Lipi Toolkit 4.0

>> Getting Started

Contents

1 Introduction

Welcome to this Getting Started guide to Lipi Toolkit 4.0.

In the following sections, you will learn how to install Lipi Toolkit 4.0 (either the source or binary package) on your Windows or Linux system, how to build the sources (if needed), and how to test your installation using the Alphanumeric Character Recognizer that comes with the package.

You will then learn how to create your own custom recognizers using one of two approaches – Lipi Designer (suitable for a quick creation using a few clicks) and Recognizer Training (suitable for creating a more robust recognizer using a large number of training samples).

In the last section, you will learn how to integrate your custom recognizer into your own client applications, using the sample application C++ source code.

This document is meant to help you get started quickly with Lipi Toolkit. For a much more comprehensive documentation please refer the Core Toolkit User Manual.

1-1 Toolkit Contents

Lipi Toolkit 4.0 is comprised of the following components:

1) **Core Toolkit**
   Core Toolkit contains the core algorithms for the creation and evaluation of recognizers for isolated shapes such as handwritten gestures and characters.

2) **Lipi Designer**

   Lipi Designer is a standalone application that provides a graphical user interface for creating shape recognizers using the Core Toolkit.

3) **Alphanumeric Character Recognizer**
   The Alphanumeric Character Recognizer recognizes handwritten English uppercase characters, lowercase characters and numerals, and is designed to be integrated into client applications using the Lipi Toolkit API (see Section 5).

   The recognizer in turn uses the Core Toolkit, and the recognition accuracies are approximately as follows:

Character Set	Accuracy (%)
Numerals	94.34
English uppercase	94.58
English lowercase	90.59
Punctuation	95.83

2 Prerequisites

This section describes the prerequisites for installing and using *Lipi Toolkit 4.0*.

2-1 Supported platforms and environment

Lipi Toolkit 4.0 has been tested on the following platforms:

- Microsoft Windows 7 – 32 bit and 64 bit
- Ubuntu Linux Version 10.10 – 32 bit and 64 bit

2-2 Disk space requirements

Lipi Toolkit 4.0 provides separate source and binary packages for Windows and Linux. The space required to extract the source package is 20MB and binary package is 30MB. To build the source package after extraction, you will need approximately 130MB of free space on Windows and 40MB on Linux.

2-3 Software requirements

Item and Description	Windows 7	Linux
Building Core Toolkit	Microsoft Visual C++ 2008 / MsBuild for VC2008	G++ 4.4 or above
Executing scripts	Perl 5.1 or above, and Archive::Zip	Perl 5.1 or above and Archive::Zip
Building Lipi Designer	Java Development Kit (JDK) jdk1.6.0_26 or above	Openjdk-6-jdk
Documents	Adobe Acrobat reader	Xpdf

Table 1: Software requirements

NOTE: Archive::Zip is a Perl package that can be downloaded from http://search.cpan.org/~adamk/Archive-Zip-1.30/lib/Archive/Zip.pm.

To install, first uncompress the downloaded file using Winzip on Windows, and the command `tar -xzvf <tar.gz file>` on Linux. Once uncompressed, go to the `lib/Archive/` folder of Archive-Zip-1.30 and copy Zip/ Zip.pm to the Archive folder of the Perl installation on your system.

---

3 Installing Packages

Lipi Toolkit 4.0 is available in the form of binary and source packages for 32 and 64-bit versions of Windows and Linux. If you are only planning to use Lipi Toolkit in your applications, and not planning to modify the source code, you only need the binary package.

Platform	Package
Windows 7	**Binary:** lipi–toolkit4.0.0-bin-x86.exe lipi-toolkit4.0.0-bin-x64.exe **Source:** lipi-toolkit4.0.0-src-x86.exe lipi-toolkit4.0.0-src-x64.exe
Linux	**Binary:** lipi-toolkit4.0.0-linux-x64.tar.gz lipi-toolkit4.0.0-linux-x86.tar.gz **Source:** lipi-toolkit4.0.0-src-linux.tar.gz

Table 2: Lipi Toolkit packages

The Windows packages are self-extracting exes and can be installed by double-clicking the exe. The Linux packages are tar balls and can be installed by executing the below command:

```
tar -xzvf  lipi-toolkit4.0.0-linux-x64.tar.gz
```

Lipi Toolkit uses an environment variable LIPI_ROOT to refer to its installation directory. This variable needs to be set for building the source code and for executing the binaries on Linux and Windows.

On Windows, the LIPI_ROOT variable is automatically set on installation of the toolkit. On Linux, this variable has to be set manually. For example, if the package was extracted to /opt/ and installation directory is lipi_toolkit_4.0.0, $LIPI_ROOT should be set as below:

```
export LIPI_ROOT=/opt/lipi_toolkit_4.0.0
```

If you want LIPI_ROOT to persist across multiple sessions of Linux, you can add this entry to the .profile or .bashrc file in your Linux home directory, by executing the command below:

```
echo "export LIPI_ROOT=/opt/lipi_toolkit_4.0.0" >> $HOME/.profile
```

CAUTION: A current limitations is that there should be no blank spaces in the path specified by LIPI_ROOT

4 Building Toolkit Source

If you have downloaded the source package and wish to build Lipi Toolkit, please follow the instructions below for building the Toolkit on Windows and Linux.

If you have downloaded the binaries package, skip this section and go to the next section on *Testing installation*.

4-1 Building on Windows for VC2008

To build *Core Toolkit* on Windows for VC2008, devenv must be included in system PATH variable. This can be done by executing <visual studio 2008 install dir>\Common7\Tools\ vsvars32.bat from the command prompt.

Execute the following command from $LIPI_ROOT/windows/vc2008

```
msbuild lipitk.targets
```

To build *Lipi Designer*, first go to $LIPI_ROOT/lipiDesigner/src/lipijniinterface/windows and execute the following command:

```
msbuild lipijniinterface.targets
```

Then, to build javauserinterface go to $LIPI_ROOT/lipiDesigner/src/javauserinterface/windows and execute:

```
buildjar.bat
```

4-2 Building on Linux

To build *Core Toolkit*, execute the following command from $LIPI_ROOT/linux on 32 bit or $LIPI_ROOT/linux-x64 on 64 bit machine.

```
make -f Makefile.linux
```

To build Lipi Designer, go to $LIPI_ROOT/lipiDesigner/src/lipijniinterface/linux on 32 bit or $LIPI_ROOT/lipiDesigner/src/ lipijniinterface/linux-x64 on 64 bit machine, and execute the following commands:

```
make -f Makefile.linux
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
sh buildjar.sh
```

5 Testing your installation

Now that you have installed (and if you downloaded the source, then built) Lipi Toolkit, you are ready to test it using the recognizers that come bundled with it.

5-1 Using shaperectstui (Windows only)

On a Windows system, you can use *shaperectstui*, a sample GUI client application for shape recognition to test your installation. Shaperectstui reads the configuration file `lipiengine.cfg` from $LIPI_ROOT/projects and displays all the available character recognizers listed there in a drop down menu (Figure 1). You can select one of the recognizers (e.g. SHAPEREC_NUMERALS), and test it by writing a numeral in the writing area (Figure 2). If the installation is correct, your input should be recognized correctly most of the time.

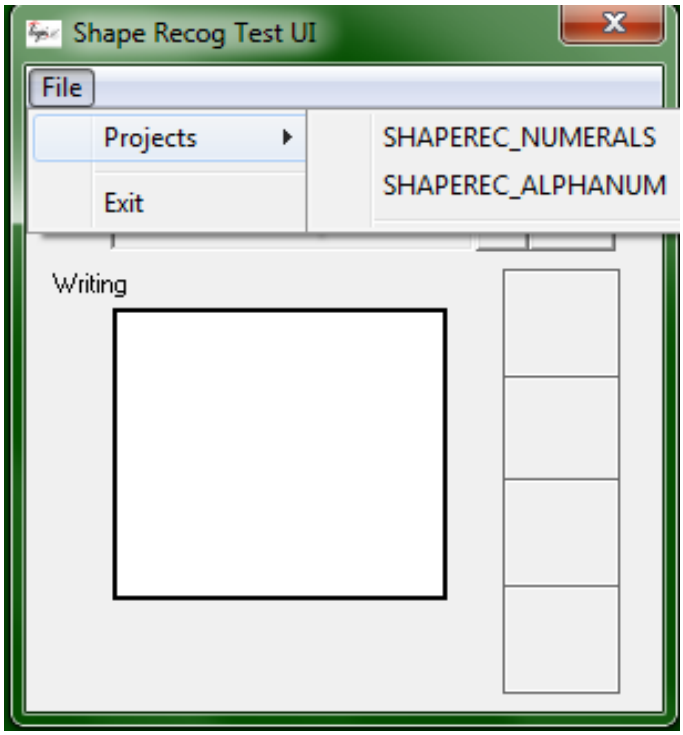

Figure 1: Load project Menu

For example if you draw '2' in the writing area, the following results should be displayed:

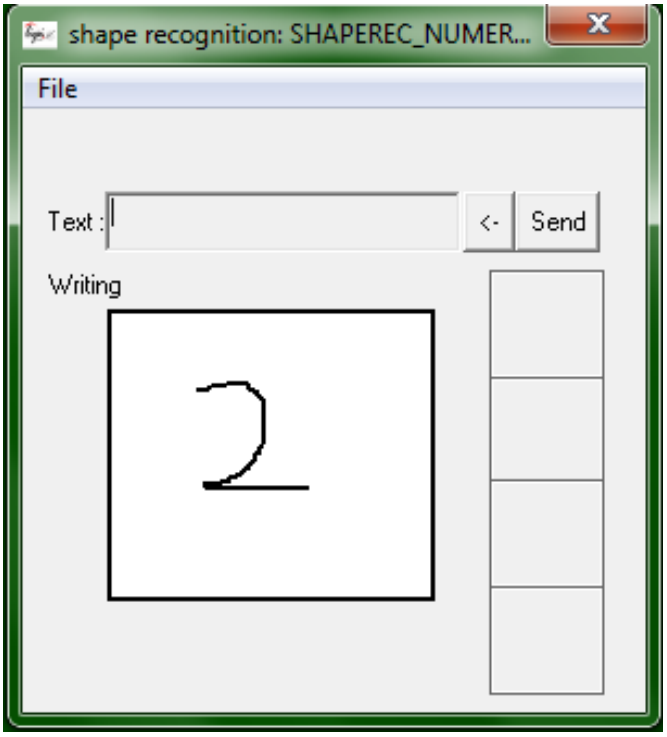

Figure2a: Draw "2" in the display area

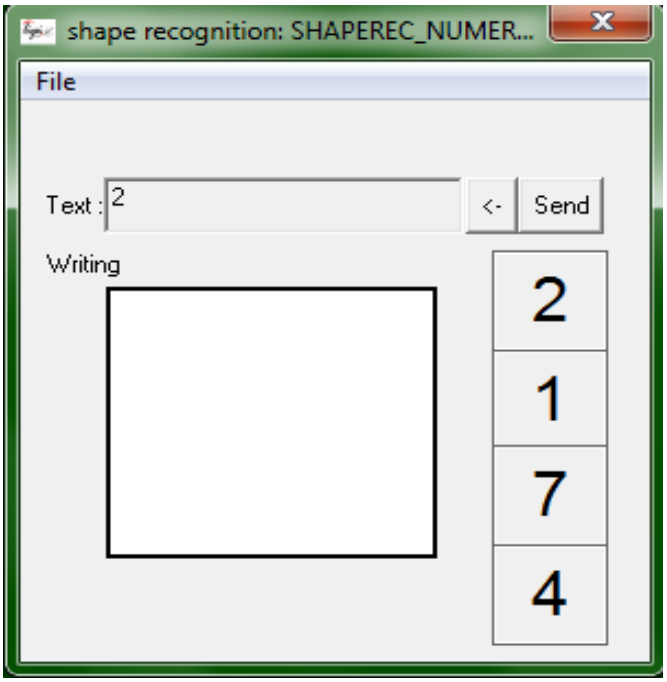

Figure 2: Displaying recognized shapes

Please refer to the Core Toolkit User Manual Section 13-4 for more details.

5-2 Using shaperectst (Windows and Linux)

shaperectst is a sample command line application that comes bundled with the toolkit. You can find it in $LIPI_ROOT/bin. *shaperectst* takes as input (i) the *logical name* of the shape recognizer you wish to use, and (ii) a handwritten character sample (in UNIPEN format), and returns the recognition result.

UNIPEN is a standard format to store on-line handwriting data as digital ink in text files, from International Unipen Foundation (www.unipen.org).

In Lipi Toolkit, each recognizer – whether already bundled with the toolkit, or created by you -- has its own *project* folder, and one or more configuration *profiles* that it uses for recognition (these terms are explained in the Core Toolkit User Manual). *Logical name* is an *alias* for *a specific shape recognizer project and recognition profile*, and provides an easy way to refer to a specific recognizer. For example, the recognizer for numerals has the logical name SHAPEREC_NUMERALS. The list of Logical names and their mappings to recognizer projects and profiles may be found in the `lipiengine.cfg` configuration file in $LIPI_ROOT/projects.

The recognition result returned by *shaperectst* is the *ShapeId* of the recognized shape. ShapeId is a label for the shape within the character set in question.

You can invoke shaperectst from the command line as follows:

```
shaperectst SHAPEREC_NUMERALS $LIPI_ROOT/projects/demonumerals/data/1/1_0.txt
```

Here we have specified the numeral recognizer as the one to use by its logical name SHAPEREC_NUMERALS, and provided a sample of a handwritten '1' as input.

The expected output is as follows:

Recognition Results

Choice[0] Recognized Shapeid = 1 Confidence = 0.549951

Choice[1] Recognized Shapeid = 2 Confidence = 0.126473

Here the Shapeids 1 and 2 - corresponding to the numerals '1' and '2' - are returned along with their confidence values. The fact that the handwriting input sample '1' has been recognized as '1' is evidence that the toolkit has been installed correctly.

6 Creating your own shape recognizer

We saw previously the use of the Alphanumeric Character Recognizer that comes bundled with the toolkit. In this section you will see how to create your own shape recognizers for new character or gesture sets. Each shape recognizer is meant to recognize a set of handwritten *shapes,* also called *classes*.

Such custom shape recognizers can be created using two different approaches:

1. **Lipi Designer** – Allows you to create a new recognizer using a small number of training samples in just a few steps, via a GUI.

2. **Recognizer Training** – Allows you to create a more robust shape recognizer, e.g. for a new script – by training a recognizer using a large number of handwriting samples and the *runshaperec* tool that is part of the Core Toolkit. This approach also allows you (if you so desire) to experiment with the many recognition algorithms available in the toolkit.

6-1 Using Lipi Designer

6-1-1 Creating a new project

Click **New** in the Project menu**,** and the **Create new project** dialog box (Figure 3) appears. The dialog box enables the user to enter a name for the project. For every new project created by the user, a new folder is created under `$LIPI_ROOT/projects`.

1. To add a new class to the project, draw a shape in the Writing Area and click on **Add Class**. You can add multi-stroke shapes also.

2. To delete a class, select the class and click on **Delete Class**.

3. To add samples to an existing class, click on **Add Sample**. The new samples are added to the class selected the **Classes** list box. By default, the most recently added class is selected. To add samples to a different class, select the desired class in the **Classes** list box before clicking on **Add Sample**.

4. To delete a sample from the class, select the sample and click on **Delete Sample.**

   NOTE: Deleting the last sample of a class results in deletion of the class from the project.

5. To train the recognizer on all the classes in a project, click on **Train**. There are two modes for training the recognizer:

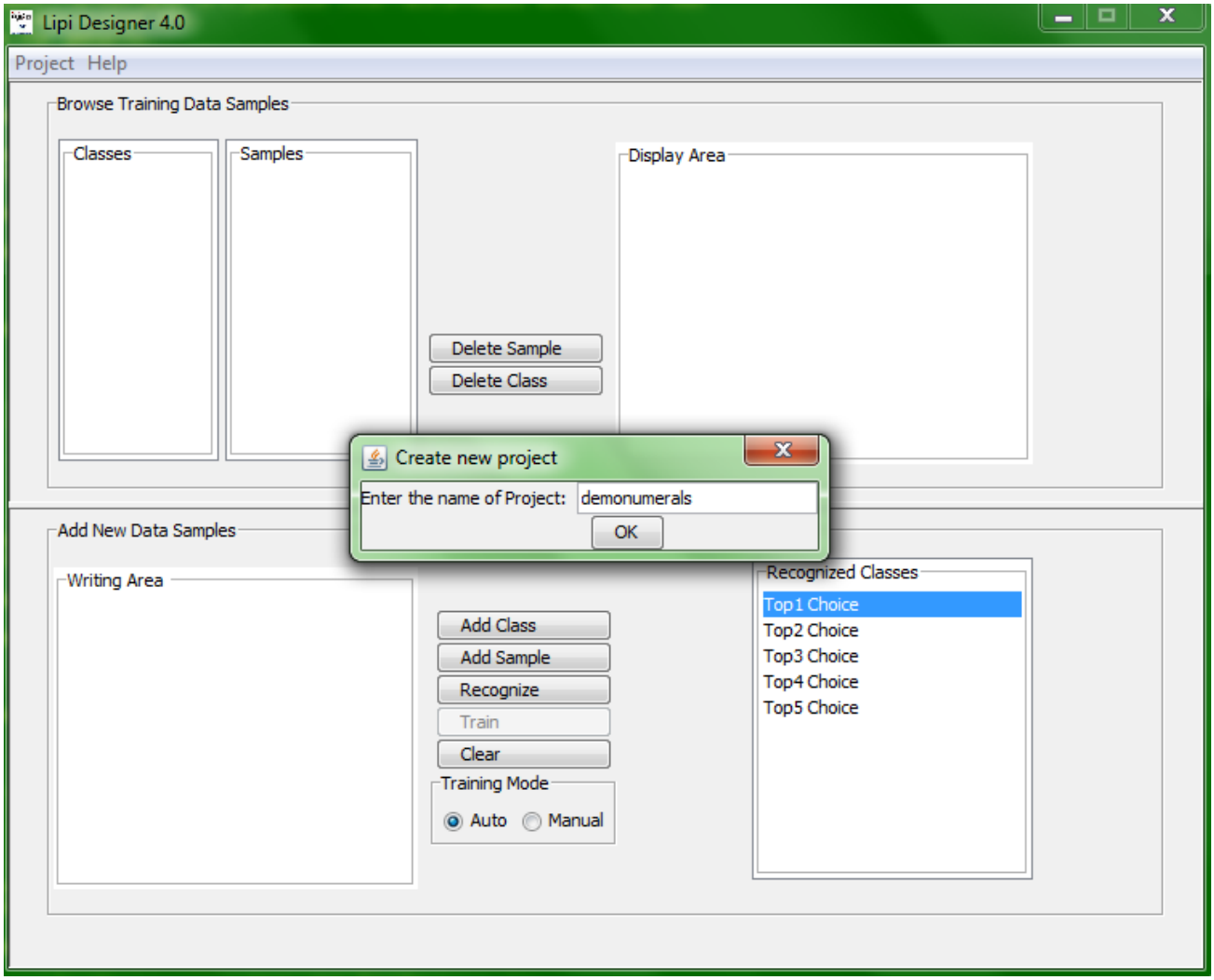

Figure 3: Create new project dialog box

Auto Mode: If the training mode is set to Auto, the application automatically trains the recognizer after every addition/deletion of a sample.

Manual Mode: In this mode, you are required to train the recognizer explicitly, by clicking on the **Train** button. You can choose to invoke training after adding or deleting multiple classes/several samples.

6. Once the recognizer has been trained, tested and found satisfactory, you should **explicitly** select "Save" in "Project" menu to create an entry for your project in "lipiengine.cfg". This will make your new recognizer available for testing, using shaperectstui and shaperectst.

7. The performance of a trained recognizer can be tested using the Recognize functionality of Lipi Designer

    1. Draw a shape in the writing area

2. Click the **Recognize** button, and the five best matching classes are listed in the **Recognized Classes** panel (Figure 4), with the first choice indicating the highest confidence result. If the number of matching classes is less than five, the system will display only the available classes, with top choice indicating the highest confidence.

8. To clear the stroke(s) written in the **Writing Area** or the result shown in the **Recognized Classes** panel, click **Clear**.

WARNING: Recognize cannot be called for an empty stroke.

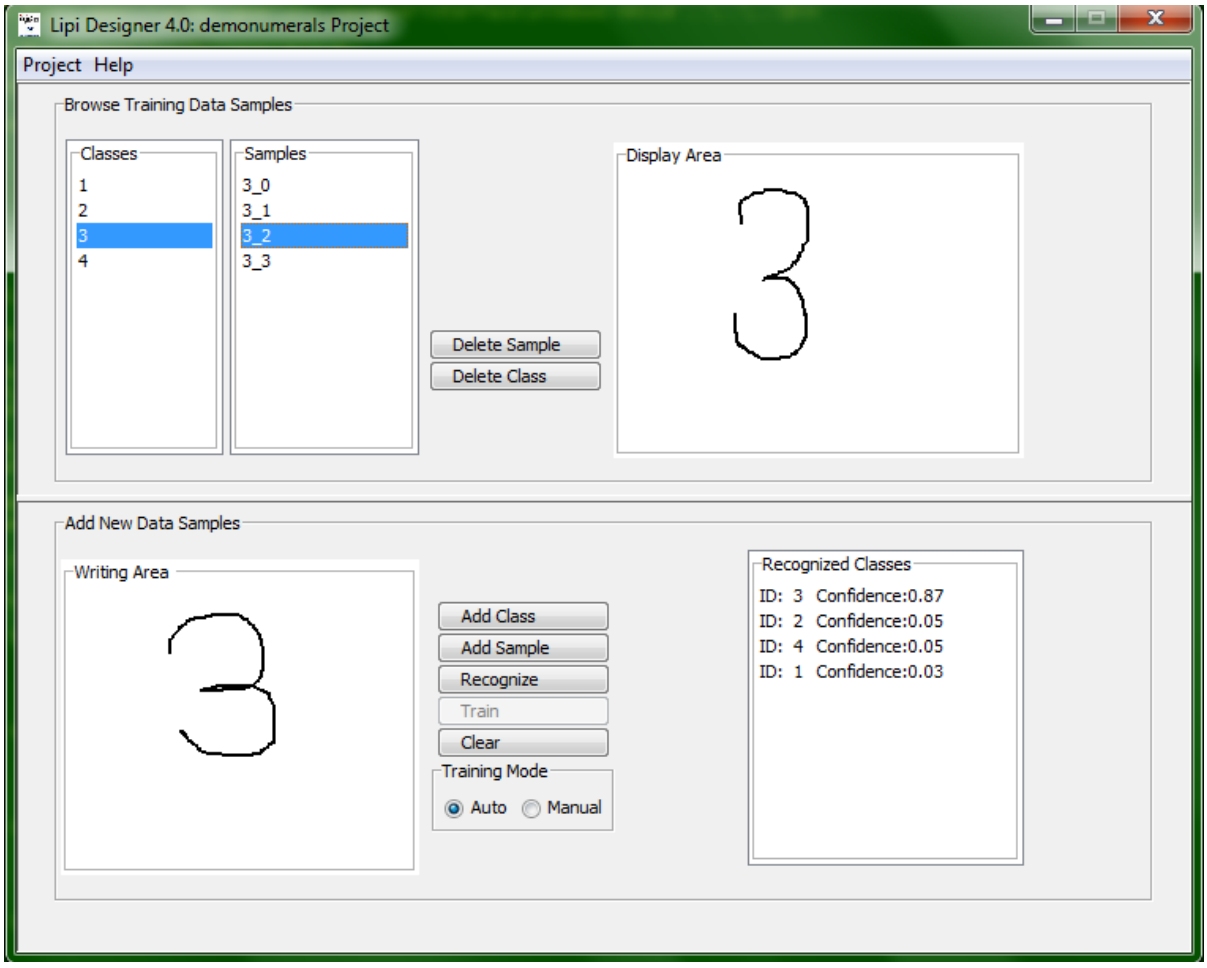

Figure 4: Recognition results

6-1-2 Loading an existing Project

To load an existing project, click **Load** in the Project menu. A window (Figure 5) displaying all the existing projects appear. The user can load a project by opening the projects directory and selecting the `<project name>`, available under `$LIPI_ROOT/projects/`

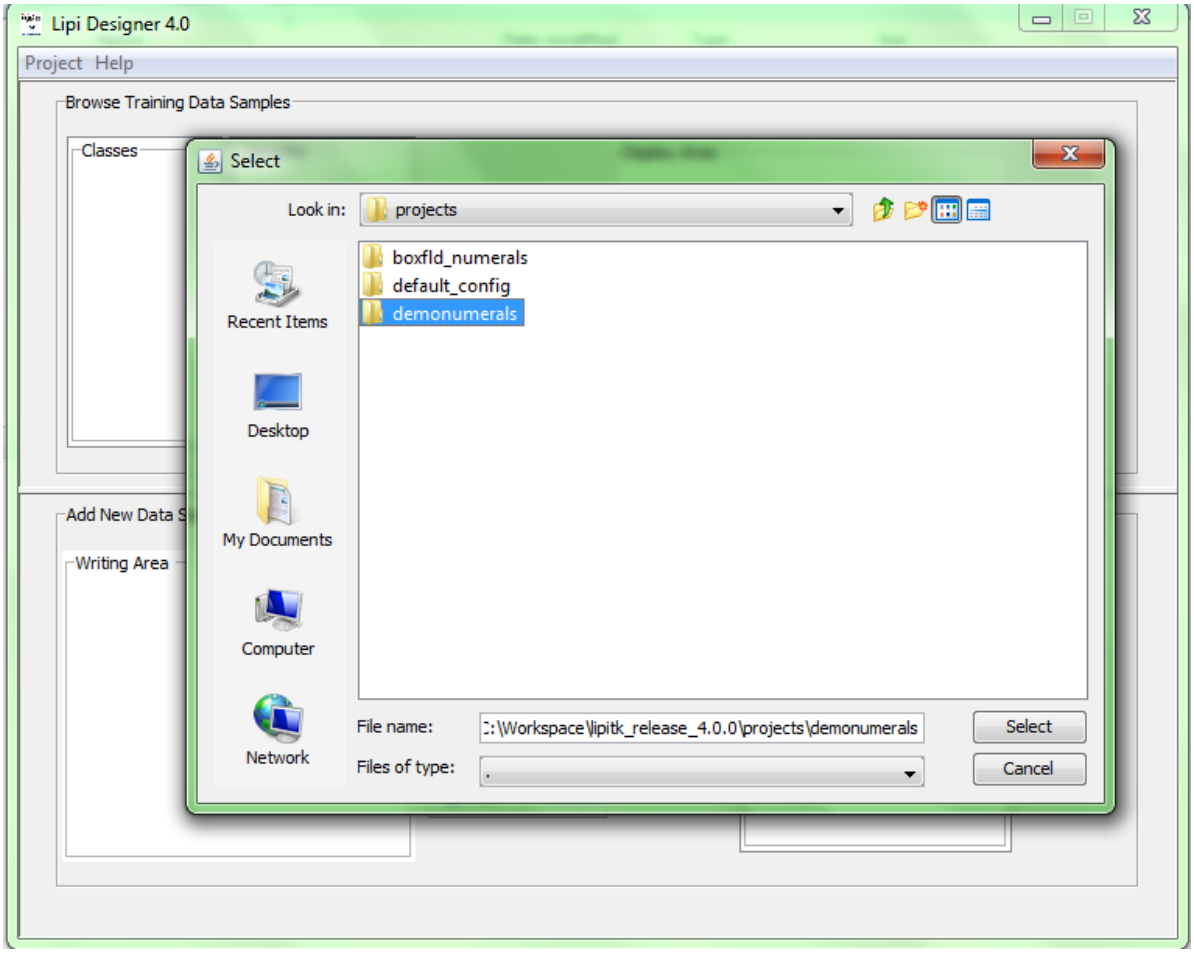

Figure 5: Load existing project

WARNING: Before loading/opening a project, the current working project must be closed.

6-2 Recognizer training using *runshaperec*

The *runshaperec* tool is an executable found under `$LIPI_ROOT/bin`, that can be used for training or testing a new shape recognizer. It uses training samples of the different shape classes it is required to recognize. It takes the recognizer project to be trained along with a file containing a list of training samples as an input:

```
runshaperec -project <project name> -train <file name>
```

A small number of samples of numerals are included in Lipi Toolkit. These may be used to train a recognizer project called "demonumerals" as follows:

```
runshaperec -project demonumerals -train
$LIPI_ROOT/projects/demonumerals/config/default/trainlist.txt
```

Ideally, and for good writer-independent accuracy, several hundred samples of handwriting from a spectrum of users covering various writing styles should be provided as training sample. Tools for such large scale data collection using a device such as a Windows TabletPC or an AceCad DigiMemo slate are available from the Lipi Toolkit website. For more details of training a recognizer using the runshaperec tool, please refer to Section 8 of the Core Toolkit User Manual.

7 Developing your first handwriting application

7-1 Introduction

The idea of Lipi Toolkit is to allow users to create interesting pen or touch applications that integrate character or gesture recognizers created using the toolkit. This section demonstrates how to use C++ APIs exposed by recognition components and modules of the Core Toolkit with sample applications that require shape or word recognition.

Lipi Toolkit 4.0 provides sample applications with C++ source code for Linux and Windows. You can use the provided source code of these applications as a starting point to build your own applications.

The sample applications are as follows:

1. *shaperectst*: sample client application that demonstrates recognition of a single shape.
2. *wordrectst*: sample client application that demonstrates recognition of a "boxed field" of shapes, such as those found on a form.

In addition to the above, there is also a sample GUI application for Windows

- *shaperectsui*

You may have already used the binaries of these applications to test your installation, in Section 5.

7-2 Typical application flow

This section describes the high level flow of the sample application *shaperectst* bundled with the toolkit. This sample program is provided as an example of how a Lipi character recognizer – bundled or custom-built - can be invoked from an application program. It illustrates the steps for instantiating the recognizers using their logical names, passing digital ink to them and obtaining recognition results.

As described in Section 5, *shaperectst* takes as input (i) the *logical name* of the shape recognizer you wish to use, and (ii) a handwritten character sample (in UNIPEN format), and returns the recognition result. The major steps in processing are listed below:

1. Get the LIPI_ROOT environment variable.
2. Load the lipiengine dll/shared project.

3. Create an instance of lipiengine module
4. Create an instance of the (bundled or custom-built) recognizer into memory using its logical name (provided as an input to *shaperectst*), before starting the recognition.
5. Read the screen context, device context (e.g. digitizer resolution information) and digital ink information corresponding to the handwriting test sample from the input UNIPEN file.
6. Invoke the Recognize function to interpret the handwriting test sample
7. Display the recognized results (a list of ShapeIds and confidence values) returned by the Recognize function.
8. Delete the shape recognizer object.
9. Unload lipiengine module from memory.

To understand the implementation of these steps, the Lipi Toolkit data structures used for storing ink, API calls for loading and recognizing handwriting input, etc, please refer to the source code available at LIPI_ROOT\src\apps\samples.

While *shaperectst* illustrates the reading of handwriting data from a UNIPEN file, *shaperectstui* illustrates the capture of handwriting from a writing area in the GUI.

For more details of *shaperectstui* and *wordrectst*, please refer to Section 13 of Core Toolkit User Manual.

8 Acknowledgements

Lipi Toolkit is the result of the efforts of a number of dedicated researchers, software developers and students at HP Labs India and collaborating universities. For a list of contributors please refer to the Lipi Toolkit website at lipitk.sourceforge.net.

We would also like to thank IRESTE, University of Nantes (France) for permitting us to use the IRONOFF handwriting database for training the Alphanumeric Character Recognizer.