LoCuS (Laboratory fOr CompUter Science) System is instructional software for teaching computer science theories through experiments. Experiments within the context of LoCuS are laboratory excursions akin to those found in other sciences like biology, chemistry, and physics. A locus (Latin for “place”) is a “collection of points which share a property” or “the path through which a point moves to satisfy a given condition.” The LoCuS logo is a Boy’s surface, named after its founder, Werner Boy. Boy’s surface is a particularly intricate locus which is described by a system of complex equations. Just like the Mobius strip, Boy’s surface has only a single side. Our goal is for LoCuS to define a new place to emphasize the science of computation as well as a new path for computer science, in the form of labs.
Contents

1 Introduction 1

2 How LoCuS Runs 3

3 LoCuS Development 5
  3.1 Checking Out the Project 5
  3.2 Importing the Working Directory into Eclipse 5
  3.3 Configuring the Project 6
  3.4 Building a Lab 6
  3.5 Running a Lab from within LoCuS 7
  3.6 Running an Individual Apparatus 7
  3.7 LoCuS Command Line Arguments 8
  3.8 Deploying a Lab to a Web Site 8
  3.9 External Tools 8

4 Workspace File Structure 9
  4.1 LoCuS Project Structure 9
  4.2 Apparatus Project Structure 9

5 SVN Repository 11
  5.1 Structure of the LoCuS SVN Repository 11
  5.2 Repository Structure and the Eclipse Workspace 13

6 Coder 15
  6.1 Coder Command Line Arguments 15
  6.2 Running Coder 15
  6.3 Coder Project structure 15

7 Troubleshooting 17
  7.1 Changing Settings on a Personal Computer 17
  7.2 Fixing a Lab 17
  7.3 Fixing an Apparatus 17
  7.4 Fixing the LoCuS System 17

8 Project-Specific Terms 19
Chapter 1

Introduction

Welcome to the LoCuS project. LoCuS stands for Laboratory for Computer Science. Please refer to the “Getting Started Guide for All Tau Projects” for SVN, Eclipse and Mantis documentation. This guide highlights LoCuS-specific details.

The purpose of this technical document is to guide new participants through the project setup process. In Chapter 3, we will discuss the process of obtaining the source code, building the system, plugins and labs, setting up the development environment and running the system for the first time.

The third chapter outlines the standard file structure for apparatuses (discussed in more detail in the Apparatus Developer’s Guide) and the platform.

The next chapter addresses the structure of the SVN repository. Although the process of checking out the code from SVN is automated for Unix-like systems (automation scripts are discussed in Chapter 3), Windows users will find this information useful since they will need to perform the checkout manually, and thus need to understand which SVN directories should and should not be checked out. Note that there are several external tools that can be used with LoCuS and they are featured in Section 3.9.

Chapter 6 outlines the Coder application including, describing the available options, detailing the interaction between Coder and the database, and outlining the structure of the application. Chapter 7 details basic troubleshooting procedures for various parts of the lab.

Finally, the last chapter is a short glossary of the project-specific terms.
Chapter 2

How LoCuS Runs

LoCuS is a complex system that is composed of several components allowing labs to be run as a standalone application or through a web site. For more information on running a lab as a standalone application see Section 3.5. A description of how the system operates through the web is show in Figure 2.1.

Figure 2.1: Overview of the LoCuS software structure.

In order to run a LoCuS lab through a web site, the web page must have a download link to a JNLP file. The JNLP file must include the location of launcher.jar, the name of the zip file containing the lab, and the student id so the answers to the lab can be properly recorded in the database. The “Chief Programmer’s Guide” Technical Document for LoCuS describes how to construct JNLP files.

Then Launcher, which is contained within launcher.jar, is opened and started by the JNLP file. The main class of Launcher is passed the name of the zip file containing the lab, and the student id. Launcher is responsible for downloading all other files required to start the lab, which are stored in locus.zip. Launcher creates the connection between Watcher and the lab itself so that student actions and errors can be recorded in the MySQL database. Note that both locus.jar, which is responsible for running the lab, and watcher.jar must be in the same folder as launcher.jar to be run correctly.

Once locus.zip is downloaded, Launcher starts both locus.jar and watcher.jar. The locus.jar file handles displaying and running the specified lab while watcher.jar starts an instance of Watcher,
which is responsible for handling any errors that occur and recording student answers and actions in the database. In order to do this, WATCHER receives Notifications from locus.jar and the lab. Both the lab and WATCHER send and listen for heartbeats so that if the lab unexpectedly quits the error can be reported. If WATCHER quits the program closes so that students do not continue to work on the lab while their answers cannot be saved.

In order for WATCHER to place answers into the database it uses PHP scripts to insert the information into a MySQL database. In addition, locus.jar makes use of PHP scripts in order to retrieve a student’s previous answers from a particular lab so a student’s work will be automatically saved.

In order to run the lab locus.jar is given the name of the zip file containing the appropriate lab. If this name is not given, the locality lab is used as a default. locus.jar parses the XML file lab.xml corresponding to that lab and displays the lab’s components including images, texts, and apparatuses. Images are taken from the lab’s resources folder from within the zipped lab folder, unless they are used across all LoCuS labs in which case the images will be in the LoCuS resources folder. The text for each lab is extracted directly from the XML file. Lastly, each apparatus is run from a jar file specific to that apparatus. These files can be found within the plugins folder located in the same directory as locus.jar. See Chapter 4 for more information on the file structure.
Chapter 3

LoCuS Development

3.1 Checking Out the Project

In order to check out the project, perform the following steps:

1. In a command shell, navigate to a folder to which you want to checkout the project. Upon the completion of this procedure a new folder called locus will be created in this directory. You will use the locus folder as a workspace in Eclipse.

2. Get the name of your branch from the Chief Programmer.

3. Execute the following commands:
   
   `svn export svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/trunk/Scripts/fetchLocus.sh`
   ./fetchLocus.sh username branchname
   
   where username is your department login and branchname is the name of your branch, e.g., branches/locus-branch-akvochko

   You will be prompted for a password several times. After the build process has finished, you should see a string saying BUILD SUCCESSFUL. A subdirectory called locus will be created containing Java projects.

3.2 Importing the Working Directory into Eclipse

1. Locate the working directory that was created by the fetchLocus.sh script. This will also serve as your Eclipse working directory.

2. Open Eclipse, open the File menu, under Switch Workspace select Other.

3. Select the locus folder created in step 3 and click OK.

4. Go to Window → Preferences (or Eclipse → Preferences on a Mac).

5. Under Team, select SVN. Make sure that SVN Interface option is set to SVNKit (Pure Java) and click OK.

6. Go to File → Import.

7. Under General select Existing Projects into Workspace and click Next.

8. Make sure that the Select root directory radio button is selected and click Browse.

9. Select the locus folder created in step 3.
10. Click Finish and wait for the projects to be imported into workspace.

If you prefer to check out the project manually, please read Chapter 5 of this document for guidelines. For specific instructions on how to check out projects using Eclipse, refer to section 7.2 of the Getting Started Guide for All Tau Projects.

After constructing an Eclipse Workspace as specified in Section 5.2 of this document, you will need to run the Ant script inside the locus project. To do that, in your newly created workspace in Eclipse, under locus open build.xml and use the Run button, which is the green arrow. Click Run → Run → OK. When finished, proceed to the next section.

### 3.3 Configuring the Project

After you checkout all the subfolders (subprojects) into a directory on your local machine, you will need to set up the Run Configuration in Eclipse.

![LoCuS Project in Eclipse](image)

Figure 3.1: LoCuS Project in Eclipse.

To do this, right click on the project locus, click Run As → Run Configurations. In the Main tab, set Project to locus and Main Class to edu.arizona.cs.locus.Locus.

In the Arguments tab set Working Directory to Other, click Workspace... and select locus/dist. Hit Apply, and hit Run.

You should see the following.

### 3.4 Building a Lab

The labs are located in the workspace under /locus/dist/labs. When you check out a project for the first time, this folder should contain several subfolders named as the labs that are currently available and a build.xml file. build.xml is an Ant script that makes it easy to convert lab source directories into .zip files used by LoCuS. You can execute Ant scripts in two ways: using the command line, or directly from Eclipse.
1. To execute an Ant script using Eclipse, using **Package Explorer**, navigate to **locus/dist/labs**

2. Right click on **build.xml**, select **Run as → 1 Ant Build**. The labs will be created in the same folder. To see them, right-click on the **labs** folder and click **Refresh**. Note that if you select **Ant Build ...** you may select to only run the Ant script on particular portions of the project. This is useful when only working on writing a lab or a particular apparatus because there is no need to build the entire project.

Note: to execute Ant script using command line, navigate to /locus/dist/labs and execute the following command:

```sh
ant
```

Anytime anything is changed, run **ant** to see the result of your changes. Note that several of the options within the Ant script will allow you to build a portion of the project.

### 3.5 Running a Lab from within LoCuS

Although when started from Eclipse, by default LoCuS loads the **locality** lab, you can configure Eclipse so that LoCuS loads the lab of your choice. To do this, follow these steps:

1. All the labs are currently located in /locus/dist/labs. Make sure that the lab you wish to start is in this folder.

2. For LoCuS to start the lab, you need to specify the name of the lab as a command-line argument when starting LoCuS.
   To do this, right click on the project **locus**, click **Run as → Run Configurations**. In the **Arguments** tab, set the **Program arguments** field to **name-of-the-lab.zip**. Hit **Apply**, and hit **Run**. For a complete list of command line arguments, see the next section. Ignore the popup stating that there are errors in the project. These errors are in the 3D view package.

### 3.6 Running an Individual Apparatus

An individual apparatus can be run from within a lab (see above), or as a standalone (see the “Apparatus Developers Guide” Technical Document).
3.7 LoCuS Command Line Arguments

For instructions on how to pass command-line arguments to LoCuS from Eclipse, see the previous section. Here we give only the list of possible command-line arguments for LoCuS.

Available arguments are as enumerated below.

1. **labname** (optional) - first argument. Specifies the name of the lab, for example, `socialnet.zip`. If not specified, the `locality` lab will be loaded.

2. **tokenname** (optional) - second argument. Specifies the token for submitting the lab. If not specified, lab submission would be disabled. Tokens should not be used in the development phase. When deployed, LoCuS takes in tokens generated by the teacher’s website.

3. **3d** (optional) - tells LoCuS to start in 3D mode. It should always be the last argument passed.

3.8 Deploying a Lab to a Web Site

In order to deploy a lab so that it may be tested over the web, use the following steps.

1. In the `settings.properties` file change `DEPLOYMENT_TYPE = ECLIPSE DEVELOPMENT` to `DEPLOYMENT_TYPE = JNLP DEVELOPMENT`. Note that the `settings.properties` file can be found at `/locus/dist/resources`.

2. Execute the Ant script using Eclipse, using **Package Explorer**, navigate to `locus/dist/labs`.

3. Right click on `build.xml`, select `Run as → 1 Ant Build ...`.

4. Select the target `preparezip` followed by the target for any of the labs you wish to deploy. Lastly select the target `deploy`. Note, order is important and the project will not be correctly built if this order is not used.

5. The next step depends on the type of deployment you plan to create.

   (a) If you wish to deploy a lab to the **private web site** at `www.cs.arizona.edu/projects/focal/ergalics/fieldguide/private/downloads.html` the Ant script has already placed a file `labnameV.zip` where `labname` is the name of the lab you just deployed and `V` is the version number of the lab (if there is only one version, this is omitted) into `www/projects/focal/ergalics/fieldguide/private`.
      - *Launcher.jar* will not be redeployed. If you need this file redeployed email the chief programmer.
      - If the lab that you are deploying was not previously available on the private web site, contact the chief programmer and they can update the site and create a new JNLP script for the lab as described in the “Chief Programmer Guide for LoCuS” Technical Document.

   (b) If you need a lab to be deployed on the **public web site** you must contact the chief programmer. The public site can be found at `cgi.cs.arizona.edu/projects/locus/labs`.

3.9 External Tools

LoCuS provides several external tools.

**Watcher** sends updates to the MySQL database including information about exceptions that occur, student answers, and any actions a student performs within a lab. See the “Watcher Feature Guide” Technical Document for more information.

**Coder** extracts student information from the database in order to collect study information or to generate grading reports for teachers. See Chapter [6] for more information.
Chapter 4

Workspace File Structure

The workspace (locus) contains Java projects, that is, each apparatus as a separate project, coreplugin and the LoCuS infrastructure (the locus project).

4.1 LoCuS Project Structure

Project locus contains the following project-specific directories:

files - contains files required to construct jar and jnlp files when deploying the application.

dist/labs - contains source files for labs. Every lab will have several versions, for example: Middle school (8th grade), High School (9-10th grade) 35 minute, High School 50 minute, High School 120 minute, College (freshman/sophomore) 120 minute. Each version of a single lab should be in a separate folder, for example, dist/labs/locality will contain folders: locality-middle, locality-high-50min, locality-high-35min, etc. Note that each undergraduate level lab has three versions so that survey questions could be in pseudo-random order. Each of these folders will contain a lab.xml file as well as all the resources necessary to run the lab.

dist/plugins - contains plugin .jar files.

dist/resources - contains resources, used by the apparatuses.

onejar - contains files used to build LoCuS as a single .jar file.

templib - contains the locus.jar file, used to build apparatuses via an Ant script.

4.2 Apparatus Project Structure

Each apparatus is a separate Java project, which uses classes and functions provided to it by the LoCuS platform. For details about apparatus development, please, refer to “Apparatus Developer’s Guide” Technical Document. The directory structure of each apparatus is uniform and is as follows:

src (required) - directory, containing all source files of the apparatus.

docs (required) - contains all the documentation for the apparatus.

resources (required) - contains resources used by the apparatus. The current design requires this folder even if apparatus does not use any resources.

docs/api (optional) - contains javadoc documentation for the apparatus.

dist (optional) - should be present in the working copy, but should not be committed to the repository. This is output folder for the apparatus.
lib (optional) - contains libraries used by the apparatus.

When checking out a project for LoCuS manually, it is important to check out projects locus, coreplugin and all apparatuses from the apparatuses SVN directory. See the Chief Programmer to learn if you need to also check out other project(s). Note that the apparatuses folder will not be present in Eclipse.
Chapter 5

SVN Repository

5.1 Structure of the LoCuS SVN Repository

The structure of SVN repository is presented in Figure 5.1. It should be noted that the structure of the SVN repository and the structure of the developer’s working copy are different. This difference is elaborated on in the next section.

Figure 5.1: Structure of the LoCuS SVN repository.

Level 0

The root of the repository contains 3 directories:

- **trunk**: contains the latest stable version of LoCuS. In general, this folder should not be checked out by users.
• **branches**: branches are created for every project participant. Development process always goes in the branch. Changes from every branch are integrated into trunk before every minor (or major) release, or when a particular feature has been implemented and tested.

• **tags**: contains every version of the project. Whenever a release happens, this release is tagged with an appropriate name, e.g., `locus-version-5.0`. Projects in this folder should not be checked out unless there is a necessity. Projects in this folder **must not** be modified.

**Level 1**

On this level, the repository contains a folder named `locus` in the trunk, `locus-branch-username` in branches and `locus-version-version-number` in tags. These directories hold the entire project (version of the project).

**Level 2**

All subfolders below are under the **trunk** folder. On this level, there are three subdirectories:

• **Deployable**: This subfolder contains all the files included in a release that is to be deployed.

• **LocalDocs**: This contains UA-specific documentation only. All documents are designated as nonsubmitted and local. Additionally, each document can be formal or informal and be part of either user documentation or internal documentation.

• **Scripts**: The scripts in this subfolder are used for checking out the project, building eclipse workspace and building the project.

**Level 3**

The subfolders below are under the **Deployable** folder.

• **src**: This subfolder contains Java subprojects of LoCuS.

• **docs**: This subfolder contains documentation that is released. All documents are designated as nonsubmitted and released. Additionally, each document can be formal or informal and be part of either user documentation or internal documentation.

**Level 4**

All subfolders below are under the **src** folder.

• **apparatuses**: contains Java projects for apparatuses.

• **coreplugin**: Java project used by apparatuses to communicate with the LoCuS platform.

• **utilities**: Java project which is a collection of tools to process submitted labs and export student answers.

• **locus**: the main Java project which contains the LoCuS platform source code.

**Level 5**

All subfolders are under **apparatuses** folder.

• **helloworld**: a demo apparatus which utilizes all functionality provided to apparatuses by LoCuS platform.
5.2 Repository Structure and the Eclipse Workspace

It is important to note that the structure of the SVN repository can (and, in general, will) be different from the structure of the Eclipse Workspace. The SVN repository structure is designed with the purpose of keeping every project’s repository structures under the TAU umbrella project uniform and consistent. It keeps the repository ordered and organized.

The Eclipse workspace, on the other hand, requires that every folder at the root level of a workspace is a Java project. Build scripts are developed with this in mind: the main build.xml script, for example, looks for plugins in the parent folder. This imposes several rules on the checkout procedure.

1. All Java projects, including locus, coreplugin, utilities and apparatuses: helloworld, colorgame, ... must be checked out into the same directory.

2. A user must check out every project from the apparatuses folder, as well as locus and coreplugin. Checking out utilities is optional.

3. The folder with checked out projects must be used as the Eclipse workspace.
Chapter 6

CODER

LoCuS uses an application known as CODER in order to extract information from the MySQL database. CODER is used to extract information from user studies, as well as generate grading reports for teachers. CODER can be found in the SVN repository under locus/Deployable/src/coder.

6.1 CODER Command Line Arguments

The CODER application makes use of several command line arguments for grading completed labs and extracting survey data.

1. **--isGrader (optional)** - Specifies if the CODER should be run to produce a grading report as a CSV file. This argument when used along will include all possible fields in the grading report.

2. **--survey (optional)** - Specifies that the CODER should be run to extract study information from the database. Using this argument alone will include all possible columns in the output CSV file and will use the defaults for all other values.

3. **--semester = ‘F2014’** - Specifies the semester that data will be retrieved for. If omitted, the default value is ‘F2013’ which is the oldest available survey data stored within the database.

6.2 Running CODER

CODER can be run through eclipse and will produce an output CSV file in coder/output.

Add information about running coder through the teacher’s web site and as a standalone, see Mantis Issue 1863

6.3 CODER Project structure

The source files for the CODER project are located at /locus/Deployable/coder. The coder is responsible for parsing the xml of the labs in order to extract pertinent information from each lab. This includes the correct answer for each multiple choice question in the lab, as well as the questions and apparatuses that are used in a lab. In addition the coder also extracts student information from the database in order to grade each student’s work and extract survey responses.

Figure 6.1 provides a diagram of the structure of the CODER application. Note that this figure contains three parts: a CODER, a Lab, and a Student. In CODER the coder portion contains the Main class which is responsible for handling any options that are passed, creating a database manager (an instance variable of Main) which is responsible for all database queries, and creating Student and Lab objects.

The Lab portion of CODER is responsible for parsing and storing the objects within the lab.xml file for each lab. Each LabItem stores the information given as XML options or attributes that are necessary in calculating student grades or survey information. Such information includes the variable
The Student portion of Coder contains objects that retrieve information directly from the MySQL database as described in Chapter 2. In order to build these objects they must interact with the Main portion of Coder which controls all connections to the database. All the information stored in these objects is obtained from the answers, actions, and students tables in the database.

The Lab and Student portions of the Coder are closely connected so that the student’s actions and answers may be matched to the original questions or apparatuses in the lab, thus, allowing for each student lab to be graded and the corresponding survey data to be analyzed.

Figure 6.1: Structure of the Coder application.
Chapter 7

Troubleshooting

Several different problems can arise in the LoCuS system. These include problems with computer settings, a particular lab, and apparatus in the lab, or with the LoCuS system.

7.1 Changing Settings on a Personal Computer

Many problems that are experienced by individual users may be due to settings on personal computers. In order to effectively run LoCuS labs a computer must have a reliable internet connection, the latest version of java, and meet the minimum screen resolution requirements. All this information can be found on the help page of the web site that is available to students.

Note that all labs must be tested and guaranteed to run on machines located in either computer lab in Gould-Simpson. If a student is unsure about changing their settings or is having difficulties direct them to use one of these labs.

Fixes to simple issues with computer settings are available on the help page that is available to students. If the student meets all the basic requirements needed to run the labs make sure that their computer and browser security settings allow them to run applications downloaded from the web.

7.2 Fixing a Lab

Issues that arise in a particular lab can usually be fixed in the xml file for that lab. This file is located under dist/labs in the folder for the particular lab. For more information about xml file please the the “Lab Developers Guide for Locus” Technical Document. Once the xml file has been changed follow the instructions in sections 2 and 3 to redeploy the labs. Note that the entire project must be redeployed to the web in order to include the new version of the apparatus in locus.zip.

7.3 Fixing an Apparatus

If you need to change an apparatus, make the necessary adjustments to the apparatus and then run the build.xml script to build the jar file. Note that the entire project must be redeployed to the web in order to include the new version of the apparatus in locus.zip. For more information on the structure of each apparatus see the “Apparatus Developer’s Guide” Technical Document.

7.4 Fixing the LoCuS System

If you make any changes to the LoCuS system you will need to redeploy the project afterwards following the instructions laid out in Chapters 2 and 3.
apparatus is a plugin to LoCuS that allows the user to do virtual experiments. For more information, please, refer to the Apparatus Developer’s Guide, located at /locus/docs/locus/ApparatusDevelopersGuide/

lab is an archive containing an XML file that is parsed by LoCuS to display the contents of the lab notebook and determine which apparatuses to show on the lab bench. Moreover, the XML file is used by LoCuS to perform basic configuration for each lab. The archive also contains all the resource files needed for the lab. For more information please refer to the “Lab Author’s Guide Technical Document” located at /locus/docs/locus/LabAuthorsGuide/

release and deploy mean the same thing for LoCuS: uploading updated jar files to the web and updating links on the website. Deployments are located at /cs/cgi/projects/locus. For definitions of the terms release and deploy, please refer to chapter “Terminology” of the “TAU Getting Started Guide Technical Document”. Note that in LoCuS we have the ability to create private and public deployments which are outlined in Section 3.8.