session directory before executing this command.

```
ltx report <test_name>
```

7.1.6 ltx remove

Remove or reset a test element. This command will function as a reset when the specified element is required. For instance, a system element can be removed, and all child elements are removed as well. A tornado element is required, so removing it will reset it to default values.

```
ltx remove <test_name> <test_element/element*/element_to_remove>
```

7.1.7 ltx run

Execute a test. Utilizes ssh, tcsh, tornado, and xterm. The nowin option suppresses the creation of xterm windows for each system (host and embedded) and directs all system output to the session window. Test must be part of current LTX environment. Since a test can be valid for multiple tags, the tag for each system can be driven into the command line. If tags are not specified, the first tag for each system will be used. The sockets option enables socket transactions between the host and embedded systems for the purpose of silently reporting hardware and software versioning data. The msglevel option sets the messaging level in the session log. A level of 2 is verbose, 1 is normal, and when set to 0, only error logging is performed. The debug switch redirects the session output to the calling shell. The serial switch allows for specification of a command or script that will enable communication with the serial port of a test stand. The tgtsvr option overrides the default target server options and replaces them with the values specified. The tornado option sets Tornado as the target communications interface. This argument must be enclosed in single quotes. The --cputime option will override the LTX imposed 7200 second limit on CPU time usage. This limit was built in to prevent stray processes from failed/crashed/hung tests from usurping shared system resources. Specify this option in seconds.

```
ltx run <test_name> [--nowin] [--tags=<sys0_tag,sys1_tag,…>] [--
sockets] [--msglevel=<level>] [--debug] [--serial=<'serial
cmd/script'>] [--tgtsvr=<'tgt_svr_opts'>] [--tornado] [--
cputime=<seconds>]
```

7.1.8 ltx save

Save a test instance to the repository. This command will archive test logs and any available data, and populate the test database. Additionally, a pass/fail status can be specified along with a LAT release tag.

```
ltx save <test_name> [--status=<PASS/FAIL>][--release=<release tag>]
```

7.1.9 ltx set

Set the value of a test element. This command is used to add and edit test element values. Some elements do not require a value, in which case a default element and any required children are created. For example, setting a system element with no id specified will created a new element and populate all required sub-elements. The value can be a comma delimited list, which can consist of package/constituent pairs separated by '/'.

```
ltx set <test_name> <test_element/element*/element_to_set>[=<value>]
```

7.1.10 ltx show

Display the contents of a test description to the screen.

ltx show <test_name>

7.1.11 ltx update

Perform an update analysis on the test description. This command compares the test with the current specification and reports discrepancies. If a test is not compatible with the current LTX version, the version that the test was built with is displayed, and the user has the option to revert to that LTX version, or update the test.

```
ltx update <test_name>
```

Elements no longer required will be removed. Elements required in a new specification but absent from the test will be created and populated with a default value. The old test file is saved as <test_name>.old and replaced with the updated file.

It is the user's responsibility change the default values as necessary and keep old tests up to date. The older the test, the more work required by the user to update.

8 LTX Scripting Interface

LTX provides an interface to the test executive that allows users to manipulate multiple systems through a single script.

8.0 The LTX_ScriptInterface Class

This class defines all of the methods available to the user script. The user can call any of the provided methods to perform the desired action on a system, provided that it has been properly specified in the test description.

8.0.0 Methods

8.0.0.0 start_sys

This method will initialize the specified systems. This action involves spawning a shell to the systems, creating I/O pipes to the shell, starting target servers, creating logs, and opening xterm windows.

start_sys(sysnum)

Parameters:

- sysnum – integer id of the system as specified in the test description or text string ‘all’ denoting the request for initialization of all systems in the test. With this parameter, LTX will start the systems in the order they are specified in the description.

Returns: nothing

8.0.0.1 stop_sys

Shuts down specified systems. Closes logs, kills xterms, kills target servers, exits and closes shells

stop_sys(sysnum)

Parameters:

- sysnum – integer id of the system as specified in the test description or text string ‘all’ denoting the request for shot down of all systems in the test. With this parameter, LTX will stop the systems in the reverse order in which they are specified in the description.

Returns: nothing

8.0.0.2 write_sys

Writes a command to the specified system's shell.

write_sys(sysnum, cmd, block)

Parameters:

- sysnum – integer id of the system as specified in the test.
- cmd – string value of the command to be written, must be enclosed in quotes.
- block – optional integer value of 0 or 1 specifying whether or not to immediately return after command execution. If 0, return status will be "None" and the next command will be executed without waiting for execution of previous command. If 1, the command will block until the prompt returns from the previous command. The default value is 1.

Returns: command status exit code

8.0.0.3 read_sys

Reads data from the specified system's output buffer. Once data is read, the systems buffer is cleared. If a new command is sent before the buffer was read, the buffer will be replaced with the response from the last command.

read_sys(sysnum, block)

Parameters:

- sysnum – integer id of the system as specified in the test.
- block - optional integer value of 0 or 1 specifiying whether or not to immediately return the current contents of the output buffer or to wait until a prompt is detected. If 0, the read will immediately return the current contents of the output buffer. If 1, the read will block until a prompt is detected. The default value is 1.

Returns: string buffer holding response to last command executed or last data output from a system.

8.0.0.4 exec_script

Instructs the executive to have the specified system execute a script. The script can be the init, main, or exit script specified in the test description. The user is responsible for the existence, compatibility, and integrity of said script.

exec_script(sysnum, arg)

Parameters

- sysnum – integer id of the system as specified in the test.
- arg - a string value containing the name of the script or one of the keywords 'init', 'main' or 'exit'.

Returns: nothing

8.0.0.5 exec_analysis

Instructs the executive to have the specified system execute its designated analysis script.nd

The user is responsible for the existence, compatibility, and integrity of said script.

exec_analysis(sysnum, script)

Parameters

- sysnum – integer id of the system as specified in the test.
- script - optional argument to override the designated analysis script. Calling this routine without the script argument will result in execution of the designated analysis script.

Returns: nothing

8.0.0.6 exec_ltx_cmds

Execute LTX generated command set and scripts on the specified system. LTX generated 'init' commands load software, display CMX asBuilt information, run specified initialization script and/or sets up the cmx environment. LTX 'exit' commands will run the specified exit script then request a reboot and/or stop the cmx environment. At this point, the 'main' command set has not been defined.

exec_ltx_cmds(sysnum, arg)

Parameters:

- sysnum – integer id of the system as specified in the test.
- arg - string value specifying 'init', 'main', or 'exit'.

Returns: nothing

8.0.0.7 close

Close the test executive interface. Closes logs and cleans up session. Records exit status code to denote success or failure of script execution.

close(exit_status, analysis_status)

Parameters:

- exit_status - integer id of the script exit status code
- analysis_status - integer id of the analysis code

Returns: nothing

8.0.0.8 chdir

Changes the current working directory of a system to either the session directory identified as "home" or the cmx /ptd directory, identified as "ptd".

chdir(sysnum,dir)

Parameters

- sysnum – integer id of the system as specified in the test or the string “all”. When no parameter is pass, the default is “all”.
- dir – string keyword “home” or “ptd”. When no parameter is pass, the default is “home”.

Returns: status code of chdir command

8.0.0.9 get_ip

Returns the specified system’s ip name.

get_ip(sysnum)

Parameters

- sysnum – integer id of the system as specified in the test.

Returns: string buffer containing ip name.

8.0.0.10 kill_all

Kill all active system processes and exits test run. This command should be executed as the result of a fatal exception as detected in the user’s main python script.

kill_all()

Parameters: none

Returns: nothing

9 Installing LTX

LTX was developed using open source and GPL software freely available to the public. It was developed for the Sun UNIX and Linux platforms and is not compatible on any other operating system.

9.0 Prerequisites

9.0.0 Prerequisite Enumeration

LTX employs a range of open source software products to handle databases, XML files and so on. The list prerequisite software can be divided into four parts:

9.0.0.0 Command Line Only LTX

- Python 2.2.2 or later
- PyXML version 0.8.1 or later
- Pexpect version 0.98 or later

9.0.0.1 Full GUI LTX

- Qt/X11 GPL/Non-Commercial version 3.1.1 or later – A platform independent C++ application development framework
- QScintilla version 1.53-x11-gpl or later – A port to Qt of the Scintilla C++ editor class
- SIP version x11-gpl-3.6 or later -- Python bindings for C++
- PyQt gpl version 3.6 or later – A python interface to Qt libraries
    - The path to the Qt /lib directory must be added to the installer's LD_LIBRARY_PATH environment variable before installing this module

9.0.0.2 Database Access

- MySQL 4.0.16 or later
- MySQL-python version 0.9.2 or later

- The path to the /lib/mysql directory of the MySQL installation must be added to the user's LD_LIBRARY_PATH environment variable

9.0.0.3 Optional Extras

The Eric3 Python IDE is an optional component that can be used to create and manage user generated python scripts. It provides a syntax sensitive Python source editor. All installation requirements for the LTX GUI version are required for Eric3.

9.0.1 Practical Experience With Prerequisites

Installing the LTX prerequisite components is not without its frustrations. The following was generated by going through the installation process and noting any variances, special cases and so on. The installer in the following exercise had root privilege and could install to the canonical Unix areas. For sites where this is not practical, please omit the "make install" step (or its equivalent) and do what is acceptable at your site to make these products "visible":

- Python-2.2.2

  Build with default options: "./configure" → "make" → "make install"

- PyXML-0.8.3

  Build with default options: "python setup.py build" → "python setup.py install"

- Pexpect-0.99

  Build with default options: "python setup.py build" → "python setup.py install"

- MySQL-4.0.16

  Build with default options: "./configure" → "make" → "make install"

- MySQL-Python-0.9.2

  Build with default options: "python setup.py build" → "python setup.py install"

- Qt-x11-gpl-3.2.2
  To support Hippo draw, we need to configure Qt with –thread option: "./configure –thread", which will an extra multi-threaded library.

  But if we only want to run LTX we can do without the multi-threaded option.

- QScintilla-1.54-x11-gpl-1.2

  Build with default options: follow instructions in README file.

- SIP-x11-gpl-3.8

  This package doesn't work straight out of the box on Sun. Apply the following patch to `siplib/sipQt.h`:

```
#include <Python.h>
#include <qobject.h>
#include <sip.h>
```

Figure 9 Section of code in `sipQt.h` before patch

```
#include <Python.h>
#if defined (truncate)
#  undef (truncate)
#endif
#include <qobject.h>
#include <sip.h>
```

Figure 10 Section of code in `sipQt.h` after patch

SIP can now we can build with default options: follow instructions in README file.

- PyQt-x11-gpl-3.8.1

  Build with default options: "build.py" → "make" → "make install"

9.1 LTX Build and System Database Installation

LTX is itself a CMX package and CMX exports the command line executable. Therefore, CMX must be installed and the LTX package must be available at the site.

Before attempting the installation:

- Ensure proper access to CMX and the flight software CVS code repository
- Choose a disk location for the LTX database directory `/LDB`. At SLAC, it is located in the LTX package source tree in parallel with the version directories.
- Pick a site name. This will be referred to later as `<site>`.

With all preparations in hand, the LTX package can be built using standard CMX methodology. Once the build is complete, it will be necessary to go on to preparing the LTX system database:

9.1.0 LDB Directory

Check out the LDB directory from the CVS repository to the chosen disk location. All LTX users must have the environment variable `LTX_C_LDB` created and set to the location of this directory. At SLAC, the variable is set in the group .cshrc script.

This directory should contain a `systems.db.template` file. This file needs to be edited to reflect the systems available at the site, and should be renamed `systems.db.<site>`.

9.1.0.0 Systems File

The `systems.db.<site>` file contains necessary information about host and embedded systems that LTX uses to execute tests. The format of a line in the file is:

<cmx_tag> <node_name> <type> <facilities> <peripherals> <port_numbers>

- cmx_tag – compilation tag as used by CMX
- node_name – IP address of specified system or board
- type – host or embedded are valid values
- facilities – a comma delimited list of capabilities available on a given system. LTX uses the values in the test description's <requirements> element, to find a system at the site meeting the requirements. Valid values for this field can be items like: ethernet, tornado, etc.

- peripherals – a comma delimitied list of hardware available on a system. LTX uses the values in the test description's <hardware> element, to find a system at the site meeting the requirements. Valid values for this field can be items like: LCB, TEM, etc. If no peripherals are available, then the value "null" must be specified.
- port numbers – The transmit and receive socket port numbers to be used exclusively by the embedded system. Listed in order as: tx,rx.

Here are a couple of lines from the systems.db.slac file:

```
sun-gcc  tersk*     host     ethernet,tornado
mcp750   lat-elf1   embedded ethernet,serial TEM 5051,5052
```

9.2 Environment Variables

LTX uses the following environment variables

- CMX_I_CMX - exported by CMX
- CMX_B_CMX - exported by CMX
- CMX_C_SITE - exported by CMX
- CMX_L_CONFIG - exported by CMX
- CMXCONFIG - exported by CMX
- CMX_I_LTX - exported by CMX
- CMX_B_LTX - exported by CMX
- LTX_C_DTD - exported by LTX
- USER - user environment
- HOME - user environment
- HOST or HOSTNAME - user environment
- LTX_C_HOME - user environment

Setting the LTX_C_HOME variable will change the default location of the /LTX directory where all tests instance data is stored. The default is $HOME/LTX.

9.3 Summary

Once all of the required software is installed, the user should have access to the LTX command line and/or the GUI interface.