About Attrasoft TransApplet

Attrasoft **TransApplet** is a .Net Class Library that enables the addition of Image Recognition capability to products & services. It can be used for:

- Image Verification (1:1 Matching);
- Image Identification (1:N Matching);
- Image Search or Retrieval (1:N Matching); and
- Multiple Verification or Identification (N:N and N:M Matching).

ImageFinder for Windows is an off-the-shelf Application Software that enables System Integrators, Solution Developers, and Individuals to quickly test their own Image Recognition ideas.

TransApplet is a .Net Class Library that enables System Integrators, Solution Developers, and Individuals to quickly add Image Recognition capability to their products and services.

Software Requirements

The software requirements are:

- Microsoft .Net Framework
- Microsoft Visual Studio .Net or C#.Net

When you install Microsoft Visual Studio .Net, the Microsoft .Net framework will be installed automatically.

Installing the Software

Copying the “CD:\transapplet70” to the C-driver will complete the installation.

Information & Support

Attrasoft TransApplet
Attrasoft
P. O. Box 13051
Savannah, GA. 31406
USA

http://attrasoft.com

imagefinder@attrasoft.com (Email Subject: Attrasoft)

Phone: (912) 484-1717

License Agreement

THIS LICENSE AGREEMENT ("AGREEMENT") IS BETWEEN YOU, THE END USER, AND Attrasoft. IT GOVERNS THE USE OF THE SOFTWARE PROGRAM AND DOCUMENTATION KNOWN AS **Attrasoft *TransApplet*** (THE "PRODUCT"). IF YOU USE THE PRODUCT, THEN YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THIS PACKAGE TO THE PLACE OF PURCHASE WITH A COPY OF THE RECEIPT, AND YOUR LICENSE FEE WILL BE REFUNDED.

Attrasoft licenses use of the PRODUCT, which may be manufactured and distributed by Attrasoft or by a third party (either, the "Manufacturer"). You, the end-user, assume responsibility for the selection of the PRODUCT to achieve your intended results, and for its installation and subsequent use.

GRANT OF LICENSE

Attrasoft hereby grants you a non-exclusive license to use the PRODUCT in object code form only, upon the terms and conditions contained in this Agreement.

You may:

1. Use the PRODUCT on the number of workstations for which you have purchased PRODUCT licenses. The workstations must be owned, leased or otherwise controlled by you, whether in a network or other configuration.
2. Create a quantity of backup copies of the PRODUCT, in any machine-readable or printed form, equal to the number of PRODUCT licenses you have purchased.
3. Transfer the PRODUCT and your rights under this Agreement to another party if the other party agrees to accept the terms and conditions of this Agreement. If you transfer the PRODUCT, you must, at the same time, either transfer all copies of PRODUCT to the same party, or destroy any copies not transferred. You must immediately notify Attrasoft of the transfer.
4. Print out one copy of the Product documentation from the Attrasoft program, **TransApplet**, for each license purchased. If you print out any part of the Product documentation from the Attrasoft program, **TransApplet**, you must reproduce and include all the copyright notices that appear in the documentation on any such copy of the documentation.

You May Not:

1. Use or copy the PRODUCT, in whole or in part, except as expressly provided in this Agreement.
2. Use the PRODUCT concurrently on more than the number of workstations for which you have purchased licenses.
3. Copy, rent, distribute, sell, license or sub-license, or otherwise transfer the PRODUCT or this license, in whole or in part, to another party, except as specifically set forth above.
4. Incorporate the PRODUCT or any portion of the PRODUCT into, or use the PRODUCT, or any portion of the PRODUCT to develop, other software without a license from Attrasoft, or otherwise modify or create a derivative work from the PRODUCT without a license from Attrasoft.
5. Reverse engineer, decompile, or disassemble the PRODUCT.

To use the PRODUCT as described in Sections 2 or 4 above, or for any other use not specifically set forth above, additional licensing from Attrasoft is required. For further information, please contact Attrasoft at:

Attrasoft, Inc.
Phone: (912) 484-1717

gina@attrasoft.com

PROPRIETARY RIGHTS
This Agreement gives you ownership only of the physical Attrasoft program, **TransApplet**, on which the PRODUCT is stored, but not of the PRODUCT itself. You acknowledge that Attrasoft owns all rights, title, and interest in the PRODUCT, and that you will acquire no rights in the PRODUCT through your use of it. You agree that you will take no action that interferes with Attrasoft's rights in the PRODUCT.

TERMS
This Agreement is effective until terminated. You may terminate it at any time by destroying the PRODUCT together with all copies and documentation in any form. This Agreement will also terminate automatically and without notice from Attrasoft if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the PRODUCT and all copies of the PRODUCT.

DISCLAIMER; LIMITED WARRANTY
EXCEPT AS PROVIDED BELOW, THE PRODUCT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PRODUCT IS WITH YOU. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. NEITHER ATTRASOFT NOR MANUFACTURER WARRANTS THAT THE FUNCTIONS CONTAINED IN THE PRODUCT WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE PRODUCT WILL BE UNINTERRUPTED OR ERROR-FREE. However, where Attrasoft is the Manufacturer, Attrasoft warrants that the Attrasoft program, **TransApplet**, on which the software is furnished will be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery as evidenced by a copy of your receipt.

LIMITATION OF REMEDIES
Where Attrasoft is the Manufacturer, Manufacturer’s entire liability and your exclusive remedy shall be:
1. The replacement of the Attrasoft program, **TransApplet**, not meeting the Limited Warranty, which is returned to Manufacturer with a copy of your receipt.
2. If Manufacturer is unable to deliver replacement Attrasoft program, **TransApplet**, which is free of defects in materials or workmanship, you may terminate this Agreement by returning the PRODUCT and a copy of your receipt to the place of purchase, and your money will be refunded. Where Attrasoft is not the Manufacturer, Attrasoft shall have no liability to replace or refund, and you agree to look to Manufacturer to meet the obligations described above.

LIMITATION OF LIABILITY
IN NO EVENT WILL ATTRASOFT OR MANUFACTURER BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING, BUT NOT LIMITED TO, ANY LOST PROFITS, LOST SAVINGS, OR OTHER INDIRECT, SPECIAL, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE THIS PRODUCT, EVEN IF ATTRASOFT OR MANUFACTURER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. FURTHER, IN NO EVENT WILL ATTRASOFT OR MANUFACTURER

BE LIABLE FOR ANY CLAIM BY ANY OTHER PARTY ARISING OUT OF YOUR USE OF THE PRODUCT. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

TRADEMARKS

Attrasoft is a trademark of Attrasoft, Inc. Microsoft, C# logo are registered trademarks of Microsoft Corporation. No rights, license, or interest in such trademarks is granted hereunder.

U.S. GOVERNMENT RESTRICTED RIGHTS IN DATA

This computer software product and documentation are provided with Restricted Rights. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Attrasoft, Inc.

EXPORT CONTROLS

You agree not to export or re-export the PRODUCT, directly or indirectly, to any countries, end-users or for any end uses that are restricted by U.S. export laws and regulations, without first obtaining permission to do so as required by the U.S. Department of Commerce's Bureau of Industry and Security, or other appropriate government agency. These restrictions change from time to time. If you have any questions regarding your obligations under U.S. export regulations, you should contact the Bureau of Industry and Security, U.S. Department of Commerce, Exporter

Counseling Division, Washington D.C. (202) 482-4811, http://www.bis.doc.gov.

TABLE OF CONTENTS

1. Introduction

Attrasoft **TransApplet** for Windows is a .Net Class Library that enables the addition of Image Recognition capability to products & services. It can be used for:

- Image Verification (1:1 Matching);
- Image Identification (1:N Matching);
- Image Search or Retrieval (1:N Matching); and
- Multiple Verification or Identification (N:N and N:M Matching).

1.1 What is TransApplet?

Attrasoft **TransApplet** for Windows is a .Net Class Library that enables the addition of Image Recognition capability to products & services.

1.1.1 ImageFinder

Before you continue, you should be familiar with the Attrasoft **ImageFinder** for two reasons:

- Learn the structure of the Attrasoft Image Recognition Approach;
- Learn the **ImageFinder** software.

The **TransApplet** will:

- Introduce the class library, "Attrasoft.TransApplet70", used in the **ImageFinder**.
- As a teaching tool, the **TransApplet** will show you how to build an **ImageFinder** via C# .Net.

Therefore, if you have not learned the **ImageFinder**, you should start with the **ImageFinder** now.

1.1.2 ImageFinder Family

The **ImageFinder** family has several members:

- **ImageFinder**
- **TransApplet**
- **PolyApplet**

To understand why there are three members in the **ImageFinder** family, we need to understand the Attrasoft software Structure, which consists of three layers:

- Application Layer
- Translation Layer
- Attrasoft Matching Engine (AME) Layer (Neural Network Layer)

Where:

- The Application Layer is the graphical user interface layer.
- The Translation Layer deals with images & videos, and formats this application data for the next layer.
- The AME Layer will do the image matching, which is independent of the type of data used. For example, the voice data, image data, text data, and numerical data will all be treated equally. The AME Layer is the same layer used in other Attrasoft products.

Within this structure:

- **ImageFinder** is a standalone software.
- **TransApplet** is the Presentation Layer programming tool, which also includes the PolyApplet.
- **PolyApplet** is the AME Layer programming tool.

Each of the three software has to be purchased separately. The **TransApplet** does include the **PolyApplet.**

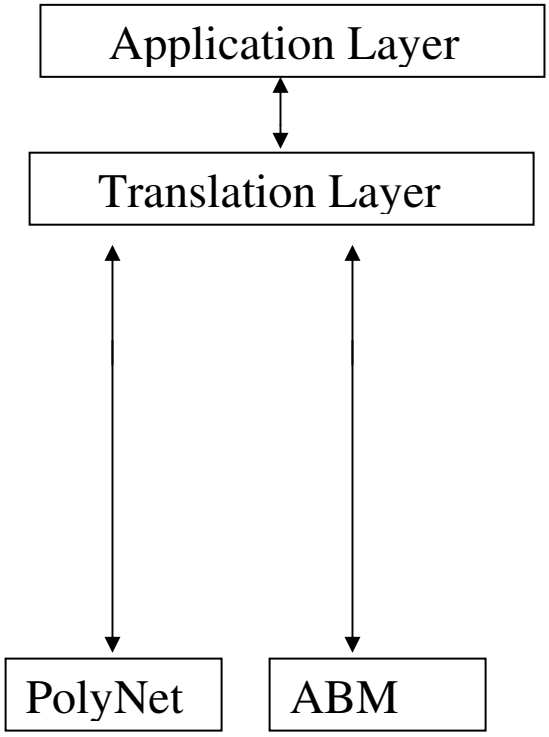

Figure 1.1 Attrasoft Component-Object Structure.

1.2 Software Requirements

The software requirements are:

- Microsoft .Net Framework
- Microsoft Visual Studio .Net or C#.Net

When you install Microsoft Visual Studio .Net, the Microsoft .Net framework will be installed automatically.

1.3 Installing the TransApplet

Copying the "CD:\transapplet70" to the C-driver will complete the installation. After copying, you should have both the Library Files and the Chapter Example.

1.3.1 Attrasoft.Transapplet70 Class Library

The **TransApplet** library has the following dll files:

C:\transapplet70\Batch70.dll
C:\transapplet70\BioFilter70.dll
…

Each Library has one main class, which has the same name as the Library-name:

Attrasoft.transapplet70.Batch70.Batch70
Attrasoft.transapplet70.BioFilter70.BioFilter70
…

The advantage of naming the class-name as the library-name is that you will know what to use, i.e.

- if you go to Library, "Attrasoft.transapplet70.Batch70", use Batch70 class;
- if you go to Library, "Attrasoft.transapplet70.BioFilter70", use BioFilter70 class, …

There is no guessing which class you should choose if the Library has more than one class.

The disadvantage is that the compiler will not be able to distinguish between the class-name and the Library-name unless you use the full name.

For example:
- "Attrasoft.transapplet70.Batch70" is a Library; and
- "Attrasoft.transapplet70.Batch70.Batch70" is a class.

Batch70 alone is not understandable by the C# compiler. To declare an object, you must use the full path:

Attrasoft.transapplet70.Batch70.Batch70 ba70.

1.3.2 Chapter Examples

Each chapter has a project. Many chapters can share the same project. Their projects provide an integration source code so you can cut and paste to produce your own application.

1.3.3 Linking the TransApplet

To link to these Libraries:

- Start C# project;
- Right click References in Solution Explorer and select Add Reference;
- Click Browse Button;
- Select "c:\transapplet70*.dll" files;
- Click the "OK" button.

1.4 Attrasoft Image Recognition Basics

The Attrasoft Image Recognition Approach has two basic phases:

- Signature Matching (matching whole images)
- Neural Matching (matching image segment)

In matching whole images, a unique image signature is computed for each image. Different images will have different signatures. The Signature Matching is based on these image signatures. **This matching can easily reach the speed of 100,000 to 1,000,000 whole images per second and is very fast.**

In the Image Segment matching, a neural net learns what the image looks like in a very similar way as the human brain does, i.e. adjusting the internal synaptic connections to remember the image. A typical Feature Space Recognition will use about 100 points and a typical Input Space Recognition will use about 10,000 points; therefore, the Signature Recognition is much faster than the Image Segment Recognition.

Image Matching is further divided into filters. It is these filters that will perform the image matching tasks.

The ability to learn is a fundamental trait of intelligence. Neural Nets are deployed in both Signature Matching and Segment Matching. Neural Nets can learn from examples. There are two basic phases in image recognition**:**

- Training; and
- Recognition.

In the training phase, data is imposed upon a neural network to force the network to remember the pattern of training data. A neural network can remember the training data pattern by adjusting its internal synaptic connections.

In the recognition phase, the neural network, based on its internal synaptic connections, will determine whether the newly captured image matches the sample image.

2. Image Recognition Overview

2.1 Image Recognition Internal Structures

An image recognition application is roughly divided into:

Level 5: User Interface;
Level 4: Data Management;
Level 3: Image-Matching Layer;
Level 2: Scanner, Camera, and their Drivers;
Level 1: PC with Windows.

The **TransApplet** Library will be used to implement the Image-Matching layer.

Stand-alone software, like the Attrasoft **ImageFinder**, is roughly divided into 3 layers:

Level 3: User Interface;
Level 2: Image-Matching Layer;
Level 1: PC with Windows.

The Attrasoft Image Recognition Approach has two basic phases:

- Signature Matching;
- Image Segment Matching

In the Signature Matching, a unique image signature is computed for each image. Different images will have different signatures. The Signature Matching is based on these image signatures. This matching can easily reach the speed of 1,000,000 per second and is very fast.

In the Image Segment Matching, a neural net learns what the image looks like in a very similar way as the human brain does, i.e. adjusting the internal synaptic connections to remember the image.

2.2 Filters

For the **ImageFinder**, the Image Matching is divided into:

- Image Preprocessing
- Image Processing
- Normalization
- Signatures
- Feature Recognition
- Image Segment Recognition

One or more filters further implement each step:

Image Preprocessing
    Preprocessing Filter
Image Processing
    Edge Filter
    Threshold Filter
    Clean Up Filter
Normalization
    Reduction Filter
Signature
    Signature Filter
Feature Recognition
    Unsupervised Filter
    BioFilter
    NeuralFilter
Image Segment Recognition
    Neural Net Filter

The image matching software design based on **TransApplet** is implemented to push an image through all these filters and at the end of pushing, the matching results will be obtained.

2.3 Image Preprocessing & Processing

Attrasoft **ImageFinder** learns an image in a way similar to human eyes:

- Ignore the background;
- Focus on an object in the image.

The Image Preprocessing is implemented by one filter, the Image Preprocessing Filter. Three filters implement image processing:

- Edge Filters;
- Threshold Filters; and
- Clean-Up Filters.

The Image Preprocessing and the Image Processing processes are not unique; there are many options available. Some are better than others.

The principle of choosing the Image Preprocessing and the Image Processing filters is to make the sample objects stand out, otherwise, change the options.

Image Preprocessing has the following functions:

Cut off the border areas;
Impose a mask;
Speed up the computation;

Skip the empty border areas;

Image Processing has the following filters:

The Edge Filters attempt to exaggerate the main features a user is looking for.
The Threshold Filters attempt to suppress the background.
The Clean-Up Filters will smooth the resulting image to reduce recognition error.

2.4 Normalization

The Normalization Sub-Layer will prepare the images for the underlying image matching engine. The Attrasoft Image Matching Engine is an internally developed algorithm, which is called the "Attrasoft Boltzmann Machine" or ABM. The ABM neural net deployed in the **ImageFinder**, by default, is a 100x100 array of neurons.

While any size of neural net can be used, when coming to a particular application, a decision has to be made. The **ImageFinder** uses 6 different sizes:

- 10,000 neurons,
- 8,100 neurons,
- 6,400 neurons,
- 4,900 neurons, or
- 2,500 neurons.

Later in the multi-layered design, the number of neurons can be much larger. The Reduction Filter will connect the images to various sets of ABM neural networks.

2.5 Signature Matching

In this phase, a unique image signature is computed for each image. Different images will have different signatures. The Signature Matching is based on these image signatures. This matching can easily reach the speed of 1,000,000 per second and is very fast.

A typical image has a dimension of say 480x640, or 300,000 pixels. A Feature Space of an image is a collection of variables computed from a given image. The number of variables in a Feature Space is usually less than 1% of the number of pixels. As a result, the matching in the Feature Space is much faster than Pixel Matching.

The following filters implement the Signature matching:

- Unsupervised Filter
- BioFilter
- Neural Filter

2.6 Image Segment Matching

The Attrasoft neural network (AME) is the Matching Engine of the **ImageFinder**. There is one filter in this sub-layer, the NeuralNet Filter. The ABM Matching Engine is responsible for all of the Attrasoft products.

3. TransApplet Overview

The **TransApplet** Class Library, "Attrasoft.TransApplet70", is used in the **ImageFinder**. It is a programming tool. This User's Guide will show you how to build an **ImageFinde**r software like the **ImageFinder** via C# .Net.

Therefore, if you have not learned how use to the **ImageFinder,** you should start with the **ImageFinder** now.

The **TransApplet** consists of a set of filter class libraries. Each chapter will introduce one of these libraries as you continue to build the "**ImageFinder**". Some minor graphical user interface issues will be ignored here because it is already complicated enough.

3.1 TransApplet API & User Interface

Chapter 4, **API**, describes the API (Application Programming Interface) of the "Attrasoft.TransApplet70" Class Library. The API is presented as the Library, "Attrasoft.transapplet70.Interface70", which consists of a set of Interfaces. Each class implements an Interface.

Chapter 5**, User Interface**, briefly describes how to set up the forms and the menu bar for the **ImageFinder,** and also how to link to class libraries**.** The **ImageFinder** GUI is divided into 3 parts:

- Input/Output
- Menu Items
- Parameters

3.2 Input

Chapter 6**, Input**, briefly describes how to enter data to the **ImageFinder.** The input images are further divided into:

- Key image – the sample image the **ImageFinder** will match against;
- Search Source – a list of images to be searched.

The search source is further divided into:

- Search-Directory (A directory containing images to be searched);
- Search-File (A file containing images to be searched);
- …

An image path specifies the training image.
A search-folder or search-file specifies the Search Source.
This chapter will introduce the class library, "Attrasoft.transapplet70.Input70".

3.3 Image Display

Chapter 7, **Image Display**, briefly describes how to display the images**.** The images are further divided into:

- Key Image;
- Search Source.

The search source is further divided into:

- Search-Directory (A directory containing images to be searched);
- Search-File (A file containing images to be searched);
- …

The Image Display will play the following role:

- Display Key Segment. No image processing will be applied to the image;
- Display the first image in search source. No image processing will be applied to the image;
- Display the next image and the previous image in search source.
- Display the first and next image in the matching results.
- Display processed images (after applying the Image Preprocessing Filter, Edge Filter, Threshold Filter, or Clean-Up Filter).

3.4 Image Preprocessing, Processing, & Normalization

Chapter 8, **Image Preprocessing**, and Chapter 9, **Image Processing**, briefly describes the image preprocessing and image processing required for the **ImageFinder.**

The Image Preprocessing sub-layer prepares the image for the **ImageFinder**. Image Preprocessing is not unique; there are many options available. Some are better than others.

The Image Preprocessing Layer consists of one filter, the Preprocessing filter.

The Image Processing Layer consists of three types of filters:

Edge Filters (Optional);
Threshold Filters (Required); and
Clean-Up Filters (Optional).

The **ImageFinder** applies these three filters in the above order.

- The Edge Filters attempt to exaggerate the main features a user is looking for.
- The Threshold Filters attempt to suppress the background.
- The Clean-Up Filters will smooth the resulting image to reduce recognition error.

From the user's point of view, Image Processing means you have to set three filters: Edge Filters, Threshold Filters, and Clean-Up Filters. The Threshold Filter is required; the other two filters are optional. This chapter will introduce the class library, "Attrasoft.transapplet70.ImageProcessing70".

Chapter 10, **Normalization**, briefly describes the Normalization process required for the **ImageFinder.** At the end of the Image Processing, the original image is transformed into a new image with the main features exaggerated and the background suppressed, i.e. the result is still an image. The underlying image matching engine will process data in a particular format, therefore, an image will need to be formatted for the internal matching engine via a normalization process.

Normalization has 1 filter, Reduction Filter, which will prepare the images for the underlying image matching engine. The Attrasoft Image Matching Engine is an internally developed algorithm, which is called the "Attrasoft Boltzmann Machine" or ABM. The ABM neural net deployed in the **ImageFinder**, by default, is a 100x100 array of neurons.

While any size of ABM neural net can be used, when coming to a particular application, a decision has to be made. The **ImageFinder** uses 6 different sizes:

- 10,000 neurons,
- 8,100 neurons,
- 6,400 neurons,
- 4,900 neurons, or
- 2,500 neurons.

3.5 Parameters

Implementing an image matching software via the **TransApplet** means pushing an image through these 10 filters:

1. Preprocessing Filter
2. Edge Filter
3. Threshold Filter
4. Clean-Up Filter
5. Reduction Filter
6. Signature Filter
7. Unsupervised Filter
8. BioFilter
9. NeuralFilter
10. Neural Net Filter

Each filter is an object (See their API in earlier chapters). Each filter will have many parameters and the 10 filters can have many parameters. It will be easier, from the programming point of view, to group all of the objects together and to group all of the parameter together.

Chapter 11, **Parameters**, will describe how to group all of the objects together and how to group all of the parameters together.

3.6 Signature Recognition

The Attrasoft Image Recognition Approach has two basic phases:

- Signature Matching
- Image Segment Matching

In the Signature Recognition, a unique image signature is computed for each image. Different images will have different signatures. The Signature Matching is based on these image signatures. **This matching can easily reach the speed of 1,000,000 per second and is very fast.**

Signature Matching is further divided into filters: Unsupervised Filter, BioFilter, and NeuralFilter. It is these filters that will perform the image matching tasks.

Chapter 12, **Signature Filter**, will describe how to generate image signatures.

Chapter 13, **Unsupervised Filter**, will describe the minimum number of steps for using the Unsupervised Filter for image recognition:

- Initialization
- Signature
- Unsupervised Matching

Chapter 14, **BioFilter**, will describe the minimum number of steps for using the BioFilter for image recognition:

- Initialization
- Signature
- Training
- Signature Matching

Initialization sets the **ImageFinder** parameters. Then, the image signatures are calculated and stored in a record. Unsupervised Matching can match images based on these records alone. Training teaches the BioFilter who matches with whom. After training, the BioFilter can be used for 1:1 and 1:N Matching.

BioFilter matches two whole images. BioFilter is better than Unsupervised Matching, but it requires a process called **training**. Training teaches the BioFilter who should match with whom. The BioFilter learns how to match the image features.

Chapter 15, **NeuralFilter**, introduces the Neural Filter. NeuralFilter matches two whole images, which is similar to the BioFilter. NeuralFilter is better than both the Unsupervised Filter and the BioFilter, but it requires a large amount of training data. Training data teaches the NeuralFilter who should match with whom. In comparison to early filters:

- The advantage of the NeuralFilter is that it is more accurate.
- The disadvantage of the NeuralFilter is that it requires more training data than BioFilter.

3.7 Dynamic Library

Chapter 16, **Dynamic Library**, introduces the dynamic library used in the **ImageFinder**. Dynamic Library allows the master library be updated via insertion, deletion, and update.

3.8 Image Segment Matching

Chapter 17, **NeuralNet Filter**, introduces the neural network used in the **ImageFinder**.

Chapter 18, **Parameters**, gives a more detailed description of all parameters used in various filters.

3.9 Input

Chapter 19, **Input Options**, introduces available input options for the **ImageFinder.**

Chapter 20, **Access Input**, introduces using the Access database as a search source for the **ImageFinder**. This chapter introduces the Database Input Library.

Chapter 21, **Avi Input**, introduces using an Avi video file as a search source for the **ImageFinder**.

Chapter 22, **Live Input**, introduces using live video as a search source for the **ImageFinder**.

3.10 Counting

Chapter 23, **Counting & Tracking Design**, introduces image-counting design. 'Counting' counts the number of objects in an image, assuming there is no overlap between objects.

Chapter 24, **ImageCounter**, introduces several examples on how to write applications to count the number of segments, and measure the areas & perimeters of each segment. This chapter also introduces an example of how to keep track of an object from frame to frame.

3.11 Batch Job

Chapter 25, **Batch**, introduces the batch command, which allows you to save your setting and execute your problem in a few clicks. This chapter will also show you how to build customized examples. This chapter will introduce the class library, "Attrasoft.transapplet70.Batch70".

At this point, you have built a software, which is very similar to the Attrasoft **ImageFinder** 7.0.

3.12 Customized Software

Chapter 26, **ImageFinder for Dos**, introduces the DOS version of the ImageFinder. ImageFinder for Dos is command line software that enables System Integrators, Solution Developers, and Individuals to make a quick & dirty system integration to test their product prototypes and services.

Chapter 27, **Attrasoft ImageHunt Web Application**, introduces an Internet version of the ImageFinder. ImageHunt is the ImageFinder for Windows with a web interface. If you need the ImageFinder for Windows, you might need the ImageFinder for Web at some point. Unlike the ImageFinder, the Internet Search Engine, by definition, does not bother users with complicated parameters; therefore, all the parameters in the ImageHunt must be fixed. The cost for such convenience is that each ImageHunt is limited to a particular type of application, i.e. the ImageHunt will require customization for a particular problem, say logos, auto parts, documents, ….

Chapter 28, **Development Process**, introduces the Image Recognition Implementation Process using the **TransApplet**.

4. API

This chapter will introduce the interfaces of the important classes. The interface for other classes will be introduced in the later chapters as they are used.

4.1 Signature

Whole Image Matching is done through Image Signature. An image has a set of computed values called features. A collection of features is grouped into a signature.

This section introduces Image Signature interface. An Image Signature consists of the following attributes:

ID
Name (For example, xyz.jpg)
Path (c:\abc\)
Attribute1
Attrabute2
…

The interface for Image Signature is:

```
public interface I_ImageSignature
{
        int getStatus ();
        string getID();
        string getImageName ();
        string getImagePath();
        string getAbsolutePath ();
        int getNumberOfAttributes ();
        int [] getSignatureAttributes();
        int getSignatureAttributes (int index);
        string toString ();
}
```

The following table lists the functions.

Function	Descriptions	Comments
int getStatus ()	Returns the status associated with the signature. Output: 1: signature ready. 0: signature not ready. -1: no image	

	-2: image segmentation specification error -3: other error.	
string getID()	Returns the ID associated with the signature. Output: image ID.	
string getImageName ()	Returns the image name associated with the signature. Output: image name.	
string getImagePath()	Returns the image path associated with the signature. Output: image path.	
string getAbsolutePath ()	Returns the absolute path associated with the signature. Output: image absolute path.	
int getNumberOfAttributes ()	Returns the number of attributes associated with the signature. Output: number of attributes.	
int [] getSignatureAttributes()	Returns the attribute array associated with the signature. Output: attribute array.	
int getSignatureAttributes() (int index)	Returns the attribute associated with the input index. Input: index. Output: attribute associated with the input index.	
string toString ()	Returns the entire image signature as a string with fields separated by Tab.	

4.2 Signature Filter

Signature filter computes signatures in the **TransApplet**: the input is an image and the output is a signature.

The filter has two classes: Signature Generator and Signature Filter. The interface for Signature Generator is:

```
public interface I_SignatureGenerator
{
        bool setSignatureFilter (int x);
        int getSignatureFilter ();

        ImageSignature getSignature (Bitmap b);
        int getSignatureFilterSize ();

        string [] getSignatureFilterNames();
}//class
```

The following table lists the functions.

Function	Descriptions	Comments
ImageSignature getSignature (Bitmap b)	Gets the signature of the input image. Input: Bitmap b Output: Signature.	
int getSignatureFilterSize ()	Gets the number of attributes the library of signatures.	
bool setSignatureFilter (int x) int getSignatureFilter ()	Selects a Signature Generator.	

The interface for Signature Filter is:

```
public interface I_SignatureFilter
    {
        bool setSignatureFilter (int x);
        int getSignatureFilter ();

        string [] getSignatureFilterNames();

        ImageSignature getSignature (string imagePath, string ID);
        ImageSignature getSignature (string imagePath);

        ImageSignature getSignature (Bitmap b, string ID);
        ImageSignature getSignature (Bitmap b);
```

```
        ImageSignature getSignature
                ( string imagePath, string ID,int x, int y, int w, int h);
        ImageSignature getSignature
                ( Bitmap bImg, string path, string name, string ID,
                int x, int y, int w, int h);

        bool getLibrary ( string [] imageAbsolutePath, string fileName );
        bool getLibrary
                ( string [] imageAbsolutePath, string [] IDs, string fileName );

        bool getSegmentLibrary ( string [] imageAbsolutePath,
                string [] IDs,
                string [] xs,
                string [] ys,
                string [] ws,
                string [] hs,
                string fileName );

        bool getSegmentLibrary ( string [] imageAbsolutePath,
                string [] IDs,
                int [] xs,
                int [] ys,
                int [] ws,
                int [] hs,
                string fileName );
    }//class
```

The following table lists the functions.

Function	Descriptions	Comments
bool setSignatureFilter (int x) int getSignatureFilter ()	Selects a Signature filter.	
string [] getSignatureFilterNames();	Gets a list of Signature filter names.	
ImageSignature getSignature (string imagePath, string ID); ImageSignature getSignature (string imagePath); ImageSignature getSignature (Bitmap b, string ID);	Gets the Signature of the input image.	

ImageSignature getSignature (Bitmap b);		
ImageSignature getSignature (string imagePath, string ID,int x, int y, int w, int h); ImageSignature getSignature (Bitmap bImg, string path, string name, string ID, int x, int y, int w, int h);	Gets the Signature of the input image segment. Input: string imagePath, or Bitmap bImg string ID int x, int y, int w, int h. Output: Signature.	
bool getLibrary (string [] imageAbsolutePath, string fileName); bool getLibrary (string [] imageAbsolutePath, string [] IDs, string fileName); bool getSegmentLibrary (string [] imageAbsolutePath, string [] IDs, string [] xs, string [] ys, string [] ws, string [] hs, string fileName); bool getSegmentLibrary (string [] imageAbsolutePath, string [] IDs, int [] xs, int [] ys, int [] ws, int [] hs, string fileName);	Generates a Signature library from all images in string [] imageAbsolutePath and produce a file that contains all signatures. Input: string [] imageAbsolutePath Output: A text file that contains the library of signatures.	

4.3 Matching Results

Attrasoft **ImageFinder** matches whole images or image segments. The **ImageFinder** can be used for:

- **Image Verification (1:1 Matching);**
- **Image Identification (1:N Matching);**
- **Image Search or Retrieval (1:N Matching); and**
- **Multiple Matching (N:N and N:M Matching).**

The results for N:N Matching always goes to a file. The results for 1:1 and 1:N Matching go to a data structure called Results_1N. The interface for class, Results_1N, is:

```
public interface I_Results_1N
        {
                bool getStatus ();

                int getNumberOfMatches();

                string getImageID (int i) ;
                string [] getImageID () ;
                string [] getImageID_N ( int N );

                string getScore(int i);
                string [] getScore();
                string [] getScore_N ( int N );

                string getImageName (int i) ;
                string [] getImageName () ;
                string [] getImageName_N ( int N ) ;

                string getImagePath (int i) ;
                string []getImagePath () ;
                string [] getImagePath_N ( int N ) ;

                string getX(int i);
                string [] getX();
                string [] getX_N ( int N );

                string getY(int i);
                string [] getY();
                string [] getY_N ( int N );

                string getW(int i);
                string [] getW();
                string [] getW_N ( int N );

                string getH(int i);
                string [] getH();
                string [] getH_N ( int N );

                Results_1N sort ();
```

```
        string toString ();
    }
```

The following table lists the functions.

Functions	Descriptions	Comments
int getStatus ()	Returns the status of the current signature comparison: > 0: OK; < 0: Error.	
int getNumberOfMatches()	Returns the number of matches of the current signature comparison.	
string getImageID (int i) string [] getImageID () string [] getImageID_N (int N)	Returns the matching IDs of the current signature.	
string getScore(int i) string [] getScore() string [] getScore_N (int N)	Returns the matching scores of the current signature.	
string getImageName (int i) string [] getImageName () string [] getImageName_N (int N)	Returns the matching Names of the current signature.	
string getImagePath (int i) string []getImagePath () string [] getImagePath_N (int N)	Returns the matching paths of the current signature.	
string getX(int i) string [] getX() string [] getX_N (int N) string getY(int i) string [] getY() string [] getY_N (int N) string getW(int i) string [] getW() string [] getW_N (int N) string getH(int i) string [] getH() string [] getH_N (int N)	Returns the matching (x, y, w, h) of the current signature.	
string toString ()	Returns the entire results as a string with fields separated by Tab.	

4.4 Signature Matching

Both BioFilter and Unsupervised Filter match two whole images. BioFilter is better than Unsupervised Matching, but it requires a process called training. Training teaches the BioFilter who should match with whom. The BioFilter learns how to match the image features.

- The advantage of the BioFilter is that it does not require a lot of training data.
- The disadvantage of the BioFilter is that it has a lower identification rate than the Neural Filter.

NeuralFilter also matches two whole images, which is similar to the BioFilter. NeuralFilter is better than both Unsupervised Filter and BioFilter, but it requires a large amount of training data. Training data teaches the NeuralFilter who should match with whom. In comparison:

- The advantage of the NeuralFilter is that it is more accurate.
- The disadvantage of the NeuralFilter is that it requires more training data than BioFilter.

The interfaces for BioFilter and Unsupervised Filter are basically the same except the Unsupervised Filter does not have the training part. The interfaces for BioFilter and Neural Filter are identical. The interface for BioFilter is:

```
public interface I_BioFilter
{
        bool training ( string a1_txt, string match_txt);

        Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11
                ( Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1,
                Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2);

        Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11
                (string path1, string path2);
        Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11
                (Bitmap left, Bitmap right);

        Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N
                (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig,
                 string a1File, string b1File);
        Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N
                (string keyuPath, string a1File, string b1File);
        Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N
                (Bitmap keyImage, string a1File, string b1File);

        bool findMatchNN (string a1File, string b1File);
        bool findMatchNM (string a1File, string a2File, string b1File);
}
```

The following table lists the functions.

Functions	Descriptions
bool training (string a1_txt, string match_txt)	Trains the BioFilter.
Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1, Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (string path1, string path2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Bitmap left, Bitmap right);	Makes a 1:1 matching
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig, string a1File, string b1File); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string keyuPath, string a1File, string b1File) Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Bitmap keyImage, string a1File, string b1File)	Makes a 1:N matching
bool findMatchNN (string a1File, string b1File); bool findMatchNM (string a1File, string a2File, string b1File);	Matches all image signatures in file a1File against all image signatures in a1File or a2File and saves the results to b1File.

4.5 ImageLibrary

Dynamic Library allows the library (N images in a 1:N Matching) to be updated via insertion, deletion, and update.

The interface for Image Library filter is:

```
public interface I_ImageLibrary
   {
            string getLibraryID ();
            bool setLibraryID (string x);

            bool load ();
            bool load ( string fileName);
            bool load ( string fileName1,  string fileName2);
```

```
        bool print ( );

        bool clear();

        bool backup ();
        bool backup (string fileName);

        bool addSignature (ImageSignature sig);

        bool deleteSignature (ImageSignature sig);
        bool deleteSignature (string ID);
        bool replaceSignature (ImageSignature sig);
        bool mergeLibrary (string libFile1, string libFile2, string outputLib);

}
```

The following table lists the functions.

Functions	Descriptions	Comments
int getLibraryID ()	Gets the Library ID (optional).	
void setLibraryID (string x)	Sets the Library ID (optional).	
bool load ()	Loads the default master library, lib1.txt.	
bool load (string fileName)	Loads master library specified by fileName.	
bool load (string fileName1, string fileName2)	Loads two libraries specified by fileName1 and fileName2.	
bool clear()	Clears the current library from RAM only.	
bool backup ()	Saves current library to the default file, lib1_bk.txt.	
bool backup (string fileName)	Saves current library to a back file.	
bool addSignature (ImageSignature sig)	Adds a signature to a loaded image library in RAM.	
bool deleteSignature (ImageSignature sig)	Deletes a signature from a loaded image library in RAM..	
bool deleteSignature (string ID)	Deletes a signature from a loaded image library in RAM..	

bool replaceSignature (ImageSignature sig)	Replaces a signature from a loaded image library in RAM..	
bool mergeLibrary (string libFile1, string libFile2, string outputLib)	Merges two signature libraries into a single library. Input: string libFile1 string libFile2 string outputLib Output: A text file that contains the library of signatures from both input libraries.	

4.6 Matching Engine

The Unsupervised Filter, the BioFilter, and the NeuralFilter all have their own matching engine. The interface is basically identical. In this section, we introduce the Neural Filter Matching Engine.

The interface for Neural Filter Matching Engine is:

```
public interface I_MatchingEngine
{
        int getLowCut();
        void setLowCut(int x);

        int getHighCut();
        void setHighCut(int x);

        void setNeuralOutputType(int x);
        int getNeuralOutputType();

        void setThreshold(int x);
        int getThreshold();

        void setSensitivity(int x);
        int getSensitivity();

        void setBlurring(int x);
        int getBlurring();

        void setUseRelativeScore(int x);
        int getUseRelativeScore();
```

```
            bool setLibrary
            (Attrasoft.TransApplet70.ImageLibrary70.ImageLibrary lib);

            bool setSignature
            (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig);

bool setNFTraining
(Attrasoft.TransApplet70.MatchingEngineTraining70.MatchingEngineTraining70 a_1);

bool setBFTraining
(Attrasoft.TransApplet70.MatchingEngineTrainingBF70.MatchingEngineTrainingBF70 bf70_1);

            Attrasoft.TransApplet70.Results_1N.Results_1N findMatch11
                  ( Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1,
                  Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2);


            Attrasoft.TransApplet70.Results_1N.Results_1N  findMatch ();
            Attrasoft.TransApplet70.Results_1N.Results_1N findMatch
            (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig);
      }
```

The following table lists the functions.

FUNCTIONS	DESCRIPTIONS	Comments
int getLowCut() void setLowCut(int x) int getHighCut() void setHighCut(int x) void setNeuralOutputType (int x) int getNeuralOutputType() void setThreshold(int x) int getThreshold() void setSensitivity (int x) int getSensitivity() void setBlurring (int x) int getBlurring() void setUseRelativeScore (int x) int getUseRelativeScore()	Gets and Sets matching engine parameters.	

bool setLibrary (ImageLibrary lib)	Sets the library to be used in matching signatures.	
bool setSignature (ImageSignature sig)	Sets the signature for matching.	
bool setNFTraining (MatchingEngineTraining70 a_1) bool setBFTraining (MatchingEngineTrainingBF70 bf70_1)	Sets the training objects for matching.	
Results_1N findMatch11 (ImageSignature sig1, ImageSignature sig2);	Gets 1:1 match results for sig1 and sig2.	
Results_1N findMatch ()	Gets the match results for the signature, sig, specified by function, setSignature (ImageSignature sig) Output: Results_1N.	
Results_1N findMatch (ImageSignature sig)	Gets the match results for the signature, sig. Input: ImageSignature sig; Output: Results_1N.	

4.7 NeuralNet Filter

The NeuralNet Filter matches a segment of an image(s). The interface for the NeuralNet Filter Matching Engine is:

```
public interface I_NeuralNetFilter
{
        bool train (Bitmap img);
        bool train (string sPath);
        bool train (Bitmap img, int x, int y, int w, int h);
        bool train (string sPath, int x, int y, int w, int h);

        bool retrain (Bitmap img);
        bool retrain (string sPath);
        bool retrain (Bitmap img, int x, int y, int w, int h);
        bool retrain (string sPath, int x, int y, int w, int h);

        Attrasoft.TransApplet70.Results_1N.Results_1N
```

```
            findMatch11 ( Bitmap img1);
        Attrasoft.TransApplet70.Results_1N.Results_1N
            findMatch11 ( string path1);
        Attrasoft.TransApplet70.Results_1N.Results_1N
            findMatch1N ( string [] fileList);

        bool findMatchNN (string [] fileList, string c1File);
        bool findMatchNM (string [] keyList, string [] libraryList, string c1File);
}
```

The following table lists the functions.

Functions	Descriptions
bool train (Bitmap img) bool train (string sPath) bool train (Bitmap img, int x, int y, int w, int h) bool train (string sPath, int x, int y, int w, int h)	Trains the neural net.
bool retrain (Bitmap img) bool retrain (string sPath) bool retrain (Bitmap img, int x, int y, int w, int h) bool retrain (string sPath, int x, int y, int w, int h)	Retrains the neural net.
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch11 (Bitmap img1); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch11 (string path1); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string [] fileList);	Makes a 1:N Matching.
bool findMatchNN (string [] fileList, string c1File) bool findMatchNM (string [] keyList, string [] libraryList, string c1File);	Makes a N:N Matching and N:M Matching.

4.8 Other API

We will introduce other API's in later chapters as we encounter them. These classes include:

- Input (Chapter 6)
- Image List (Chapter 7)
- Image Preprocessing (Chapter 8)
- Image Processing (Chapter 9)
- Normalization (Chapter 10)
- Database Input (Chapter 20)
- Video Input (Chapter 21 & 22)
- Parameters (Chapter 11 & 18)
- Batch (Chapter 25)

- Counting (Chapter 23 & 24)
- Tracking (Chapter 23)
- …

5. User Interface

This User's Guide is written as if you will create your own version of the **ImageFinder**. We will do a step-by-step instruction for implementing an "ImageFinder". Eventually, you will have your own "ImageFinder".

This chapter briefly describes how to set up the form and menu bar for the **ImageFinder.** The **ImageFinder** GUI is divided into 3 parts:

- Input/Output
- Menu Items
- Parameters

We will use C# .Net as the implementation language. We will build the user interface form with a Menu Bar and Input buttons.

5.1 How to Create a Project

To create a new C# project:

1. Start the Visual Studio .Net, (see Figure 5.1).
2. Click File ➔ New Project command. The New Project dialog box is displayed (Figure 5.2).
3. Language and Templates: Highlight the Visual C# project folder in the Project Type list to display the templates that are available for C#. Then, highlight the Windows Application template (Figure 5.2).
4. Location and Name: Enter a name for the project and select the location for the project. A folder with the same name as the project is automatically added to the location you specify. We will use chap4 as the project name.
5. Click the 'OK' button to start the new project (Figure 5.3).

Now go to the property window and set (Figure 5.4):

- Form1.text to "ImageFinder";
- Resize Form1.

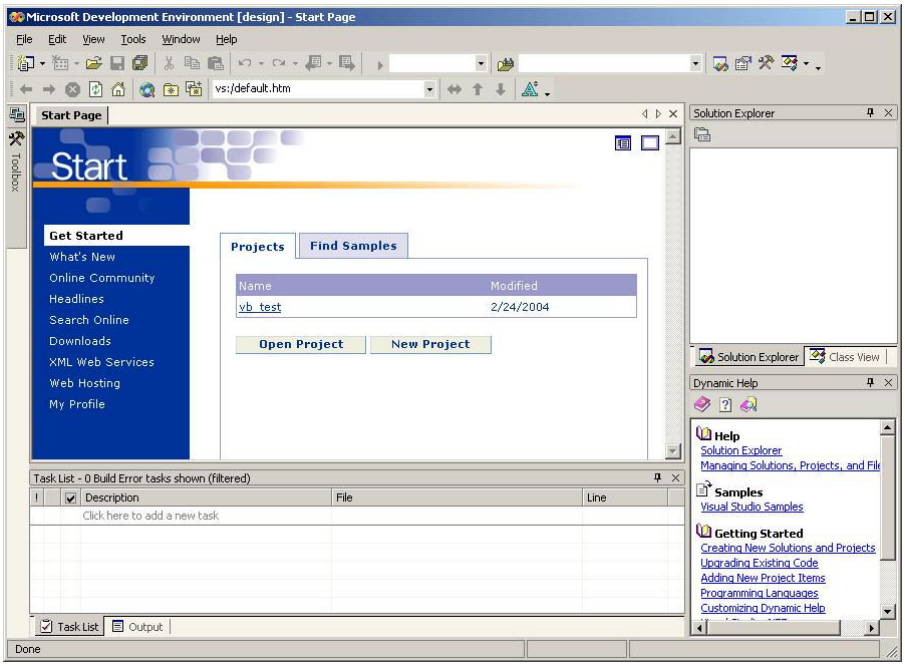

Figure 5.1 Create C# Project.

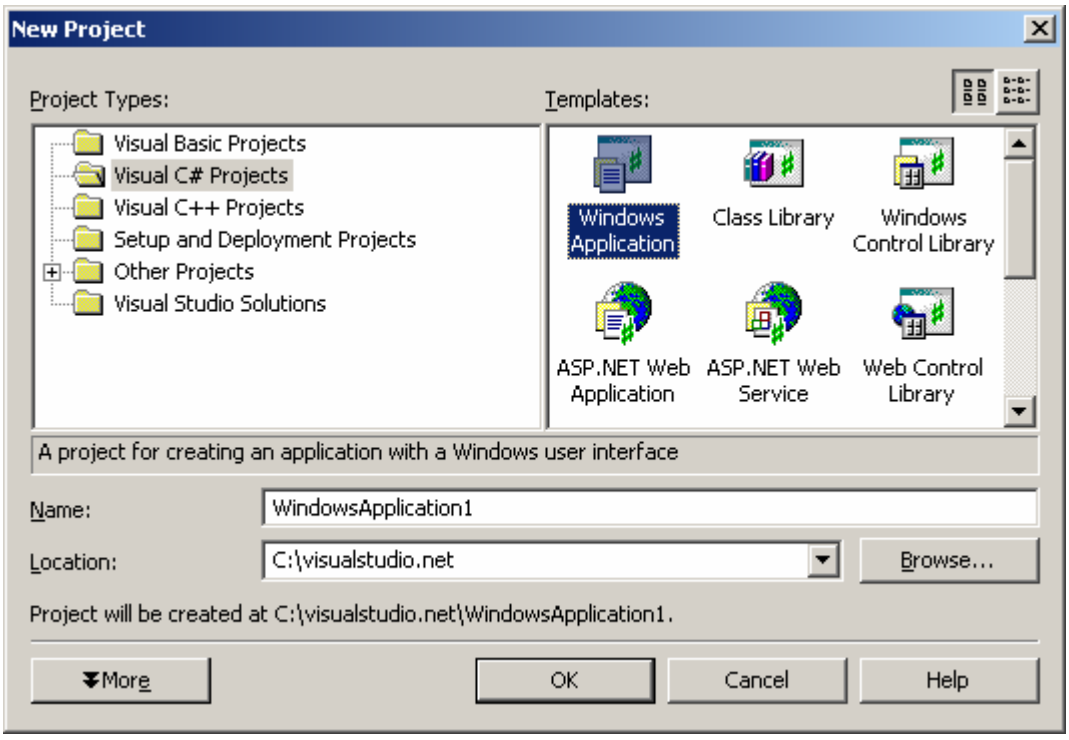

Figure 5.2 Select Project Templates.

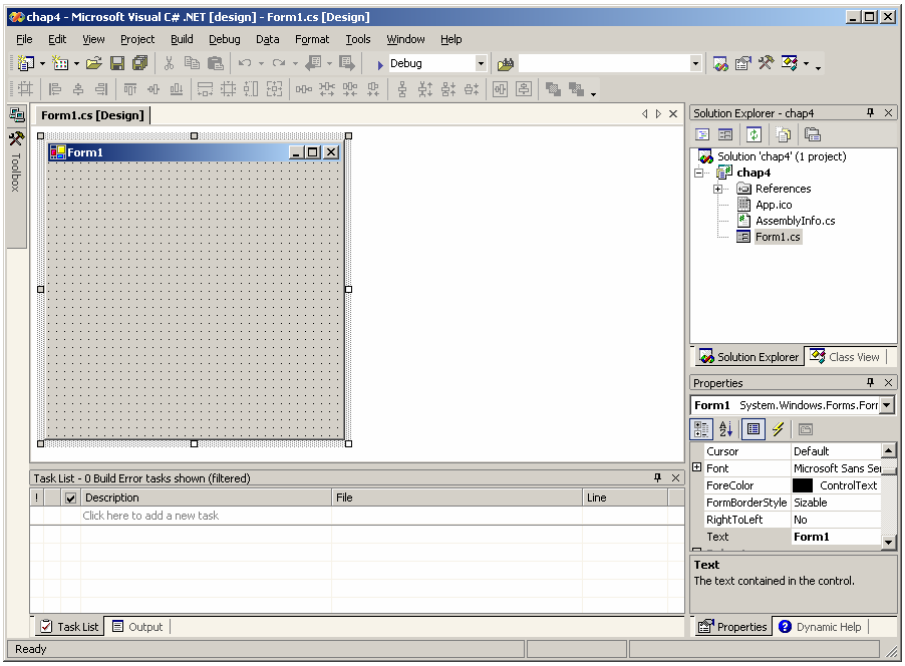

Figure 5.3 Form1.

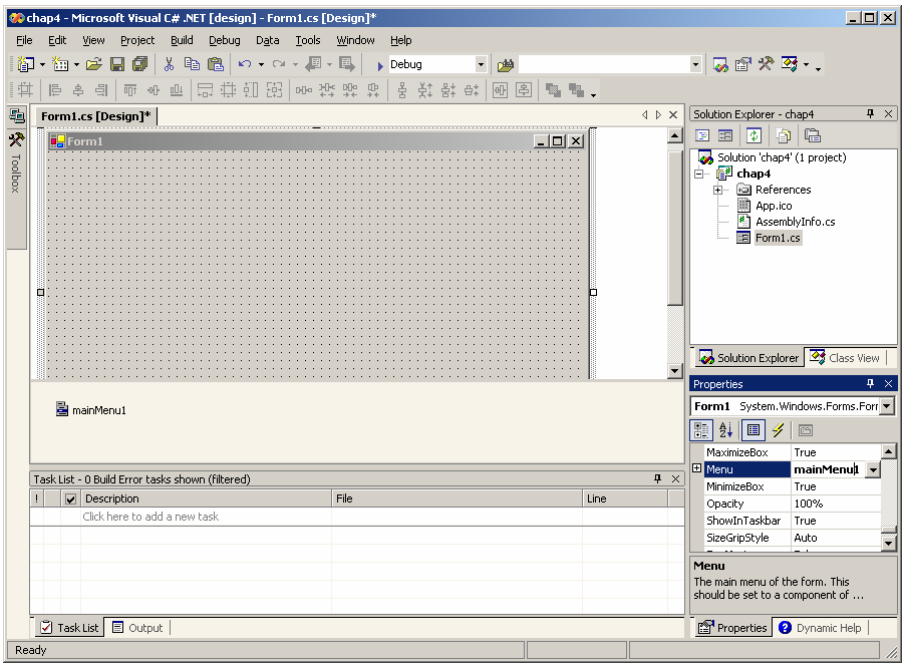

Figure 5.4 Set Form1.Size.

5.2 How to Create Menus

To add a menu bar to a form:

1. Add a “MainMenu” control to a form from the toolbox. The control will appear in the component Design Tray at the bottom of the Form Design window. The menu of a form is

determined by the Menu property. The menu property of the form will be set to the name of the "MainMenu" control you just added.

2. To add items to the menu, click the "MainMenu" control and click wherever it says "Type Here" in the menu designer. Then, type the text of the menu item and press the Enter key. Additional entry areas appear below and to the right of the entry.
3. To edit the property for each menu item, right-click the menu and select the Property command.

5.3 Link to Class Library

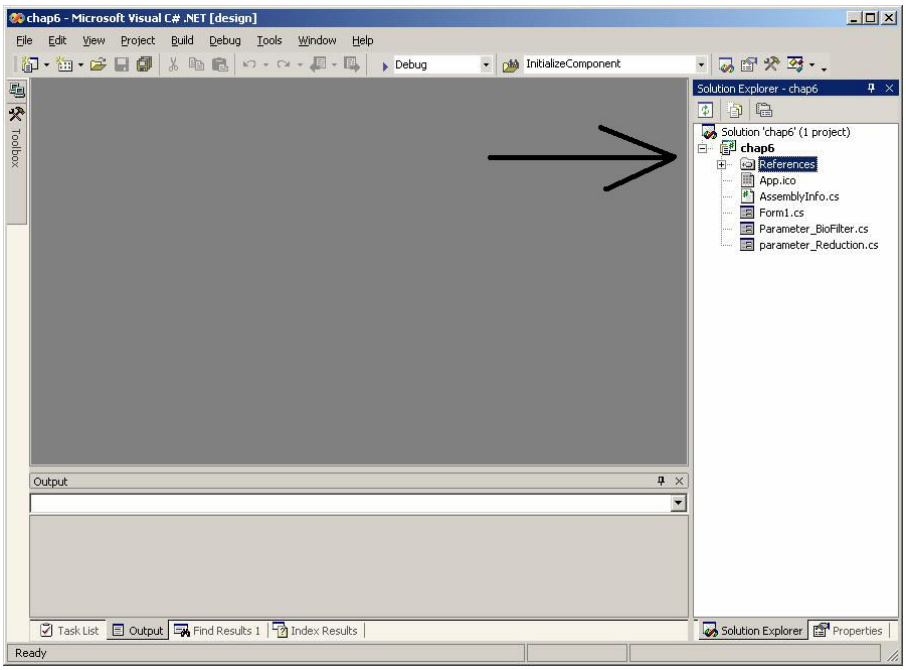

Figure 5.5 References In Solution Explorer.

In this section, XYZ denotes any class located in c:\transapplet70\. To use the class library, "Attrasoft.TransApplet70.XYZ":

(1) Add a "using" statement:

```
using Attrasoft.TransApplet70.XYZ;
```

(2) Right click References and select Add reference in the Solution Explorer (Figure 5.5);
(3) Use the Browse button to find "XYZ.dll"; highlight the library under "Selected Components" and click the "OK" button.

Now the class library, "Attrasoft.TransApple70.XYZ", is ready to be used. To make sure the link is successful, you should see XYZ under References in the Solution Explorer.

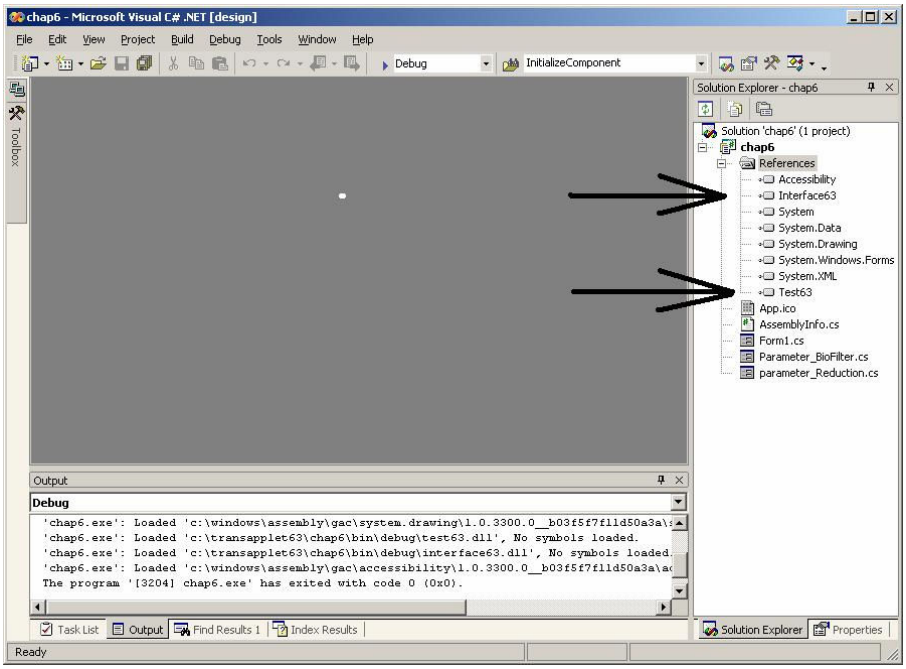

Figure 5.6 Solution Explorer after linking a class library.

5.4 Declare Objects

Each Library has one main class, which has the same name as the Library-name:

Attrasoft.transapplet70.Batch70.Batch70
Attrasoft.transapplet70.BioFilter70.BioFilter70
…

<u>The advantage</u> of naming the class-name as the library-name is that you will know what to use, i.e.

- if you go to Library, "Attrasoft.transapplet70.Batch70", use Batch70 class;
- if you go to Library, "Attrasoft.transapplet70.BioFilter70", use BioFilter70 class, …

There is no guessing which class you should choose if the Library has more than one class.

<u>The disadvantage</u> is the compiler will not be able to distinguish between the class-name and the Library-name unless you use the full name.

To make a long story short, declare all **TransApplet** objects using the full class-name like this:

```
public Attrasoft.TransApplet70.Test70.ImageProcessing70 ip70
        = new Attrasoft.TransApplet70.Test70.ImageProcessing70 ();
public Attrasoft.TransApplet70.Test70.BioFilter70 bf70
        = new Attrasoft.TransApplet70.Test70.BioFilter70 ();
```

6. Input

This chapter describes how to enter data to the **ImageFinder.** The input images are further divided into:

- Key
- Search Source.

The Search Source is further divided into:

- Search-Directory (A directory containing images to be searched);
- Search-File (A file containing the images to be searched);
- Sub-Directory (A directory containing sub-directories with images to be searched);
- Segment-File (A file containing image segments to be searched);
- …

An Image Path specifies the training image. A Search-Folder or Search-File specifies the Search Source.

This chapter will introduce the class library, "Attrasoft.transapplet70.Input70". The chapter project is in "c:\transapplet70\ transapplet70.chap6".

6.1 Class Library Name

The class library is:

Attrasoft.TransApplet70.Input70.

The main class in this library will be:

Attrasoft.TransApplet70.Input70.Input70.

The Input70 interface is:

```
public interface I_Input
{
        string [] getDirList ( string sInput);

        string [] getSubDirList ( string sInput);

        string [] getFileList ( string sInput);
        string []  getFileSegmentList ( string sInput);

        string [] getAccessList ( string sInput, string sSQL);
        string []  getAccessSegmentList ( string sInput, string sSQL);
```

```
        string [] getID ( );
        string [] getName ( );
        string [] getPath ( );
        string [] getAbsolutePath ( );
        string [] getX ();
        string [] getY ();
        string [] getW ();
        string [] getH ();
}
```

6.2 Class Library Overview

The functions in Input70 class are listed in the following table.

Function	Description
string [] getDirList (string sInput)	Gets a string list of the absolute paths of all images in directory, sInput.
string [] getSubDirList (string sInput)	Gets a string list of the absolute paths of all images in all sub-directories of sInput.
string [] getFileList (string sInput)	Gets a string list of the absolute paths of all images in file, sInput.
string [] getFileSegmentList (string sInput)	Gets a string lists of the absolute paths of all images in file, sInput. This command will also populate other arrays so they can be obtained through the following function: string [] getID ( ); string [] getName ( ); string [] getPath ( ); string [] getAbsolutePath ( ); string [] getX (); string [] getY (); string [] getW (); string [] getH ().
string [] getAccessList (string sInput, string sSQL)	Gets a string list of the absolute paths of all images in access file, sInput, specified by a SQL statement, sSQL.
string [] getAccessSegmentList (string sInput, string sSQL)	Gets a string list of the absolute paths of all images in access file, sInput, specified by a SQL statement, sSQL. This command will also populate other arrays so they can be obtained through the following function: string [] getID ( ); string [] getName ( ); string [] getPath ( ); string [] getAbsolutePath ( );

	string [] getX (); string [] getY (); string [] getW (); string [] getH ().
string [] getID ( ) string [] getName ( ) string [] getPath ( ) string [] getAbsolutePath ( ) string [] getX () string [] getY () string [] getW () string [] getH ()	Gets a string list of the ID, Name, Path, Absolute Path, X, Y, W, and H of all images in a search source.

6.3 Set Up the Chapter Project

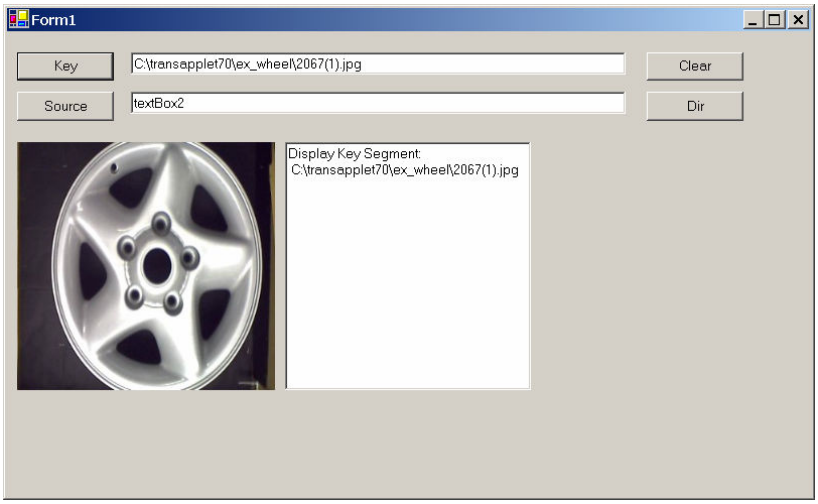

Figure 6.1 Chapter 6 Project.

We will add the following controls to our form:

Key Button, Key Text Box

Use the "Key" button to display the selected key image and print the image path in the key text box. For example, click the button, go to the directory, "C:\transapplet70\ex_wheel", and then select an image. The selected image will be shown in the **ImageFinder** to indicate the specification is successful and the key-image is ready for training or retraining.

Source Button, Source Text Box

Use the "Source" button to select a search source and display the source in the text box.

Clear Button

Use the "Clear" button to clear the rich text box.

Type Button

Use the “Type” button to select a search source type, which includes:
string [] strMode = { "Dir", "File", "Sub Dir", "F Segment" };

Rich Text Box

Use Rich Text Box to display messages.

Left Picture Box

Use the Left Picture Box to display key images.

Right Picture Box

Use the Right Picture Box to display images in the search source.

After adding these controls, the form will look like Figure 6.1.

First, we want to add an Open File Dialog to the chapter 6 project. To add Open File Dialog to a form:

- Add “OpenFileDialog” control to a form from the toolbox. The control will appear in the component Design Tray at the bottom of the Form Design window.
- Click the proper buttons to add the code, which will be the next section.

6.4 Link to Class Library

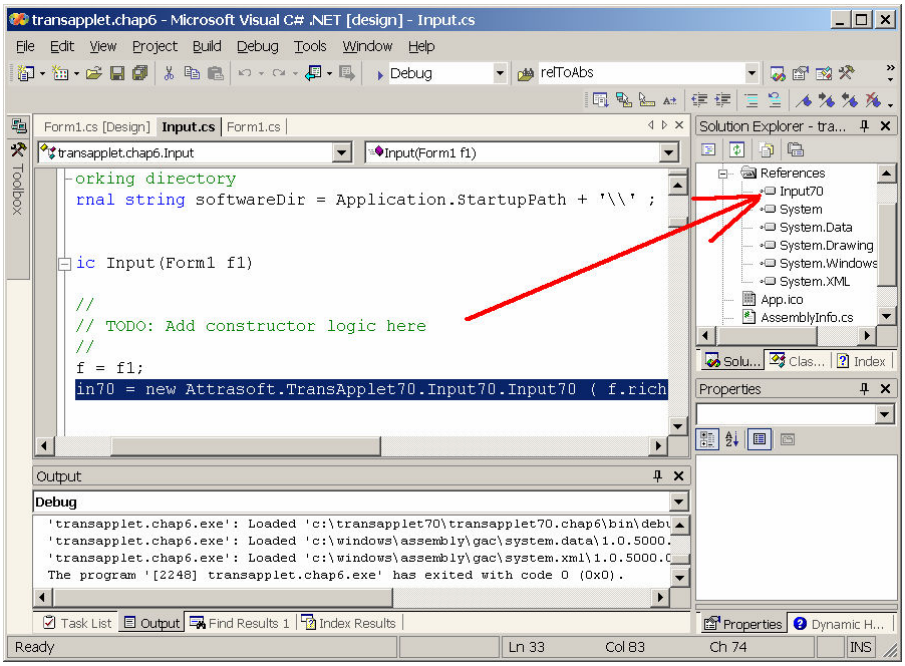

Figure 6.2 Link to Class library Input70.

To include the class library to the project,

- Right click References and select Add Reference in the Solution Explorer;
- Browse to find “Input70.dll” in “c:\transapplet70”;
- Highlight it and click the “OK” button (Figure 6.2).

To use the class library, add:

```
using Attrasoft.TransApplet70.Input70;
```

To declare an object, please use the full path for class:

```
public Attrasoft.TransApplet70.Input70.Input70 in70;
in70 = new Attrasoft.TransApplet70.Input70.Input70 ( f.richTextBox1 );
```

Now, the Input70 object, in70, is ready to use.

6.5 Key Segment

We will add a class, “Input”, in the project. We will implement the input buttons in this class. In this way, the main class is simpler. We will declare an object, input, as follows:

```
Input input;
public Form1()
        {
                InitializeComponent();
                input = new Input (this);
        }
```

We can double click the Key Segment buttons to complete the programming:

```
private void button1_Click(object sender, System.EventArgs e)
        {
                input.keySegment();
        }
```

which in turn, calls a function in the following class:

```
public class Input
{
        Form1 f;
        public Attrasoft.TransApplet70.Input70.Input70 in70 ;
        internal Bitmap bTrain;
        internal string softwareDir = Application.StartupPath + '\\' ;

        public Input(Form1 f1)
        {
                f = f1;
                in70 = new Attrasoft.TransApplet70.Input70.Input70
```

```
                ( f.richTextBox1 );
        }

        void appendText (string s)
        {
                f.richTextBox1.AppendText ( s);
        }

        public bool keySegment()
        {
                if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                        return false;

                string fileName = f.openFileDialog1.FileName;
                try
                {
                        bTrain =new Bitmap (fileName);
                        f.pictureBox1.Image = bTrain;
                        f.textBox1.Text = fileName ;
                        f.richTextBox1.AppendText
                        ( "Display Key Segment:\n " + fileName +"\n");
                }
                catch
                {
                        appendText ("Invalid key image !\n");
                        return false;
                }
                return true;
        }//keySegment()
}
```

The first section of code opens a file dialog so the user can select a file:

```
if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
        return false;
```

The next section of code gets the selected key:

```
string fileName = f.openFileDialog1.FileName;
```

The last section of code creates an image and displays the image:

```
bTrain =new Bitmap (fileName);
f.pictureBox1.Image = bTrain;
f.textBox1.Text = fileName ;
f.richTextBox1.AppendText ( "Display Key Segment:\n " + fileName +"\n");
```

6.6 Search Source Button

The search source can be:

- Search-Directory,
- Search-File,
- Sub-Directory, and
- Segment-File.

For more information, please see the **ImageFinder** User's Guide. The "Type" button specifies the search-source:

```
string [] strMode = { "Dir", "File", "Sub Dir", "F Segment" };
internal int iMode = 0;

private void button3_Click(object sender, System.EventArgs e)
        {
                iMode = (iMode + 1 )% strMode.Length ;

                if (iMode == 0 )
                        button3.Text = strMode[0];
                else if (iMode == 1 )
                        button3.Text = strMode[1];
                else if (iMode == 2 )
                        button3.Text = strMode[2];
                else if (iMode == 3 )
                        button3.Text = strMode[3];
}
```

Now, we implement the "Source" button:

```
private void button2_Click(object sender, System.EventArgs e)
        {
                input.searchSource( iMode );
        }
```

which in turn, calls the following function in the Input class:

```
public void searchSource (int iMode)
        {
                if ( iMode == 0 )
                        searchSource0 ();
                else if ( iMode == 1 )
                        searchSource1 ();
                else if ( iMode == 2 )
                        searchSource2 ();
                else if ( iMode == 3 )
                        searchSource3 ();
                else
```

```
                searchSource0 ();
        }//searchSource
```

The search source can be:

- Search-Directory,
- Search-File,
- Sub-Directory, and
- Segment-File.

For Search-Directory, the implementation is:

```
    private void searchSource0 ()//dir
    {
        if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
            return;
        string cDir = System.IO.Directory.GetCurrentDirectory();

        f.textBox2.Text = cDir;
        f.richTextBox1.AppendText ( cDir +"\n" );

        filelist = in70.getDirList(cDir);

        if ( filelist == null )
            return;
        if ( filelist.Length == 0 )
            return;

        f.richTextBox1.Text ="";
        for (int i= 0; i < filelist.Length ; i++ )
            f.richTextBox1.AppendText ("" + i + " " +filelist[i] +"\n" );

        f.pictureBox2.Image = new Bitmap  (filelist[0] );
        f.richTextBox1.AppendText ( "Display the first image!\n" );
    }
```

The first section of code gets the current search directory via the open file dialog:

```
    if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
        return;
    string cDir = System.IO.Directory.GetCurrentDirectory();
```

The next section gets a string list of the image paths:

```
    filelist = in70.getDirList(cDir);
```

The third section prints the string list of the image paths:

```
    f.richTextBox1.Text ="";
```

```
for (int i= 0; i < filelist.Length ; i++ )
        f.richTextBox1.AppendText ("" + i + " " +filelist[i] +"\n" );
```

The last section displays the first images:

```
f.pictureBox2.Image = new Bitmap  (filelist[0] );
```

To test, run the software and click the first image in folder, “c:\transapplet70\ex_wheel”.

Now, we implement Search-File:

```
private void searchSource1 ()//file
{
        if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                return;
        string cDir = f.openFileDialog1.FileName ;

        f.textBox2.Text = cDir;
        f.richTextBox1.AppendText ( cDir +"\n" );

        filelist = in70.getFileList (cDir);

        if ( filelist == null )
                return;
        if ( filelist.Length == 0 )
                return;

        f.richTextBox1.Text ="";
        for (int i= 0; i < filelist.Length ; i++ )
            f.richTextBox1.AppendText ( "" + i + " " +filelist[i] +"\n" );

        f.pictureBox2.Image = new Bitmap  (filelist[0] );
        f.richTextBox1.AppendText ( "Display the first image!\n" );
}
```

There is only one line difference between Search-Directory and Search-File:

```
filelist = in70.getFileList (cDir);
```

The rest of the code is identical to Search Directory.

To test, run the software and select file, “c:\transapplet70\ex_wheel\input_file1.txt”.

Now, we implement Sub-Directory and File-Segment:

```
private void searchSource2 ()//sub
{
        if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                return;
```

```
        string cDir = System.IO.Directory.GetCurrentDirectory();

        f.textBox2.Text = cDir;
        f.richTextBox1.AppendText ( cDir +"\n" );

        filelist = in70.getSubDirList (cDir);
        if ( filelist == null )
            return;
        if ( filelist.Length == 0 )
            return;

        f.richTextBox1.Text ="";
        for (int i= 0; i < filelist.Length ; i++ )
           f.richTextBox1.AppendText ( "" + i + " " +filelist[i] +"\n" );

        f.pictureBox2.Image = new Bitmap  (filelist[0] );
        f.richTextBox1.AppendText ( "Display the first image!\n" );
}

        private void searchSource3 ()
        {
            if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                return;
            string cDir = f.openFileDialog1.FileName ;

            f.textBox2.Text = cDir;
            f.richTextBox1.AppendText ( cDir +"\n" );

            filelist = in70.getFileSegmentList (cDir);

            if ( filelist == null )
                return;
            if ( filelist.Length == 0 )
                return;

            f.richTextBox1.Text ="";
            for (int i= 0; i < filelist.Length ; i++ )
               f.richTextBox1.AppendText ( "" + i + " " +filelist[i] +"\n" );

            f.pictureBox2.Image = new Bitmap  (filelist[0] );
            f.richTextBox1.AppendText ( "Display the first image!\n" );
        }
```

Again, there is only one line difference between Search-Directory and these two functions.

To test the Sub-Directory, run the software and click the first image in folder, “c:\transapplet70\ex_wheel\”.

To test Segment-File, run the software and select file, “c:\transapplet70\ex_wheel\ input_filesegment1.txt”.

7. Image Display

This chapter describes how to display images in a search source**.** The images are further divided into:

- Search-Directory (A directory containing images to be searched);
- Search-File (A file containing the images to be searched);
- Sub-Directory (A directory containing sub-directories with images to be searched);
- Segment-File (A file containing image segments to be searched);

To display these search images, three buttons will do:

First Button
Use the "First" button to display the first image in the search source.
> (Next) Button
Use the "Next" button to display the Next image in the search source.
< (Previous) Button
Use the "Previous" button to display the Previous image in the search source.

After adding the three buttons, the form looks like Figure 7.1.

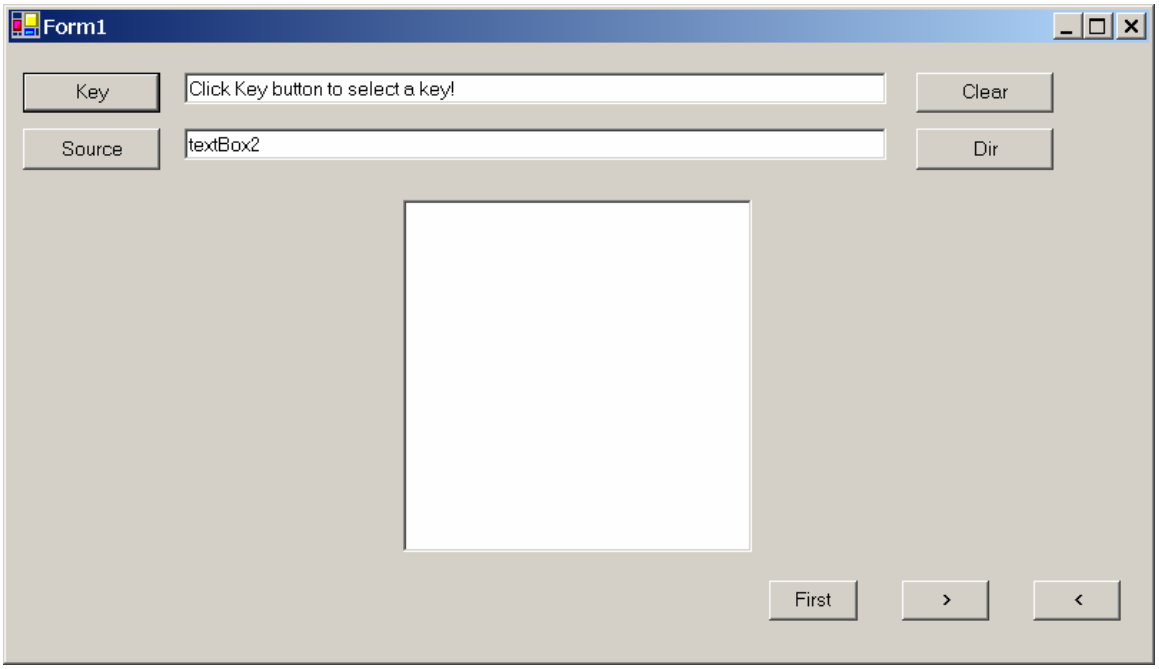

Figure 7.1 First, Next, and Previous Buttons.

7.1 Class Library Name

The class library is:

Attrasoft.TransApplet70.ImageList70.

The class in this library will be:

Attrasoft.TransApplet70. ImageList70. ImageList70.

The functions in this class are:

Function	Description
void setList (string [] seachDirList1)	Sets an input list of image paths.
bool getStatus ()	Gets the status.
string getFirst ()	Gets the first image path.
string getNext ()	Gets the next image path.
string getPrevious ()	Gets the previous image path.

In the last chapter, we obtained the string list of the image path. Basically, we will enter this list to an ImageList70 object and then, use the getFirst (), getNext(), and getPrevious () functions to display the images.

7.2 Link to Class Library

To include the class library to the project:

- Right click References and select Add Reference in the Solution Explorer;
- Browse to find "ImageList70.dll" in "c:\transapplet70\";
- Highlight it and click the "OK" button.

To use the class library, add:

using Attrasoft.TransApplet70.ImageList70;

To declare an object, please use the full path for class:

```
Attrasoft.TransApplet70.ImageList70.ImageList70 imageAbsoultePathList
= new Attrasoft.TransApplet70.ImageList70.ImageList70 ();
```

7.3 Implementing Buttons

In this section, we will implement the First, Next, and Previous buttons.
To see images in a search source, start with the "First" button and then, keep clicking the "Next" button to display the next image; eventually, all of the images in the search source will be shown.

The code for the three buttons consists of three parts:

- Populating ImageList70 Object;

- Functions for buttons, which in turn will call functions in the Input objects;
- Implementation functions in the Input objects.

First of all, the ImageList70 object has to be populated. This can be accomplished by adding a single line of code in the last chapter:

```
private void searchSource0 ()
        {
                …
                f.pictureBox2.Image = new Bitmap  (filelist[0] );
                f.richTextBox1.AppendText ( "Display the first image!\n" );

                imageAbsoultePathList.setList ( filelist );
        }

private void searchSource1 ()
        {
                …
                f.pictureBox2.Image = new Bitmap  (filelist[0] );
                f.richTextBox1.AppendText ( "Display the first image!\n" );

                imageAbsoultePathList.setList ( filelist );
        }

private void searchSource2 ()
        {
                …
                f.pictureBox2.Image = new Bitmap  (filelist[0] );
                f.richTextBox1.AppendText ( "Display the first image!\n" );

                imageAbsoultePathList.setList ( filelist );
        }

private void searchSource3 ()
        {
                …
                f.pictureBox2.Image = new Bitmap  (filelist[0] );
                f.richTextBox1.AppendText ( "Display the first image!\n" );

                imageAbsoultePathList.setList ( filelist );
        }
```

Secondly, the code for the “First” button is:

```
private void button5_Click(object sender, System.EventArgs e)
        {
                input.firstButton( iMode );
```

```
        }
```

Similarly, the code for the “Next” button and the “Previous” button are:

```
    private void button6_Click(object sender, System.EventArgs e)
        {
            input.nextButton ( iMode );
        }

    private void button7_Click(object sender, System.EventArgs e)
        {
            input.previousButton ( iMode );
        }
```

Finally, in the Input class, these functions are implemented:

```
    public void firstButton (int iMode)
        {
            try
            {
            appendText ( imageAbsoultePathList.getFirst ()+"\n"   );
            bSearch =new Bitmap ( imageAbsoultePathList.getFirst () );
            f.pictureBox2.Image = bSearch;
            }
            catch
            {
                appendText ("Invalid image!\n");
                return;
            }
        }

    public void nextButton (int iMode)
        {
            try
            {
            appendText ( imageAbsoultePathList.getNext  () +"\n"   );
            bSearch =new Bitmap ( imageAbsoultePathList.getNext  () );
            f.pictureBox2.Image = bSearch;
            }
            catch
            {
                appendText ("Invalid image!\n");
                return;
            }
        }

        public void previousButton (int iMode)
        {
            try
```

```
        {
        appendText ( imageAbsoultePathList.getPrevious () +"\n"  );
        bSearch =new Bitmap ( imageAbsoultePathList.getPrevious () );
        f.pictureBox2.Image = bSearch;
        }
        catch
        {
                appendText ("Invalid image!\n");
                return;
        }
}
```

7.4 Test

This section tests the project:

(1) Run the project;
(2) Click the “Key” button, go to “c:\transapplet70\ex_wheel\”, and select a key image;
(3) Click the “Source” button, go to “c:\transapplet70\ex_wheel\”, and select an image.
(4) Click the “First” button;
(5) Click the “Next” button several times;
(6) Click the “Previous” button several times.

7.5 Output Images

In the last a few sections, we demonstrated how to display Input Source images. After a matching, the output images will be displayed in a similar fashion. We will enter a string list of matched images to an ImageList70 object and then, use the getFirst (), getNext(), and getPrevious () functions to display the images.

8. Image Preprocessing

Attrasoft ImageFinder learns an image in a way similar to human eyes:

- Ignore the background;
- Focus on an object in the image.

The Image Preprocessing in this chapter and the Image Processing in the next chapter prepare the image for the **ImageFinder**. The Image Processing process is not unique; there are many options available. Some are better than others.

The principle of choosing the Image Preprocessing and Image Processing filters is to make the **sample objects stand out, otherwise change the options**.

Image Preprocessing has the following functions:

- Cut off the border areas;
- Impose a mask;
- Speed up the computation;
- Skip the empty border areas;

If you do not have a good Image Preprocessing/Processing filter in the off-the-shelf **ImageFinder**, **a customized filter has to be built**.

Do not make too many things stand out, i.e. as long as the area of interest stands out, the rest should show as little as possible.

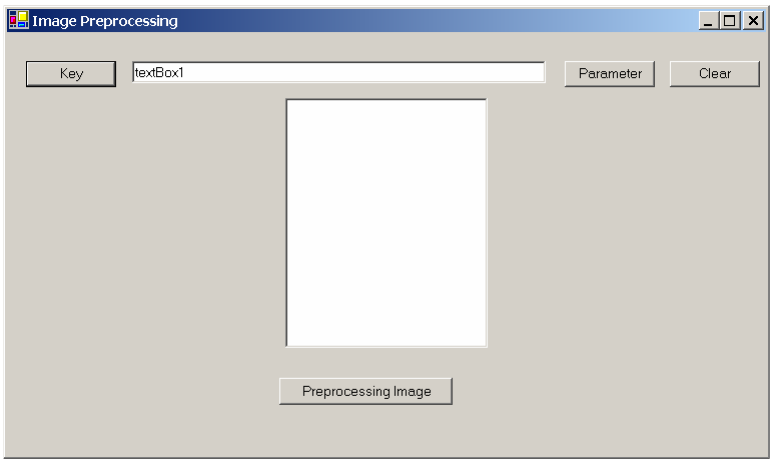

Figure 8.1 Chapter 8 project.

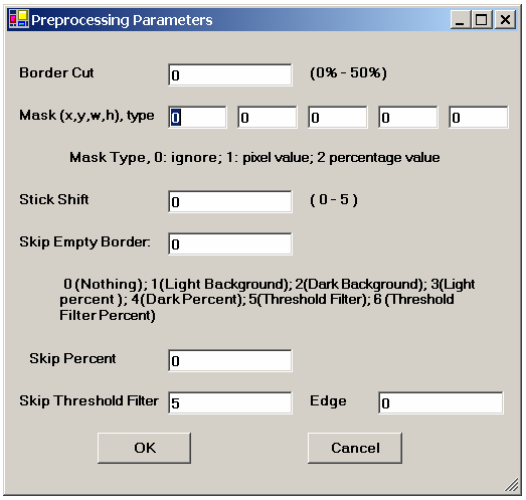

Figure 8.2 Image Preprocessing Parameters.

8.1 Image Preprocessing Filter

To set the Image Preprocessing Filter parameters, click the Parameter button in Figure 8.1; then click the Parameter button again, and you will see Figure 8.2. You will set the Image Preprocessing parameters on Figure 8.2. The "OK" button in Figure 8.2 enters the parameters to the Preprocessing filter.

To see the effect of the Image Preprocessing and Image Processing filters, you have to select a key image via the Key button. The image will be displayed on the left. Then click the "Preprocessing Image" button to apply the preprocessing filter and see the processed image on the right.

8.2 PreProcessing API

The Image Preprocessing interface is:

```
public interface I_ImagePreProcessing70
{
        int getMaskX();
        void setMaskX(int x);
        int getMaskY();
        void setMaskY(int x);
        int getMaskW();
        void setMaskW(int x);
        int getMaskH();
        void setMaskH(int x);
        int getMaskType();
        void setMaskType(int x);

        int getBorderCut();
        void setBorderCut(int x);
```

```
        int getStickShift();
        void setStickShift(int x);

        void setSkipEmptySpaceType(int x);
        int getSkipEmptySpaceType();

        void setSkipEmptySpacePercentage(int x);
        int getSkipEmptySpacePercentage();

        void setSkipEmptySpaceThresholdFilter(int x);
        int getSkipEmptySpaceThresholdFilter();

        void setSkipEmptySpaceEdgeFilter(int x);
        int getSkipEmptySpaceEdgeFilter();

        Bitmap getPreProcessingImageBW (Bitmap bImg);
        Bitmap getPreProcessingImageColor (Bitmap bImg);

}
```

The following table lists the functions.

Functions	Description

int getMaskX();
void setMaskX(int x);
int getMaskY();
void setMaskY(int x);
int getMaskW();
void setMaskW(int x);
int getMaskH();
void setMaskH(int x);
int getMaskType();
void setMaskType(int x);
	Sets and gets mask (x, y, w, h) and mask type.

int getBorderCut(); void setBorderCut(int x);	Sets and gets Border Cut.
int getStickShift(); void setStickShift(int x);	Sets and gets Stick Shift.

void setSkipEmptySpaceType(int x);
int getSkipEmptySpaceType();
 	
void setSkipEmptySpacePercentage(int x);
int getSkipEmptySpacePercentage();
 	
void
setSkipEmptySpaceThresholdFilter(int x);
int getSkipEmptySpaceThresholdFilter();
	Sets and gets parameters for Skip Empty Space.

void setSkipEmptySpaceEdgeFilter(int x); int getSkipEmptySpaceEdgeFilter();	
Bitmap getPreProcessingImageBW (Bitmap bImg); Bitmap getPreProcessingImageColor (Bitmap bImg);	Get preprocessed image from an input image, bImg.

To add an image preprocessing object, add:

```
Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 ipre70;

private void Form1_Load(object sender, System.EventArgs e)
        {
        ipre70 = new Attrasoft.TransApplet70.ImagePreProcessing70
                .ImagePreProcessing70 (richTextBox1);
        richTextBox1.Clear ();
        }
```

8.3 Enter Parameters

Clicking the "OK" button on Figure 8.2 will enter the parameters to the Image Pre-processing filter. The "OK" button is implemented as follows:

```
        private void button1_Click(object sender, System.EventArgs e)
        {
        try
        {
                ipre70.setMaskX (int.Parse (textBox1.Text)) ;
                ipre70.setMaskY  (int.Parse (textBox2.Text )) ;
                ipre70.setMaskW  (int.Parse (textBox3.Text )) ;
                ipre70.setMaskH  (int.Parse (textBox4.Text )) ;
                ipre70.setMaskType  (int.Parse (textBox11.Text )) ;

                ipre70.setBorderCut   (int.Parse (textBox5.Text )) ;

                ipre70.setStickShift   (int.Parse (textBox6.Text )) ;

                ipre70.setSkipEmptySpaceType   (int.Parse (textBox7.Text )) ;

                ipre70.setSkipEmptySpacePercentage    (int.Parse (textBox8.Text )) ;

                ipre70.setSkipEmptySpaceThresholdFilter     (int.Parse (textBox9.Text )) ;
                ipre70.setSkipEmptySpaceEdgeFilter      (int.Parse (textBox10.Text )) ;
```

```
            this.Close();
}
catch
{
            MessageBox.Show ("Please enter valid integers", "Entry Error");
}
}
```

8.4 Cut Off the Border Areas

Let us assume we want to cut off 10 % of the border; click the "Parameter" button; and then click the parameter button again. You will see Figure 8.2. Enter 10 to the first textbox in Figure 8.2. Click the "Key" button and select an image from "c:\transapplet70\ex_wheel\". Click the "Preprocessing Image" button and you will have Figure 8.3. You will see that the second image is the first image with 10% of the border cut off.

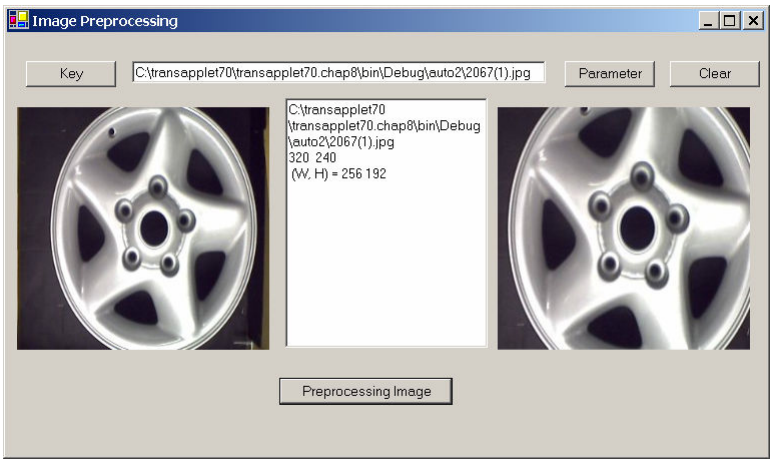

Figure 8.3 Five Percent Border Cut.

The implementation of the "Preprocessing Image" button is:

```
private void button11_Click(object sender, System.EventArgs e)
            {
                        if (b1 == null)
                        {
                                    richTextBox1.AppendText ("Please select an image");
                                    return;
                        }

                        b2 = ipre70.getPreProcessingImageColor  (b1);
                        pictureBox2.Image = b2;
                        richTextBox1.AppendText
```

```
                (" (W, H) = " + b2.Width + " " + b2.Height + "\n");
        }
```

8.5 Impose a Mask

Let us assume we want to cut off 20% of the border on top, 20% on the left, 10% on the right, and 10% on the bottom, then enter (20, 20, 70, 70, 2) to the second row textboxes in Figure 8.2. Click the "Preprocessing Image" button and we will have Figure 8.4.

In Figure 8.4, you will see the second image is the first image with a mask (20%, 20%, 70%, 70%). Here 2 in (20, 20, 70, 70, 2) means in percentage rather than pixels. You will see that the second image is the first image with the following cut: cut off 20 % of the border on top, 20% on the left, 10% on the right, and 10% on the bottom.

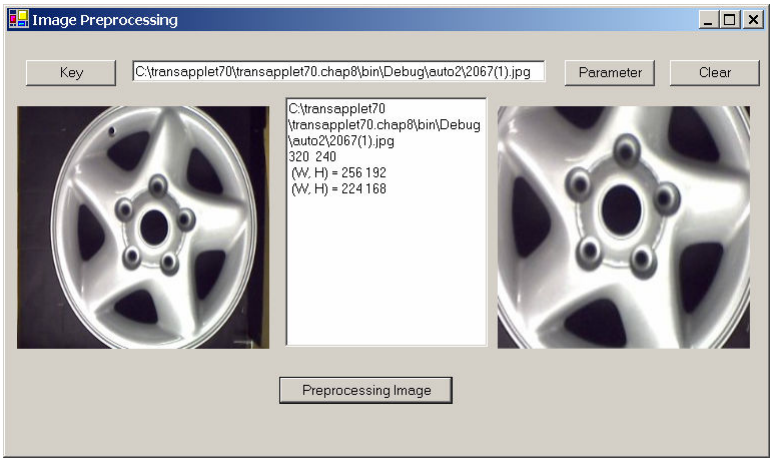

Figure 8.4 Mask (20, 20, 70, 70, 2).

Now, if we use pixels rather than percentage, enter (20, 20, 70, 70, 1) to the second row textboxes in Figure 8.2 and we will have Figure 8.5.

Figure 8.5 Mask (20, 20, 70, 70, 1).

8.6 Speed Up the Computation

To speed up the computation, set the parameter "Stick Shift" in Figure 8.2 between 0 and 5, with 0 being the slowest and 5 being the fastest.

8.7 Skip the Empty Border by Content Percent

The last function for the Preprocessing filter is to skip border by content percent, not pixel percent.

The "Skip Empty Border" field in Figure 8.2 specifies the type:

0 No skip;
1 Skip the white empty border space;
2 Skip the black empty border space;
3 Skip x percent of the contents on the white background space;
4 Skip x percent of the contents on the black background space;
5 Skip empty border space on the user defined Threshold Filter;
6 Skip x percent of the contents on the user defined Threshold/Edge Filters.

We will discuss Threshold/Edge Filters in the next chapter, Image Processing.

Example 1. Set "Skip Empty Border" = 2, means skipping the black empty space on the edge. The result is shown in 8.6; you will see that the second image is the first image without a black border.

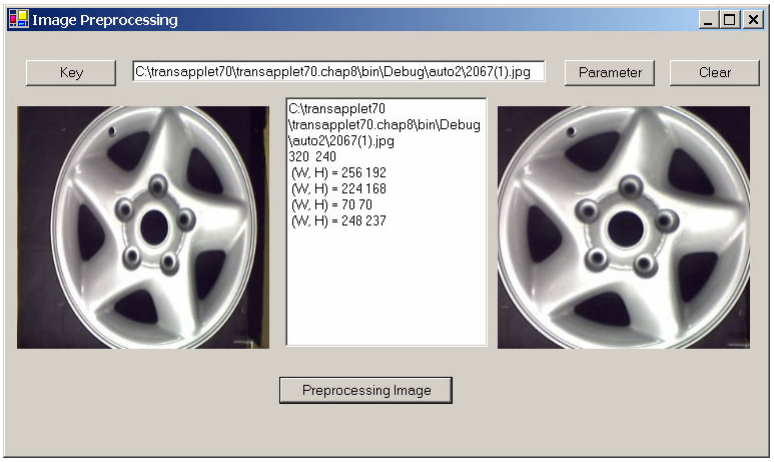

Figure 8.6 Skipping the black empty space.

Example 2. Set "Skip Empty Border" = 4, means skipping the black empty space on the edge. Set "Skip percent" = 10. The result is shown in Figure 8.7; you will see that the second image is the first image with 10% of the contents cut on each side.

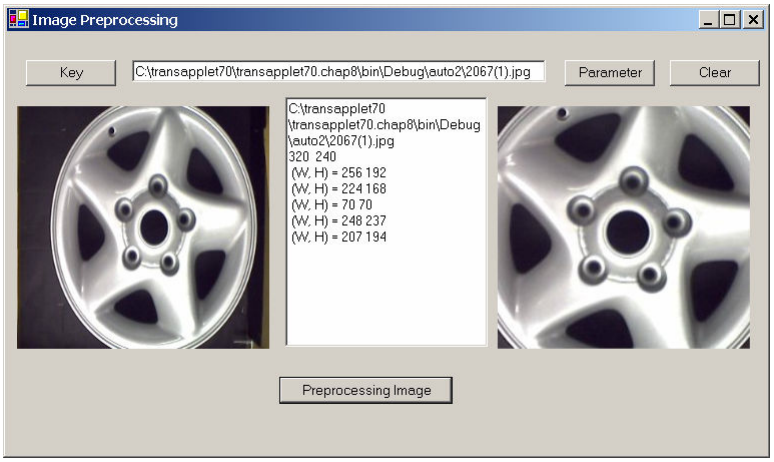

Figure 8.7 Skipping 10% of the contents on the black background.

9. Image Processing

Attrasoft ImageFinder learns an image in a way similar to human eyes:

- Ignore the background;
- Focus on an object in the image.

The Image Preprocessing in the last chapter and the Image Processing in this chapter prepare the image for the **ImageFinder**.

The Image Processing process is not unique; there are many options available. Some are better than others. **Image Processing can make it or break it. For many problems like fingerprints, palm prints, …, special image processing filters will be required.**

9.1 Good & Bad

The principle of choosing the Image Preprocessing and Image Processing filters is to make the **sample objects stand out, otherwise change the options**.

Do not make too many things stand out, i.e. as long as the area of interest stands out, the rest should show as little as possible.

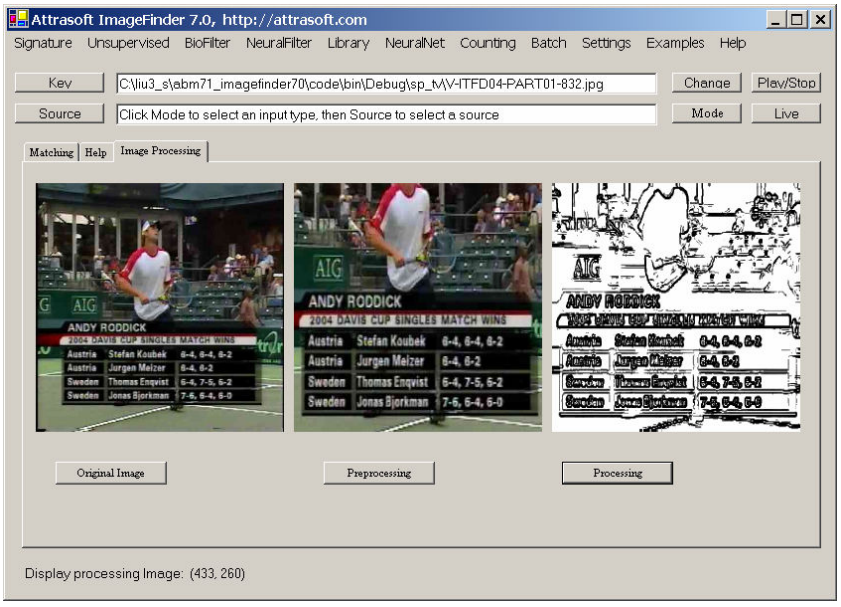

Figure 9.1 Bad Image Processing Example.

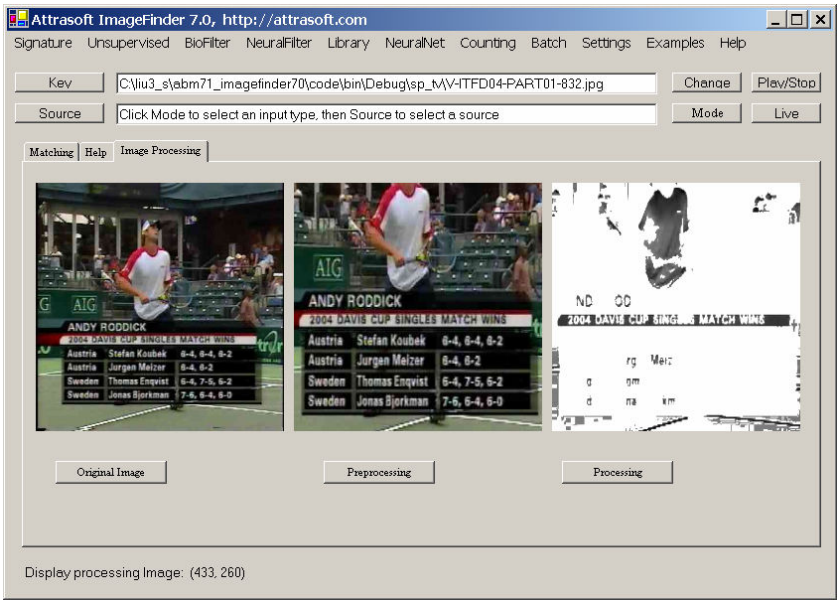

Figure 9.2 Good Image Processing Example.

In Figure 9.1, the first image is the selected key image. The objective is to identify the logo, "2004 Davis Cup …".

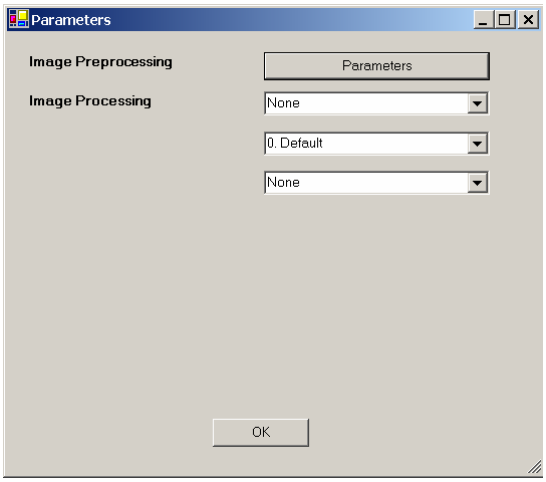

Figure 9.3 Image Processing Setting.

First of all, let us use the setting:

Edge Filter = 2
Threshold Filter = 1
Clean-Up Filter = 2

The second image in Figure 9.1 is a preprocessed image; and the third image in Figure 9.1 is the processed image. This is a bad example of image processing, because the **sample object, "2004 Davis Cup …", does not stand out.**

Now, we will try a different setting:

Edge Filter = 0
Threshold Filter = 1
Clean-Up Filter = 16

The third image in Figure 9.2 is the processed image. This is a good example of image processing, because the **sample object, "2004 Davis Cup …", does stand out.**

9.2 Processing API

The Image Processing interface is:

```
public interface I_ImageProcessing70
{
        string [] getEdgeFilterNames();
        int getEdgeFilterDefaultIndex();
        string [] getThresholdFilterNames();
        int getThresholdFilterDefaultIndex();
        string [] getCleanUpFilterNames();
        int getCleanUpFilterDefaultIndex();

        int getDoubleImageProcessing ();
        void setDoubleImageProcessing (int x);

        bool setImage (Bitmap sImageName );

        bool imageProcess();

        void setEdgeFilter(int x);
        int getEdgeFilter();
        bool getEdgeFiltersStatus();

        void setThresholdFilter (int x);
        int getThresholdFilter ();
        bool getThresholdFiltersStatus();

        void setCleanUpFilter (int x);
        int getCleanUpFilter ();
        bool getCleanUpFilterStatus();

        void setR1(int x);
        int getR1();
        void setR2(int x);
        int getR2();
        void setR3(int x);
        int getR3();
```

```
        void setG1(int x);
        int getG1();
        void setG2(int x);
        int getG2();
        void setG3(int x);
        int getG3();
        void setB1(int x);
        int getB1();
        void setB2(int x);
        int getB2();
        void setB3(int x);
        int getB3();
}//image processing
```

The following table lists the functions.

Functions	Description

string [] getEdgeFilterNames();
int getEdgeFilterDefaultIndex();
string [] getThresholdFilterNames();
int getThresholdFilterDefaultIndex();
string [] getCleanUpFilterNames();
int getCleanUpFilterDefaultIndex();
	Gets the basic info about Edge filter, Threshold filter, and Clean-Up filter.

int getDoubleImageProcessing (); void setDoubleImageProcessing (int x);	Sets and gets the "Double Processing" parameter.
bool setImage (Bitmap sImageName);	Sets the input image for the Image Processing filters, which is also the output image; i.e. the image processing is applied to the image directly.

void setEdgeFilter(int x);
int getEdgeFilter();
bool getEdgeFiltersStatus();
 	
void setThresholdFilter (int x);
int getThresholdFilter ();
bool getThresholdFiltersStatus();
 	
void setCleanUpFilter (int x);
int getCleanUpFilter ();
bool getCleanUpFilterStatus();
	Sets and gets Image Processing filters.

bool imageProcess();	Apply the image-processing filter to the input image. The input image is also the output image, because the filter is applied to the image directly.
void setR1(int x);	Sets and gets Threshold Filter parameters.

int getR1();
void setR2(int x);
int getR2();
void setR3(int x);
int getR3();
void setG1(int x);
int getG1();
void setG2(int x);
int getG2();
void setG3(int x);
int getG3();
void setB1(int x);
int getB1();
void setB2(int x);
int getB2();
void setB3(int x);
int getB3();	

To add an image-processing object, add:

```
Attrasoft.TransApplet70.ImageProcessing70.ImageProcessing70 ip70;
private void Form1_Load(object sender, System.EventArgs e)
        {
        ipre70 = new Attrasoft.TransApplet70.ImagePreProcessing70 .
                ImagePreProcessing70 (richTextBox1);
        ip70 = new Attrasoft.TransApplet70.ImageProcessing70.ImageProcessing70 ();
        richTextBox1.Clear ();
        }
```

9.3 Set Image Processing Filters

The image processing will be applied to all images before recognition. As far as the operation is concerned, this means setting three filters:

Edge Filters;
Threshold Filters; and
Clean-Up Filters.

In Figure 9.3,

- to select an Edge Filter, click the Edge Filter Drop Down List, which is the first List in the red box;
- to select a Threshold Filter, click the Threshold Filter Drop Down List, which is the second List;
- to select a Clean-Up Filter, click the Clean-Up Filter Drop Down List, which is the third List.

The Edge Filters attempt to exaggerate the main features a user is looking for.
The Threshold Filters attempt to suppress the background.

The Clean-Up Filters will smooth the resulting image to reduce recognition error.

The default setting in the **ImageFinder** is:

Edge Filter = 2
Threshold Filter = 1
Clean-Up Filter = 2

The code for the three Drop Down Lists is:

```
private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
        {
        f.ip70.setEdgeFilter (comboBox1.SelectedIndex);
        f.richTextBox1.AppendText ( "Set EdgeFilter " +f.ip70.getEdgeFilter () + ".\n" ) ;
        }

private void comboBox2_SelectedIndexChanged(object sender, System.EventArgs e)
        {
        f.ip70.setThresholdFilter (comboBox2.SelectedIndex );
        f.richTextBox1.AppendText ( "Set ThresholdFilter "
                +comboBox2.SelectedIndex + ".\n" );
        }

private void comboBox3_SelectedIndexChanged(object sender, System.EventArgs e)
        {
        f.ip70.setCleanUpFilter ( comboBox3.SelectedIndex );
        f.richTextBox1.AppendText  ( "Set CleanUpFilter "
                +comboBox3.SelectedIndex + ".\n" );
        }
```

9.4 First Two Settings

The default setting should be:

Edge Filter = 2 or “Sobel 2”.
Threshold Filter =1 or “Dark Background 128”.
Clean-Up Filter = 2

Your second choice should be:

Edge Filter = 0
Threshold Filter =5
Clean-Up Filter = 2

9.5 Chapter Projects

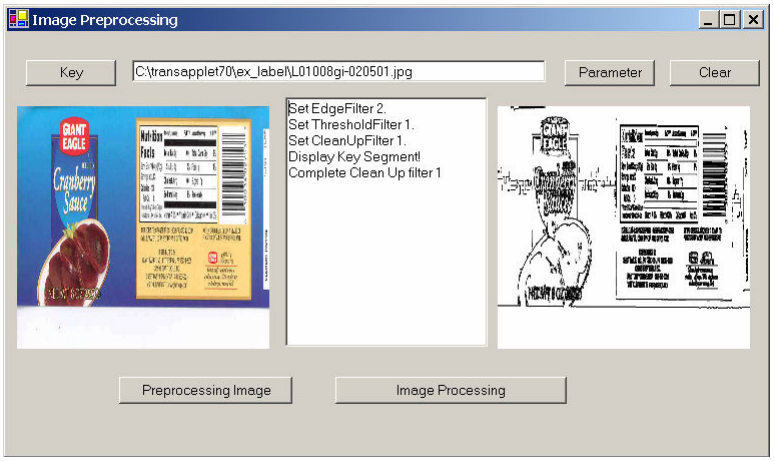

Figure 9.4 Chapter Project.

To see how image processing works, click the "Parameter" button in Figure 9.4 to get Figure 9.3. In Figure 9.3, set the:

Edge Filter = 2 or "Sobel 2".
Threshold Filter =1 or "Dark Background 128".
Clean-Up Filter = 2

You should see the following message in the text window:

Set EdgeFilter 2.
Set ThresholdFilter 1.
Set CleanUpFilter 1.
Display Key Segment!

Click the "Key" button and select image, "C:\transapplet70\ex_label\L01008gi-020501.jpg". Click the "Image Processing" button and Figure 9.4 will show the results of image processing.

10. Normalization

This chapter introduces a filter, Reduction Filter. From the user's and programmer's point of view, it means the setting of several parameters. This filter, together with the Image PreProcessing Filter and Image Processing Filters, will be passed to later layers as parameters.

If you are not interested in the details, you can skip this chapter.

The Normalization sub-layer will prepare the images for the underlying NeuralNet filters. The neural net deployed in the **ImageFinder**, by default, is a 100x100 array of neurons. While any size of ABM neural net can be used, when coming to a particular application, a decision has to be made. The **ImageFinder** uses 6 different sizes:

- 10,000 neurons,
- 8,100 neurons,
- 6,400 neurons,
- 4,900 neurons, or
- 2,500 neurons.

10.1 Class Library Name

The class library is:

Attrasoft.TransApplet70.ReductionFilter70.

The class in this library will be:

ReductionFilter70.

The interface, which will be used by ReductionFilter70, is:

```
public interface I_ReductionFilter70
    {
        string [] getReductionFilterNames();
        int getReductionFilterDefaultIndex();

        void setReductionFilter(int x);
        int getReductionFilter();
        bool getReductionFilterStatus();
        int getNeuralNetWAndH ( );
        void setNeuralNetWAndH (int i );

        void setSegmentCut(int x);
        int getSegmentCut();
```

```
        void setSizeCut(int x);
        int getSizeCut();

        void setBorderCut(int x);
        int getBorderCut();

        void setLookAtX(int x);
        int getLookAtX();

        void setLookAtY(int x);
        int getLookAtY();

        void setLookAtXLength(int x);
        int getLookAtXLength();

        void setLookAtYLength(int x);
        int getLookAtYLength();

        string getMessage();
        string getInfomation();
        string toString();

        void setImage (Bitmap b );
        int [] getPixels ();
        int[] getArray ();
        int getArrayWidth();
        int getArrayHeight();
}
```

10.2 Class Library Overview

This class library will address a single filter in the Normalization layer, the Reduction Filter.

The Reduction Filter selections are:

```
string [] getReductionFilterNames();
int getReductionFilterDefaultIndex();
```

To set the Reduction Filter, use:

```
void setReductionFilter(int x);
int getReductionFilter();
bool getReductionFilterStatus();
```

To set the Reduction Filter parameters, use:

```
void setSegmentCut(int x);
```

```
int getSegmentCut();

void setSizeCut(int x);
int getSizeCut();

void setBorderCut(int x);
int getBorderCut();

void setLookAtX(int x);
int getLookAtX();

void setLookAtY(int x);
int getLookAtY();

void setLookAtXLength(int x);
int getLookAtXLength();

void setLookAtYLength(int x);
int getLookAtYLength();

string getMessage();
String getInfomation();
String toString();
```

To process an image via the Reduction Filter, specify the image via:

```
void setImage (Bitmap b );
```

To use the filter, call:

```
int[] getArray ();
int getArrayWidth();
int getArrayHeight();
```

10.3 Link to Class Library

To include the class library in the project,

- Right click References and select Add Reference in the Solution Explorer;
- Browse to find “ReductionFilter70.dll” in “c:\transapplet70\”;
- Highlight it and click the “OK” button.

To use the class library, add:

```
using Attrasoft.TransApplet70.ReductionFilter70;
```

To declare an object, add:

```
public Attrasoft.TransApplet70.ReductionFilter70.ReductionFilter70 rd70
= new Attrasoft.TransApplet70.ReductionFilter70.ReductionFilter70 ();
```

Now ReductionFilter70, rd70, is ready to use.

10.4 Parameters

The Reduction Filter parameters are:

Segment-Cut Button

Use the "Segment-Cut" button to shape the segment considered by the **ImageFinder**. The Segment-Cut parameter ranges from 0 to 12. This parameter deals with the edges of segments in the images. The larger this parameter is, the smaller the segment the **ImageFinder** will use. The possible settings of this parameter in the user interface are: 0, 1, 2, .., and 12.

Size-Cut Button

Use the "Size-Cut" button to limit the dimensions of images to be searched. In some applications, the users only want to search images of certain dimensions and ignore other images. The dimension setting ranges from 0 to 9.

- If the setting is 0, this parameter will be ignored.
- If the parameter is 1, then the longest edge of the image to be considered must be at least 100, but less than 199.
- If the parameter is 2, then the longest edge of the image to be considered must be at least 200, but less than 299, …

Border Cut

Use the "Border-Cut" button to ignore the sections of the image near the borders. The Border-Cut parameter ranges from 0 (no cut) to 9 (18% border cut). The possible settings in the user interface are: 0, 1, 2, .., and 9. Assume an image is (0,0; 1,1); setting Border-Cut to 1 means the **ImageFinder** will look at the section (0.02, 0.02; 0.98, 0.98); setting Border-Cut to 2 means the **ImageFinder** will look at the section (0.04, 0.04; 0.96, 0.96); … .

Look-At Area

The "Look-At Area" is the area the **ImageFinder** will use. A 100x100 window specifies a whole image. In the Integer-Reduction, the actual area can be less than 100x100. The Look-At Area is specified by 4 numbers:

(x, y, w, h)

(x, y) are the coordinates of the upper-left corner and (w, h) are the width and height of the Look-At Window.

11. Parameter Class

ImageFinder has many parameters. For the **ImageFinder**, the Image Matching is divided into:

- Image Preprocessing
- Image Processing
- Normalization
- Signatures
- Feature Recognition
- Image Segment Recognition

One or more filters further implements each step:

Image Preprocessing
    Preprocessing Filter
Image Processing
    Edge Filter
    Threshold Filter
    Clean-Up Filter
Normalization
    Reduction Filter
Signature
    Signature Filter
Feature Recognition
    Unsupervised Filter
    BioFilter
    NeuralFilter
Image Segment Recognition
    Neural Net Filter

The image-matching software design, based on the **TransApplet,** is implemented to push an image through all of these filters. At the end of this pushing, the matching results will be obtained.

11.1 Pushing Images Through Filters

How to push an image through the following 10 filters:

1. Preprocessing Filter
2. Edge Filter
3. Threshold Filter
4. Clean-Up Filter
5. Reduction Filter
6. Signature Filter
7. Unsupervised Filter

8. BioFilter
9. NeuralFilter
10. Neural Net Filter

The implementation is as follows: the second filter will use the first filter as a parameter; the third filter will use the first and second filters as parameters; …

Each filter is an object (See their API in earlier chapters). Each filter will have many parameters and all 10 filters can have many parameters.

It will be easier, from the programming point of view, to group all of the objects together and to group all of the parameters together.

- Section 11.2 will group all of the objects together; and
- Section 11.3 will group all of the parameters together.

11.2 Predefined Objects

You can either declare objects yourself or use the predefined objects in your project. There is one class in the **TransApplet**:

Attrasoft.TransApplet70.EntryPoint70,

that has all of the objects required for the **ImageFinder**. In particular, the objects are:

```
public Attrasoft.TransApplet70.ImageSignature70.ImageSignature imageSignature;
public Attrasoft.TransApplet70.ImageSignature70.ImageSignature imageSignature1;
public Attrasoft.TransApplet70.Results_1N.Results_1N results_1N;

public Attrasoft.TransApplet70.Input70.Input70 input;

public Attrasoft.TransApplet70.VideoInput70.VideoInput70 aviVideo70
= new Attrasoft.TransApplet70.VideoInput70.VideoInput70 ();

public Attrasoft.TransApplet70.ImageSignatureFilter70.ImageSignatureFilter signatureFilter;

public Attrasoft.TransApplet70.ImagePacket70.ImagePacket70 imagePacket ;
public Attrasoft.TransApplet70.TransAppletParameters.TransAppletParameters  transAppletPara;
public Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 imagePreProcessingFilter ;
public Attrasoft.TransApplet70.ImageProcessing70.ImageProcessing70 imageProcessingFilter ;
public Attrasoft.TransApplet70.ReductionFilter70.ReductionFilter70 reductionFilter ;

public Attrasoft.TransApplet70.UnsupervisedFilter70 .UnsupervisedFilter unsupervisedFilter;

public Attrasoft.TransApplet70.BioFilter70.BioFilter bioFilter;

public Attrasoft.TransApplet70.NeuralFilter70.NeuralFilter neuralFilter;

public Attrasoft.TransApplet70.ImageLibrary70.ImageLibrary imageLibrary;

public Attrasoft.TransApplet70.NeuralNet70.NeuralNet70 neuralNetFilter;
```

```
public Attrasoft.TransApplet70.Counting70 .Counting70 counting;

public Attrasoft.TransApplet70.OutputTracking70.OutputTracking70 ott70
= new Attrasoft.TransApplet70.OutputTracking70.OutputTracking70 ();
public Attrasoft.TransApplet70.TrackingAuto70 .TrackingAuto70  autoTracking;

public Attrasoft.TransApplet70.Analysis70.Analysis analysis;
public Attrasoft.TransApplet70.Batch70.Batch70 batch;
```

To use these objects, simply declare an object of the following class:

```
Attrasoft.TransApplet70.EntryPoint70.EntryPoint70 script;
```

In the constructor, add:

```
script = new Attrasoft.TransApplet70.EntryPoint70.EntryPoint70 (richTextBox1);
```

Now, you can use all of the objects:

Objects	Descriptions
Script. ImageSignature	Image Signature
Script. signatureFilter	Signature Filter
Script. bioFilter	BioFilter
Script. unsupervisedFilter	Unsupervised Filter
…	…

11.3 Grouping Parameters Together

The class that groups all parameters together is called TransAppletParameters. The class interface is:

```
public class TransAppletParameters
                : TransAppletParametersData
        {
        public bool setPreProcessingFilterData
        ( Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 ipre70);
        public bool getPreProcessingFilterData
        ( Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 ipre70);

        public bool setImageProcessingFilterData
         ( Attrasoft.TransApplet70.ImageProcessing70.ImageProcessing70 ip70);
        public bool getImageProcessingFilterData
         ( Attrasoft.TransApplet70.ImageProcessing70.ImageProcessing70 ip70);

        public bool setReductionFilterData
         ( Attrasoft.TransApplet70.ReductionFilter70 .ReductionFilter70 rd70);
        public bool getReductionFilterData
         ( Attrasoft.TransApplet70.ReductionFilter70 .ReductionFilter70 rd70);

        public bool setSignatureFilterData
```

```
 ( Attrasoft.TransApplet70.ImageSignatureFilter70.ImageSignatureFilter sig70);
public bool getSignatureFilterData
 ( Attrasoft.TransApplet70.ImageSignatureFilter70.ImageSignatureFilter sig70);

public bool setUnsupervisedFilterData
 ( Attrasoft.TransApplet70.UnsupervisedFilter70 .UnsupervisedFilter  usf70);
public bool getUnsupervisedFilterData
 ( Attrasoft.TransApplet70.UnsupervisedFilter70 .UnsupervisedFilter  usf70);

public bool setBioFilterData
 ( Attrasoft.TransApplet70.BioFilter70.BioFilter  bf70);
public bool getBioFilterData
 ( Attrasoft.TransApplet70.BioFilter70 .BioFilter   bf70);

public bool setNeuralFilterData
        ( Attrasoft.TransApplet70.NeuralFilter70.NeuralFilter nf70);
public bool getNeuralFilterData
        ( Attrasoft.TransApplet70.NeuralFilter70 .NeuralFilter  nf70);

public bool setNeuralNetData
        ( Attrasoft.TransApplet70.NeuralNet70 .NeuralNet70   nn70);
public bool getNeuralNetData
        ( Attrasoft.TransApplet70.NeuralNet70 .NeuralNet70   nn70);
}
```

Before we explain the functions, let us first see the base class, which contains all of the parameters:

```
public class TransAppletParametersData
{
//[ImageFinder 7.0]
public int executionCode =0;

//[Input]
public string keyImage_absolutePath="None";
public string keyImage_segmentX="0";
public string keyImage_segmentY="0";
public string keyImage_segmentXlength="0";
public string keyImage_segmentYLength="0";
public string searchSource_Name="None";
public string searchSource_Type = "0";
public string searchSource_sqlStatement ="NA";

//[Image PreProcessing Filters]
public int imagePreProcessing_BorderCut =0;
public int imagePreProcessing_MaskX =0;
public int imagePreProcessing_MaskY =0;
public int imagePreProcessing_MaskW =0;
public int imagePreProcessing_MaskH = 0;
```

```
public int imagePreProcessing_MaskType = 0;
public int imagePreProcessing_StickShift = 0;
public int imagePreProcessing_SkipEmptyBorderType  =0;
public int imagePreProcessing_SkipPercent =0;
public int imagePreProcessing_SkipEdgeFilter = 0;

public int imagePreProcessing_SkipThresholdFilter = 0;

//[Image Processing Filters]
public int imageProcessing_edgeFilter=2;
public int imageProcessing_thresholdFilter=1;
public int imageProcessing_cleanUpFilter=2;
public int imageProcessing_doubleProcessing=0;

public int imageProcessing_R1=0;
public int imageProcessing_R2=128;
public int imageProcessing_R3=2;
public int imageProcessing_G1=0;
public int imageProcessing_G2=128;
public int imageProcessing_G3=2;
public int imageProcessing_B1=0;
public int imageProcessing_B2=128;
public int imageProcessing_B3=2;

//[Reduction Filter]
public int reduction_Filter=0;
public int reduction_SegmentCut=0;
public int reduction_SizeCut=0;
public int reduction_BorderCut=0;
public int reduction_LookAtX=0;
public int reduction_LookAtY=0;
public int reduction_LookAtXLength=0;
public int reduction_LookAtYLength=0;

//[SignatureFilter]
public int signatureFilter=0;

//[UnsupervisedFilter]
public int UnsupervisedFilterFilter=0;
public int UnsupervisedFilter_Percent=20;
public int UnsupervisedFilter_Mode=0;
public int UnsupervisedFilter_CutOff=0;
public int UnsupervisedFilter_OutputFileType=0;

public int UnsupervisedFilter_ShowFile=1;
public int UnsupervisedFilter_Blurring=2;
public int UnsupervisedFilter_Sensitivity=4;
public int UnsupervisedFilter_UseRelativeScore=0;
public int UnsupervisedFilter_ShowScore = 1 ;
```

```
public int UnsupervisedFilter_AutoSegment=0;

//[BioFilter]
public int bioFilter=0;
public int bioFilter_Percent=20;
public int bioFilter_Mode=0;
public int bioFilter_CutOff=0;
public int bioFilter_OutputFileType=0;

public int bioFilter_ShowFile=1;
public int bioFilter_Blurring=2;
public int bioFilter_Sensitivity=4;
public int bioFilter_UseRelativeScore=0;
public int bioFilter_ShowScore = 1 ;

public int bioFilter_AutoSegment=0;

//[NeuralFilter]
public int neuralFilter=2;
public int neuralFilter_Percent=20;
public int neuralFilter_Mode=0;
public int neuralFilter_Size=2;
public int neuralFilter_CutOff=0;

public int neuralFilter_OutputFileType=0;
public int neuralFilter_ShowFile=1;
public int neuralFilter_Blurring=2;
public int neuralFilter_Sensitivity=4;
public int neuralFilter_UseRelativeScore=0;

public int neuralFilter_ShowScore = 1 ;
public int neuralFilter_AutoSegment=0;

//[Neural Net]
public int neuralNetFilter=0;
public int neuralNetFilter_symmetry=3;
public int neuralNetFilter_rotationType=0;
public int neuralNetFilter_translationType=0;
public int neuralNetFilter_scalingType=0;

public int neuralNetFilter_sensitivity=50;
public int neuralNetFilter_blurring=10;
public int neuralNetFilter_internalWeightCut=100;
public int neuralNetFilter_externalWeightCut=0;
public int neuralNetFilter_segmentSize=0;

public int neuralNetFilter_imageType=1;
public int neuralNetFilter_fileDisplayType=0;
```

```
public int neuralNetFilter_autoSegment=0;
public int neuralNetFilter_neuralNetMode=0;
}
```

The base class holds all parameters for the **TransApplet**. The main class, TransAppletParameters, has 8 pairs of functions like this:

```
public bool setPreProcessingFilterData
(Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 ipre70);

public bool getPreProcessingFilterData
(Attrasoft.TransApplet70.ImagePreProcessing70.ImagePreProcessing70 ipre70);
```

The first function, setPreProcessingFilterData, passes the parameters in the TransAppletParameters object to a filter.

The second function, getPreProcessingFilterData, gets all of the parameters in a filter and stores them into the current TransAppletParameters object.

In this way, we get and set all of the parameters of a filter together. We also treat all of the filters exactly the same way.

11.4 Chapter Project

The chapter project is located at c:\transapplet70\imagefinder\.

Starting from this chapter, we will start to build software like the **ImageFinder**.
This section will focus on the forms used in the project and the parameters. The main form will not implement any functions; rather, it will link to functions in other classes. The functions will be implemented in two classes:

- MainMenuToAPI
- GUIRelated

Class, **MainMenuToAPI,** will implement all **TransApplet** related functions. This class will declare an object of "Attrasoft.TransApplet70.EntryPoint70.EntryPoint70", which contains all of the required objects.

Class, **GUIRelated**, will implement all functions related to GUI (Graphical User Interface), such as input/output, display images, messaging, …

The form, **Parameters**, will handle all of the parameters used in the **TransApplet**. It also links to many other forms used to specify the parameter. Each of these forms specifies parameters for one filter.

11.5 Creating Forms

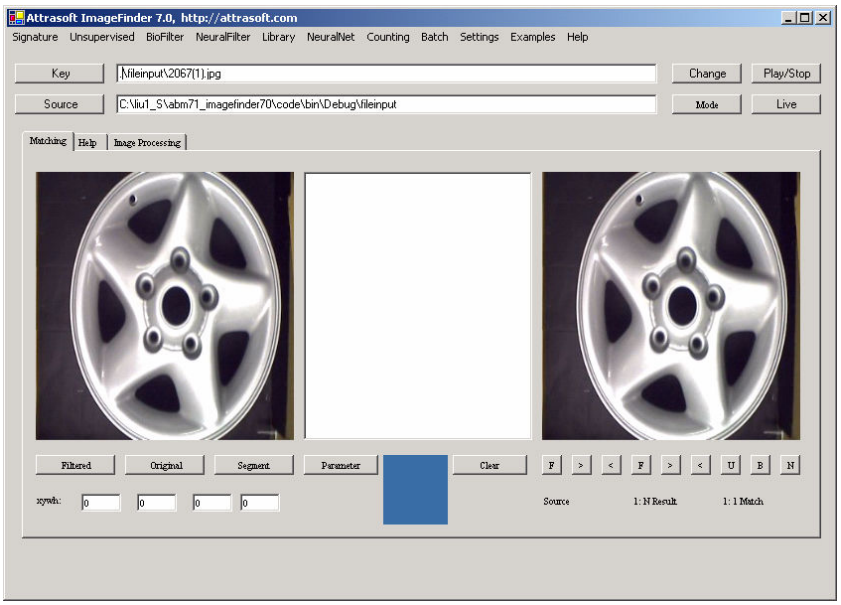

Figure 11.1 The main form.

The project will have the following forms and classes. You can either start your own **ImageFinder** project to build these forms and classes, or use the project located at "c:\transapplet70\imagefinder\".

If you start your own project, please create the following forms and classes:

MainMenu Form

Rename Form1 to Mainmenu. This will be the main form. The main form will not implement any functions. Its sole purpose is to link the buttons and menu items in the main form to functions in other classes. This form looks like Figure 11.1.

GUIRelated Class

Create a class, GUIRelated, to implement all functions related to GUI (Graphical User Interface), such as input/output, display images, messaging, ...

MainMenuToAPI Class

Create a class, MainMenuToAPI, to implement all **TransApplet** related functions. This class will declare an object of "Attrasoft.TransApplet70.EntryPoint70.EntryPoint70", which contains all of the required objects.

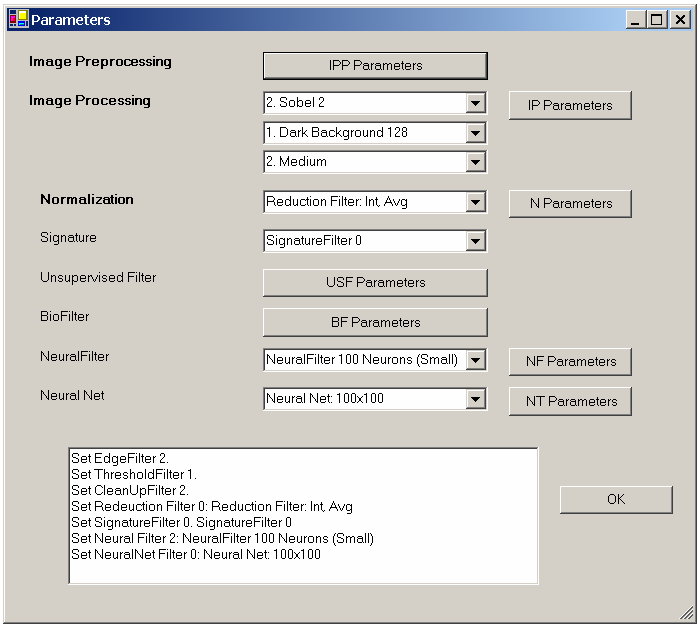

Figure 11.2 The Parameters form.

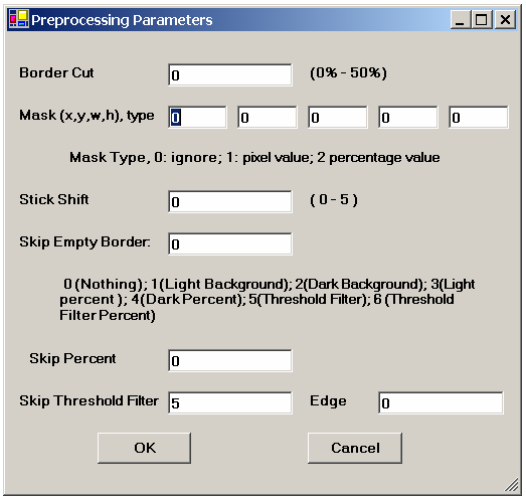

Figure 11.3 Image Preprocessing Parameter Form.

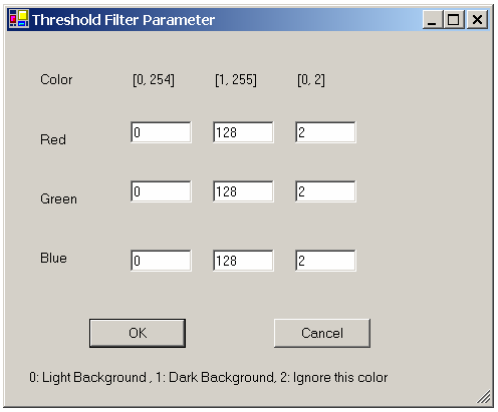

Figure 11.4 Image Processing Parameter Form.

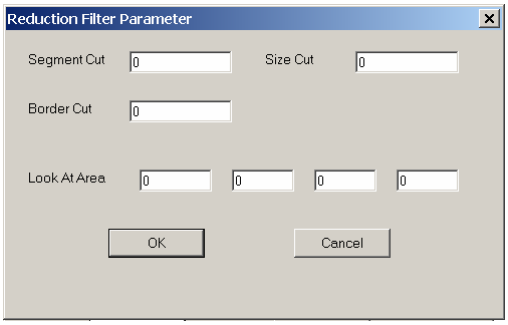

Figure 11.5 Normalization Parameter Form.

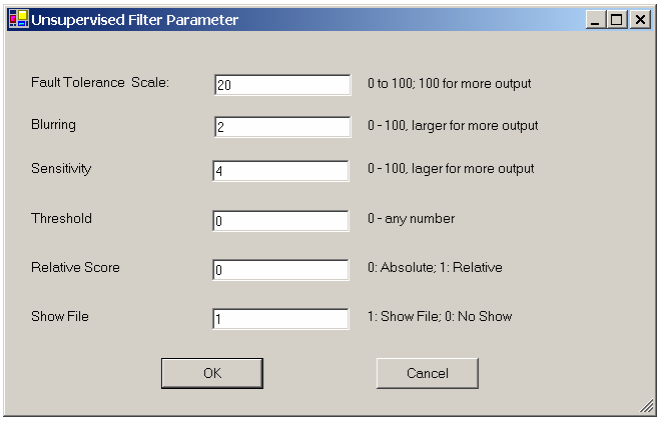

Figure 11.6 Unsupervised Filter Parameter Form.

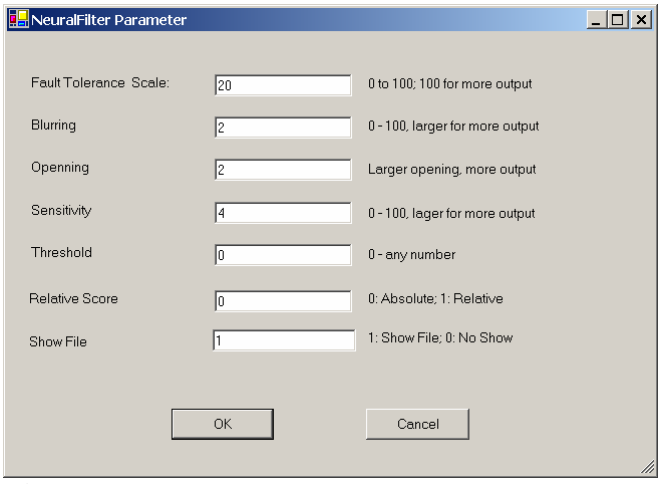

Figure 11.7 Neural Filter Parameter Form.

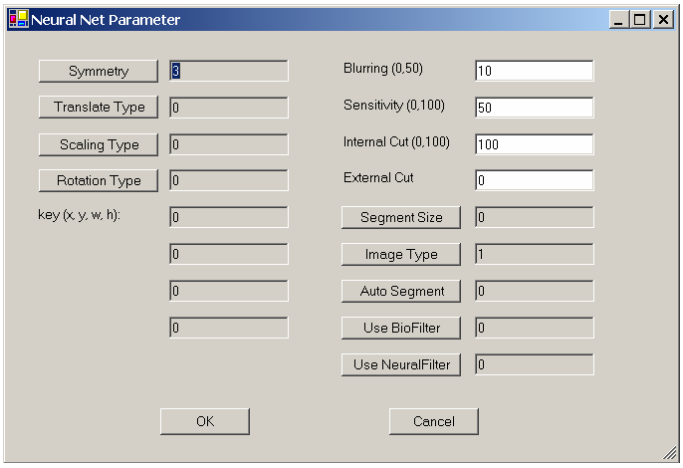

Figure 11.8 NeuralNet Filter Parameter Form.

Parameter Form

Implements all of the **TransApplet** parameters. This form will link to many other forms used to specify parameters. Each of these forms specifies parameters for one filter. Figure 11.2 shows the form.

Image Preprocessing Parameter Form

Implements all of the Image Preprocessing parameters. Figure 11.3 shows the form.

Image Processing Parameter Form

Implements all of the Image Processing parameters. Figure 11.4 shows the form.

Normalization Parameter Form

Implements all of the Normalization parameters. Figure 11.5 shows the form.

Unsupervised Filter Parameter Form

Implements all of the Unsupervised Filter parameters. Figure 11.6 shows the form.

BioFilter Parameter Form

Implements all of the BioFilter parameters. The BioFilter Parameter Form looks like the Unsupervised Filter parameters form in Figure 11.6.

NeuralFilter Parameter Form

Implements all of the NeuralFilter parameters. Figure 11.7 shows the form.

NeuralNet Filter Parameter Form

Implements all of the NeuralNet Filter parameters. Figure 11.8 shows the form.

11.6 TransApplet Objects

The first thing we will do is to create all of the objects required for the **ImageFinder**. As we discussed earlier in this chapter, all of the objects required for the **ImageFinder** project is grouped into a single class. All we have to do is to create an object of this class and we will call it script. We will create this object in class, MainMenuToAPI. The object is called script.

```
internal Attrasoft.TransApplet70.EntryPoint70.EntryPoint70 script;
MainMenu f;

public MainMenuToAPI(MainMenu f1)
{
try
        {
        f = f1;
        script = new Attrasoft.TransApplet70.EntryPoint70.EntryPoint70
 (f.richTextBox1);
        }
        catch (Exception e )
        {
                f.richTextBox1.Text = e.ToString () +"\n";
        }
}
```

11.7 Selecting Filters

The "Parameter" button in Figure 11.1 opens the Parameter Form in Figure 11.2, as follows:

```
private void button10_Click(object sender, System.EventArgs e)
{
Parameters paForm = new Parameters (this.mainMenuToAPI .script );
paForm.ShowDialog ();
}
```

Once in the Parameter Form in Figure 11.2, you can set the filters directly. To set the filter parameters, you have to open the following additional forms:

Image Preprocessing Parameters Form
Image Processing Parameters Form
Normalization Parameters Form
Unsupervised Filter Parameters Form
BioFilter Parameters Form
NeuralFilter Parameters Form
NeuralNet Filter Parameters Form

There are two parts related to the parameters:

- Selecting Filters
- Setting Filter Parameters

Selecting filters are done in the Parameter Form in Figure 11.2.
Setting filter parameters requires opening another form in Figure 11.3 – 11.8 and it is in these forms that the filter parameters are set.

We now discuss how to set each filter:

Image Preprocessing Filter

There is only one Image Preprocessing Filter, so you do not have to set this filter.

Image Processing Filters

There are three image processing filters: Edge Filters, Threshold Filters, and Clean-Up Filters. You can select these filters from the Drop Down Lists in Figure 11.2.

The Edge Filter is implemented as follows:

```
void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
        script.transAppletPara.imageProcessing_edgeFilter =
                comboBox1.SelectedIndex;
        script.transAppletPara.setImageProcessingFilterData
                (script.imageProcessingFilter );
        richTextBox1.AppendText  ( "Set EdgeFilter "
         +script.transAppletPara.imageProcessing_edgeFilter + ".\n" ) ;
}
```

The first line gets the new setting from the Combo Box (Drop Down List).
The second line assigns the new data to the Image Processing filter object.
The third line prints a message.

The Threshold Filter and the Clean-Up Filter are handled in a similar fashion.

Normalization Filter

You can select the Normalization filters from the Drop Down List in Figure 11.2. The Normalization Filter is implemented as follows:

```
void comboBox4_SelectedIndexChanged(object sender, System.EventArgs e)
{
        script.transAppletPara.reduction_Filter
                =  comboBox4.SelectedIndex ;

        script.transAppletPara.setReductionFilterData
                 (script.reductionFilter  );

        richTextBox1.AppendText ( "Set Redeuction Filter "
        +comboBox4.SelectedIndex + ": " +  comboBox4.SelectedItem + "\n" );
}
```

The Unsupervised Filter selection, BioFilter selection, NeuralFilter selection, and NeuralNet Filter selection will be implemented in a similar fashion.

11.8 Set Filter Parameters

In the last section, we have shown that you can select a filter in the Parameter Form shown in Figure 11.2. After selecting a filter, to set the filter parameters, you have to open the following additional forms and specify the parameter:

Image Preprocessing Parameter Form
Image Processing Parameter Form
Normalization Parameter Form
Unsupervised Filter Parameter Form
BioFilter Parameter Form
NeuralFilter Parameter Form
NeuralNet Filter Parameter Form

We will use the Image Preprocessing Parameter Form in Figure 11.3 to show the implementation. The "OK" button in Figure 11.3 is implemented as follows:

```
private void button1_Click(object sender, System.EventArgs e)
{
try
{
script.transAppletPara.imagePreProcessing_MaskX = int.Parse (textBox1.Text) ;
script.transAppletPara.imagePreProcessing_MaskY = int.Parse (textBox2.Text ) ;
script.transAppletPara.imagePreProcessing_MaskW = int.Parse (textBox3.Text ) ;
script.transAppletPara.imagePreProcessing_MaskH =  int.Parse (textBox4.Text ) ;
script.transAppletPara.imagePreProcessing_MaskType
= int.Parse (textBox11.Text );

script.transAppletPara.imagePreProcessing_BorderCut
=   int.Parse (textBox5.Text ) ;

script.transAppletPara.imagePreProcessing_StickShift
=   int.Parse (textBox6.Text ));


script.transAppletPara.imagePreProcessing_SkipEmptyBorderType
=    int.Parse (textBox7.Text ) ;
script.transAppletPara.imagePreProcessing_SkipPercent
=    int.Parse (textBox8.Text ) ;
script.transAppletPara.imagePreProcessing_SkipThresholdFilter
=   int.Parse (textBox9.Text ) ;
script.transAppletPara.imagePreProcessing_SkipEdgeFilter
=   int.Parse (textBox10.Text ) ;

script.transAppletPara.setPreProcessingFilterData
 (script.imagePreProcessingFilter );

this.Close();
```

```
}
catch
{
MessageBox.Show ("Please enter valid integers", "Entry Error");
}
}
```

The first section takes the input data from the text boxes and stores them in object, script.transAppletPara.

The second section,

```
script.transAppletPara.setPreProcessingFilterData (script.imagePreProcessingFilter );
```

passes the data in object, “script.transAppletPara”, to the Image Preprocessing filter,

```
script.imagePreProcessingFilter.
```

In a similar fashion, we can implement the following parameter forms:

Image Processing Parameter Form
Normalization Parameter Form
Unsupervised Filter Parameter Form
BioFilter Parameter Form
NeuralFilter Parameter Form
NeuralNet Filter Parameter Form

12. Image Signatures

Image Matching is done through something called Image Signature. An image has a set of computed values called features. A collection of features is grouped into a signature.

In this project, we will use Signature Menu Items to compute. The input is an image and the output is a signature. We will show how to use the Image Signatures in image matching in the next chapter. This chapter introduces how to compute signatures only, which will take an input image and produce a signature.

12.1 Signature Menu

First of all, we will create the menu shown in Figure 12.1 and create the menu items under the Signature Menu.

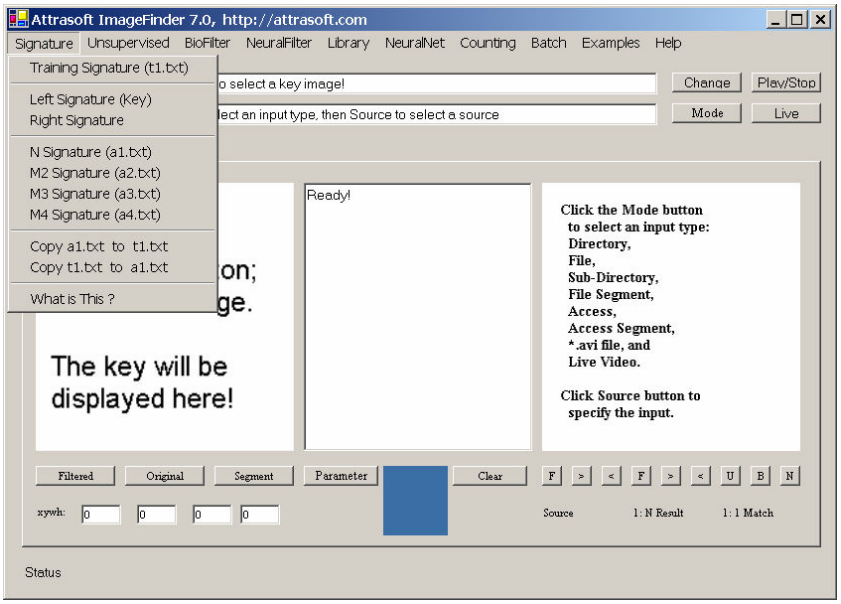

Figure 12.1 Signature Menu.

12.2 API

The basic idea is:

Image Signature = SignatureFilter. getSiganture (image);

As we can see, there are two objects here, Signature and Signature Filter. The Signature functions are:

Function	Descriptions	Comments
int getStatus ()	Returns the status associated with the signature. Output: 1: signature ready. 0: signature not ready. -1: no image -2: image segmentation specification error -3: other error.	
string getID()	Returns the ID associated with the signature. Output: image ID.	
string getImageName ()	Returns the image name associated with the signature. Output: image name.	
string getImagePath()	Returns the image path associated with the signature. Output: image path.	
string getAbsolutePath ()	Returns the absolute path associated with the signature. Output: image absolute path.	
int getNumberOfAttributes ()	Returns the number of attributes associated with the signature. Output: number of attributes.	
int [] getSignatureAttributes()	Returns the attribute array associated with the signature. Output: attribute array.	
int int [] getSignatureAttributes() (int index)	Returns the attribute associated with the input index. Input: index. Output: attribute associated with the input index.	
string toString ()	Returns the entire image signature as a string with	

	fields separated by Tab.	

The Signature Filter functions are:

Function	Descriptions	Comments
bool setSignatureFilter (int x) int getSignatureFilter ()	Selects a Signature filter.	
string [] getSignatureFilterNames();	Gets a list of Signature filter names.	
ImageSignature getSignature (string imagePath, string ID); ImageSignature getSignature (string imagePath); ImageSignature getSignature (Bitmap b, string ID); ImageSignature getSignature (Bitmap b);	Gets the Signature of the input image.	
ImageSignature getSignature (string imagePath, string ID,int x, int y, int w, int h); ImageSignature getSignature (Bitmap bImg, string path, string name, string ID, int x, int y, int w, int h);	Gets the Signature of the input image segment. Input: string imagePath, or Bitmap bImg string ID int x, int y, int w, int h. Output: Signature.	
bool getLibrary (string [] imageAbsolutePath, string fileName); bool getLibrary (string [] imageAbsolutePath, string [] IDs, string fileName);	Generates a Signature library from all images in string [] imageAbsolutePath and produces a file that contains all signatures. Input: string [] imageAbsolutePath	

bool getSegmentLibrary (string [] imageAbsolutePath, string [] IDs, string [] xs, string [] ys, string [] ws, string [] hs, string fileName); bool getSegmentLibrary (string [] imageAbsolutePath, string [] IDs, int [] xs, int [] ys, int [] ws, int [] hs, string fileName);	Output: A text file that contains the library of signatures.	

12.3 TransApplet Objects

The first thing we will do is to create all of the objects required for the **ImageFinder**. As we discussed earlier in this chapter, all of the objects required for the **ImageFinder** project is grouped into a single class. All we have to do is to create an object of this class and we will call it script. We will create this object in class, MainMenuToAPI.

```
internal Attrasoft.TransApplet70.EntryPoint70.EntryPoint70 script;
MainMenu f;

public MainMenuToAPI(MainMenu f1)
{
        try
        {
        f = f1;
        script = new Attrasoft.TransApplet70.EntryPoint70.EntryPoint70
                        (f.richTextBox1);
        }
        catch (Exception e )
        {
                f.richTextBox1.Text = e.ToString () +"\n";
        }
        }
```

12.4 Key Signature

You can collect the signature(s) of the left image, the right image, and all of the images in a folder. To compute the signature of an image:

- Click the "Key" button to select a key image or sample image;
- Click the "Signature/Left Signature (Key)" to compute the signature.

Example. Left Signature.

To continue from Figure 3.4, click the "Clear" button to clear the text box. Click "Signature/Left Signature (Key)" and you will see Figure 12.2.

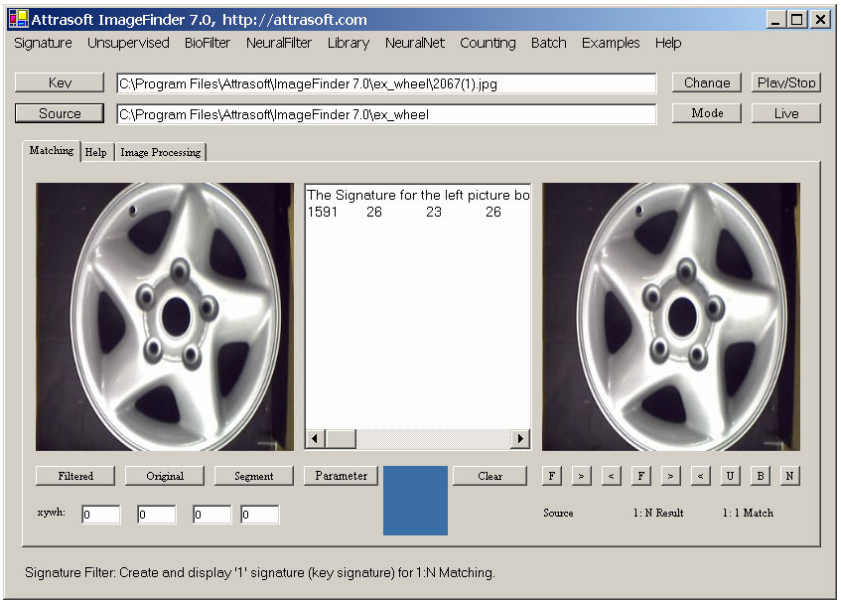

Figure 12.2 Click "Signature/Left Signature (Key)".

The Signature looks like:

1591 26 23 26 23 …

Similarly, click "Signature/Right Signature" and you will get an image signature for the right image.

In 1:1 Matching or 1:N Matching, the signatures are computed behind the screen; you do not need to use the signature directly. The menu item "Signature/Left Signature (Key)" shows you the mechanism behind the screen so you can see what a signature looks like.

Now we implement these two menu items. Double click these menu items and enter:

```
private void menuItem2_Click(object sender, System.EventArgs e)
{
this.mainMenuToAPI.signature_Key_Signature ( textBox1.Text );
}
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed in the last chapter, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool signature_Key_Signature ( string key)
{
        if ( ! System.IO .File .Exists (key) )
        {
                appendText ( "Please enter a valid Key!\n");
                return false;
        }

        int x =0, y = 0, w =0, h = 0;
        try
        {
                x = int.Parse (f.textBox3.Text );
                y = int.Parse (f.textBox4.Text );
                w = int.Parse (f.textBox5.Text );
                h = int.Parse (f.textBox6.Text );
        }
        catch
        {
                setText ("Invalid integers!\n");
                x= 0;
                y = 0;
                w = 0;
                h = 0;
                return false;
        }

        script.ImageSignature
                = script.signatureFilter.getSignature (key, "1 of 1:N", x, y, w, h );

        if ( script.imageSignature != null )
                {
                        setLine ("The Signature for the left picture box:");
                        appendLine (script.imageSignature.toString () );
                        return true;
                }
        else
                return false;
        }
```

The following section of code simply makes sure the key image exists:

```
if ( ! System.IO .File .Exists (key) )
        {
                appendText ( "Please enter a valid Key!\n");
```

```
        return false;
    }
```

The following section of code computes (x, y, w, h) of the key segment:

```
    int x =0, y = 0, w =0, h = 0;
    try
    {
        x = int.Parse (f.textBox3.Text );
        y = int.Parse (f.textBox4.Text );
        w = int.Parse (f.textBox5.Text );
        h = int.Parse (f.textBox6.Text );
    }
    catch
    {
        setText ("Invalid integers!\n");
        x= 0;
        y = 0;
        w = 0;
        h = 0;
        return false;
    }
```

The following section of code computes the signature of the key segment:

```
script.ImageSignature
    = script.signatureFilter.getSignature (key, "1 of 1:N", x, y, w, h );
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, “script.ImageSignature” is the image signature object; and script.signatureFilter is the signature filter object. The parameter, "1 of 1:N", is the ID, which we arbitrarily assigned to the key image. The following section of code prints the signature of the key segment in the text window:

```
if ( script.imageSignature != null )
    {
        setLine ("The Signature for the left picture box:");
        appendLine (script.imageSignature.toString () );
        return true;
    }
```

The menu item, “Signature/Right Signature”, is implemented in a similar fashion.

12.5 Signature File Concepts

Signatures files are divided into three groups: Training Signature File, N-Signature File, and M-Signature File. In a 1:N Match, the 1-signature is computed at run time and the N-signature is computed in advance.

Training teaches the **ImageFinder** who should match with whom. Training is done through two files, a training signature file with a fixed name ".\data\t1.txt" and a match file with a fixed name ".\data\match.txt". Here, ".\" is the folder where the executable files stay. Menu item, "Signature/Training Signature (t1.txt)", generates training signatures from a specified directory of images.

In 1:N Matching, the Key image will be matched against all images in a directory. The key signature is computed first, as we demonstrated in the last section. Then this signature will be matched against all signatures in the N-signature file. The N-signature file has a fixed name, ".\data\a1.txt".

In N:M Matching, the N-signature file, a1.txt, and the M-signature file, a2.txt, a3.txt, a12.txt, are computed first. Then all of the signatures in a1.txt will be matched against all of the signatures in a2.txt. The three M-signature files have fixed names, ".\data\a2.txt", ".\data\a3.txt", and ".\data\a12.txt".

There are 5 menu items:

Signature/Training Signature (t1.txt)
Signature/N Signature (a1.txt)
Signature/M2 Signature (a2.txt)
Signature/M3 Signature (a3.txt)
Signature/M4 Signature (a12.txt)

These commands compute the signatures for all images in a directory. The only difference is where to save the signatures.

Menu item, Signature/Training Signature (t1.txt), computes the signatures for all images in a directory and saves the signatures to t1.txt, the training signature file.

Menu item, Signature/N Signature (a1.txt), computes the signatures for all images in a directory and saves the signatures to a1.txt, the N-signature file.

Menu item, Signature/M2 Signature (a2.txt), computes the signatures for all images in a directory and saves the signatures to a2.txt, the M-signature file.

Menu item, Signature/M3 Signature (a3.txt), computes the signatures for all images in a directory and saves the signatures to a3.txt, the M-signature file.

Menu item, Signature/M4 Signature (a12.txt), computes the signatures for all images in a directory and saves the signatures to a12.txt, the M-signature file.

12.6 Signature File Implementation

Now we implement menu item, Signature/N Signature (a1.txt). Double click this menu item and enter:

```
private void menuItem6_Click(object sender, System.EventArgs e)
```

```
{
this.mainMenuToAPI.signature_Training_Segment_Signature
        (dataDir + "a1.txt");
}
```

Again, mainMenuToAPI is an object, which will implement all functions. As we discussed in the last chapter, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public void signature_Training_Segment_Signature   (string outputFile)
        {
        f.mainMenuToAPI.script.signatureFilter.getSegmentLibrary
                        ( f.gui.imageAbsoultePath,
                        f.gui.imageID ,
                        f.gui.imageX,
                        f.gui.imageY,
                        f.gui.imageW,
                        f.gui.imageH ,
                        outputFile );
        }
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, "script.signatureFilter" is the image signature filter object. The parameters, "f.gui.imageAbsoultePath", is a string list that has the absolute paths of all source images to be matched. After the completion of signature computation, the results will be stored in a file specified by the last parameter, outputFile.

The only difference between the following menu items is the output file; therefore, all of them will be implemented in a similar fashion:

Signature/Training Signature (t1.txt)
Signature/N Signature (a1.txt)
Signature/M2 Signature (a2.txt)
Signature/M3 Signature (a3.txt)
Signature/M4 Signature (a12.txt)

12.7 Examples

To compute the N-signatures:

- Click the "Source" button to select a directory;
- Click "Signature/N Signature (a1.txt)" to compute the signatures in a1.txt.

Example. Select N-images:

- Click the "Source" button;

- Select image, "./ex_wheel/2067(1).jpg", here "./" means the folder where the **ImageFinder** is located. The default location for our project will be:
  "c:\transapplet70\imagefinder\bin\Release\".
  i.e. ".\" = "c:\transapplet70\imagefinder\bin\Release\".
- Click "Signature/N Signature (a1.txt)";
- Open "./data/a1.txt" to see the results.

You can repeat the above example for the following menu items:

Signature/Training Signature (t1.txt)
Signature/M2 Signature (a2.txt)
Signature/M3 Signature (a3.txt)
Signature/M4 Signature (a12.txt)

And the results will go to t1.txt, a2.txt, a3.txt and a12.txt, respectively.

13. Unsupervised Filters

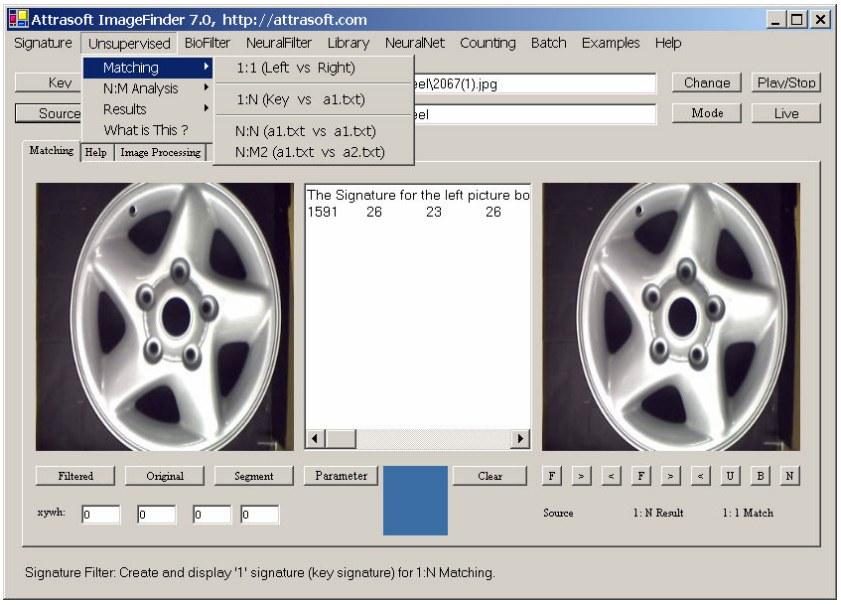

Figure 13.1 Unsupervised Filter.

This chapter will demonstrate how the Unsupervised Filter works using a **Label Matching example**. The project is located at:

c:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\". The data is located at: ".\ex_label" or "c:\transapplet70\imagefinder\bin\Release\ ex_label".

13.1 Unsupervised Filter Menu

Figure 13.2 Newly Captured Image.

First of all, we will create the menu and menu items shown in Figure 13.1.

The data used is shown in Figure 13.2. The matching problem is: assuming we have a newly captured image in Figure 13.2, let us match it against the existing master library and see if there is a match. There are 304 images in 152 pairs. The data is stored at ".\ex_label\" folder.

13.2 Unsupervised Filter API

The following table lists the functions.

Functions	Descriptions
Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1, Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (string path1, string path2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Bitmap left, Bitmap right);	Makes a 1:1 matching
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig, string a1File, string b1File); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string keyuPath, string a1File, string b1File) Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Bitmap keyImage, string a1File, string b1File)	Makes a 1:N matching
bool findMatchNN (string a1File, string b1File); bool findMatchNM (string a1File, string a2File, string b1File);	Matches all image signatures in file a1File against all image signatures in a1File or a2File and saves the results to b1File.

13.3 N-Signature

An Unsupervised Matching process has three steps:

- Signature;
- Matching;
- Results and Analysis.

To get the N-signature file, a1.txt:

- Click the "Source" button, go to the ".\ex_label" directory and select any file in the folder. This will specify the input directory.
- Click the Source ">" button a few times to see the images;
- Click menu item "Signature/N Signature (a1.txt)" to get the signatures in a1.txt file.

13.4 N:N Matching Design

Continuing from the last section, we will do N:N Matching first:

- Click menu item "Unsupervised/Matching/N:N (a1.txt vs. a1.txt)" button to complete a N:N Match.

The results are in a file, b1.txt, which will be opened at this time. The name, b1.txt, is fixed. The file looks like this:

C:\…\L01008gi-020501.jpg
C:\...\L01008gi-020501.jpg
110
C:\...\L01008gi_r90.jpg
95
C:\...\L02016alb-090100_m.jpg
65

C:\...\L01008gi_r90.jpg
C:\...\L01008gi-020501.jpg
95
C:\...\L01008gi_r90.jpg
110
…

The result file contains many blocks. The number of blocks is the same as the number of images in the search directory, i.e. each image has a block. Line 1 in each block is the input and the rest of the lines are output; i.e. the first line is the image matched against all images in the search directory; the rest of the lines represent the matched images. For example, "C:\…\L01008gi_r90.jpg" is matched against all 304 images in the search directory; there are three matches.

The first match is:

C:\...\L01008gi-020501.jpg
110

Here "110" is a score.

The second match is:

C:\...\L01008gi_r90.jpg
95

and the third match is:

C:\...\L02016alb-090100_m.jpg
65

Higher scores indicate a closer match.

13.5 N:N Matching Implementation

Double click menu item “Unsupervised/Matching/N:N (a1.txt vs. a1.txt)” and enter:

```
private void menuItem23_Click(object sender, System.EventArgs e)
            {
            this.mainMenuToAPI .unsupervised_Matching_NToM
                  (dataDir + "a1.txt", dataDir + "a1.txt", dataDir + "b1.txt" );

            }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
      public bool unsupervised_Matching_NToM
            ( string a1File, string a2File, string b1File)
            {
                  try
                  {
                  script.unsupervisedFilter .findMatchNM
                        ( a1File, a2File, b1File);
                  }
                  catch (Exception e)
                  {
                        appendText (  e.ToString () + "\n");
                        return false;
                  }
                  return true;
            }
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, “script.unsupervisedFilter” is the unsupervised filter object. The parameters, (a1File, a2File, b1File), indicate all signatures in a1File will be matched against all signatures in a2File. After completion of the matching, the results will be stored in a file specified by the last parameter, b1File.

13.6 1:N Matching Design

1:N Matching compares one key image with the images in a search directory; the key image is selected by the "Key" button. In a 1:N match, the 1-signature is computed at run time and the N-signature is computed in advance.

To continue the **Label Recognition problem** for 1:N Matching:

- Click the "Key" button, in the ".\ex_label"directory, select the first image "L01008gi-020501.jpg";
- Click the menu item "Unsupervised/Matching/1:N (Key vs. a1.txt)" button to complete a 1:N Match.

The results are in file, b1.txt, which will be opened at this point:

ID	Name	Path	Score	X	Y	W	H
L01008gi-020501	L01008gi-020501.jpg	C:\...\	110	0	0	0	0
L01008gi_r90	L01008gi_r90.jpg	C:\...\	95	0	0	0	0
L02016alb-090100_m	L02016alb-090100_m.jpg	C:\...\	65	0	0	0	0

The output will always go to b1.txt and will overwrite earlier results. If you need to save the results, simply save it to a different file.

13.7 1:N Matching Implementation

Double click menu item "Unsupervised/Matching/1:N (Key vs. a1.txt)" and enter:

```
private void menuItem21_Click(object sender, System.EventArgs e)
{
bool b = this.mainMenuToAPI .unsupervised_Matching_1ToN
        textBox1.Text , dataDir + "a1.txt",    dataDir + "b1.txt" );
}
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool unsupervised_Matching_1ToN
        ( string key, string a1File, string b1File)
{
        bool b = false;
        if ( ! System.IO .File .Exists (key) )
                {
                        appendText ( "Please enter a valid Key!\n");
                        return false;
                }

        b = this.signature_Key_Signature (key);
```

```
if ( ! b )
{
        appendText ( "Key Signature computation fails!\n");
        return false;
}

try
{
script.results_1N = script.unsupervisedFilter  .findMatch1N
         ( script.imageSignature ,  a1File,  b1File);
}
catch (Exception e)
{
        appendText ( "UnsupervisedFilter 1:N Matching fails:\n"
                + e.ToString () + "\n");
        return false;
}
if ( script.results_1N == null )
        {
                appendText ( "UnsupervisedFilter 1:N Matching fails!\n" );
                return false;
        }

if ( script.results_1N.getStatus () )
        {
        setText ( script.results_1N.toString () + "\n");
        appendText ("" + script.results_1N.getNumberOfMatches ()
        +" matches!\n");
        }
        else
        {
                appendText ( "No Match!\n");
                return false;
        }
}
```

The following code simply makes sure the key image exists:

```
if ( ! System.IO .File .Exists (key) )
        {
                appendText ( "Please enter a valid Key!\n");
                return false;
        }
```

The next section of code computes the key signature:

```
b = this.signature_Key_Signature (key);
```

The next section code makes a 1:N Match:

```
try
{
script.results_1N = script.unsupervisedFilter  .findMatch1N
        ( script.imageSignature ,  a1File,  b1File);
}
catch (Exception e)
{
        appendText ( "UnsupervisedFilter 1:N Matching fails:\n"
                + e.ToString () + "\n");
        return false;
}
```

The final section of code prints the 1:N Matching results:

```
if ( script.results_1N.getStatus () )
        {
        setText ( script.results_1N.toString () + "\n");
        appendText ("" + script.results_1N.getNumberOfMatches ()
        +" matches!\n");
        }
        else
        {
                appendText ( "No Match!\n");
                return false;
        }
```

14. BioFilters

The BioFilter matches two whole images. The BioFilter is better than Unsupervised Matching, but it requires a process called **training**. Training teaches the BioFilter who should match with whom. The BioFilter learns how to match the image features.

- The advantage of the BioFilter is that it does not require a lot of training data.
- The disadvantage of the BioFilter is that it has a lower identification rate than the Neural Filter.

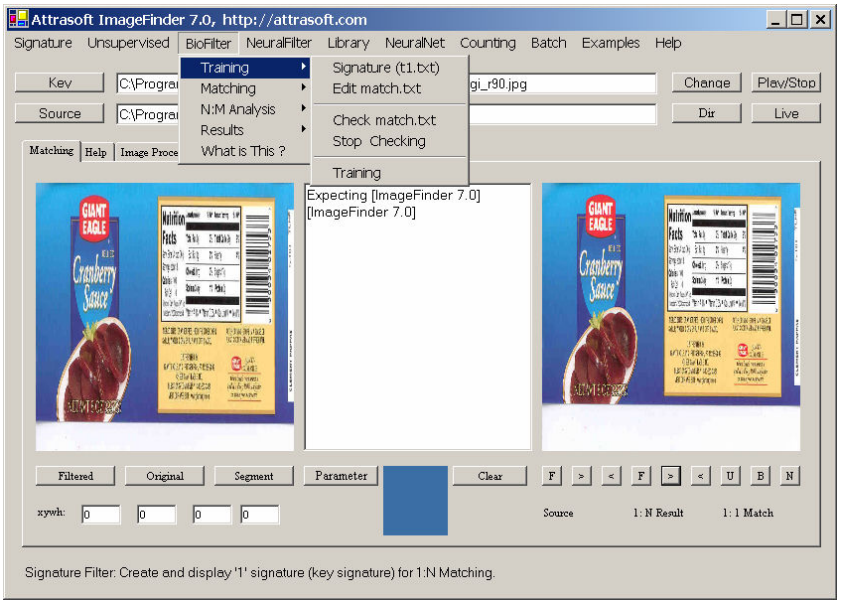

Figure 14.1 Training Menu.

The chapter project is located at:

c:\transapplet70\imagefinder**.**

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\". The data is located at: ".\ex_label" or "c:\transapplet70\imagefinder\bin\Release\ ex_label".

14.1 BioFilter Menu

The BioFilter Matching will have four steps:

- Signatures
- Training
- Matching
- Analysis

We discussed Signature Computation in earlier chapters. The BioFilter Matching is very similar to the Unsupervised Matching in the last chapter.

Training uses the data collected in advance to teach the BioFilter how to match. Training requires two files, t1.txt and match.txt.

- T1.txt is the signature file, which contains many signatures. Each image is converted into a signature.
- Match.txt is a list of matching pairs. This file will teach the **ImageFinder** who will match with whom. You must prepare this file.

14.2 BioFilter API

The following table lists the BioFilter functions.

Functions	Descriptions
bool training (string a1_txt, string match_txt)	Trains the BioFilter.
Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1, Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (string path1, string path2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Bitmap left, Bitmap right);	Makes a 1:1 matching
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig, string a1File, string b1File); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string keyuPath, string a1File, string b1File) Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Bitmap keyImage, string a1File, string b1File)	Makes a 1:N matching
bool findMatchNN (string a1File, string b1File); bool findMatchNM (string a1File, string a2File, string b1File);	Matches all image signatures in file a1File against all image signatures in a1File or a2File

	and saves the results to b1File.

14.3 Training Design

Training teaches the **ImageFinder** what to look for. The BioFilter training requires two files, t1.txt and match.txt:

- T1.txt is the signature file, which contains many signatures. Each image is converted into a signature.
- Match.txt is a list of matching pairs. This file will teach the **ImageFinder** who will match with whom.

These two file names, t1.txt and match.txt, for training are fixed for users; users cannot change the names of these two files. Users obtain t1.txt through the signature computation process discussed in earlier chapters. Users have to prepare match.txt for each problem.

The match.txt looks like this:

```
152
1    L01008gi_r90      L01008gi-020501
2    L01008KEY_m       L01008key-082301_m
3    L010103C          L010103C-081502_m
4    L01010co_m        L01010CODE_m
5    L010163C_m        L010163C-083100_m
…
```

Line 1 is the number of matches in this file. This match file indicates images, L01008gi_r90, will match with image, L01008gi-020501. Each line has the following format:

Number, tab, filename, tab, filename**.**

Note:

You MUST have a tab between the three columns;
The file names do not contain ".jpg".

There are two common errors:

(1) The Tab is replaced by a space;
(2) The number of rows is less than the first number in the file.

Once you get the two files prepared, click "BioFilter\Training\Training" to train the BioFilter. (Figure 14.1)

14.4 Training Implementation

Double click menu item “BioFilter\Training\Training” and enter:

```
private void menuItem39_Click(object sender, System.EventArgs e)
        {
                this.mainMenuToAPI .bioFilter_Train_Train
                        ( dataDir + "t1.txt" , dataDir + "match.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
        public bool bioFilter_Train_Train ( string a1_txt, string match_txt)
        {
                script.bioFilter.training (  a1_txt, match_txt );
                return true;
        }
```

14.5 Parameters

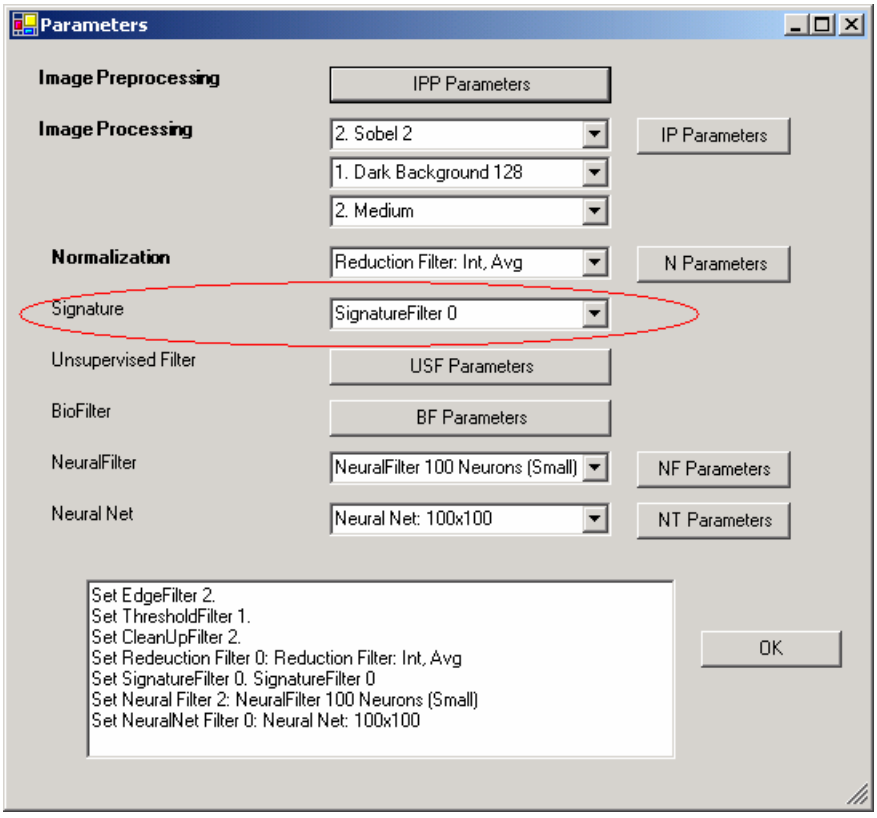

Figure 14.2 Parameter Window.

The **ImageFinder** has many parameters. Changing these parameters will change the output of the computations. The parameters are set by clicking the "Parameter" button in the main form, which will open the Parameter Window in Figure 14.2. You will adjust the **ImageFinder** parameters here.

The default setting uses the Signature Filter 0, which is the least accurate Signature Filter. There are 20 Signature filters in the current version of the **ImageFinder**. In general, when you have less data, use less accurate Signature filters; when you have more data, use more accurate Signature filters.

There are really no hard guidelines; the following is only a rough reference for the BioFilter:

Signature Filter	Training Pairs
0	10
1	30
2	30
3	70
4	70
5	150
6	150
7	310
8	310
9	630
10	630
> 11	1000

We have 152 pairs, so we will choose Signature Filter 6. In Figure 14.2, select Signature Filter 6 from line 6.

14.6 Example: Label Recognition Training

We now revisit the **Label Recognition example** first introduced in the Unsupervised Filter. We must prepare the match.txt file for training. This file is already prepared for you and we will simply open it and save it to match.txt. The steps are:

Match.txt

- Open the file, ".\data\match_ex_label.txt". This file lists 152 matching pairs. Save it to match.txt (overwrite the existing file). Now the training file is prepared.

T1.txt:

- Click the "Source" button, go to "ex_label" directory and select any file in the folder. This will specify the input directory.
- Click the Source ">" button a few times to see the images;
- Click menu item "Signature/N Signature (a1.txt)" to get signature file, a1.txt file;
- Click menu item "Signature/Copy a1.txt to t1.txt" to get the training file, t1.txt.

Note: Here t1.txt is for training and a1.txt is for 1:N Matching and N:N Matching.

Training

- Click “BioFilter\Training\Training” to train the BioFilter.

You should get this message at the end of the text window:

Total Number of Matches = 152
Number of Images that have No Match = 152

There are 304 images in 152 pairs. The match.txt listed 152 pairs.

- The first line, Total Number of Matches = 152, indicates the training used 152 pairs.
- The second line, Number of Images that have No Match = 152, indicates 152 out of 304 images does not have a match, which is correct. This is because in match.txt which has 152 pairs, (A, B), only A will match with B, but B will not match with A.

Now, the **BioFilter** is trained for the **Label Recognition problem.** We will continue this example in the next section, N:N Matching.

14.7 N:N Matching Design

N: N Matching compares each image, a1.txt, with every image in the a1.txt:

- Click menu item “BioFilter/Matching/N:N (a1.txt vs. a1.txt)” button to complete a N:N Match.

The results will go to a file, b1.txt, which will be opened right after the click. The file will look like this:

C:\...\L01008gi-020501.jpg
C:\...\L01008gi-020501.jpg
638
C:\...\L01008gi_r90.jpg
510

C:\...\L01008gi_r90.jpg
C:\...\L01008gi-020501.jpg
510
C:\...\L01008gi_r90.jpg
638
…

Again, line 1 in each block is the input and the rest of the lines are output. Go all the way to the end of the file; the last line indicates the number of matches in the N:N Matching.

14.8 N:N Matching Implementation

Double click menu item “BioFilter/Matching/N:N (a1.txt vs. a1.txt)” and enter:

```
private void menuItem43_Click(object sender, System.EventArgs e)
{
   this.mainMenuToAPI.bioFilter_Matching_NToM
        ( dataDir + "a1.txt", dataDir + "a1.txt", dataDir + "b1.txt" );
}
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool bioFilter_Matching_NToM
        ( string a1File, string a2File, string b1File)
        {
                try
                {
                script.bioFilter .findMatchNM ( a1File, a2File, b1File);
                }
                catch (Exception e)
                {
                        appendText (  e.ToString () + "\n");
                        return false;
                }
                return true;
        }
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, “script.BioFilter” is the BioFilter object. The parameters, (a1File, a2File, b1File), indicate all signatures in a1File will be matched against all signatures in a2File. After completion of the matching, the results will be stored in a file specified by the last parameter, b1File.

14.9 1:N Matching Design

1:N Matching compares one key image with the images in a search directory; the key image is selected by the “Key” button.

To continue the **Label Recognition problem** for 1:N Matching:

- Click the “Key” button, in the “ex_label” directory, select the first image “L01008gi-020501.jpg”;
- Click menu item “BioFilter/Matching/1:N (Key vs. a1.txt)” button to complete a 1:N Match.
- The results are in file, b1.txt, which will be opened at this point:

ID	Name	Path	Score	X	Y	W	H
L01008gi-020501	L01008gi-020501.jpg	C:\...\ex_label\	638	0	0	0	0
L01008gi_r90	L01008gi_r90.jpg	C:\...\ex_label\	510	0	0	0	0

14.10 1:N Matching Implementation

Double click menu item “BioFilter/Matching/1:N (Key vs. a1.tx)” and enter:

```
private void menuItem41_Click(object sender, System.EventArgs e)
        {
        bool b = this.mainMenuToAPI .bioFilter_Matching_1ToN
                ( textBox1.Text , dataDir + "a1.txt",  dataDir + "b1.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool bioFilter_Matching_1ToN
        ( string key, string a1File, string b1File)
        {
        bool b = false;

        if ( ! System.IO .File .Exists (key) )
                {
                        appendText ( "Please enter a valid Key!\n");
                        return false;
                }

        b = this.signature_Key_Signature (key);
                if ( ! b )
                {
                        appendText ( "Key Signature computation fails!\n");
                        return false;
                }

        try
                {
                script.results_1N = script.bioFilter .findMatch1N
                        (  script.imageSignature,  a1File,  b1File);
                }
                catch (Exception e)
                {
                        appendText ( "BioFilter 1:N Matching fails:\n"
                        + e.ToString () + "\n");
                        return false;
                }

        if ( script.results_1N == null )
                {
```

```
                    appendText ( "BioFilter 1:N Matching fails!\n" );
                    return false;
                }

            if ( script.results_1N.getStatus () )
                {
                setText ( script.results_1N.toString () + "\n");
                appendText ("" + script.results_1N.getNumberOfMatches ()
                +" matches!\n");
                }
                else
                {
                    appendText ( "No Match!\n");
                    return false;
                }
            }
```

The following code simply makes sure the key image exists:

```
            if ( ! System.IO .File .Exists (key) )
                {
                    appendText ( "Please enter a valid Key!\n");
                    return false;
                }
```

The next section of code computes the key signature:

```
    b = this.signature_Key_Signature (key);
```

The next section code makes a 1:N Match:

```
            try
                {
                script.results_1N = script.bioFilter .findMatch1N
                    (  script.imageSignature,  a1File,  b1File);
                }
                catch (Exception e)
                {
                    appendText ( "BioFilter 1:N Matching fails:\n"
                    + e.ToString () + "\n");
                    return false;
                }
```

The final section of code prints the 1:N Matching results:

```
            if ( script.results_1N.getStatus () )
                {
                setText ( script.results_1N.toString () + "\n");
                appendText ("" + script.results_1N.getNumberOfMatches ()
```

```
+" matches!\n");
}
else
{
        appendText ( "No Match!\n");
        return false;
}
```

15. NeuralFilters

The NeuralFilter matches two whole images, which is similar to the BioFilter. The NeuralFilter is better than both Unsupervised Filter and BioFilter, but it requires a large amount of training data. Training data teaches the NeuralFilter who should match with whom. In comparison to early filters:

- The advantage of the NeuralFilter is that it is more accurate.
- The disadvantage of the NeuralFilter is that it requires more training data than the BioFilter.

The chapter project is located at:

C:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\". The data is located at: ".\ex_label" or "c:\transapplet70\imagefinder\bin\Release\ ex_label".

15.1 NeuralFilter Menu

The NeuralFilter Matching will have four steps:

- Signatures
- Training
- Matching
- Analysis

This process is identical to the Bio Filter in the last chapter. The NeuralFilter menu items are identical to the BioFilter menu items.

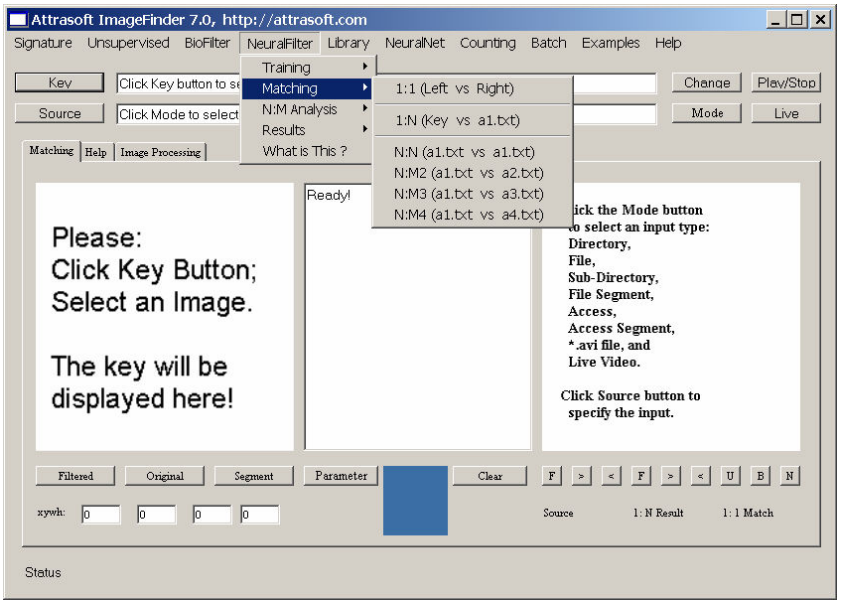

Figure 15.1 NeuralFilter Menu.

15.2 NeuralFilter API

The following table lists the NeuralFilter functions.

Functions	Descriptions
bool training (string a1_txt, string match_txt)	Trains the NeuralFilter.
Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig1, Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (string path1, string path2); Attrasoft.TransApplet70.Results_1N .Results_1N findMatch11 (Bitmap left, Bitmap right);	Makes a 1:1 matching
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Attrasoft.TransApplet70.ImageSignature70.ImageSignature sig, string a1File, string b1File); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string keyuPath, string a1File, string b1File) Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (Bitmap keyImage, string a1File, string b1File)	Makes a 1:N matching
bool findMatchNN (string a1File, string b1File); bool findMatchNM (string a1File, string a2File, string b1File);	Matches all image signatures in file

	a1File against all image signatures in a1File or a2File and saves the results to b1File.

15.3 Parameters

The **ImageFinder** has many parameters. Changing these parameters will change the output of the computations. The parameters are set by clicking the "Parameter button", which will open the Parameter Window in Figure 15.2. You will adjust the **ImageFinder** parameters here.

The default setting uses the Signature Filter 0, which is the least accurate Signature Filter. There are 20 Signature filters in the current version of the **ImageFinder**. In general, when you have less data, use less accurate Signature filters; when you have more data, use more accurate Signature filters.

There are really no hard guidelines; the following is only a rough reference for the NeuralFilter:

Signature Filter	BioFilter (Pairs)	Neural Filter (Pairs)
0	10	50
1	30	150
2	30	150
3	70	200
4	70	200
5	150	400
6	150	400
7	310	600
8	310	600
9	630	1000
10	630	1000
> 11	1000	2000

15.4 Training Design

We introduced the **Label Recognition problem** with both the Unsupervised Filter and the BioFilter. We will choose "Signature Filter 9" in this chapter. In Figure 15.2, select Signature Filter 9 from line 6.

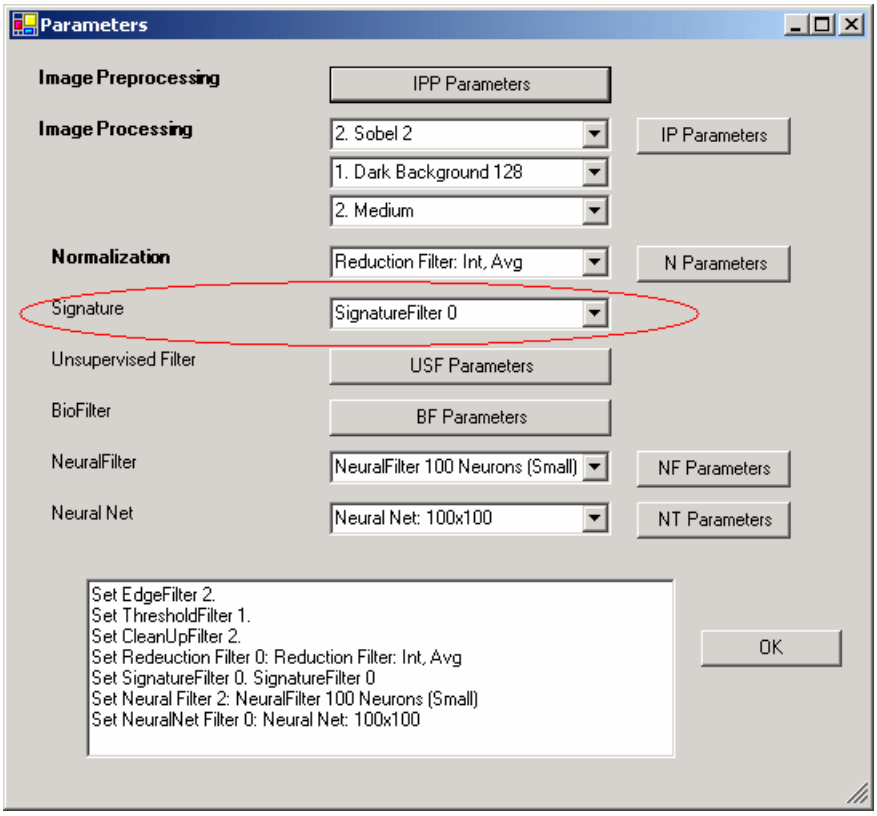

Figure 15.2 Parameter Window.

We now revisit the **Label Recognition example**. We must prepare the match.txt file for training. This file is already prepared for you and we will simply open it and save it to match.txt. The steps are:

Match.txt

- Open the file, ".\data\match_ex_label.txt". This file lists 152 matching pairs. Save it to match.txt (overwrite the existing file). Now the training file is prepared.

T1.txt:

- Click the "Source" button, go to ".\ex_label" directory and select any file in the folder. This will specify the input directory.
- Click the Source ">" button a few times to see the images;
- Click menu item "Signature/N Signature (a1.txt)" to get signature file, a1.txt file;
- Click menu item "Signature/Copy a1.txt to t1.txt" to get the training file, t1.txt.

Note: Here t1.txt is for training and a1.txt is for 1:N Matching and N:N Matching.

Training

- Click "Neural Filter\Training\Training" to train the NeuralFilter.

You should get this message at the end of the text window:

Number of Matches = 152
Neural Filter Training Completed!

The match.txt listed 152 pairs. The first line, Total Number of Matches = 152, indicates the training used 152 pairs. Now, the **Neural Filter** is trained for the **Label Recognition problem**. We will continue this example in the next section, N:N Matching.

15.5 Training Implementation

Double click menu item "NeuralFilter\Training\Training" and enter:

```
private void menuItem64_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI .neuralFilter_Train_Train
                ( dataDir + "t1.txt" , dataDir + "match.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool neuralFilter_Train_Train ( string a1_txt, string match_txt)
        {
        bool b = script.neuralFilter  .training (  a1_txt, match_txt );
        return b;
        }
```

15.6 N:N Matching Design

N: N Matching compares each image in a1.txt with every image in the a1.txt:

- Click menu item "NeuralFilter/Matching/N:N (a1.txt vs. a1.txt)" button to complete a N:N Match.

The results will go to a file, b1.txt, which will be opened right after the click. The file will look like this:

```
C:\...\ex_label\L01008gi-020501.jpg
C:\...\ex_label\L01008gi-020501.jpg
242307
C:\...\ex_label\L01008gi_r90.jpg
153038

C:\...\ex_label\L01008gi_r90.jpg
C:\...\ex_label\L01008gi-020501.jpg
153038
C:\...\ex_label\L01008gi_r90.jpg
242307
```

…

Again, line 1 in each block is the input and the rest of the lines are output. Go all the way to the end of the file; the last line indicates the number of matches in the N:N Matching.

15.7 N:N Matching Implementation

Double click menu item “NeuralFilter/Matching/N:N (a1.txt vs. a1.txt)” and enter:

```
private void menuItem71_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI.neuralFilter_Matching_NToM
                ( dataDir + "a1.txt", dataDir + "a1.txt", dataDir + "b1.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
Public bool neuralFilter_Matching_NToM
        ( string a1File, string a2File, string b1File)
        {
                try{
                script.neuralFilter.findMatchNM ( a1File, a2File, b1File);
                }
                catch (Exception e)
                {
                        appendText (  e.ToString () + "\n");
                        return false;
                }
                return true;
        }
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, script.neuralFilter is the NeuralFilter object. The parameters, (a1File, a2File, b1File), indicate all signatures in a1File will be matched against all signatures in a2File. After completion of the matching, the results will be stored in a file specified by the last parameter, b1File.

15.8 1:N Matching Design

1:N Matching compares one key image with the images in the a1.txt; the key image is selected by the “Key” button.

To continue the **Label Recognition problem** for 1:N Matching:

- Click the “Key” button, in the “ex_label”directory, select the first image “L01008gi-020501.jpg”;
- Click menu item “NeuralFilter/Matching/1:N (Key vs. a1.txt)” button to complete a 1:N Match.
- The results are in file, b1.txt, which will be opened at this point:

ID	Name	Path	Score	X	Y	W	H
L01008gi-020501	L01008gi-020501.jpg	C:\...\ex_label\	242307	0	0	0	0
L01008gi_r90	L01008gi_r90.jpg	C:\...\ex_label\	153038	0	0	0	0

15.9 1:N Matching Implementation

Double click menu item “NeuralFilter/Matching/1:N (Key vs. a1.tx)” and enter:

```
private void menuItem66_Click(object sender, System.EventArgs e)
        {
                this.mainMenuToAPI .neuralFilter_Matching_1ToN
                        ( textBox1.Text , dataDir + "a1.txt",  dataDir + "b1.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool neuralFilter_Matching_1ToN
        ( string key, string a1File, string b1File)
        {
        bool b = false;

        if ( ! System.IO .File .Exists (key) )
                {
                        appendText ( "Please enter a valid Key!\n");
                        return false;
                }

        b = this.signature_Key_Signature (key);
                if ( ! b )
                {
                        appendText ( "Key Signature computation fails!\n");
                        return false;
                }

        try
                {
                script.results_1N = script.neuralFilter.findMatch1N
                        ( script.imageSignature,  a1File,  b1File);
                }
                catch (Exception e)
                {
```

```
                    appendText ( "NeuralFilter 1:N Matching fails:\n"
                    + e.ToString () + "\n");
                    return false;
                }
            if ( script.results_1N == null )
                {
                    appendText ( "NeuralFilter 1:N Matching fails!\n" );
                    return false;
                }

            if ( script.results_1N.getStatus () )
                {
                setText ( script.results_1N.toString () + "\n");
                appendText ("" + script.results_1N.getNumberOfMatches ()
                    +" matches!\n");
                }
            else
                {
                    appendText ( "No Match!\n");
                    return false;
                }

                return createFile (key, b1File);
            }
```

The following code simply makes sure the key image exists:

```
            if ( ! System.IO .File .Exists (key) )
                {
                    appendText ( "Please enter a valid Key!\n");
                    return false;
                }
```

The next section of code computes the key signature:

```
        b = this.signature_Key_Signature (key);
```

The next section code makes a 1:N Match:

```
            try
                {
                script.results_1N = script.neuralFilter.findMatch1N
                    ( script.imageSignature,  a1File,  b1File);
                }
                catch (Exception e)
                {
                    appendText ( "NeuralFilter 1:N Matching fails:\n"
                    + e.ToString () + "\n");
                    return false;
```

```
}
```

The final section of code prints the 1:N Matching results:

```
if ( script.results_1N.getStatus () )
        {
        setText ( script.results_1N.toString () + "\n");
        appendText ("" + script.results_1N.getNumberOfMatches ()
                +" matches!\n");
        }
else
        {
                appendText ( "No Match!\n");
                return false;
        }
```

16. Dynamic Library

In the last a few chapters, we introduced 1:N Match and N:N Match, where N is fixed. This chapter will introduce the Dynamic Library where N can be updated via:

- insertion,
- deletion, and
- replacement.

The chapter project is located at:

c:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\". We will introduce a **Logo Recognition** example. The data is located at:

.\ex_dynamic_lib\	Original logo data
.\ex_dynamic_lib**add**	Add to the library later

16.1 Dynamic Library Menu

The Dynamic Library works only for the NeuralFilter Matching; there are six steps:

- Parameters
- Signatures
- Training
- Matching
- Update Dynamic Library and Match Again
- Analysis

Figure 16.1 and Figure 16.2 shows the dynamic library menu. Please create the menu items accordingly.

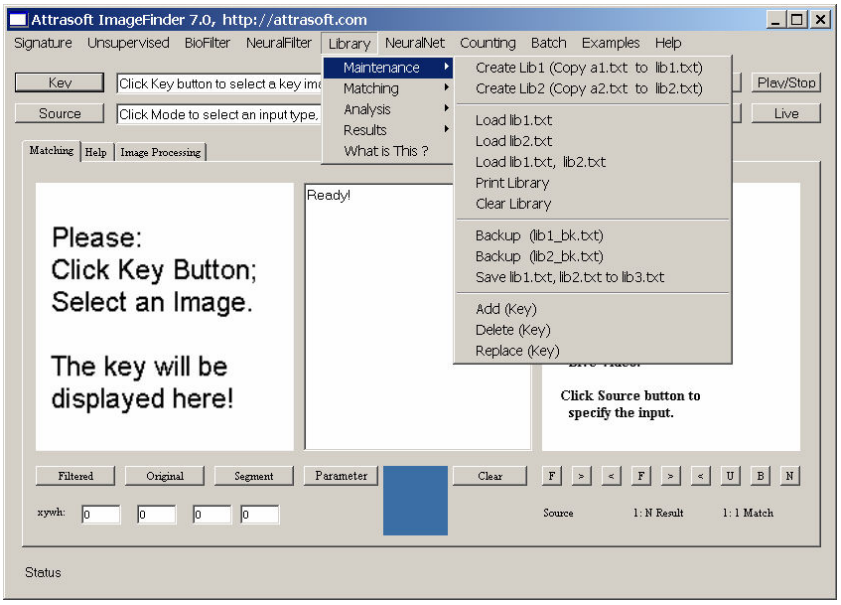

Figure 16.1 Dynamic Library Menu, 1 of 2.

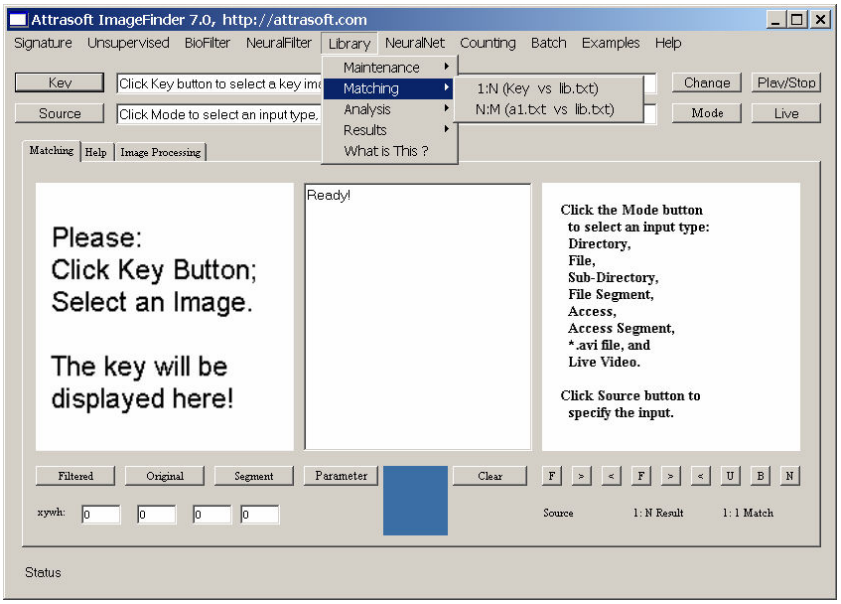

Figure 16.2 Dynamic Library Menu, 2 of 2.

16.2 Dynamic Library API

The following table lists the dynamic library functions.

Functions	Descriptions	Comments
int getLibraryID ()	Gets the Library ID (optional).	
void setLibraryID (string	Sets the Library ID	

x)	(optional).	
bool load ()	Loads the default master library, lib1.txt.	
bool load (string fileName)	Loads master library specified by fileName.	
bool load (string fileName1, string fileName2)	Loads two libraries specified by fileName1 and fileName2.	
bool clear()	Clears the current library from RAM only.	
bool backup ()	Saves current library to the default file, lib1_bk.txt.	
bool backup (string fileName)	Saves current library to a back file.	
bool addSignature (ImageSignature sig)	Adds a signature to a loaded image library in RAM.	
bool deleteSignature (ImageSignature sig)	Deletes a signature from a loaded image library in RAM.	
bool deleteSignature (string ID)	Deletes a signature from a loaded image library in RAM.	
bool replaceSignature (ImageSignature sig)	Replaces a signature from a loaded image library in RAM.	
bool mergeLibrary (string libFile1, string libFile2, string outputLib)	Merges two signature libraries into a single library. Input: string libFile1 string libFile2 string outputLib Output: A text file that contains the library of signatures from both input libraries.	

16.3 Creating Master Library

Up to this point when we have done 1:N or N:N Matching, the N-images have been fixed. The Dynamic Library allows you to update the N-images, including inserting, deleting, and replacing signatures. We now create the master library. **The master library file is lib1.txt.**

First of all, we will set the parameters for the **Logo Recognition example**. The **ImageFinder** has many parameters. Changing these parameters will change the output of the computations. Click the Parameter button, and set:

- Signature Filter = "Signature Filter 15";
- NeuralFilter/Fault Tolerance scale = 10;
- NeuralFilter/Blurring = 0;
- NeuralFilter/Sensitivity = 0;

To compute the N-signatures:

- Click the "Source" button to select directory, ".\ex_dynamic_lib\", then select any file in this folder;
- Click "Signature/N Signature (a1.txt)" to compute the signatures in a1.txt.

Now we have created the a1.txt file. To use the Dynamic Library, you have to create a library file lib1.txt. To create a library file, open a1.txt and save it to lib1.txt.

16.4 Training Design

The dynamic library will use the same NeuralFilter object in the last chapter. Therefore, the training implementation is already completed. Again, training requires two files:

Match.txt

- Open the file, ".\data\match_ex_dynamic_lib.txt". Save it to match.txt (overwrite the existing file). Now the training file is prepared.

T1.txt:

- In the last step, we obtained a1.txt;
- Click menu item "Signature/Copy a1.txt to t1.txt" to get the training file, t1.txt.

Training

- Click "NeuralFilter\Training\Training" to train the Neural Filter.

16.5 Load Dynamic Library

The dynamic library uses a file lib1.txt as a master library file. Before we can use the dynamic library, this file must be loaded into the dynamic library. To load the library file, click "Library/Maintenance/Load lib1.txt".

Double click menu item "Library/Maintenance/Load lib1.txt" and enter:

```
private void menuItem89_Click (object sender, System.EventArgs e)
        {
        this.mainMenuToAPI.library_loadLib (dataDir +"lib1.txt" );
```

```
}
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool library_loadLib    ( string lib1_txt)
        {
        if (script.imageLibrary == null )
                {
                        appendText ( "Unable to find a library !\n");
                        return false;
                }
        try
                {
                        bool b = script.imageLibrary.load  ( lib1_txt);
                        if ( b )
                        {
                                appendText ("Library 1 loaded!\n");

                        }
                }
        catch (Exception ee )
                {
                        appendText ( ee.ToString () + "\n" );
                        return false;
                }
                return true;
        }
```

The following statement loads the library:

```
bool b = script.imageLibrary.load  ( lib1_txt);
```

16.6 Library M:N Matching

In an N:M Matching, the N-images are in a1.txt and M-images are in lib1.txt. To make a M:N Matching,

- Click menu item “Library/Matching/N:M (a1.txt vs. lib1.txt)” menu item to complete a N:M Match.

Double click menu item “Library/Matching/N:M (a1.txt vs. lib1.txt)” and enter:

```
private void menuItem100_Click(object sender, System.EventArgs e)
        {
        bool b = this.mainMenuToAPI.library_Matching_N_M
                ( dataDir + "a1.txt",  dataDir + "b1.txt" );
```

```
}
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool library_Matching_N_M  ( string a1_txt,    string b1_txt)
        {
        if ( script.imageLibrary == null )
        {
                appendText ("Dynamic Library not available!\n");
                return false;
        }

        bool b = script.neuralFilter .setLibrary ( script.imageLibrary );
        if (!b)
        {
                appendText ("Dynamic Library Assignment Fails!\n");
                return false;
        }
        try
        {

                script.neuralFilter.findMatchNN ( a1_txt,  b1_txt);
        }
        catch (Exception e)
        {
                appendText (  e.ToString () + "\n");
                return false;
        }
        return true;
        }
```

In this statement, the script object contains all of the objects required for the **ImageFinder** project. In particular, “script.imageLibrary” is the dynamic library object.

The following statement assigns the dynamic library to the NeuralFilter object, “script.NeuralFilter”:

```
bool b = script.neuralFilter .setLibrary ( script.imageLibrary );
        if (!b)
        {
                appendText ("Dynamic Library Assignment Fails!\n");
                return false;
        }
```

Once the NeuralFilter object obtains this library, it will match images against this master library.

16.7 Library 1:N Matching

To make 1:N Matching via the library,

- Click the “Key” button, in the “.\ex_dynamic_lib” directory, select the first image “A10.jpg”;
- Click menu item “Library/Matching/1:N (Key vs. lib1.txt)” button to complete a 1:N Match.

Double click menu item “Library/Matching/1:N (Key vs. lib1.txt)” and enter:

```
private void menuItem98_Click(object sender, System.EventArgs e)
        {
        bool b = this.mainMenuToAPI.library_Matching_1_N
                (textBox1.Text  , dataDir + "b1.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool library_Matching_1_N( string key, string b1File)
        {
        if ( script.imageLibrary == null )
        {
                appendText ("Dynamic Library not available!\n");
                return false;
        }
        bool b = script.neuralFilter .setLibrary ( script.imageLibrary );
        if (!b)
        {
                appendText ("Dynamic Library Assignment Fails!\n");
                return false;
        }
        b = false;

        if ( ! System.IO .File .Exists (key) )
        {
                appendText ( "Please enter a valid Key!\n");
                return false;
        }

        b = this.signature_Key_Signature (key);
        if ( ! b )
        {
                appendText ( "Key Signature computation fails!\n");
                return false;
        }

        try
```

```
{
script.results_1N = script.neuralFilter.findMatch1N
                    ( script.imageSignature);
}
catch (Exception e)
{
        appendText ( "NeuralFilter 1:N Matching fails:\n"
        + e.ToString () + "\n");
        return false;
}
if ( script.results_1N == null )
{
        appendText ( "NeuralFilter 1:N Matching fails!\n" );
        return false;
}

if ( script.results_1N.getStatus () )
{
        setText ( script.results_1N.toString () + "\n");
        appendText ("" + script.results_1N.getNumberOfMatches ()
        +" matches!\n");
}
else
{
        appendText ( "No Match!\n");
        return false;
}
}
```

The only difference between this 1:N Matching and the NeuralFilter 1:N Matching introduced earlier is the following statement:

```
bool b = script.neuralFilter .setLibrary ( script.imageLibrary );
```

In a normal 1:N Matching, the neural filter gets the library from a1.txt. This statement assigns the dynamic library, "script.imageLibrary", to the NeuralFilter object, "script.NeuralFilter".

16.8 Library Updating Design

Now, we will add images in folder, ".\ex_dynamic_lib**add**\". To make 1:N Matching via the library,

- Click the "Key" button, in the ".\ex_dynamic_lib**add**\" directory; select the image "**A416.jpg**";
- Click menu item "Library/Matching/1:N (Key vs. lib1.txt)" button to complete a 1:N Match.

You will get:

ID	Name	Path	Score	X	Y	W	H
A324	A324.jpg	C:\...\ex_dynamic_lib\	142953	0	0	0	0

B10	B10.jpg	C:\...\ex_dynamic_lib\	137606	0	0	0	0
A166	A166.jpg	C:\...\ex_dynamic_lib\	135926	0	0	0	0
A142	A142.jpg	C:\...\ex_dynamic_lib\	133745	0	0	0	0
B206	B206.jpg	C:\...\ex_dynamic_lib\	132967	0	0	0	0
B11	B11.jpg	C:\...\ex_dynamic_lib\	130253	0	0	0	0
B35	B35.jpg	C:\...\ex_dynamic_lib\	124340	0	0	0	0
B76	B76.jpg	C:\...\ex_dynamic_lib\	123182	0	0	0	0

To add this image, A416.jpg, to the library, click "Library/Maintenance/Add (Key)".

Now, make a 1:N Match again:

- Click menu item "Library/Matching/1:N (Key vs. lib1.txt)" button to complete a 1:N Match.

You will get:

ID	Name	Path	Score	X	Y	W	H
A416	**A416.jpg**	**C:\...\ex_dynamic_lib\add**	**345857**	**0**	**0**	**0**	**0**
A324	**A324.jpg**	**C:\...\ex_dynamic_lib**	**142953**	**0**	**0**	**0**	**0**
B10	**B10.jpg**	**C:\...\ex_dynamic_lib**	**137606**	**0**	**0**	**0**	**0**
A166	**A166.jpg**	**C:\...\ex_dynamic_lib**	**135926**	**0**	**0**	**0**	**0**
A142	**A142.jpg**	**C:\...\ex_dynamic_lib**	**133745**	**0**	**0**	**0**	**0**
B206	**B206.jpg**	**C:\...\ex_dynamic_lib**	**132967**	**0**	**0**	**0**	**0**
B11	**B11.jpg**	**C:\...\ex_dynamic_lib**	**130253**	**0**	**0**	**0**	**0**
B35	**B35.jpg**	**C:\...\ex_dynamic_lib**	**124340**	**0**	**0**	**0**	**0**
B76	**B76.jpg**	**C:\...\ex_dynamic_lib**	**123182**	**0**	**0**	**0**	**0**

As you can see, A416.jpg has been added to the library. Now we will delete it from the library by clicking "Library/Maintenance/Delete (Key)". Now make a 1:N Match again:

- Click menu item "Library/Matching/1:N (Key vs. lib1.txt)" button to complete a 1:N Match.

Now **A416.jpg** will no longer be in the output.

16.9 Update Implementation

Double click menu item "**Library/Maintenance/Add (Key)"** and enter:

```
private void menuItem95_Click (object sender, System.EventArgs e)
        {
        if ( ! System.IO .File .Exists (textBox1.Text  ) )
        {
                this.mainMenuToAPI.appendText
                ("File does not exist: \n" + textBox1.Text + "\n");
                return;
        }
        this.mainMenuToAPI.library_add (textBox1.Text  );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool library_add ( string sImage)
        {
        script.imageSignature
                = script.signatureFilter.getSignature (sImage);

        if (script.imageSignature == null )
        {
                this.setLine ("Image Signature computation fails!");
                return false;
        }

        if (script.imageSignature.getStatus () <= 0 )
        {
                this.setLine ("Image Signature computation fails!");
                return false;
        }
        bool b =  script.imageLibrary.addSignature (script.imageSignature);
        return b;
        }
```

The script object contains all of the objects required for the **ImageFinder** project. In particular, "script.imageLibrary" is the dynamic library object. The first section of the code computes the image signature. The next section of code checks the validity of the signature. The following statement inserts the signature to the dynamic library:

```
bool b =  script.imageLibrary.addSignature (script.imageSignature);
```

The menu item Delete and the menu item Replace can be implemented in a similar fashion and the codes are provided in the project.

17. NeuralNet Filter

Up to this point, we have focused on matching whole images. The NeuralNet Filter matches a segment of an image(s).

As we have seen, accurate matching via the NeuralFilter requires many matching pairs. Preparing matching pairs for whole images means listing all pairs in the match.txt file.

Preparing matching pairs for image segments is much harder; therefore, rather than using the NeuralFilter for image segments, we will use the Unsupervised Filter for image segments. As we have seen, the Unsupervised Matching for image segments is not as accurate as the NeuralFilter.

Figure 17.1 Locating an Image Segment.

The chapter project is located at:

c:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\".

In this chapter, we will introduce a **Trademark Recognition example**. The objective is to identify and locate the trademarks in an image. Figure 17.1 shows a typical image. The data is stored in the folder, ".\input_auto_track".

17.1 Key Segment Specification

A path, such as "c:\abc\xyz.jpg", specifies the key image. The key segment is specified by a path and (x, y, w, h). Here (x, y) is the coordinate of the upper left corner, w is the width, and h is the height.

An image segment is specified via its pixel values. For example, let an image be 256x256, then (0, 0, 128, 128) specifies a quarter of the image located in the upper left corner.

Figure 17.2 shows how to specify an image segment in 2 steps:

- Enter (x, y, w, h) into the four textboxes in Figure 17.2;
- Click the Segment button to enter (x, y, w, h) to the software.

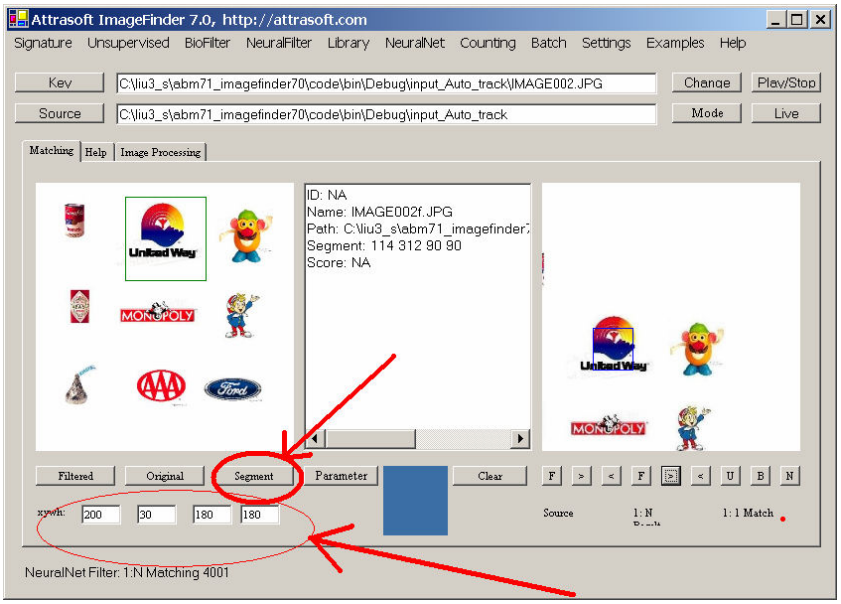

Figure 17.2 Specify an Image Segment.

Example. Specifying a Key Segment:

- Click the "Key" button, in the ".\input_auto_track" directory, select image "IMAGE002.jpg";
- Enter (200, 30, 180, 180) to the segment textboxes in Figure 17.2.
- Click the "Segment" button and the segment is marked by a square in Figure 17.2..

17.2 NeuralNet Filter Menu

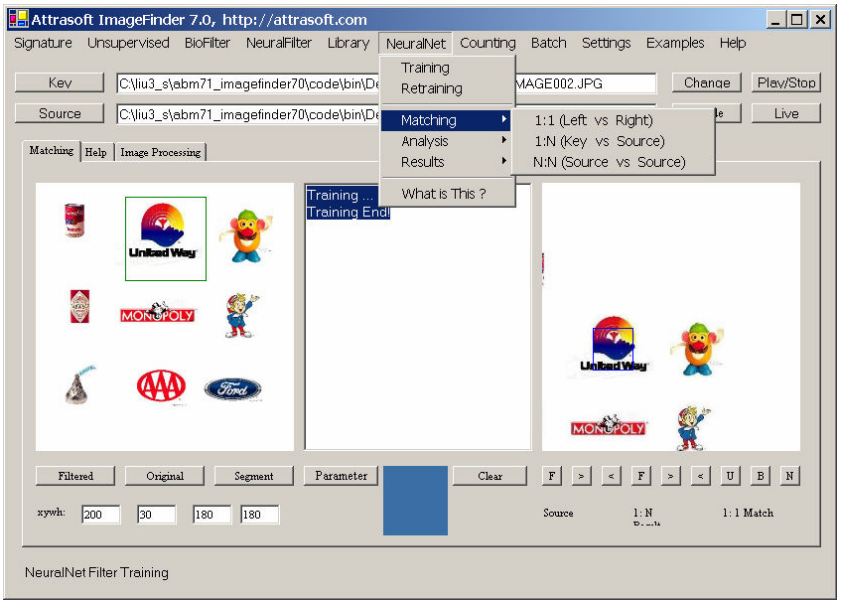

Figure 17.3 Neural Net Training & Matching.

Figure 17.3 shows the Neural Net menu item. Please implement the menu items accordingly.

17.3 NeuralNet Filter API

The following table lists the NeuralNet Filter functions.

Functions	Descriptions
bool train (Bitmap img) bool train (string sPath) bool train (Bitmap img, int x, int y, int w, int h) bool train (string sPath, int x, int y, int w, int h)	Trains the neural net.
bool retrain (Bitmap img) bool retrain (string sPath) bool retrain (Bitmap img, int x, int y, int w, int h) bool retrain (string sPath, int x, int y, int w, int h)	Retrains the neural net.
Attrasoft.TransApplet70.Results_1N.Results_1N findMatch11 (Bitmap img1); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch11 (string path1); Attrasoft.TransApplet70.Results_1N.Results_1N findMatch1N (string [] fileList);	Makes a 1:1 and 1:N Matching.
bool findMatchNN (string [] fileList, string c1File) bool findMatchNM (string [] keyList, string [] libraryList, string c1File);	Makes a N:N Matching and N:M Matching.

17.4 Training

Training here means setting up the NeuralNet Filter. After specifying a key segment, click "NeuralNet/Training" in Figure 17.3 to complete the training. You should see this message:

Training ...
Training End!

Double click menu item "NeuralNet/Training" and enter:

```
private void menuItem109_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI.neuralNet_Training ( textBox1.Text  );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool neuralNet_Training ( string b)
        {
        if ( ! System.IO .File .Exists (b) )
        {
                appendText ( "Please enter a valid Key!\n");
                return false;
        }

        int x =0, y = 0, w =0, h = 0;
        try
        {
                x = int.Parse (f.textBox3.Text );
                y = int.Parse (f.textBox4.Text );
                w = int.Parse (f.textBox5.Text );
                h = int.Parse (f.textBox6.Text );
        }
        catch
        {
                setText ("Invalid Training Segment!\n");
                x = 0;
                y = 0;
                w = 0;
                h = 0;
                return false;
        }

        script.neuralNetFilter.train ( b, x, y, w, h) ;
        return true;
        }
```

The following code simply makes sure the key image exists:

```
if ( ! System.IO .File .Exists (b) )
        {
                appendText ( "Please enter a valid Key!\n");
                return false;
        }
```

The next section of code computes (x, y, w, h):

```
int x =0, y = 0, w =0, h = 0;
        try
        {
                x = int.Parse (f.textBox3.Text );
                y = int.Parse (f.textBox4.Text );
                w = int.Parse (f.textBox5.Text );
                h = int.Parse (f.textBox6.Text );
        }
        catch
        {
                setText ("Invalid Training Segment!\n");
                x = 0;
                y = 0;
                w = 0;
                h = 0;
                return false;
        }
```

The last section of code trains the neural net:

```
script.neuralNetFilter.train ( b, x, y, w, h) ;
```

17.5 1:N Matching Design

To make a 1:N Matching:

- Click the "Source" button to select directory, ".\input_auto_track"; then select any file in this folder;
- Click "NeuralNet/Matching/1:N (Key vs Source)" to make a 1:N Matching (See Figure 17.3).

The output will look like this:

ID	Name	Path	Score	X	Y	W	H	R
IMAGE002	IMAGE002.JPG	C:\...\input_Auto_track\	74240	204	42	162	126	0
IMAGE002a	IMAGE002a.JPG	C:\...\input_Auto_track\	70080	228	42	162	126	0
IMAGE002b	IMAGE002b.JPG	C:\...\input_Auto_track\	71616	240	30	162	126	0
IMAGE002c	IMAGE002c.JPG	C:\...\input_Auto_track\	72640	246	66	162	126	0
IMAGE002d	IMAGE002d.JPG	C:\...\input_Auto_track\	70720	246	120	162	126	0

IMAGE002e	IMAGE002e.JPG	C:\...\input_Auto_track\	71104	138	150	162	126	0
IMAGE002f	IMAGE002f.JPG	C:\...\input_Auto_track\	108096	84	288	162	126	0
IMAGE004	IMAGE004.JPG	C:\...\input_Auto_track\	70080	12	24	162	126	0
IMAGE006	IMAGE006.JPG	C:\...\input_Auto_track\	67008	378	204	162	126	0
IMAGE008	IMAGE008.JPG	C:\...\input_Auto_track\	67200	48	60	162	126	0

17.6 1:N Matching Implementation

Double click menu item “NeuralNet/Matching/1:N (Key vs Source)” and enter:

```
private void menuItem116_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI.neuralNet_Matching_1N
                ( textBox1.Text  , gui.imageAbsoultePath, dataDir + "c1.txt" );
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public bool neuralNet_Matching_1N
        ( string key, string [] imageAbsoultePath, string c1_txt)
        {
        bool b = false;

        if ( imageAbsoultePath == null )
        {
        f.richTextBox1.Text =
        "Search Image Source Fail!\nPlease specify search source!\n" ;
        return false;
        }

        try
        {
        script.results_1N =
        script.neuralNetFilter.findMatch1N ( imageAbsoultePath ) ;
        }
        catch (Exception e)
        {
                appendText ( "NeuralFilter 1:N Matching fails:\n"
                + e.ToString () + "\n");
                return false;
        }

        if ( script.results_1N == null )
        {
                appendText ( "NeuralNet 1:N Matching fails!\n" );
                return false;
        }
        return true;
```

```
}
```

The following code simply makes sure search source exists:

```
if ( imageAbsoultePath == null )
        {
        f.richTextBox1.Text =
        "Search Image Source Fail!\nPlease specify search source!\n" ;
        return false;
        }
```

The next section of code makes a 1:N Match:

```
try
        {
        script.results_1N =
        script.neuralNetFilter.findMatch1N ( imageAbsoultePath ) ;
        }
        catch (Exception e)
        {
                appendText ( "NeuralFilter 1:N Matching fails:\n"
                + e.ToString () + "\n");
                return false;
        }
```

Remember, the neural net is already trained with the key image at this point.

17.7 Results

Although the output file specifies the segment location, it is not obviously where (x, y, w, h) is in a given image. For example, it is not clear where (202, 42, 162, 126) is in image, image002.jpg.

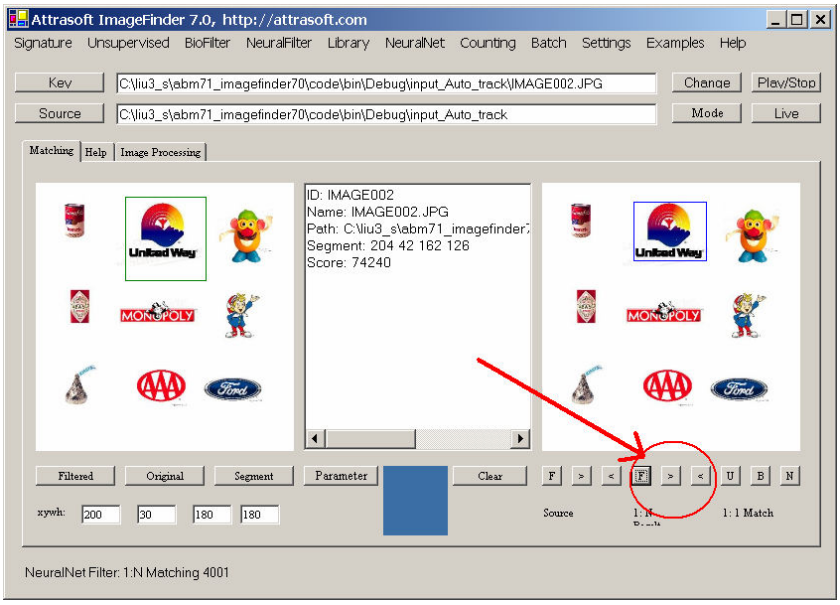

Figure 17.4. Matched Segment.

In Figure 17.4, the first picture box shows the training segment; and the second segment shows the matched segment.

To see where the matching segment is, there are three buttons in Figure 17.4:
F, > (Next), and < (Previous), that can be used to show where the matched segment is:

Click the "F" button to see the first matched segment;
Click the ">" to see the next matched segment;
Click the "<" button to see the previous matched button.

Figure 17.5 shows another matched segment.

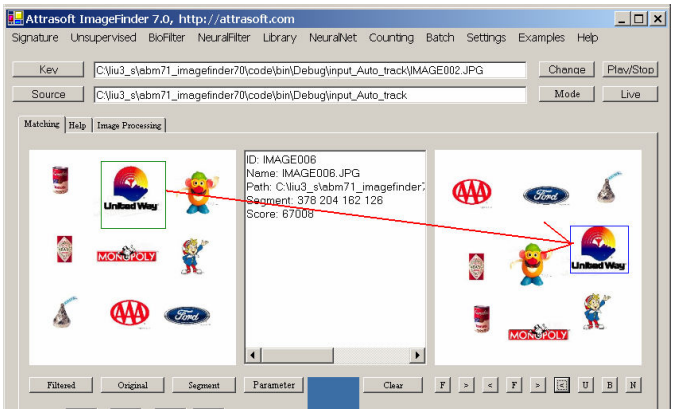

Figure 17.5 Matched Segment.

The GUI (Graphical User Interface) Implementation is beyond the scope of this software package. The source code for the input and output image display is given in the chapter project.

17.8 Another Test: Mr. Potato

Now, we want a match of the **Mr. Potato trademark** in Figure 17.3:

- Click the "Key" button, in the ".\input_auto_track" directory, select image "IMAGE002.jpg";
- Enter (400, 30, 150, 180) to the segment textboxes in Figure 17.2.
- Click the Segment button.

To train the NeuralNet Filter:

- Click "NeuralNet/training" in Figure 17.3 to complete the training.

To make a 1:N Matching:

- Click the "Source" button to select directory, ".\input_auto_track"; then select any file in this folder;
- Click "NeuralNet/Matching/1:N (Key vs Source)" to make a 1:N Matching (See Figure 17.3).

To see where the matching segment is, there are three buttons in Figure 17.4:
F, > (Next), and < (Previous), that can be used to show where the matched segment is:

Click the "F" button to see the first matched segment;
Click the ">" to see the next matched segment;
Click the "<" button to see the previous matched button.

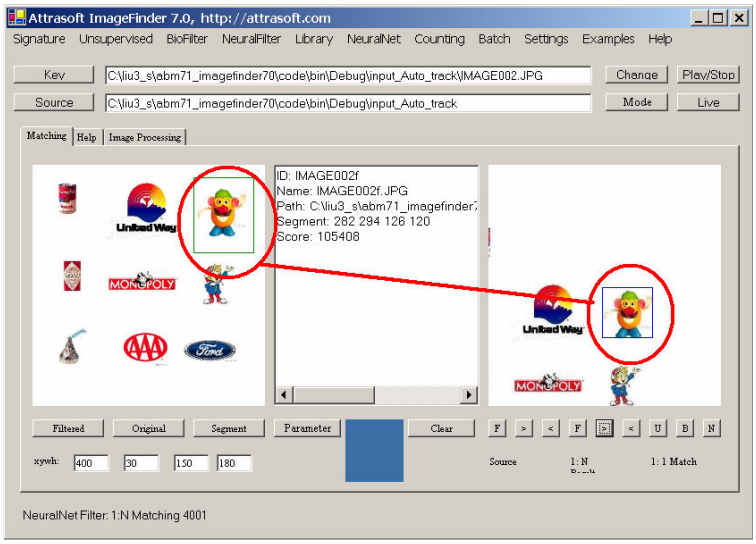

Figure 17.6 Mr. Potato.

18. Parameters

This chapter will describe the parameters in the **ImageFinder**.

18.1 Overview

Attrasoft ImageFinder can:

- Match whole images;
- Match a portion of an image.

When matching a portion of an image, similar images are defined as images containing the sample segments, or:

- Translated segments;
- Rotated segments;
- Scaled segments;
- Rotated & Scaled segments;
- Brighter or Darker segments.

To match an image, the **ImageFinder** pushes the image through many filters. For example, a set of filters could be:

Preprocessing Filters
Edge Filters
Threshold Filters
Clean-Up Filters
Reduction Filters
Unsupervised Filters
BioFilters
NeuralFilters
NeuralNet Filters

Many parameters and options of the **ImageFinder** are hidden. The users have only limited control of the parameters. Still, the **ImageFinder** has many parameters, which can be adjusted by users.

The **ImageFinder** for Windows has 70 open parameters for the user to adjust for their particular image type. You should get Identification Rates ranging from 60% to 89%; this is because the off-the-shelf **ImageFinder** only has 70 open parameters for users to adjust. The best rate, one of our customers (without any customization) was able to obtain, was an 89% Identification Rate.

However, the **ImageFinder** itself has 3000+ internal parameters, which the users have no access to at all. Fine-tuning these 3000+ internal parameters is called customization, which is Attrasoft's area of expertise. If you need increased accuracy beyond what you are able to achieve when using the **ImageFinder** for Windows, then customization will provide you with ***Your*** desired level of accuracy (ranging from 95% to 99.9%).

If you need a customized version of the **ImageFinder**, please contact imagefinder@attrasoft.com .

In a typical search, you will set these parameters and leave the other parameters with default values.

Click the "Parameter" button; you will see Figure 18.1, where you can set the parameters.

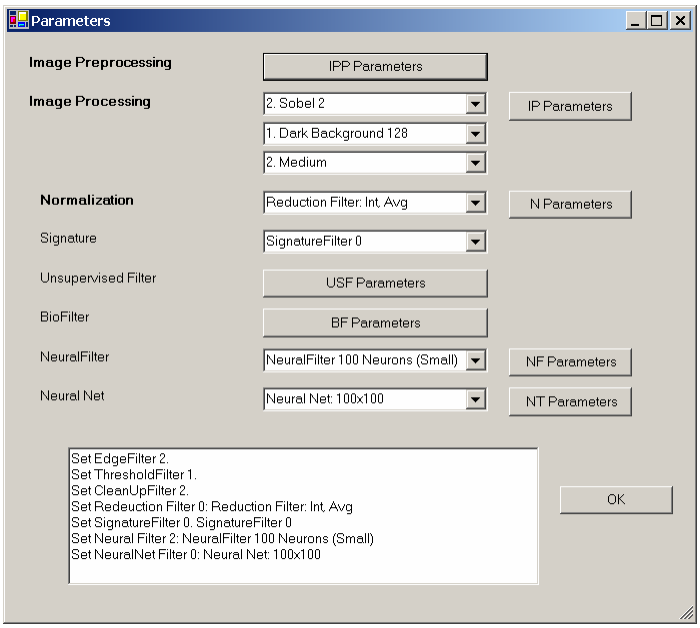

Figure 18.1 Parameter Window.

18.2 Image Preprocessing

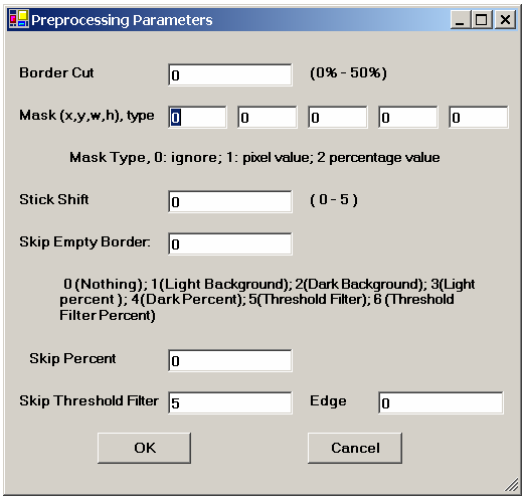

Figure 18.2 Image Preprocessing.

Border Cut

Use the “Border Cut” parameter to cut the border areas. Enter N to the first textbox in Figure 18.2, then N% near the border will be cut off.

Mask

Use the “Mask” parameter to impose a mask on the input images. Enter (x, y, w, h, 2) to the second row in Figure 18.2; then a mask in percent value will be imposed on the input images. Enter (x, y, w, h, 1) to the second row in Figure 18.2; then a mask in pixel value will be imposed on the input images.

Stick Shift

Use the “Stick Shift” parameter to speed up the computation. Set “Stick Shift” in Figure 18.2 between 0 and 5, with 0 being the slowest and 5 being the fastest.

Skip Empty Board
Skip Percent
Skip Threshold Filter
Skip Edge Filter

Use these parameters to skip the border area by cutting off N% percent of the contents.
The “Skip Empty Border” parameter in Figure 18.2 specifies the type:

0 No skip;
1 Skip the white empty border space;
2 Skip the black empty border space;
3 Skip x percent of the contents on the white background space;
4 Skip x percent of the contents on the black background space;
5 Skip empty border space on user defined Threshold Filter;
6 Skip x percent of the contents on user defined Threshold/Edge Filters.

Use the “Skip Percent” parameter to specify the percentage of content you want to cut off for Options 3, 4, and 6.

Options 1, 2, and 5 use the default setting, which is 2%. Use the “Skip Threshold Filter” and “Skip Edge Filter” to set the Edge Filter and Threshold Filter, respectively.

18.3 Image Processing

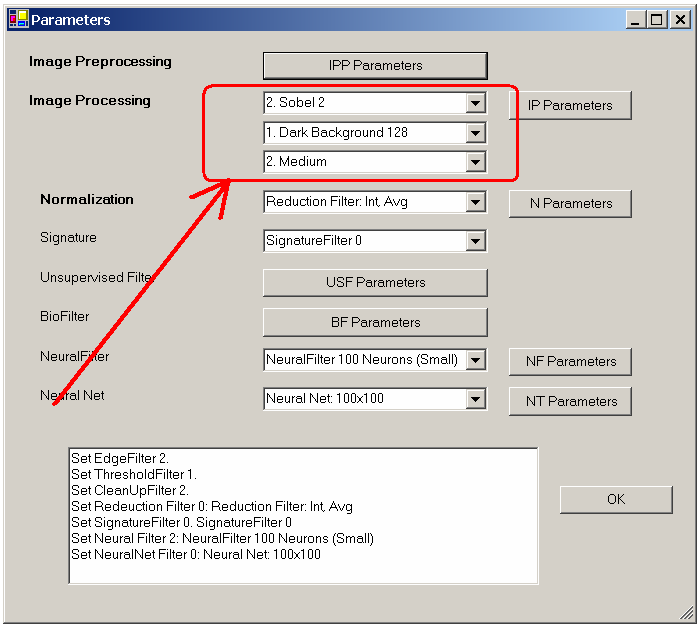

Figure 18.3 Image Processing.

18.3.1 Edge Filters

Edge Filters extract and enhance edges & contours in an image by expressing intensity differences (gradients) between neighboring pixels as an intensity value. The basic variables are the differences between the top and bottom rows, the differences between the left and right columns, and the differences between the center point and its neighbors.

Edge Filters have the following selections:

Code	Meaning
0	No Edge Filter
1	Sobel 1 (Prewitt)
2	Sobel 2 (Sobel)
3	Sobel 3
4	Sobel 4
5	Gradient
6	Gradient, 45°
7	Sobel 1, 45°
8	Sobel 1, - 45°
9	Laplacian 4
10	CD 11
11	FD 11
12	FD 9
13	FD 7
14	FD 13
15	Laplacian 5
16	Laplacian 8
17	Laplacian 9

18	Laplacian 16
19	Laplacian 17

All other filters have to be ordered in a Customized Version.

These names really do not make any sense to common users; the best way to figure out what these filters are, is to select a training image and try each of the filters. In general, these filters require the "Dark Background 128" Threshold Filter.

If you do not want to know the details, please skip the rest of this section.

The details will be given below so you will know how to order a customized filter:

Sobel 1:

-1 0 1	-1 -1 -1
-1 0 1	0 0 0
-1 0 1	1 1 1

Sobel 2:

-1 0 1	-1 -2 -1
-2 0 2	0 0 0
-1 0 1	1 2 1

Sobel 3:

-1 0 1	-1 -3 -1
-3 0 3	0 0 0
-1 0 1	1 3 1

Sobel 4:

-1 0 1	-1 -4 -1
-4 0 4	0 0 0
-1 0 1	1 4 1

Gradient:

0 0 0	0 -1 0
-1 0 1	0 0 0
0 0 0	0 1 0

Gradient, 45°

0 0 1	-1 0 0
0 0 0	0 0 0
-1 0 0	0 0 1

Sobel 1, 45°

0 1 1	1 1 0
-1 0 1	1 0 -1
-1 -1 0	0 -1 -1

Sobel 2, 45°

0 1 2	2 1 0

```
-1  0  1          1   0   -1
-2 -1  0          0  -1  -2
```

Laplacian 4

```
 0  -1  0
-1   4  -1
 0   -1  0
```

Laplacian 5

```
 0  -1  0
-1   5  -1
 0  -1  0
```

Laplacian 8

```
-1  -1  -1
-1   8  -1
-1  -1  -1
```

Laplacian 9

```
-1  -1  -1
-1   9  -1
-1  -1  -1
```

Laplacian 16

```
0   0   -1   0    0
0  -1   -2  -1   0
-1  -2  16 -2  -1
0   -1  -2  -1   0
0    0   -1   0   0
```

Laplacian 17

```
0   0   -1   0    0
0  -1   -2  -1   0
-1  -2  17 -2  -1
0   -1  -2  -1   0
0    0   -1   0   0
```

18.3.2 Threshold Filters

After Edge Filters, the Threshold Filter will be applied to the images. **Choose these two filters where the sample objects stand out, otherwise change the filters**.

If no filter in this version fits your problem, **a Customized Filter has to be built**. DO NOT make too many things stand out, i.e. as long as the area of interest stands out, the rest should show as little as possible.

Once you make a selection, the objects in the images are black and the background is white (like a book: white paper, black print). **You should make the black area as small as possible, as long as it covers the key-segment(s). Otherwise, switch to a different background.**

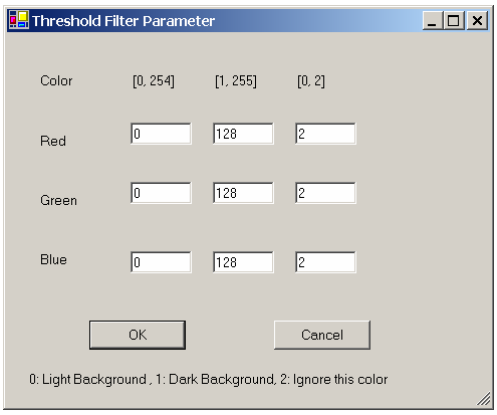

Figure 18.4 Threshold Filter Parameters.

There are 30 Threshold filters in the **ImageFinder**.

A few filters, including the average-filter and the customized-filter, allow you to specify any color range. Color is specified by three separate colors: Color = (red, green, blue). Each of the colors ranges from 0 to 255. (0, 0, 0) is black; (255, 255, 255) is white.

You should choose a filter where the sample object(s) stand out. You may want to know the meaning of the filters; example, "Light Background 128" means:

- "RGB Average in 0 – 127 " ➔ objects; and
- "RGB Average in 128 - 255" ➔ background.

To Summarize:

- **Choose an Edge Filter and a Threshold Filter where the sample object(s) stand out**;
- **Choose an Edge Filter and a Threshold Filter where the black area is as small as possible, as long as it covers the key-segment(s).**

18.3.3 Clean-Up Filters

Clean-Up Filters will clear noise off the image, but it will take more computation time.

18.4 Normalization Filter

The Normalization Filter connects the image to the underlying neural nets.

Let the underlying neural net be 100x100: if an image is larger than 100x100, say 350x230, then this image will be reduced to 100x100 or smaller.

When reducing images, a scaling factor can be introduced easily. Although scaling symmetry can compensate for this scaling factor, scaling symmetry is computationally expensive.

It is important to know that the Reduction Filter will match the selected underlying neural net, therefore, the behavior of the Reduction Filter not only depends on the selection of this filter itself, but also depends on the NeuralNet Filter chosen.

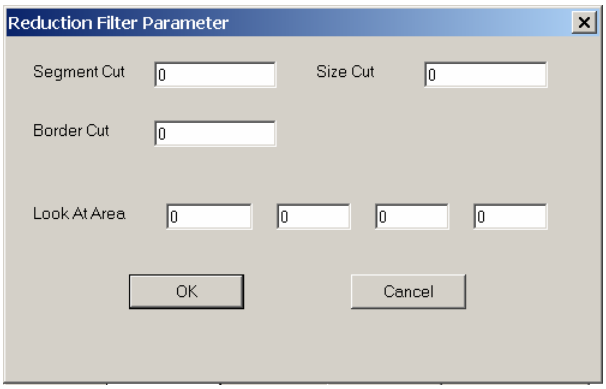

Figure 18.5 Selecting Reduction Filter.

There are several ways to reduce images:

- Integer,
- Real, or
- All images are reduced by the same amount.

Integer Reduction

Images are reduced by an integer factor to maximally fit 100x100 without distortion. For example, a 350x230 image will be reduced to 87x57.

Real Reduction

Images are reduced by a real number to maximally fit 100x100 without distortion. For example, a 350x230 image will be reduced to 100x65.

Within each type of reduction, there are 3 more settings. Assume a 3x3 pixel array is reduced to 1 pixel,

- Avg: Assign the average of the 3x3 pixel array to the new pixel;
- Max: Assign the maximum of the 3x3 pixel array to the new pixel; or
- Min: Assign the minimum of the 3x3 pixel array to the new pixel.

To select the Reduction Filter, use the fourth drop down list. The Reduction Filter has seven parameters.

Segment Cut

This parameter deals with the edges of the segments in the images. The Segment Cut parameter ranges from 0 to 12. The larger this parameter is, the smaller the segment the **ImageFinder** will use. The possible settings in the user interface are: 0, 1, 2, .., and 12.

Size Cut

In some applications, the users only want to search images of certain dimensions and ignore other images. An example is given below:

In this example, the two stamps belong to two different classes based on the image dimension alone.

The Size Cut parameter ranges from 0 to 9. If the setting is 0, this parameter will be ignored.

- If the parameter is 1, then the longest edge of the image to be considered must be at least 100, but less than 199.
- If the parameter is 2, then the longest edge of the image to be considered must be at least 200, but less than 299; …

Border Cut

The Border Cut parameter ranges from 0 (no cut) to 9 (18% border cut). For some images (see the picture below), you might want to get rid of the sections of images close to the borders. To get rid of the border section, use the Border Cut.

The possible settings in the user interface are: 0, 1, 2, .., and 9.
Assume an image is (0,0; 1,1),

- setting Border Cut to 1 means the **ImageFinder** will look at the section (0.02, 0.02; 0.98, 0.98);
- setting Border Cut to 2 means the **ImageFinder** will look at the section (0.04, 0.04; 0.96, 0.96); … .

Look-At Area

The Look-At Area is the area the **ImageFinder** will use in a matching operation. A 100 x 100 window specifies a whole image. If an integer Reduction Filter is used, the actual area can be less than 100x100.

Four numbers specify the Look-At Area:

(x, y, w, h)

(x, y) are the coordinates of the upper-left corner and (w, h) are the width and height of the Look-At window.

To use this Look-At window, enter (x, y, w, h) to the 4 text boxes.

18.5 Unsupervised Filter & BioFilter

The Unsupervised Filter and the BioFilter have similar parameters, so we have combined these two filters together.

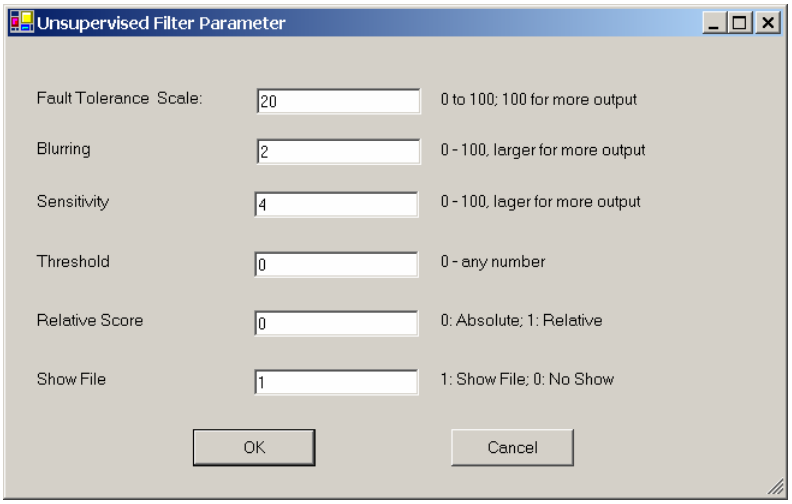

Figure 18.6 Unsupervised Filter or BioFilter Parameter.

Fault Tolerance Scale

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Blurring

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Sensitivity

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Threshold

The result of image comparison is a "score", indicating the degree to which a match exists. This score is then compared to a pre-set Threshold to determine whether or not to declare a match. This parameter sets the threshold. To decide what threshold to use, you should make a test run first and look at the scores. Matching images have higher scores; unmatched images have lower scores. Select a threshold to separate these two groups. There will be a few images in the middle, representing both groups. Under these circumstances, the threshold selection depends on your application.

Relative Score

Use the relative score to set the range of matching score between 0 and 100.

Show File

This parameter is set to 1 by default, which will show the output file. If this parameter is set to 0, then output file will not be shown.

18.6 Neural Filters

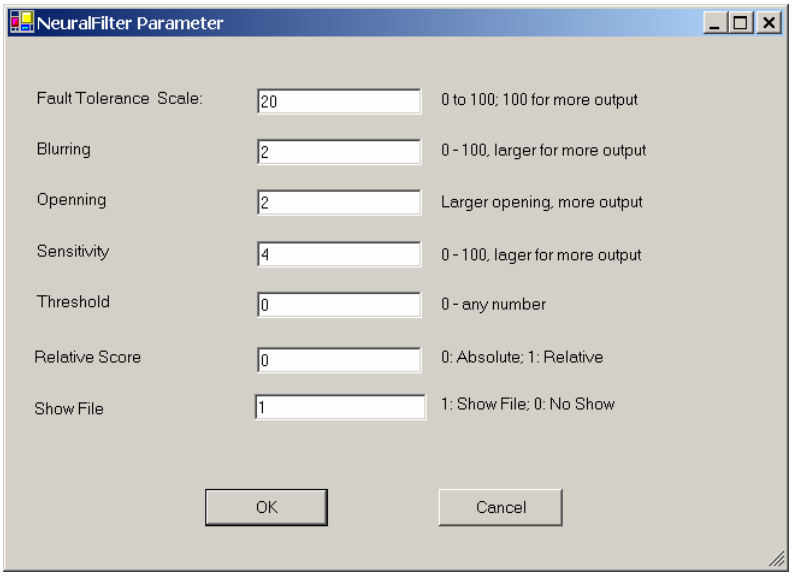

Figure 18.7 NeuralFilter Parameter.

Fault Tolerance Scale

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Blurring

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Sensitivity

Use this parameter to control the amount of output. This parameter ranges from 0 to 100. The larger this number is, the more matches you will get. To set this parameter, enter a number between 0 and 100 to the text box.

Threshold

The result of image comparison is a "score", indicating the degree to which a match exists. This score is then compared to a pre-set Threshold to determine whether or not to declare a match. This parameter sets the threshold. To decide what threshold to use, you should make a test run first and look at the scores. Matching images have higher scores; unmatched images have lower scores. Select a threshold to separate these two groups. There will be a few images in the middle, representing both groups. Under these circumstances, the threshold selection depends on your application.

Relative Score

Use the relative score to set the range of matching scores between 0 and 100.

Neural Filter Opening

This parameter controls the amount of output. This parameter has 5 settings:

Very Large
Large
Normal
Small
Very Small

Large openings will allow more output than small openings. To set the parameter, keep clicking the button; the setting will switch from one to the next each time you click the Blurring button.

Show File

This parameter is set to 1 by default, which will show the output file. If this parameter is set to 0, then output file will not be shown.

18.7 NeuralNet Filter

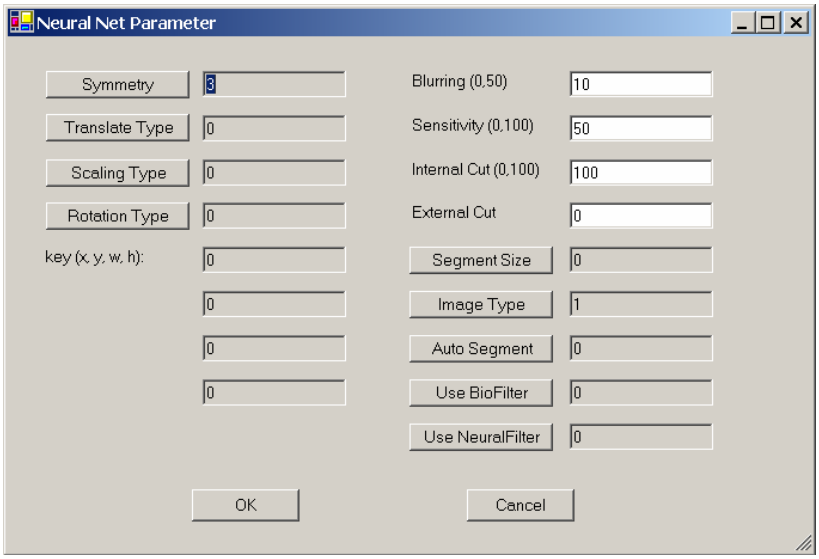

Figure 18.8 Neural Net Parameter.

The available NeuralNet filters are:

- 100x100 (Most Accurate)
- 90x90
- 80x80
- 70x70
- 60x60
- 50x50 (Least Accurate)

Let the speed of a 100x100 filter be a base, then the overall speed for:

- 90x90 filter is 1 times faster;
- 80x80 filter is 1.6 times faster;
- 70x70 filter is 2.7 times faster;
- 60x60 filter is 5 times faster; and
- 50x50 filter is 10 times faster.

The NeuralNet Filter has many parameters. The following sections will explain these parameters.

18.7.1 Symmetry

Symmetry or Invariance means similarity under certain types of changes. For example, considering two images, one with a face in the middle and the other with the face moved to the edge; we say these two images are similar because of the face.

The symmetry defines "similar images". The **Attrasoft ImageFinder** supports five symmetry settings:

- No symmetry (0);
- Translation symmetry (3);
- Scaling symmetry (4);
- Rotation symmetry (5); and
- Rotation & Scaling symmetries (6).

The numbers are the codes in the batch file. Currently, Scaling symmetry and Oblique symmetry are the same.

Other symmetries, or combination of symmetries, can be built for Customized Orders.

A customized Attrasoft ImageFinder can implement any symmetry (or combination of symmetries), which can be described by mathematics.

However, symmetries are computationally expensive.

Every symmetry setting has the Translation symmetry, except "No Symmetry". In addition, each of the above settings support:

- Intensity symmetry.

Symmetries are computationally expensive, meaning it will take a longer time to do the job. You should use them only when they are required.

To set the Symmetry, keep clicking the “Symmetry” button; the setting will switch from one to the next each time you click the button. The default setting in this version is Translation Symmetry.

For example, it seems that **Stamp Recognition** requires Translation and Rotation symmetries. But because the edges of a stamp can be detected easily, the stamp can be rotated and shifted to a fixed position where the horizontal side is longer than the vertical side. All you need to do is recognize a stamp or an upside-down stamp. Therefore, Stamp Recognition does not really require Translation and Rotation symmetries.

18.7.2 Translation Type

The Translation Type defines the accuracy of the Translation symmetry.

The Translation Type settings (and their codes) are:

- Most Accurate (0);
- Accurate (1); and
- Least Accurate (2).

To set the Translation Type, keep clicking the “T Type” button; the setting will switch from one to the next each time you click the button. The default setting is 0, the most accurate setting.

18.7.3 Scaling Type

The Scaling Type defines the accuracy of the Scaling symmetry.

The Scaling Type settings (and their codes) are:

- Least Accurate (0);
- Accurate (1);
- Accurate (2); and
- Most Accurate (3).

To set the Scaling Type, keep clicking the "T Type" button; the setting will switch from one to the next each time you click the button. The default setting is 0, the least accurate setting.

18.7.4 Rotation Type

The Rotation Type defines the accuracy of the Rotation symmetry.

The Rotation Type settings (and their codes) are:

- 360° rotation, least accurate (0);
- -5° to 5° rotation (1);
- -10° to 10° rotation (2);
- 360° rotation, accurate (3);
- 360° rotation, more accurate (4);
- 360° rotation, most accurate (5).

To set the Rotation Type, keep clicking the "Rotation Type" button; the setting will switch from one to the next each time you click the button. The default setting is 360° rotation, the least accurate setting (0).

18.7.5 Area of Interest (AOI)

Selecting an image segment is very important for training.

- Use image segments for searching **similar images**.
- Only use the whole image for **exact matches**.

Training requires an "Area of Interest" (AOI) or "Set Focus", which selects a key-segment. If an AOI is not chosen, the whole image is the AOI. Four numbers specify AOI: the upper-left corner coordinates, and the length & width. Once the segment specification is successful, a box will cover the selected area. When you look at the training image, if the selected area is not what you want, just re-select the area again and click the "Segment" button.

The default setting is the whole image; the code is (x y w h) = (0 0 0 0). (0000) means ignore the segment. The units are pixels.

There are two situations where you should create a new sample image out of a sample segment:

- You repeatedly use an image segment;
- The image segment is not a rectangle; say a polygon.

The Windows Paint program will help you to create an image from a segment. When you create an image segment, please do not change the original image size. For example, if your image is 512x512 and you want create a segment of 400x200, please paste the 400x200 segment into a 512x512 empty image.

18.7.6 Blurring

This is one of the most important search parameters and the first parameter you should adjust.

Blurring compensates for minor image changes, which are not visible to human eyes. For example, if you use software to compress an image, to change the intensity of an image, or to translate, scale, or rotate an image, the image will be distorted a bit at the pixel level. You have to set "Blurring" to compensate for this.

The Blurring setting ranges from 0 to 50. The default setting is 10. You should set the parameters in the following order:

Blurring, Internal Weight Cut, Sensitivity, External Weight Cut.

To Summarize:

- **When a search yields no results, increase Blurring;**
- **When a search yields too many results, decrease Blurring.**

18.7.7 Sensitivity

The Sensitivity parameter ranges from 0 (least sensitive) to 100 (most sensitive).

- To search small segment(s), use high sensitivity search.
- To search large segment(s), use low sensitivity search.
- **The higher this parameter is, the more results you will get.**

The Sensitivity parameter ranges from 0 to 100. The default is 50.

To Summarize:

- **When a search yields no results, increase sensitivity;**
- **When a search yields too much result, decrease sensitivity.**

18.7.8 Internal/External Weight Cut

You can set the "**Internal Weight Cut**" (Internal Cut) or "**External Weight Cut**" (External Cut) to list only those retrieved images with scores or weights greater than a certain value (called Threshold).

It is better to give no answer than a wrong answer.

Assume you are searching images and all similar images have weights ranging from 1,000 to 10,000. It is possible that some other images pop up with weights ranging from 10 to 100. To eliminate these images, you can set the “External Weight Cut” to 1,000.

The Internal Cut plays a similar role as the External Cut. There are two differences between these two cuts:

- The Internal Cut ranges from 0 to 99; the External Cut can be any number;
- The Internal Cut stops the images from coming out, whereas the External Cut can bring the eliminated images back if you set the External Cut to 0. You might need to see the eliminated images sometimes for the purpose of adjusting the parameters.

To Summarize:
- **Set the “Internal Cut” or “External Cut” to eliminate errors.**

18.7.9 Segment Size

The **ImageFinder** is currently tuned to search for large image segments (size of the whole image). It can look for small segments via the "Small Segment" setting; however, only Translation symmetry is supported for small segments.

A Customized Version can be ordered for other symmetries.

To search large segments, use setting 0.
To search small segments, use setting 1.

For example:
- If a sample segment is one quarter of the sample image, it is a large segment.
- If the segment is 1/20 of the sample image, it is a small segment.

Currently, "S Segment" only supports Translation symmetry. If you need Rotation or/and Scaling symmetry, please use "L Segment".

Other symmetries can be added in a Customized Version.

18.7.10 Image Type

There are BW and Color images. For each of them, there are “sum-search”, “maximum-search”, and “average-search”. This generates 6 image types:

- BW Sum
- BW Max
- BW Avg
- Color Sum
- Color Max
- Color Avg

"BW Sum" is like an integration of function f (x).
"BW Max" is like a maximum value of f (x); and
"BW Avg" is the average of the above two.

"Color Sum" is like an integration of function f (x).
"Color Max" is like a maximum value of f (x); and
"Color Avg" is the average of the above two.

To set the image type, keep clicking the Image Type button; the setting will switch from one to the next each time you click the Image Type button.

18.7.11 Use BioFilter & Use Neural Filter

These two parameters, "Use BioFilter" and "Use NeuralFilter" will decide whether the BioFilter and NeuralFilter will be used before the NeuralNet Filter is used to eliminate some images.

18.7.12 Auto Segment

The training segment can be specified in two ways:

Manual Specification
Automatic Specification

The default is Manual Specification. In this setting the segment will be specified by the four text boxes (x, y, w, h), as we discussed earlier.

If you do not want to pick up a training segment, then let the **ImageFinder** pick up the segment for you by using the Automatic Specification. This parameter has several settings:

NO Auto Segment
Very Large Segment
Very Large Segment
Large Segment
Large Segment
Medium Segment
Medium Segment

18.7.13 Summary

The NeuralNet Filter is hard to use because it has so many parameters. Not all parameters are equal. We divide the parameters into two groups. The beginners should use only parameters in the first group. Note that:

- **The most important parameters for Training are Image Processing, AOI, Symmetry, and Segment Cut (in the Reduction Filter).**
- **The most important parameters for Matching are Blurring, and Sensitivity.**

In a typical search, you will set these parameters and leave other parameters with default values. These are the 7 parameters you should focus on first**:**

Training (3 parameters):

- Segment: selecting "Large AutoSeg 3" so the **ImageFinder** will select a training segment for you at the beginning.
- Symmetries
- Segment Cut (in the Reduction Filter)

Matching (4 parameters):

- Sensitivity
- Blurring
- External Weight Cut
- Internal Weight Cut

Ignore the rest of the parameters at the beginning.

19. Input Options

The chapter project is located at:

c:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

We also call this folder ".\". That is, ".\" is "c:\transapplet70\imagefinder\bin\Release\". When you start the software, the first thing you will see is Figure 19.1. **ImageFinder** requires a key image and a search source. Now, we will introduce a few input options, including:

- Microsoft Access,
- Avi video file, and
- Live video.

This chapter will introduce the input design for Microsoft Access, avi video file, and live video; and the next three chapters will introduce the implementation.

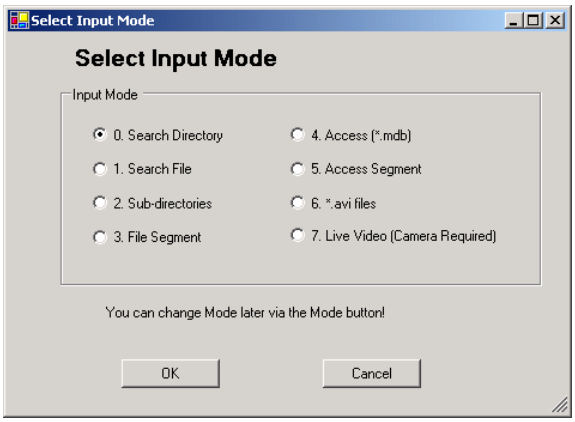

Figure 19.1 Input options.

The default search source is directory input, i.e. the **ImageFinder** will search through the images in a directory. The **ImageFinder** also will support a number of other options. See Figure 19.1.

19.1 File Input

You can specify the search source with a file. The Input Files must list one image per line. Each line specifies an absolute path. For example,

C:\xyz1\0001.jpg

C:\xyz1\0002.jpg
C:\xyz2\0003.jpg
C:\xyz2\0004.jpg
…

The only difference between the Directory Input and the File Input is how images are entered into the **ImageFinder**; after that, all other steps are the same.

You can specify File Input either in Figure 19.1, or in Figure 19.2.

- In Figure 19.1, select the second option for File Input.
- In Figure 19.2, keep clicking the "Mode" button and the setting will switch from one to the next.

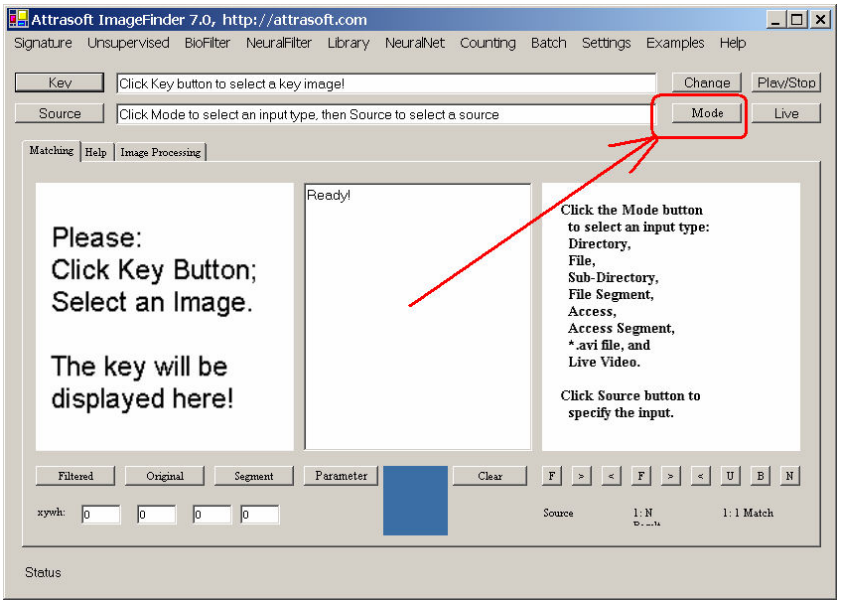

Figure 19.2 Input Mode Button.

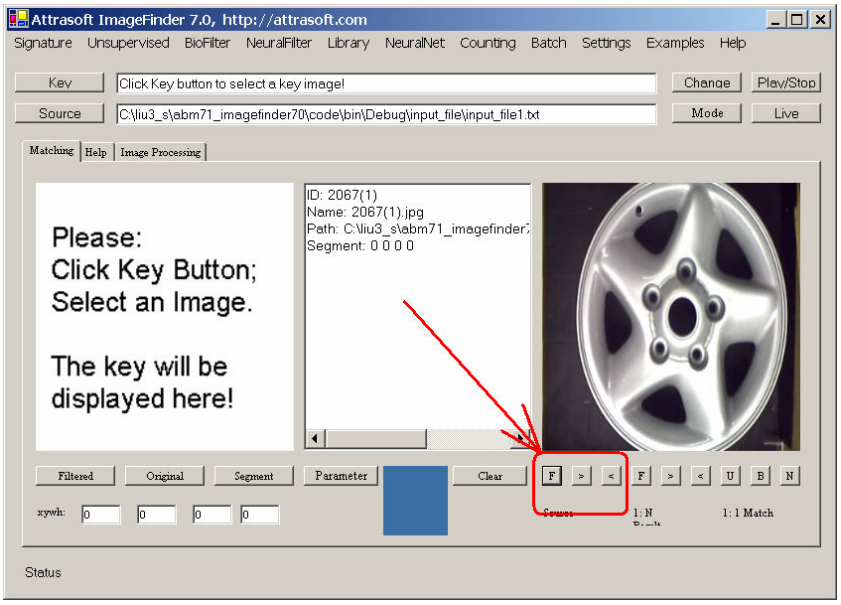

Figure 19.3 File Input example.

Example. File Input

- Start the software;
- Select option, "Search File";
- Click the "Source" button, and select file, ".\input_file\input_file1.txt";
- Click the "F" button in Figure 19.3 to see the first image, and click the ">" button to see the next image.

19.2 Sub-Directory Input

The default search source is Directory Input only. If the directory has sub-directories, the **ImageFinder** will not search through the sub-directories.

To search through the sub-directories, use the Sub-directory Input option.

You can specify sub-directory source either in Figure 19.1, or in Figure 19.2.

- In Figure 19.1, select the third option for File Input.
- In Figure 19.2, keep clicking the Mode button and the setting will switch from one to the next.

Example. Sub-Directory Input:

- Start the software;
- Select option, "Sub-directory";
- Click the Source Button, and select file, ".\input_subdir\1001A.jpg";
- Click the "F" button in Figure 19.3 to see the first image, and click the ">" button to see the next image.

19.3 Segment File Input

The first three options are for whole images.

If you want compute the signature for a segment of an image, you have to select File Segment Input.

The Input Files must list one image per line. Each line specifies an absolute path, Image_ID, x, y, w, and h. A sample segment file looks like this:

```
.\2067(1).jpg   1       20      20      280     200
.\2067(2).jpg   2       20      20      280     200
.\2067(3).jpg   3       20      20      280     200
.\2067(4).jpg   4       20      20      280     200
.\2071(1).jpg   5       20      20      280     200
.\2071(2).jpg   6       20      20      280     200
.\2071(3).jpg   7       20      20      280     200
```

Here (x, y, w, h) are in the unit of pixels. You can specify File Segment source either in Figure 19.1, or in Figure 19.2:

- In Figure 19.1, select the "File Segment".
- In Figure 19.2, keep clicking the "Mode" button and the setting will switch from one to the next.

Example. File Segment Input:

- Start the software;
- Select Option, "File Segment";
- Click the "Source" button, and select file, ".\input_file\input_filesegment1.txt";
- Click the "F" button in Figure 19.3 to see the first image, and click the ">" button to see the next image.
- The segment is marked by a box in Figure 19.4.

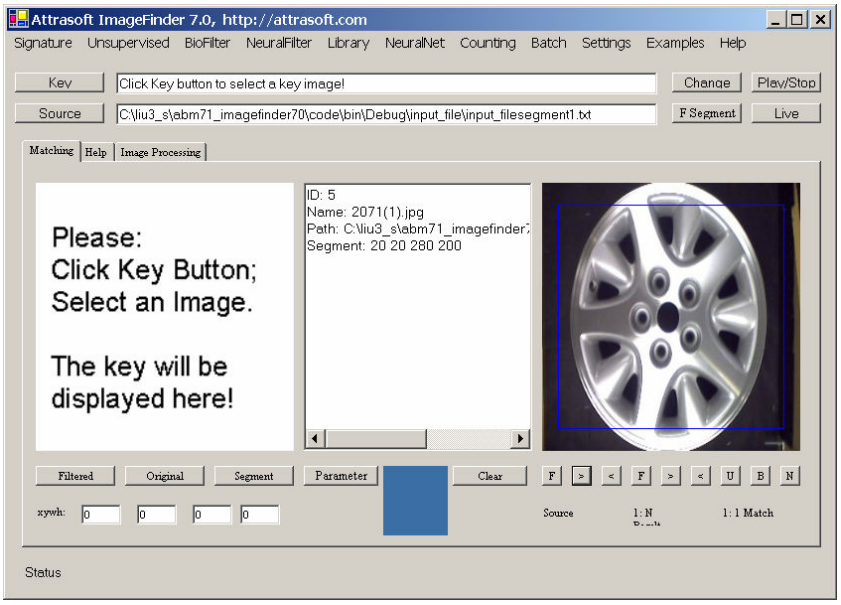

Figure 19.4 File Segment.

19.4 Database Input, Whole Image

The only database supported in this version is Microsoft Access. If you need a different Database supported, customization will solve this problem.

The Database consists of a set of tables. The table contains the locations of the images.

The data in a table is obtained by a SQL statement; therefore, to use this option, you must be able to write a SQL statement. The SQL output must list one image per row. When all fields in each row are combined, it must specify an absolute path.

A sample access table is:

ID	Name
1	.//2067(1).jpg
2	.//2067(2).jpg
3	.//2067(3).jpg
4	.//2067(4).jpg
5	.//2071(1).jpg
6	.//2071(2).jpg
7	.//2071(3).jpg
8	.//2071(4).jpg
9	.//2082(1).jpg
10	.//2082(2).jpg

Some sample SQL statements are:

select Path, Name from List1
select Name from List2

Do not add the semicolon, ";" at the end of the SQL statement!

The result of a query must produce a list of paths for images. The result is either a single column or two columns, like path and name, which forms an absolute path when combined together.

Example. Database Input:

- Start the software;
- Select the option, "Access";
- Click the "Source" button, and select file, ".\input_access\db1.mdb";
- Enter the SQL statement, "Select Name from List2";
- Click the "F" button in Figure 19.3 to see the first image, and click the ">" button to see the next image.

19.5 Database Input, Image Segment

The only database supported in this version is Microsoft Access. If you need a different Database supported, customization will solve this problem.

The Database consists of a set of tables. The table contains the locations of the images segments. Six fields specify the image segments: path, ID, x, y, w, and h. Database retrieval is specified by a query. For example, "Select Path, ID, x, y, w, h from List3". **Do not add ";" at the end of the SQL statement!**

The result of a query must produce a list of (Path, ID, x, y, w, h). No other formats are accepted!

Example. Database Segment Input:

- Start the software;
- Select the Option, "Access Segment";
- Click the "Source" button, and select file, ".\input_access\db1.mdb";
- Enter the SQL statement, "Select Path, ID, x, y, w, h from List3";
- Click the "F" button in Figure 19.3 to see the first image, and click the ">" button to see the next image.

19.6 Converting AVI Video to Images

The only video format supported in this version is *.avi. If you need a different video format supported, customization will solve this problem.

After selecting an *.avi file, the **ImageFinder** will convert the avi video file to a set of images. After that, they are the same as the rest of the images.

Example. Converting AVI Video:

- Start the software;
- Select the option, "*.avi File";
- Click the "Source" button, and select file, ".\input_avi\clock.avi";
- Click the "Play/Stop" button in Figure 19.5 to convert the button to a "Play" button;
- Click the "Play" button to Play, which also converts the button to a "Stop" button;
- Click the "Stop" button to stop the video;
- Click the "Change" button in Figure 19.5 to get Figure 19.6;
- Click the "*.avi to Images" button to convert the avi file to images.

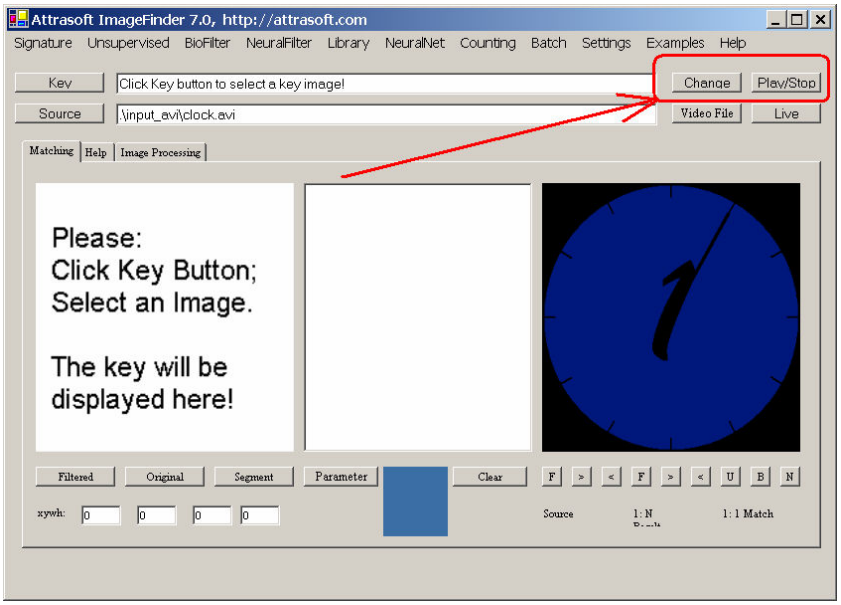

Figure 19.5 Converting AVI Video File to Images, Step 1.

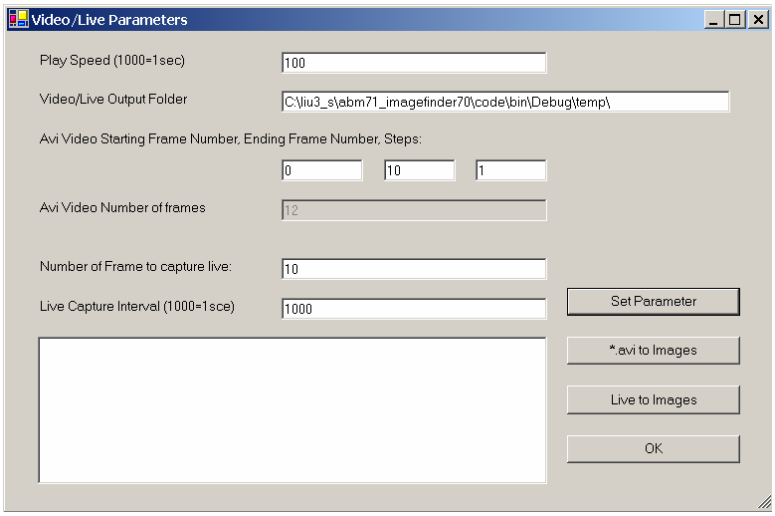

Figure 19.6 Converting AVI Video File to Images, Step 2.

19.7 Converting Live Video to Images

You will need a Logitech camera for this section.

After starting the camera live, the **ImageFinder** will convert the live video file to a set of images. After that, they are the same as the rest of the images.

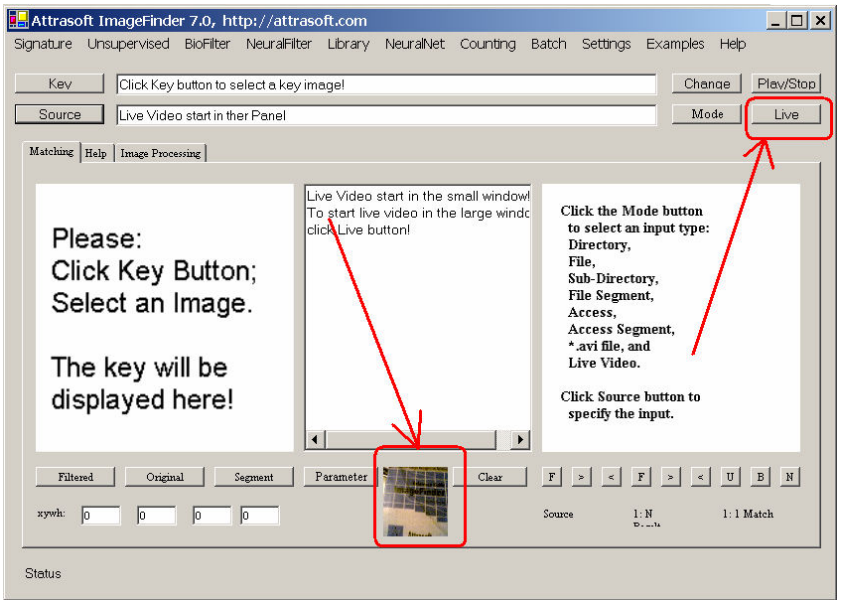

Figure 19.7 Converting Live Video File to Images, before clicking the "Live" button.

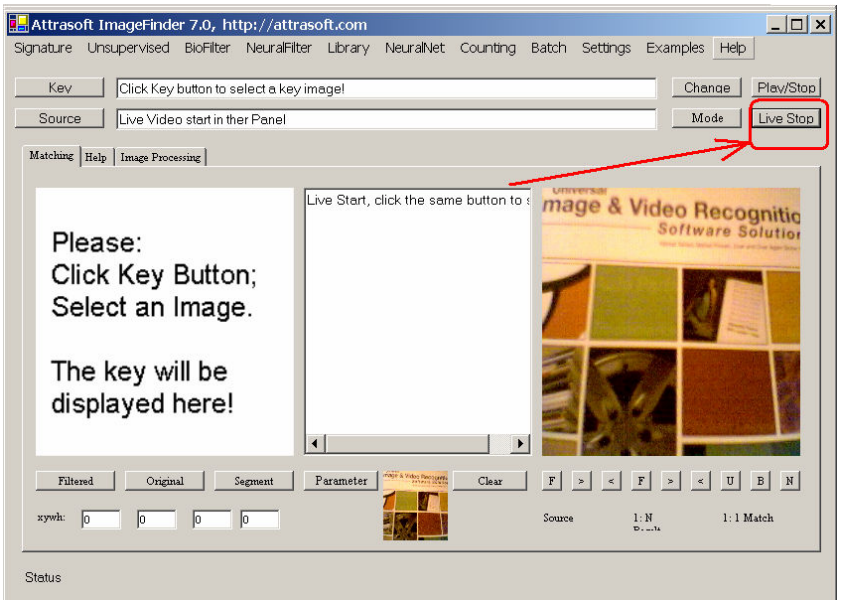

Figure 19.8 Converting Live Video File to Images, after clicking the "Live" button.

Example. Converting Live Video:

- Start the software;

- Select the option, “Live Video”;
- Click the “Source” button to get Figure 19.7, where live video is shown on a small window on Figure 19.7;
- Click the “Live” button on 19.7 to get Figure 19.8.
- Click the “Change” button in Figure 19.7 to get Figure 19.8;
- Click the “Live to Images” button in get Figure 19.6 to convert the Live Video to images;
- Click the “Live Stop” button in Figure 19.8 to stop.

20. Database Input

In this chapter, we introduce Microsoft Access Input. Other DBMS (Database Management System) can be ordered via a customized **TransApplet.**

The database input requires:

Database File;
SQL statement, which specifies how the data is obtained from the database.

There are two options you can use to implement the Access Input:

Attrasoft.TransApplet70. InputDBMS70;
Attrasoft.TransApplet70. Input70.

The chapter project is located at:

c:\transapplet70\imagefinder\.

The executable file is located at:

c:\transapplet70\imagefinder\bin\Release\.

20.1 Basic Access Class

The Class Library is:

Attrasoft.TransApplet70.InputDBMS70,

The class in this library will be:

Attrasoft.TransApplet70. InputDBMS70. InputDBMS70.

The interface, which will be used by InputDBMS70, is:

```
public interface I_InputDBMS
{
        bool setDBMS (string dbms, string query);
        bool setDBMSSegment (string dbms, string query);

        string getDBMS ();
        string getQuery ();

        bool isDBMSFile ( string s);
```

```
        bool getStatus();

        int getN();
        string [] getID ( );
        string [] getName ( );
        string [] getPath ( );
        string [] getAbsolutePath ( );
        string [] getX ();
        string [] getY ();
        string [] getW ();
        string [] getH ();

        string getMessage();

}
```

This Database Input Class will have a constructor that will take a RichTextBox:

```
public InputDBMS70( RichTextBox r1)
        {
        }
```

To declare an object, write:

```
public Attrasoft.TransApplet70.InputDBMS70.InputDBMS70 indb70 ;
```

To create an object, write:

```
indb70 = new
Attrasoft.TransApplet70.InputDBMS70.InputDBMS70 (richTextBox1 );
```

The Input parameters to this class are:

- Database File;
- **SQL statement that specifies how the data is obtained from the database.**

The Output of this class is a string list of images. The first thing you will do is to enter information to this class:

```
indb70.setDBMS (sDBMS, sQuery);
```

At this point, you can proceed to get data:

```
        int getN();
        string [] getID ( );
        string [] getName ( );
        string [] getPath ( );
        string [] getAbsolutePath ( );
```

```
string [] getX ();
string [] getY ();
string [] getW ();
string [] getH ();
```

The status function will tell you whether the database computation is successful:

indb70.getInputDBMSStatus ()

If it is successful, the retrieved image list can be obtained:

```
if ( ! indb70.getInputDBMSStatus ()  )
        {
                appendText ( "Database input Fail!\n" );
                return;
        }
filelist = indb70. getAbsolutePath ( );
```

20.2 Input Class

An alternation is to use Input70 class introduced earlier.

The class library is:

Attrasoft.TransApplet70.Input70.

The main class in this library will be:

Attrasoft.TransApplet70.Input70.Input70.

The Input70 interface is:

```
public interface I_Input
{
        string [] getDirList ( string sInput);

        string [] getSubDirList ( string sInput);

        string [] getFileList ( string sInput);
        string []  getFileSegmentList ( string sInput);

        string [] getAccessList ( string sInput, string sSQL);
        string []  getAccessSegmentList ( string sInput, string sSQL);

        string [] getID ( );
        string [] getName ( );
        string [] getPath ( );
```

```
        string [] getAbsolutePath ( );
        string [] getX ();
        string [] getY ();
        string [] getW ();
        string [] getH ();
}
```

The functions in Input70 class are listed in the following table.

Function	Description
string [] getDirList (string sInput)	Gets a string list of the absolute paths of all images in directory, sInput.
string [] getSubDirList (string sInput)	Gets a string list of the absolute paths of all images in all sub-directories of sInput.
string [] getFileList (string sInput)	Gets a string list of the absolute paths of all images in file, sInput.
string [] getFileSegmentList (string sInput)	Gets a string lists of the absolute paths of all images in file, sInput. This command will also populate other arrays so they can be obtained through the following functions: string [] getID ( ); string [] getName ( ); string [] getPath ( ); string [] getAbsolutePath ( ); string [] getX (); string [] getY (); string [] getW (); string [] getH ().
string [] getAccessList (string sInput, string sSQL)	Gets a string list of the absolute paths of all images in access file, sInput, specified by a SQL statement, sSQL.
string [] getAccessSegmentList (string sInput, string sSQL)	Gets a string list of the absolute paths of all images in access file, sInput, specified by a SQL statement, sSQL. This command will also populate other arrays so they can be obtained through the following functions: string [] getID ( ); string [] getName ( ); string [] getPath ( ); string [] getAbsolutePath ( ); string [] getX (); string [] getY (); string [] getW (); string [] getH ().

string [] getID ( ) string [] getName ( ) string [] getPath ( ) string [] getAbsolutePath ( ) string [] getX () string [] getY () string [] getW () string [] getH ()	Gets a string list of the ID, Name, Path, Absolute Path, X, Y, W, and H of all images in a search source.

20.3 Input Selection

The input options are:

0. Directory,
1. File,
2. Sub-Directory,
3. File Segment,
4. Access,
5. Access Segment,
6. *.avi file, and
7. Live Video.

The selection is implemented by the "Mode" button in Figure 19.2:

- In Figure 19.1, select an input option.
- In Figure 19.2, keep clicking the Mode button and the setting will switch from one to the next.

Double click the "Mode" button and enter:

```
private void button3_Click(object sender, System.EventArgs e)
        {
        gui.Button_Mode ();
        }
```

Here, mainMenuToAPI is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the mainMenuToAPI object. The implementation is:

```
public void Button_Mode ()
        {
        iMode = (iMode + 1 )% strMode.Length ;
        Button_Mode_consistent ();
        }
```

where the third line is:

```
public void Button_Mode_consistent ()
	{

		if (iMode == 0 )
			f.button3.Text = strMode[0];
		else if (iMode == 1 )
			f.button3.Text = strMode[1];
		else if (iMode == 2 )
			f.button3.Text = strMode[2];
		else if (iMode == 3 )
			f.button3.Text = strMode[3];
		else if (iMode == 4 )
			f.button3.Text = strMode[4];
		else if (iMode == 5 )
			f.button3.Text = strMode[5];
		else if (iMode == 6 )
			f.button3.Text = strMode[6];
		else if (iMode == 7 )
			f.button3.Text = strMode[7];
	else
	{
		iMode = 0;
		f.button3.Text = strMode[0];
	}

	f.richTextBox1.AppendText ( "Input source: " + f.button3.Text + "\n");
}
```

The access options are:

```
		iMode = 4 (Access);
		iMode = 5 (Access Segment).
```

The “Source” button specifies the input source based on the variable, iMode. Double click the “Source” button and enter:

```
public void button2_Click(object sender, System.EventArgs e)
	{
		gui.Button_SearachSource ( gui.iMode );
	}
```

The implementation is:

```
public void Button_SearachSource (int iMode)
	{
	canLiveStart = false;

	if ( iMode == 0 )
		searachSource0 ();
```

```
else if ( iMode == 1 )
        searachSource1 ();
else if ( iMode == 2 )
        searachSource2 ();
else if ( iMode == 3 )
        searachSource3 ();
else if ( iMode == 4 )
        searachSource4 ();
else if ( iMode == 5 )
        searachSource5 ();
else if ( iMode == 6 )
        searachSource6 ();
else if ( iMode == 7 )
        searachSource7 ();
else
        searachSource0 ();
}
```

Therefore, the database implementation is given in the following functions:

```
searachSource4 ();
searachSource5 ().
```

20.4 Database Parameter Input

The database input requires:

- Database File;
- SQL statement, which specifies how the data is obtained from the database.

The first variable is used to build the connection string for a database:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + database
```

The second SQL statement will get data. Examples are:

```
Select image from imagetable1.
Select path, name from imagetable2.
```

If the query selects multiple fields, this library will build the image path by path+name.

The chapter project will obtain both input parameters in a single click of the “Source” button. Now the project will accommodate 8 different types of input; you must select an input type when the program runs:

- When the program starts, select Input source;
- After the program has started, click the “Mode” button to select the Input source.

When the program starts, Figure 19.1 will be displayed, which allows the user to make a selection. After the project starts, the Input source can be changed via the "Mode" button in the Figure 19.2. Click the "Source" button to select Microsoft Access file; the program will also prompt you to enter the SQL statement with the second Dialog box:

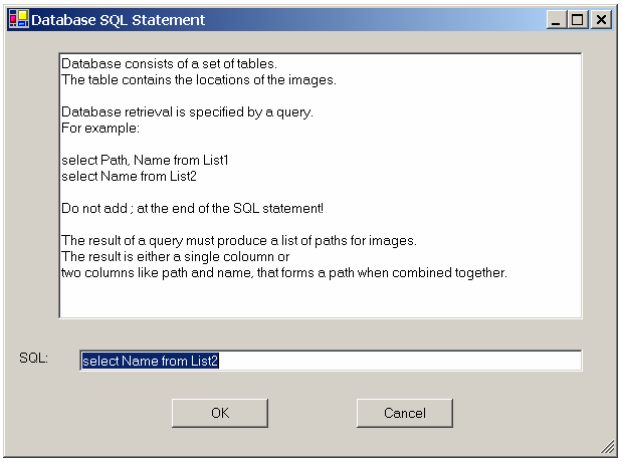

Figure 20.1 SQL Statement.

Users will enter the SQL statement in Figure 20.1.

20.5 Database Input Implementation

The database input is implemented by the following codes:

```
internal string dbmsSQL ="NA";
internal void setSQL (string s)
        {
        dbmsSQL = s;
        }
internal string getSQL ()
        {
        return dbmsSQL ;
        }

private bool searachSource4 ()
        {
        try{

        //Step 1. Database file
        if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                return false;
        string sDBMS =  f.openFileDialog1.FileName  ;
        f.textBox2.Text = sDBMS ;
```

```
// Step 2. SQL Statement
string sQuery ="";
Parameter_DbmsInput pd = new Parameter_DbmsInput (this, 0);
pd.ShowDialog ();
sQuery = getSQL ();
f.richTextBox1.AppendText ( "Database: " +sDBMS +"\n" );
f.richTextBox1.AppendText ( "Query: " + sQuery +"\n" );

//Step 3. Database Input
this.imageAbsoultePath = script.input.getAccessList  (sDBMS, sQuery);
if ( imageAbsoultePath  == null )
{

        f.richTextBox1.AppendText ( "Data Source fails!\n");
        return false;
}
}
catch (Exception e )
{
        f.richTextBox1.AppendText ( e.ToString () +"\n" );
        return false;

}
…
return true;
}

private bool searachSource5 ()
        {
        …
        }
```

Now we briefly explain the code:

- The code in section “Step 1. Database file” will get a Microsoft Access file.
- The code in section “Step 2. SQL Statement” will get the SQL statement. Here class, “Parameter_DbmsInput”, will produce Figure 20.1, which will call function, “setSQL (string s)”, to set the SQL statement.
- The code in section “Step 3. Database Input” will enter the two parameters (Access file and SQL statement) into the Input class; and retrieve the image list from the database. This is similar to other input options. The result is a string list and each string is a path of an image.

The implementation, searchSource5 (), for the Access Segment is similar.

20.6 Testing

To test, please refer to the last chapter.

21. Video Input

The **ImageFinder** breaks a video file into frames and matches the frames one by one. The **Attrasoft ImageFinder** provides the users with a tool for Video Matching, including:

- Recognizing an object in a video;
- Reporting the time an object starts to appear;
- Tracking an object in a video and reporting its coordinates;
- Counting the number of frames in which the object appears; and
- Selecting a video from a video database.

This library supports only *.avi files. This chapter simply shows how to break an *.avi video file into images. From that point on, it will be an image recognition application.

21.1 Class Library Name

The class library is:

Attrasoft.TransApplet70.VideoInput70,

The class in this library will be:

Attrasoft.TransApplet70.VideoInput70.VideoInput70.

The interface, which will be used by VideoInput70, is:

```
public  interface I_VideoInput70
{
        string getMessage();

        void setVideoFile (string s);
        string getVideoFile ();
        bool getVideoFileStatus();

        void setVideoToImagesDir (string s);
        string getVideoToImagesDir ();
        bool getVideoToImagesDirStatus();

        int getFrameNumberStart ();
        void setFrameNumberStart (int i );
        int getFrameNumberEnd ();
        void setFrameNumberEnd (int i );
        int getFrameNumberStep ();
        void setFrameNumberStep (int i );

        int getNumberOfFrames();
```

```
        bool videoToImages ();
        Bitmap getBitmap (int i);

        string getInfomation();
        string toString();
}
```

21.2 Class Library Overview

This Class Library will break a *.avi video into images by an algorithm like this:

```
For ( I = start; I <= end; I += step )
{
        get Frame I from the video;
        save the image to image_I.jpg;
}
```

To enter a video *.avi file, use these functions:

```
void setVideoFile (String s);
string getVideoFile ();
bool getVideoFileStatus();
```

The converted images are saved to a folder, called “Video To Image” directory. To set the “Video To Image” directory, use these functions:

```
void setVideoToImagesDir (String s);
string getVideoToImagesDir ();
bool getVideoToImagesDirStatus();
```

The breaking algorithm requires the starting frame number, the ending frame number, and the skip steps. The following functions are for this purpose:

```
int getFrameNumberStart ();
void setFrameNumberStart (int i );
int getFrameNumberEnd ();
void setFrameNumberEnd (int i );
int getFrameNumberStep ();
void setFrameNumberStep (int i );
```

To get the number of frames from the video file, use:

```
int getNumberOfFrames();
```

To get a particular frame, use:

```
Bitmap getBitmap (int i);
```

To convert *.avi video to images, use:

```
bool videoToImages ();
```

Before calling videoToImages(), you should set the:

- "Video To Image" Directory;
- Starting Frame Number;
- Ending Frame Number; and the
- Skip Steps.

21.3 Link to Class Library

To include the Class Library in the project,

- Right click References and select Add Reference in the Solution Explorer;
- Browse to find "VideoInput70.dll" in "c:\transapplet70\";
- Highlight it and click the "OK" button.

To use the class library, add:

```
using Attrasoft.TransApplet70.VideoInput70;
```

To declare an object, please use the full path for class:

```
public Attrasoft.TransApplet70. VideoInput70. VideoInput70 vi70
= new Attrasoft.TransApplet70. VideoInput70. VideoInput70 ();
```

Now the VideoInput70 object, vi70, is ready to use. Once again, in the remaining part of this chapter, we will use script object, which has all of the objects required for the **ImageFinder** project. In particular, the object for *.avi video input is "script.aviVideo70".

21.4 AVI Video Selection

The basic Video Match operation is:

(1) Specify a Video File;
(2) Convert Video to Images;
(3) Treat the Converted Images as Directory Input covered earlier.

The only video format supported in this version is *.avi. If you need a different video format supported, customization will solve this problem.

After starting the chapter projects software, selecting an *.avi file in Figure 19.1, the **ImageFinder** will convert the avi video file to a set of images. After that, video recognition is converted into image recognition.

Example. Converting AVI Video:

- Start the software;
- Select the option, "*.avi File";
- Click the "Source" button, and select file, ".\input_avi\clock.avi";
- Click the "Play/Stop" button in Figure 21.1 to convert the button to a "Play" button;
- Click the "Play" button to play, which also converts the button to a "Stop" button;
- Click the "Stop" to stop the video;
- Click the "Change" button in Figure 21.1 to get Figure 21.2;
- Click the "*.avi to Images" button to convert the avi file to images.

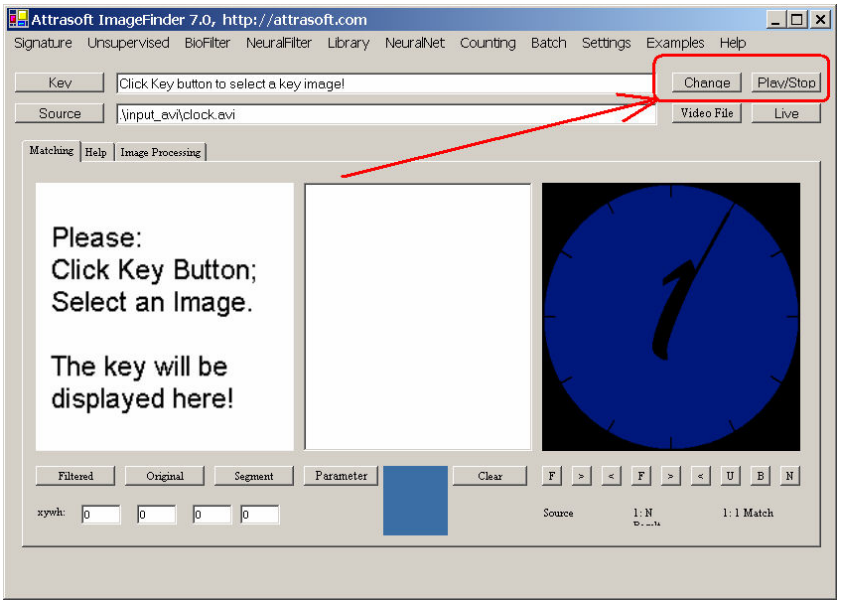

Figure 21.1 Converting AVI Video File to Images, Step 1.

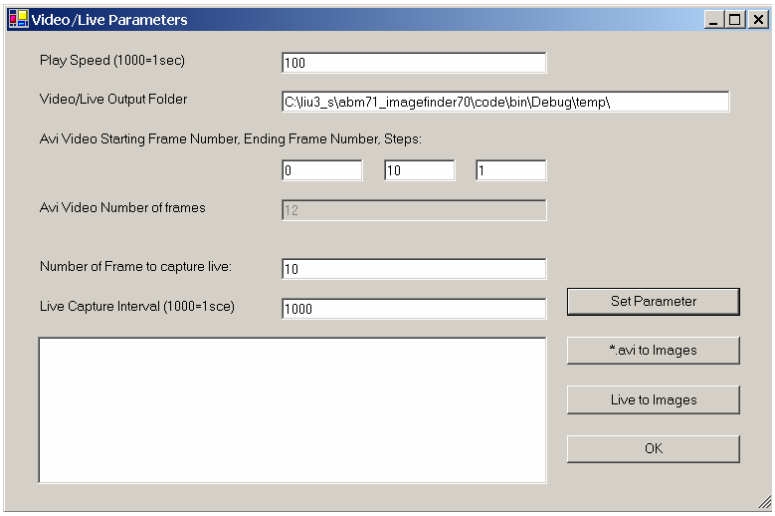

Figure 21.2 Converting AVI Video File to Images, Step 2.

The "Source" button selects an *.avi file, which in turn, calls the following functions:

```
int videoFilePointer = 0;
private bool searachSource6 ()
{
        try
        {
                if ( f.openFileDialog1.ShowDialog () != DialogResult.OK )
                        return true;

                bool b = f.openFileDialog1.FileName.EndsWith (".avi")
                        || f.openFileDialog1.FileName.EndsWith (".AVI")
                        || f.openFileDialog1.FileName.EndsWith (".Avi");
                if (!b)
                {
                        f.richTextBox1.Text = "Not *.avi file\n";
                        return false;
                }
                script.aviVideo70.setVideoFile  ( f.openFileDialog1.FileName   );
                if ( ! script.aviVideo70.getVideoFileStatus() )
                {
                        f.richTextBox1.Text = "Video Selection Fail!\n"
                        + script.aviVideo70.getMessage ();
                        return false;
                }
                f.textBox2.Text = script.aviVideo70.getVideoFile () ;

                script.aviVideo70.setVideoToImagesDir
                         (this.videoLiveParameters .imageOutputFolder);
                f.richTextBox1.AppendText ( "Video To Image Directory: \n"
                        + script.aviVideo70.getVideoToImagesDir () +"\n");
                videoFilePointer = 0;
```

```
                f.pictureBox2.Image =
                        script.aviVideo70.getBitmap (videoFilePointer);
        }
        catch (Exception ee )
        {
                f.richTextBox1.AppendText ( ee.ToString () +"\n");
                return false;
        }
        return true;
}
```

This code will:

- Select an *.avi video file;
- Set the selected file to object, “script.aviVideo70”; and
- Set the Video-To-Image Directory.

We will now explain this code. The first statement,

```
if ( openFileDialog1.ShowDialog () != DialogResult.OK )
        return;
```

opens a file dialog box and selects an *.avi file. The next statement,

```
script.aviVideo70.setVideoFile  ( f.openFileDialog1.FileName   );
```

assigns the selected *.avi file to the script.aviVideo70 object. The next two statements,

```
script.aviVideo70.setVideoToImagesDir
        (this.videoLiveParameters .imageOutputFolder);
f.richTextBox1.AppendText ( "Video To Image Directory: \n"
+ script.aviVideo70.getVideoToImagesD
```

creates the “Video To Image” directory, and prints a message to the text Box. The last statement,

```
f.pictureBox2.Image = script.aviVideo70.getBitmap (videoFilePointer);
```

prints the first image in the video.

21.5 Converting Video to Images

To convert the video to images, a new button is added, “Change” (Figure 21.1). Clicking the button will bring up the form on Figure 21.2, which has a button, “*.avi To Image”. The roles of the three buttons are:

- The “Set Parameter” button will specify the:

- Video-To-Image Directory
- Starting Frame Number
- Ending Frame Number
- Skip Step

- The “*.Avi To Image” button will do the image conversion.

- The “OK” button will end the form.

Double clicking the “*.Avi To Image” button will convert the AVI video to images, which is implemented as follows:

```
private void button2_Click_1(object sender, System.EventArgs e)
{
        if ( ! f.gui.script.aviVideo70.getVideoFileStatus() )
        {
        richTextBox1.Text =
                "Please select a video source first.\n"
                + "Please click the Mode button, then click the source button.\n"
                + "The Mode button can looks like Dir, File, Sub Dir, ...\n";
        return;
        }

        try{
        f.mainMenuToAPI.script.aviVideo70.
        setVideoToImagesDir (f.gui.videoLiveParameters .imageOutputFolder  );
        f.mainMenuToAPI.script.aviVideo70.setFrameNumberStart
                (f.gui.videoLiveParameters .videoStart ) ;
        f.mainMenuToAPI.script.aviVideo70.setFrameNumberEnd
         ( f.gui.videoLiveParameters .videoEnd )  ;
        f.mainMenuToAPI.script.aviVideo70.setFrameNumberStep
                (f.gui.videoLiveParameters .videoStep  );
        }
        catch (Exception ee)
        {
                richTextBox1.Text = ee.ToString () + "\n";
                return;
        }

        if ( ! f.mainMenuToAPI.script.aviVideo70.videoToImages () )
                richTextBox1.Text =  "Conversion fails!";
        else
                richTextBox1.AppendText ( "Conversion completed!\n" );
}
```

The last statement,

```
f.mainMenuToAPI.script.aviVideo70.videoToImages ()
```

will convert the *.avi file into images.

21.6 Testing

To test, please refer to chapter 19.

22. Live Video Input

In this chapter, we extend the **ImageFinder** to live video. In particular, the **ImageFinder** breaks a live video stream into images and matches the images one by one. The Attrasoft **ImageFinder** provides the users with a tool for Live Video Matching, including:

- Recognizing an object in a live video;
- Reporting the time an object starts to appear;
- Tracking an object in a video and reporting its coordinates;
- Counting the number of frames in which the object appears; and
- Selecting a video from a video database.

The library in this chapter supports converting live video to images. The project for this chapter is to build an Attrasoft **ImageFinder** that provides the users with a tool for live video matching. This chapter will only focus on how to convert live video into images. From that point on, live video recognition becomes image recognition. This chapter simply shows how to break an *.avi video file into images. From that point on, it will be an image recognition application.

22.1 Class Library Name

The class library is:

Attrasoft.TransApplet70.LiveVideoInput70,

The class in this library will be:

Attrasoft.TransApplet70.LiveVideoInput70.LiveVideoInput70.

The interface, which will be used by VideoInput70, is:

```
public  interface I_LiveVideoInput70
{
        //1. parameters
        bool getCaptureStatus();

        bool getVideoToImagesDirStatus();
        void setVideoToImagesDirStatus( bool b );

        string getVideoToImagesDir();
        void setVideoToImagesDir( string s);

        // 2. action
        bool initialization ();

        bool attach ();
        bool detach ();
```

```
        string getMessage();

        string getInfomation();
        string toString();
}
```

The class, “LiveVideoInput70”, will have a constructor that will take a Picture Box and Panel:

```
public LiveVideoInput70( PictureBox pictureBox1a, Panel panel1a)
        {
                pictureBox1 = pictureBox1a;
                panel1 = panel1a;
        }
```

The Live Video will be displayed in the Panel and the Picture Box. **You can obtain a live image at anytime via the Picture Box.**

To declare an object, write:

```
internal Attrasoft.TransApplet70.LiveVideoInput70.LiveVideoInput70 lv70;
```

To create an object, write:

```
lv70 = new
Attrasoft.TransApplet70.LiveVideoInput70.LiveVideoInput70 (pictureBox2, panel1) ;
```

Make sure the Picture Box and Panel objects are created first. The first thing you will do is to initialize the LiveVideoInput70 object:

```
lv70.initialization ();
```

This will bring the live video to the Panel. To get images, we will need to bring the live image to the Picture Box. The following two commands will display and stop displaying live images to the Picture Box:

```
bool attach ();
bool detach ();
```

To make the Picture Box show live images, use the attach function. To stop the Picture Box from showing live images, use the detach function.

22.2 Link to Class Library

To include the Class Library in the project:

- In the Solution Explorer, right click References and select Add Reference;
- Browse to find “LiveVideoInput70.dll” in c:\transapplet70\;
- Highlight it and click the “OK” button.

To use the class library, add:

```
using Attrasoft.TransApplet70.LiveVideoInput70;
```

To declare an object, please use the full path for class:

```
internal Attrasoft.TransApplet70.LiveVideoInput70.LiveVideoInput70 lv70;
```

To create an object:

```
lv70 = new
Attrasoft.TransApplet70.LiveVideoInput70.LiveVideoInput70
 (pictureBox2, panel1) ;
```

22.3 ImageFinder Input

When the program starts, Figure 19.1 will be displayed. After the project is started, the Input Source can be changed via the “Mode” button in the Figure19.2. The implementation of the “Mode” button was given earlier.

22.4 Initialization

The LiveVideoInput70 object is implemented via the DirectShow component in DirectX. It will detect any cameras in the USB ports and will select the first camera automatically.

There are three buttons relevant to live video:

- Source
- Live, Live Stop, Live Start;
- Live To Image.

The “Source” button will initialize the LiveVideoInput70 object and you should be able to see the live video in the Panel:

```
private void searchSource7 ()
        {

          f.mainMenuToAPI .lv63.initialization ();
          canLiveStart = true;
          f.textBox2.Text = "Live Video start in the Panel";
          f.richTextBox1.Text = "Live Video start in the small window!\n"
                + "To start live video in the large window, click Live button!";
```

}

22.5 Video to Image Design

You will need a Logitech camera for this section. After starting the camera live, the **ImageFinder** will convert the live video file to a set of images. After that, live video recognition becomes image recognition.

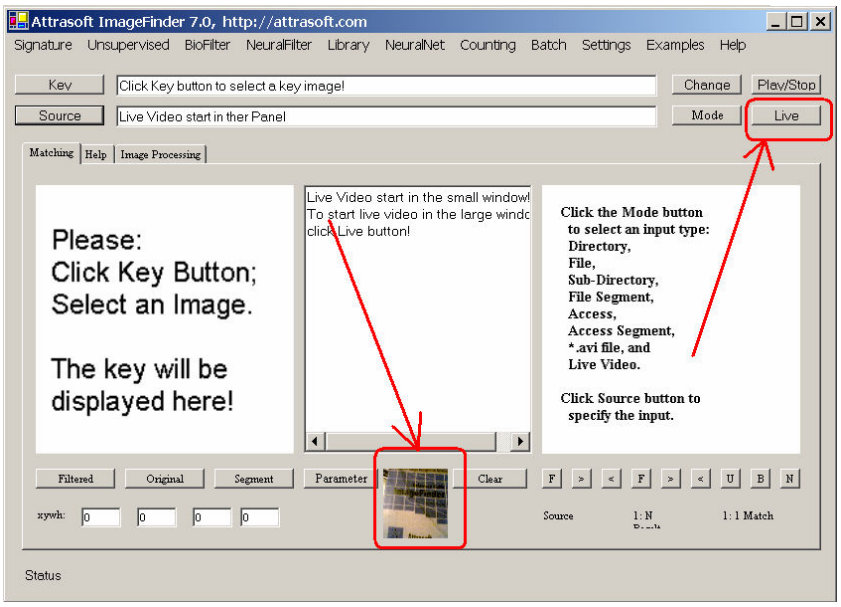

Figure 22.1 Converting Live Video File to Images, before clicking the "Live" button.

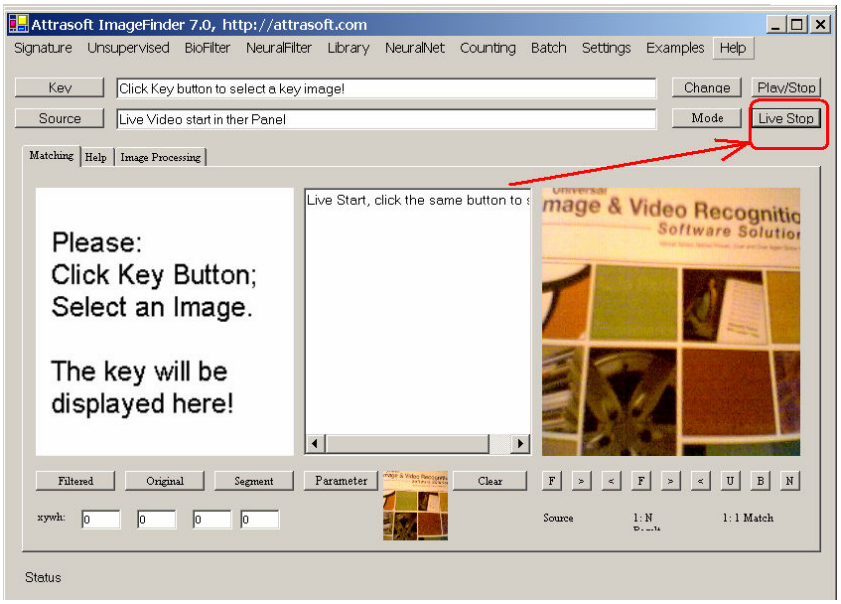

Figure 22.2 Converting Live Video File to Images, after clicking the "Live" button.

Example. Converting Live Video:

- Start the software;
- Select the option, “Live Video” (Figure 22.1);
- Click the “Source” button to get Figure 22.1, where live video is shown on a small panel on Figure 22.1;
- Click the “Live” button on Figure 22.1 to get Figure 22.2.
- Click the “Change” button in Figure 22.2 to get Figure 22.3;
- Click the “Live to Images” button to convert the Live Video to images;
- Click “OK” to exit Figure 22.3.
- Click the “Live Stop” button in Figure 22.2 to stop live video input.

You can find the image from live video in the folder specified by Line 2 in Figure 22.3. The default folder is:

C:\transapplet70\imagefinder\bin\debug\temp\.

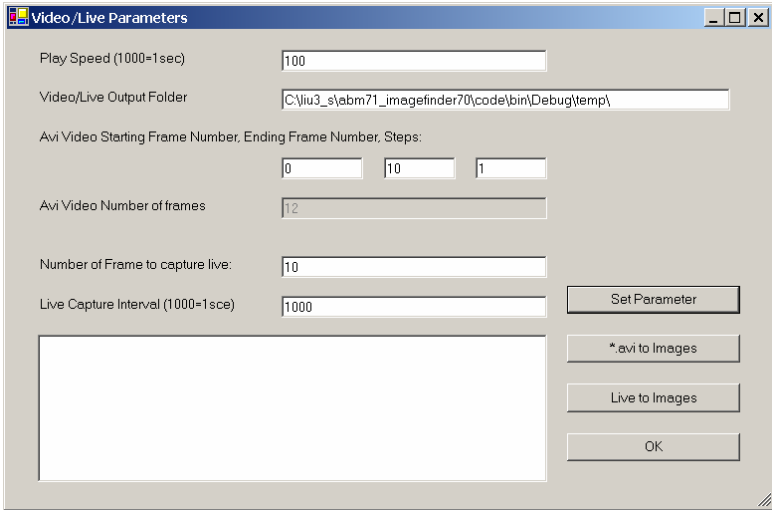

Figure 22.3 Converting Live Video to Images.

22.6 Display Live Image in Picture Box

To convert the live video to images, we have to activate the Picture Box via the “Live” button. The button is implemented as:

```
private void button6_Click(object sender, System.EventArgs e)
        {
        gui. Button_Live ();
        }
public bool Button_Live ()
        {
        if (! canLiveStart )
```

```
        {
                f.richTextBox1.Text =
                "Please set the mode to Live Video and click the source button!\n"
                + "The Mode button is right in front of the Live button.\n";
                return false;
        }
        if (iMode != 7 )
        {
                f.richTextBox1.Text =
                "Please set the mode to Live Video and click the source button!\n"
                + "The Mode button is right in front of the Live button.\n";
                return false;
        }
        iLive = (iLive + 1 )% 2;
        if (iLive == 0 )
        {
                f.button6.Text = "Live Stop";
                f.mainMenuToAPI .lv63.attach ();
                f.richTextBox1.Text =  "Live Start, click the same button to stop!";
        }
        else if (iLive == 1 )
        {
                f.button6.Text = "Live Start";
                f.mainMenuToAPI .lv63.detach ();
                f.richTextBox1.Text = "Live Stop, click the same button to start!";
        }
        return true;
}
```

If the LiveVideoInput70 object is not initialized, or the Input Source is not "Live Video", the button will not do anything. This button will either show the live video in the Picture Box or stop showing the live video in the Picture Box.

Once the live video is displayed in the Picture Box, the live images can be obtained by:

```
PictureBox.Image.Save
( image_name, System.Drawing.Imaging.ImageFormat.Jpeg );
```

22.7 Converting Live Video to Images

The "Live To Image" button in Figure 22.3 will convert the live video to images. The "Live To Image" code is:

```
private void button4_Click(object sender, System.EventArgs e)
        {
        if ( ! f.mainMenuToAPI .lv63 .getCaptureStatus () )
        {
```

```
                richTextBox1.Text = "Please set the mode to Live Video\n,
                        click the Source button, then Live Button!\n"
                        + "The Mode button is right in front of the Live button.\n";
                return;
        }

        try{
                f.mainMenuToAPI .lv63.setVideoToImagesDir
                        ( f.gui.videoLiveParameters .imageOutputFolder);
                f.mainMenuToAPI .lv63.setNumOfFrames
                         ( f.gui.videoLiveParameters .captureNumberOfFrames  ) ;
                f.mainMenuToAPI .lv63.setTimeInterval
                        ( f.gui.videoLiveParameters .captureTimeInterval   ) ;
                mer1.Interval =  f.gui.videoLiveParameters .captureTimeInterval ;
        }
        catch (Exception ee)
        {
                richTextBox1.Text = ee.ToString () + "\n";
                return;
        }
        if ( !  f.mainMenuToAPI .lv63.getVideoToImagesDirStatus () )
        {
                richTextBox1.Text = "Set Live Video To Image Directory Fails!" ;
                return;
        }

        richTextBox1.Text = "Output directory:\n"
                + f.mainMenuToAPI .lv63.getVideoToImagesDir ();

        liveID = 0;
        timer1.Enabled = true;
        }
```

Basically, this button simply starts the timer via the following statement:

```
timer1.Enabled = true;
```

The rest of the statements are for various tests. Figure 22.3 has the following constructor:

```
public Parameter_VideoInput(Form1 f1)
        {
        InitializeComponent();
        f = f1;
        img = f.pictureBox2 ;
}
```

Here object, img, is the picture box where the live image will be obtained. The timer will convert the live video into images as follows:

```
private void timer1_Tick(object sender, System.EventArgs e)
        {
                string s =
                 f.mainMenuToAPI .lv63.getVideoToImagesDir ()
                + DateTime.Now.Year+ "_"
                + DateTime.Now.Month + "_"
                + DateTime.Now.Day  + "_"
                + DateTime.Now.Hour + "_"
                + DateTime.Now.Minute + "_"
                + DateTime.Now.Second + "_"
                + ( DateTime.Now.Millisecond  /10 )
                + ".jpg";
        richTextBox1.Text = "" + (liveID ++) + ".   " + s;
        try
        {
                img.Image.Save ( s,System.Drawing.Imaging.ImageFormat.Jpeg );
        }
        catch (Exception ei )
        {
                richTextBox1.Text = ei.ToString () ;
        }
        if (liveID >= f.mainMenuToAPI .lv63.getNumOfFrames () )
        {
                timer1.Enabled = false;
                richTextBox1.Text = "Live Video To Image Conversion End! ";
        }
}
```

The key statement is:

```
img.Image.Save (  s,System.Drawing.Imaging.ImageFormat.Jpeg );
```

which will save the image in the Picture Box to a jpeg file. The following statement simply stops the timer when the number of pre-determined frames has been reached:

```
if (liveID >= f.mainMenuToAPI .lv63.getNumOfFrames () )
        {
                timer1.Enabled = false;
                richTextBox1.Text = "Live Video To Image Conversion End! ";
        }
```

23. Counting & Tracking Design

'Counting' counts the number of objects in an image, assuming there is no overlap between objects. If you need to count objects, which are NOT physically separated, then you need a customized version. 'Tracking' finds the most obvious object in an image and tracks it from image frame to image frame.

In this chapter, we will show the **counting design**. In the next chapter, we will show **counting implementation**.

23.1 Data

The data is located in the following folder, ".\input_auto_track". The example has one basic image in Figure 23.1. This image is shifted, so the **ImageFinder** can track the largest segment in the images, see Figure 23.2.

Figure 23.1 Original Images.

Figure 23.2 Shifted Images.

23.2 Counting the Left Image

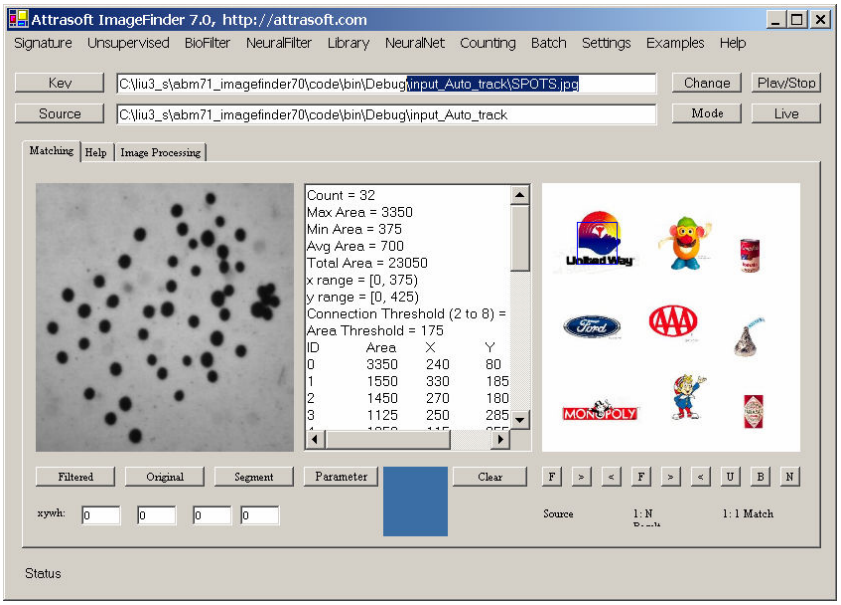

Figure 23.3 Counting the Left Image.

Please restart your ImageFinder so you do not carry over the old setting.

'Counting' will count the number of physically separated objects. Note: Customization will allow you to count non-physically separated objects.

To count the number of segments in the key image:

Input:

Key: ".\ input_Auto_track\SPOTS.jpg"

Parameters

All default values.

Operation

- Click the "Key" button and select ".\input_Auto_track\SPOTS.jpg";
- Click the "Source" button and select ".\input_Auto_track \";
- Set the Parameters as specified above;
- Click the "Counting/Count Left" button to count the number of segments on the left image.

Results

Count = 32
Max Area = 3350
Min Area = 375
Avg Area = 700
Total Area = 23050
x range = [0, 375)

y range = [0, 425)
Connection Threshold (2 to 8) = 3
Area Threshold = 175

ID	Area	X	Y	Perimeter
0	3350	240	80	610
1	1550	330	185	285
2	1450	270	180	245
3	1125	250	285	200
4	1050	115	355	200
5	1050	175	195	190
6	875	175	285	140
7	650	260	240	120
8	625	140	395	115
9	600	90	150	100
10	600	150	75	100
11	575	115	290	105
12	575	65	195	110
13	550	105	180	110
14	550	125	120	110
15	525	95	220	95
16	525	225	210	95
17	525	45	170	105
18	500	65	250	95
19	500	295	75	95
20	475	75	325	85
21	475	35	270	90
22	475	20	225	90
23	450	295	260	85
24	450	150	235	80
25	450	195	160	90
26	450	195	120	80
27	425	215	275	75
28	425	230	135	75
29	425	295	130	75
30	425	250	20	75
31	375	205	330	75

Here:

(x, y) is the center of the segment;
Area is the area of each segment in pixel-squares;
Perimeter is the perimeter of the segment in pixels;
x range = [0, 375) and y range = [0, 425) are the x- and y-dimensions of the image.

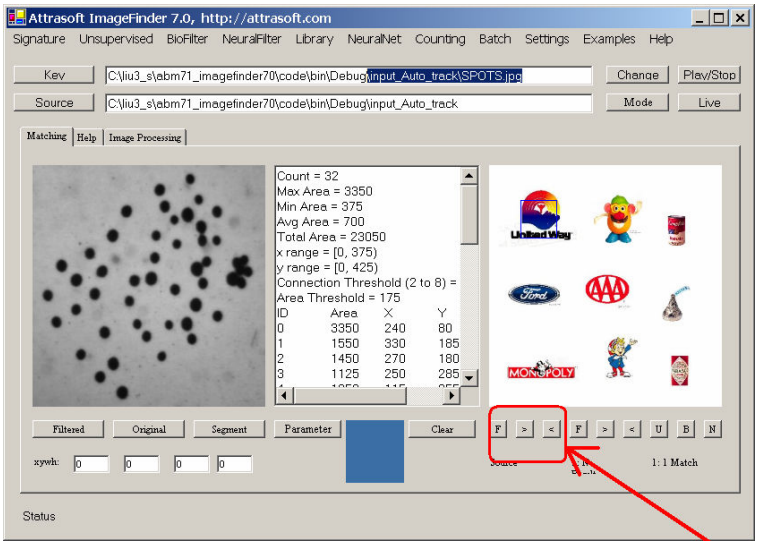

Figure 23.4 Counting the Right Image.

23.3 Counting the Right Image

In the last section, we counted the number of physically separated objects in the key image. We can count any image in the search source also. Note: Customization will allow you to count non-physically separated objects.

To count, select an image in a search source, use the F (First), > (Next), and < (Previous) buttons in Figure 23.4:

- Click the "F" button to see the first image in the search directory;
- Click the ">" to see the next image in the search directory;
- Click the "<" button to see the previous image in the search directory.

To count, click the "Counting/Count Right" button to count the number of segments on the right image. For Figure 23.3, the results are:

Count = 9
Max Area = 3024
Min Area = 1116
Avg Area = 1920
Total Area = 17280
x range = [0, 288)
y range = [0, 384)
Connection Threshold (2 to 8) = 3
Area Threshold = 84

ID	Area	X	Y	Perimeter
0	3024	60	84	616
1	2328	159	80	580
2	2256	144	196	452

3	2196	63	324	316
4	2064	54	200	312
5	1812	159	300	288
6	1320	234	320	204
7	1164	228	100	168
8	1116	225	220	168

23.4 Automatic Tracking

'Tracking' finds the most obvious object in an image and tracks it from image frame to image frame. To track these images, you have to specify the source and click the "Counting/ Track Largest segment" button.

Input:

Search directory: ".\ input_Auto_track\"

Parameters

All default values.

Operation

- Click the "Source" button and select ".\input_Auto_track \";
- Set the Parameters as specified above;
- Click the "Counting/Track Largest segment" button to track.

Results

IMAGE002.JPG	234	66	90	90
IMAGE002a.JPG	264	66	84	84
IMAGE002b.JPG	270	54	90	90
IMAGE002c.JPG	276	90	90	90
IMAGE002d.JPG	282	144	90	90
IMAGE002e.JPG	168	174	90	90
IMAGE002f.JPG	114	312	90	90
IMAGE004.JPG	42	48	90	90
IMAGE006.JPG	402	222	96	96
IMAGE008.JPG	78	84	90	90
SPOTS.jpg	216	55	55	55

To see where the matching segment is, there are three buttons in Figure 13.4:
F (First), > (Next), and < (Previous), that can be used to show where the matched segment is:

- Click the "F" button to see the first matched segment;
- Click the ">" to see the next matched segment;
- Click the "<" button to see the previous matched button.

24. Counting

In this chapter, we will introduce **Counting,** which is a component of the **ImageFinder**. The **Counting** is implemented via the following class library:

Attrasoft.TransApplet70.Counting70.

24.1 Introduction

The **Counting** Filter will:

- Locate all segments in an image;
- Print their center coordinates; and

The input for the Counting Filter is an image and the outputs are attributes. Each attribute is labeled by an integer and can be obtained individually. Among the attributes are areas of segments and locations of segments.

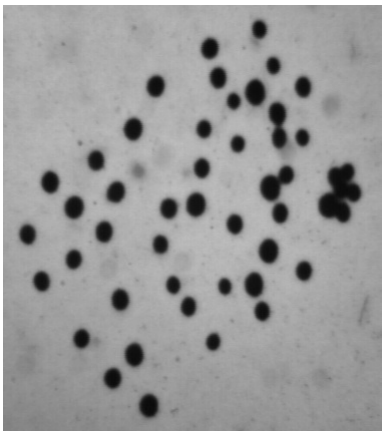

Figure 24.1 An image with 9 logos and an image with many spots.

24.2 Class Library Name

The class library is:

Attrasoft.TransApplet70.Counting70.

The class in this library will be:

Attrasoft.TransApplet70.Counting70.Counting70.

The interface, which will be used by **Counting70**, is:

```
public interface I_Counting70
{
        // 1. Parameters
        void setMinAreaCut(int x);
        int getMinAreaCut();
        void setMinConnectCut(int x);
        int getMinConnectCut();

        //2. Input
        bool setInput   ( int [] inputArray1,int new_w1, int new_h1);
        bool setInput   ( Bitmap bTrain);
        bool setInput   ( string s);

        // 3. Output
        int getCount();
        int getMaxArea();
        int getMinArea();
        int getAvgArea();
        int getTotalArea();

        int getNormalizedW();
        int getNormalizedH();
        int [] getArea();
        int [] getX();
        int [] getY();

        string toString();
}
```

24.3 Class Library Overview

To set the **Counting** parameters, use:

```
void setMinAreaCut(int x);
int getMinAreaCut();

void setMinConnectCut(int x);
int getMinConnectCut();
```

There are three commands for **Counting** input image**:**

```
bool setInput   ( int [] inputArray1, int new_w1, int new_h1);
bool setInput   ( Bitmap bTrain);
bool setInput   ( string s);
```

Once an image is entered into the **Counting70**, the following attributes can be obtained:

```
int getCount();
int getMaxArea();
int getMinArea();
int getAvgArea();
int getTotalArea();

int getNormalizedW();
int getNormalizedH();
int [] getArea();
int [] getX();
int [] getY();

string toString();
```

24.4 Link to Class Library

To include the class library in the project,

- Right click References and select Add Reference in the Solution Explorer;
- Browse to find "Counting70.dll" in "c:\transapplet70\";
- Highlight it and click the "OK" button.

To declare an object, add:

```
Attrasoft.TransApplet70.Counting70.Counting70 ct70
= new Attrasoft.TransApplet70.Counting70.Counting70 (richTextBox1, ip70, rd70);
```

Now **Counting70** object, ct70, is ready to use.

24.5 Counting Parameters

Minimum Area

If the segment area is too small, it can be considered as noise. This parameter is used to distinguish a small segment from background noise. The default value is 7. The “set” and “get” functions for this parameter are:

```
void setMinAreaCut(int x);
int getMinAreaCut();
```

Minimum Connectivity

If the segment is connected to another segment via only 1 pixel, are these two segments are really connected? This parameter is used to set a threshold for two segments being connected. The possible values are 2 to 8 and the default value is 3. The “set” and “get” functions for this parameter are:

```
void setMinConnectCut(int x);
int getMinConnectCut();
```

24.6 Implementing Counting

In this section, we will use the training image as an input and obtain the attributes from the Counting object**.** We will add a button to the last chapter project. Double click the “Counting\Count Left” and enter:

```
private void menuItem165_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI.count ( textBox1.Text );
        }

public bool count( string sInput)
        {
        if ( ! System.IO .File .Exists (sInput) )
        {
                appendText ( "Please enter a valid image!\n");
                return false;
        }

        bool b  = script.counting.setInput  (sInput  );
        if (!b )
                return false;

        setText (  countingResults( ) + "\n" );

        return true;

}
```

The first section makes sure a training image has been selected:

System.IO .File .Exists (sInput).

The second section uses the Counting object in the script object, “script.counting”. The object uses the training image as the input:

```
bool b  = script.counting.setInput  (sInput );
```

The last section calls the following function to print the attributes calculated by the **Counting:**

```
setText (  countingResults( ) + "\n" );
```

24.7 Tracking

In the **TransApplet**, tracking is a special type of counting.

- The Counting Filter selects 1 or 2 or … 5 of the largest segments.
- The Tracking Filter stores their coordinates and sizes. It will do this for all images in a folder and the results are a stream of attributes (centers of gravity, sizes) of images. In this way, the tracking is performed for the largest segment, second largest segment, …

For the tracking implementation, please see the project in:

“c:\transapplet70\imagefinder\”.

24.8 Testing

Please see the last chapter.

25. Batch Job

In this chapter, we introduce the Batch commands. When matching images, you will need to select many filters. For each selected filter, you will need to select many parameters. If this is your first matching and you do not like the default values, you will have go through a trial and error process to select an optimal set of parameters**.**

However, <u>if this is your second matching</u>, you can save everything in the first matching and then use a Batch command. Click the "Batch/Save" menu command; you will get the batch file in the text area.

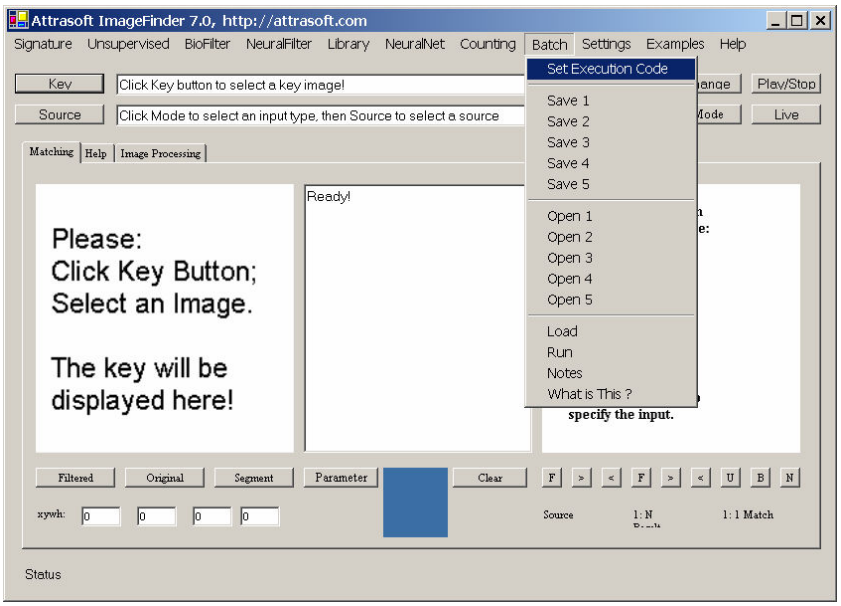

Figure 25.1 Batch Menu.

25.1 Batch Code

Batch Code is a set of text files that saves the **ImageFinder** parameters. The Filter selection and Parameter setting can be saved in one of 5 files by the following commands:

Batch/Save
Batch/Save 2
Batch/Save 3
Batch/Save 4
Batch/Save 5

These 5 commands create the batch codes and saves them to 5 different files. The batch codes can also be recalled later by clicking the following commands:

Batch/Open
Batch/Open 2
Batch/Open 3
Batch/Open 4
Batch/Open 5

These 5 commands open existing batch codes.

25.2 Sample Batch Files

Click "Example/Special Example/Document Duplication"; you will get the following batch code, which is very typical.

```
[ImageFinder 7.0]
        ExecutionCode=3002
[Input]
        keyFileName=Click Key button to select a key image!
        keysegmentX=0
        keysegmentY=0
        keysegmentW=0
        keysegmentH=0
        searchSource=.\Debug\sp_document
        searchSourceType=2
        searchSQLStatement=NA
[ImagePreProcessing]
        BorderCut=0
        MaskX=0
        MaskY=0
        MaskW=0
        MaskH=0
        MaskType=0
        StickShift=0
        SkipEmptyBorder=0
        SkipEmptyBorderPercent=0
        SkipEmptyBorderEdgeFilter=0
        SkipEmptyBorderThresholdFilter=5
        Parameter12=0
        Parameter13=0
        Parameter14=0
        Parameter15=0
[Image Processing Filters]
        EdgeFilter=2
        ThresholdFilter=1
        CleanUpFilter=2
        DoubleProcessing=0
        R1=0
        R2=128
        R3=2
        G1=0
        G2=128
        G3=2
        B1=0
```

B2=128
B3=2
Parameter14=0
Parameter15=0
Parameter16=0
Parameter17=0
Parameter18=0
Parameter19=0
[Reduction Filter]
ReductionFilter=0
SegmentCut=0
SizeCut=0
BorderCut=0
lookAtX=0
lookAtY=0
lookAtXLength=0
lookAtYLength=0
[Signature Filter]
SignatureFilter=9
[Unsupervised Filter]
UnsupervisedFilter=0
FaultToleranceScale=20
Mode=0
Threshold=0
OutputFileType=0
Show File=1
Blurring=2
Sensitivity=4
UseRelativeScore=0
ShowScore=1
AutoSegment=0
Parameter12=0
Parameter13=0
Parameter14=0
Parameter15=0
Parameter16=0
Parameter17=0
Parameter18=0
Parameter19=0
[BioFilter]
bioFilter=0
FaultToleranceScale=20
Mode=0
Threshold=0
OutputType=0
ShowFile=1
Blurring=2
Sensitivity=4
UseRelativeScore=0
ShowScore=1
AutoSegment=0
Parameter12=0
Parameter13=0

```
        Parameter14=0
        Parameter15=0
        Parameter16=0
        Parameter17=0
        Parameter18=0
        Parameter19=0
[NeuralFilter]
        neuralFilter=2
        FaultToleranceScale=20
        Mode=0
        NeuralFilterSize=2
        Threshold=0
        OutputFileType=0
        ShowFile=1
        Blurring=0
        Sensitivity=0
        UseRelativeScore=0
        ShowScore=1
        AutoSegment=0
        Parameter13=0
        Parameter14=0
        Parameter15=0
        Parameter16=0
        Parameter17=0
        Parameter18=0
        Parameter19=0
[Neural Net]
        neuralNetFilter=0
        symmetry=3
        rotationType=0
        translationType=0
        scalingType=0
        sensitivity=50
        blurring=10
        internalWeightCut=100
        Threshold=0
        segmentSize=0
        imageType=1
        OutputFileType=0
        AutoSegment=0
        Mode=0
        Parameter15=0
        Parameter16=0
        Parameter17=0
        Parameter18=0
        Parameter19=0

End
```

When you create the batch code by command, Batch/Save, you will see the above code in the text area.

When you open a batch file by command, Batch/Open, you will see the above code in the text area.

25.3 Batch Design

This section will explain how the Batch code is used.

(1) Create an application using the **ImageFinde**r;
(2) Save the setting to a batch code with the following commands:

Batch/Save
Batch/Save 2
Batch/Save 3
Batch/Save 4
Batch/Save 5

You might find the following online note useful in helping you remember what you saved into these 5 batch files:

Batch/Notes

(3) Later, you can open the Batch file with the following commands:

Batch/Open
Batch/Open 2
Batch/Open 3
Batch/Open 4
Batch/Open 5

(4) To load the parameter without running, click:
Batch/Load.

(5) To load the parameter and run, click:
Batch/Run.

The Batch/Save command saves the following information:

- Filter selection and their Parameter settings;
- The signature file, which contains the signatures from images.

25.4 Batch Execution Code

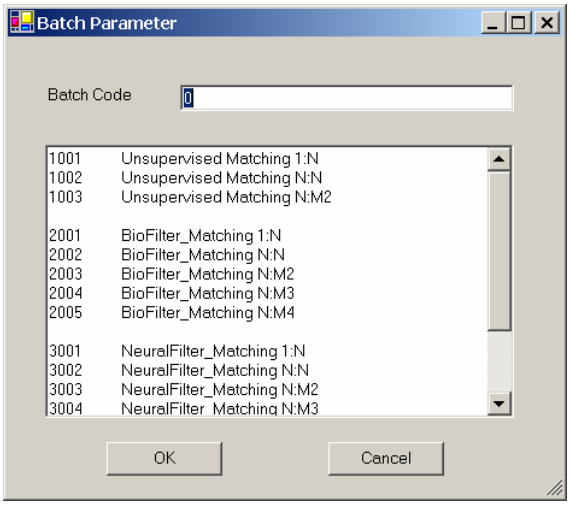

Figure 25.2 Execution Code Window.

There are many commands in the **ImageFinder.** Each command has an integer for identification. This integer is called **Batch Execution Code**. The "Batch/Run" command uses this code to run the command specified by the batch file.

To find the batch code for each command, click:

Batch/Set Execution Code

You will see a textbox and the following codes:

1001 Unsupervised Matching 1:N
1002 Unsupervised Matching N:N
1003 Unsupervised Matching N:M2

2001 BioFilter_Matching 1:N
2002 BioFilter_Matching N:N
2003 BioFilter_Matching N:M2
2004 BioFilter_Matching N:M3
2005 BioFilter_Matching N:M4

3001 NeuralFilter_Matching 1:N
3002 NeuralFilter_Matching N:N
3003 NeuralFilter_Matching N:M2
3004 NeuralFilter_Matching N:M3
3005 NeuralFilter_Matching N:M4

4001 NeuralNet_Matching 1:N
4002 NeuralNet_Matching N:N

The current batch code is the code of the last run. When the software started, the current batch code is 1001. To change it:

- Click Batch/Set Execution Code;
- Enter the Batch Execution Code to the text box and click the “OK” button.

You can also make changes directly in the batch files. The batch files are abm70.txt, abm70_2.txt, abm70_3.txt, abm70_4.txt, abm70_5.txt.

25.5 API

The Batch API is:

```
public interface I_Batch
        {
        void setExecutionCode(int x);
        int getExecutionCode();
        string [] getExecutionCodeName();
        int [] getExecutionCodeValue();
        bool getExecutionCodeStatus();

        bool save (int i);
        bool open (int i );
        bool load ();
        bool run();
  }
```

These functions match the menu items introduced earlier.

25.6 Implementation

Double click menu item “Batch/Run” and enter:

```
private void menuItem142_Click(object sender, System.EventArgs e)
        {
        this.mainMenuToAPI .batch_Run ();
        }
```

Here, “mainMenuToAPI” is an object, which will implement all functions. As we discussed earlier, the main form simply links menu items to functions in the “mainMenuToAPI” object. The implementation is:

```
public bool batch_Run ()
        {
        bool b = false;
        try
        {
                b  = script.batch.run () ;
                if ( !b)
```

```
                {
                        return false;

                }

                b = batch_GUI_LoadInput (  );
                if ( !b)
                {
                        appendText ( "Batch/Run: No Search Source Available\n");
                        return false;

                }

                script.results_1N = script.tranAppletPara .results_1N ;
                b = batch_GUI_LoadOutput (  );
                if ( !b)
                {
                        appendText( "Batch/Run: No Search Results Available\n");
                        return false;
                }
        }

        catch (Exception e )
        {
                appendText (e.ToString () + "\n" );
                return false;

        }
        …

        return b;
        }
```

The first section,

```
b  = script.batch.run () ;
```

makes a batch run.

The second section,

```
b = batch_GUI_LoadInput (  );
```

retrieves and displays search source images.

The third section,

```
script.results_1N = script.tranAppletPara .results_1N ;
b = batch_GUI_LoadOutput (  );
```

displays result images.

The details are given in the chapter project located at “c:\transapplet70\imagefinder\”.

26. ImageFinder for Dos

ImageFinder for Dos is a quick and easy programming tool. **ImageFinder for Dos** is the **ImageFinder for Windows** without the Graphical User Interface (GUI). Other than this, the two software are exactly the same.

The chapter project is located at:

c:\transapplet70\transapplet70.chap26\.

The executable file is located at:

c:\transapplet70\ transapplet70.chap26\bin\Debug\.

26.1 Why Dos Version?

ImageFinder for Dos is the **ImageFinder for Windows** without the Graphical User Interface (GUI). If you have developed a solution with the **ImageFinder** and want some minimum programming capability, the **ImageFinder for Dos** is a really simple and easy option.

Example. The **ImageFinder for Dos** is useful for the following 1:N Matching while the training image is:

- A newly captured image via a camera;
- A newly captured image via a scanner;
- A newly uploaded image via the Internet;
- A newly uploaded image via a video-phone.

The following are the potential applications:

- Cell-phone point & click:
  - Identifying pictures or other art in a museum, where it is desired to provide additional information about such art objects to museum visitors as well as related advertisement;
- Cell-phone point & click:
  - Identifying wine labels, where it is desired to provide additional information about this wine as well as related advertisement;
- Cell-phone point & click:
  - Identifying sport logo, where it is desired to provide additional information about ticket purchases for this team as well as related advertisement;
- Cell-phone point & click:
  - Identifying a tourist attractions, such as the Space Needle in Seattle, where it is desired to provide additional information about the tourist attraction as well as related advertisement;
- Cell-phone point & click:

- To buy certain products, for example "pointing and clicking" on a theatre advertisement to buy tickets as well as related advertisements (people who see this movie will often see that movie).

26.2 The Idea

There are also ten menu items under the menu, “Batch”, in the **ImageFinder** for Windows:

Batch/Save
Batch/Save 2
Batch/Save 3
Batch/Save 4
Batch/Save 5

Batch/Open
Batch/Open 2
Batch/Open 3
Batch/Open 4
Batch/Open 5

The first 5 commands create the batch codes and saves them to 5 different files. The batch codes can be opened later by the next 5 commands.

ImageFinder for Dos allows you to run these five batch files from the Dos prompt, Dos batch file, Visual Basic program. This allows developers to quickly integrate the **ImageFinder** into their applications.

26.3 Batch Design

The **ImageFinder for Dos commands are:**

C:\>…\chap26 x

Where **x = 1, 2, 3, 4, or 5** (used to specify one of the five batch files). The Dos version uses the same files as the Batch70 object. If you use the command:

C:\>…\chap26 1

You will need the following files:

Abm70.txt
Bf70.txt
Bf270.txt
Nf70.txt
Nf270.txt
Tp70.txt

If you use the command:

C:\>…\chap26 2

You will need the following files:

Abm70_2.txt
Ba70_2.txt
Bf270_2.txt
Nf70_2.txt
Nf270_2.txt
Tp70_2.txt

Since this is basically the Windows version without the form, it will stop when the Window's version stops. If you want to stop the **ImageFinder** at the Dos prompt,

- Hit "Ctrl+Alt+Del" to open the Windows Task Manager;
- Go to the Processes Tab;
- Stop chap26.exe.

26.4 Class Library Name

The class library is:

Attrasoft.TransApplet70.Dos70.

The class in this library will be:

Attrasoft.TransApplet70.Dos70.Dos70.

The interface, which is implemented by Dos70, is:

```
public interface I_Dos70
{
  bool save (int i);
  bool open (int i );
  bool load ();
  bool run();
  bool ImageFinderForDos (int i);
}
```

26.5 Class Library Overview

The class library, Dos70, will introduce the **ImageFinder for Dos**.

There is one only command:

bool ImageFinderForDos (int x).

The parameter, x, can have the following values: 1, 2, 3, 4, 5.

- Value 1 corresponds to the ImageFinder command: “Batch/Open” + “Batch/Run”;
- Value 2 corresponds to the ImageFinder command: “Batch/Open 2” + “Batch/Run”; …

26.6 Creating Console Project

To create a new C# Console project:

- Start the Visual Studio .Net, (see Figure 4.1).
- Click File ➔ New Project command. The New Project dialog box is displayed (Figure 4.2).
- Highlight the Visual C# project folder in the Project Type list to display the templates that are available for C#. Then, highlight the **Console Application template** (Figure 4.2).
- Enter a name for the project and select the location for the project. A folder with the same name as the project is automatically added to the location you specify. We will use chap26 as the project name.
- Click the “OK” button to start the new project.

The Console project will also has a parameter. To set it:

- In the Solution Explorer, right click Property;
- In the Property Window, go to “Configuration property”;
- In “Command Line Arguments”, **enter 1.**

26.7 Link to Class Library

To include the class library in the project,

- In the Solution Explorer, right click References and select Add Reference;
- Browse to find “*.dll” in “c:\transapplet70\”;
- Highlight it and click the “OK” button.

To declare an object, add:

```
Attrasoft.TransApplet70.Dos70.Dos70 d70
= new Attrasoft.TransApplet70.Dos70.Dos70 ();
```

26.8 Implementation the Project

The implementation is fairly straightforward. The first thing to do is to read the argument, which determines which batch file is to be used. This argument can be 1, 2, 3, 4, or 5. Then it will call the run function in the Dos object.

The algorithm is:
Step 1. Object d70
The project first creates an object, d70. If this object cannot be created, the program terminates.

Step 2. Argument
The project will then find the argument, which determines the batch file to be used. If there is any error, it will use the default 1.

Step 3. Data Files.
If you use the command:

C:\>…\chap26 1

You will need the following files:

Abm70.txt
Bf70.txt
Bf270.txt
Nf70.txt
Nf270.txt
Tp70.txt

These text files are used by the **ImageFinder for Dos** and are stored in the ".\data\" folder. If you did not change the installation folder, it is:

"c:\transapplet70\ transapplet70.chap26\bin\Debug\data".

Before you can run the Dos version, you must prepare these files using the Windows version for your application.

Step 4. Code

```
class Class1
{
	static string sCode = "0";
	static void Main(string[] args)
	{
		Attrasoft.TransApplet70.Dos70.Dos70 d70;
		try
		{
			d70 = new Attrasoft.TransApplet70.Dos70.Dos70 ();
		}
		catch (Exception ee)
		{
			Console.Write (ee.ToString () );
```

```
                return;
        }

        int theCode = 0;
        if (args.Length == 0 )
                theCode = 0;
        else
        {
                sCode = args[0];
                theCode = getCode ();
        }
        Console.WriteLine ("Batch Option " + theCode );

        try
        {
                bool b = d70.ImageFinderForDos (theCode ) ;
                Console.Write ( d70.toString () );
                if ( !b )
                        Console.Write ( "ImageFinder for Dos fails!" );
        }
        catch (Exception ee)
        {
                Console.Write (ee.ToString () );
        }
}

static int getCode (  )
        {
                int i = 0;
                try
                {
                        i = int.Parse (sCode);
                }
                catch
                {
                        Console.WriteLine ("Invalid args!");
                        return 0;
                }
                if ( (i>=0 ) && (i <= 5 ) )
                        return i;
                return 0;
        }
}
```

The first section,

```
d70 = new Attrasoft.TransApplet70.Dos70.Dos70 ();
```

creates a Dos object.

The second section,

```
sCode = args[0];
theCode = getCode (); (which calls int.Parse (sCode) )
```

gets the "Command Line Arguments".

The last section,

```
bool b = d70.ImageFinderForDos (theCode ) ;
```

calls the dos command.

26.9 Example

The default folder for the project is:

"C:\transapplet70\transapplet70.chap26\".

The executable file folder is:

"C:\transapplet70\transapplet70.chap26\bin\debug".

The data folder is:

"C:\transapplet70\transapplet70.chap26\data\.

Go to Dos command and enter:

"C:\transapplet70\transapplet70.chap26\bin\debug\chap26 1",

You will duplicate the project under the following menu item in the Windows version:

"Examples\BioFilter\Label N:N".

26.10 How to Use ImageFinder For DOS

The typical use of the dos version is a 1:N Matching, where the "1" is the newly captured image and the "N" is the previously stored images. The "N" image will not change, but the "1" image changes all the time. How is this change entered into the **ImageFinder for Dos**? The answer is the parameter file.

The **ImageFinder for Dos** uses a set of parameter files, Abm70.txt for argument x = 1, Abm70_2.txt for argument x = 2, ….

If you use the following command:

"C:\transapplet70\transapplet70.chap26\bin\debug\chap26 1",

you will need the following files:

Abm70.txt
Bf70.txt
Bf270.txt
Nf70.txt
Nf270.txt
Tp70.txt

stored the following folder:

"C:\transapplet70\transapplet70.chap26\bin\debug\data\".

The Abm70.txt looks like this:

```
[ImageFinder 7.0]
        ExecutionCode=3002
[Input]
        keyFileName=Click Key button to select a key image!
        keysegmentX=0
        keysegmentY=0
        keysegmentW=0
        keysegmentH=0
        searchSource=.\Debug\sp_document
        searchSourceType=2
        searchSQLStatement=NA
[ImagePreProcessing]
        BorderCut=0
        MaskX=0
        MaskY=0
        MaskW=0
        MaskH=0
…
```

The following line,

KeyFileName = Click Key button to select a key image!

specifies the key image. Before the **ImageFinder for Dos** is called, your Dos application must create a new Abm70.txt, with the above line specifying the newly captured image. This can be achieved by writing a simple program, which creates a new abm70.txt by copying from a master file. The simple program will copy line by line from a master file to abm70.exe with the exception of line 4, which specifies the newly captured image. In this way, when the **ImageFinder for Dos** is called, the newly captured image will be used.

27. Introduction To ImageHunt

We will briefly introduce the **Attrasoft ImageHunt** in this chapter. **Attrasoft ImageHunt** is an Internet Image Search Engine. The **ImageHunt** is the Internet version of the ImageFinder.

Unlike the **ImageFinder**, the Internet Search Engine, by definition, does not bother users with complicated parameters; therefore, all the parameters in the **ImageHunt** must be fixed. The **ImageFinder**, however, will require customization for a particular problem, say logos, auto parts, documents, ….

If you order the **ImageHunt** from Attrasoft, the following will be required:

(1) Proof-Of-Concept (POC) Project. The POC project will tweak and fix the internal parameters for a particular problem.
(2) **ImageHunt** License and Support.

If you plan to build the **ImageHunt** yourself, this chapter is for you:

(1) Fix the ImageFinder parameters yourself;
(2) The code for building the **ImageHunt** can be found in this chapter.

This chapter will:

- Introduce the **ImageHunt**;
- Introduce Asp.Net (Active Server Page);
- Introduce the **ImageHunt** programming.

27.1 Why ImageHunt?

ImageHunt is the **ImageFinder for Windows** with a web interface. If you need the ImageFinder for Windows, you might need the ImageFinder for Web at some point. The web users, however, are not supposed to know how to tune the ImageFinder parameters, so that job has to be done in advance. The cost for such convenience is that each **ImageHunt** is limited to a particular type of application.

27.2 ImageHunt Design

This chapter will show you how to program the **ImageHunt**. The basic idea is this:

- User selects an image;
- User uploads the image;
- **ImageHunt** creates a batch file based on the newly uploaded image;
- **ImageHunt** runs the batch file;

- **ImageHunt** sends the resulting html page back to users.

The main job is to tweak the parameters via the ImageFinder. After that, simply add a web interface to the **ImageFinder for Dos** to complete the **ImageHunt**. The **ImageHunt** works like this:

(1) Start the **ImageHunt** (Figure 27.1).

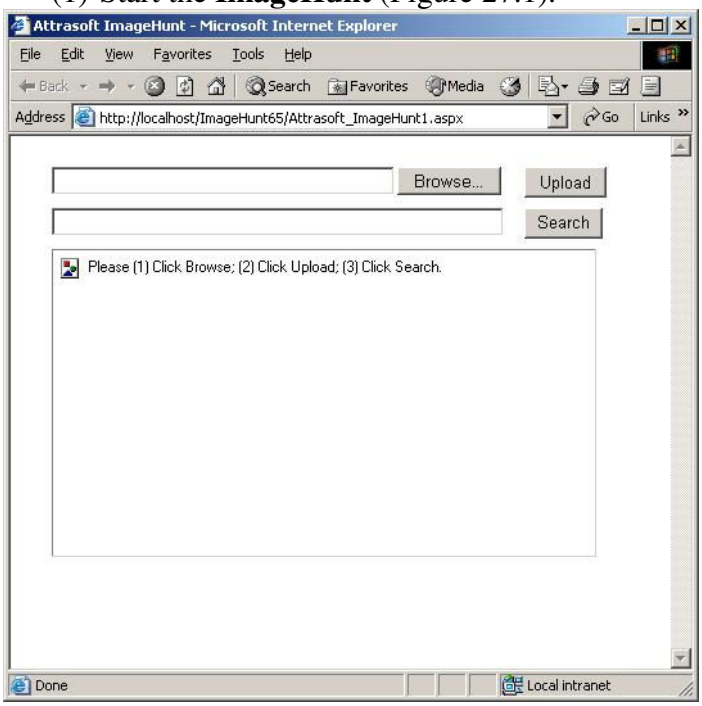

Figure 27.1 The **ImageHunt**.

(2) Click the "Browse" button to select an image (Figure 27.2).

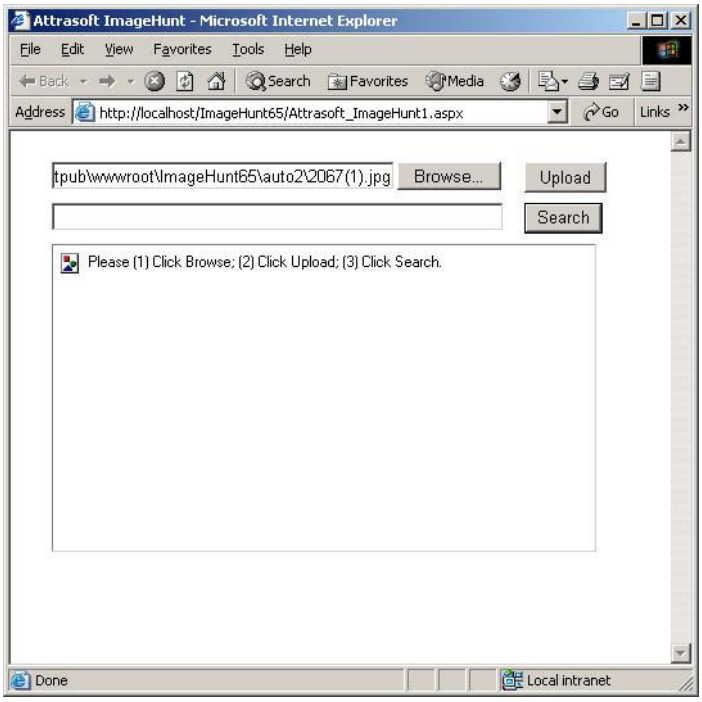

Figure 27.2 Select an image.

(3) Click the “Upload” button to upload the image (Figure 27.3).

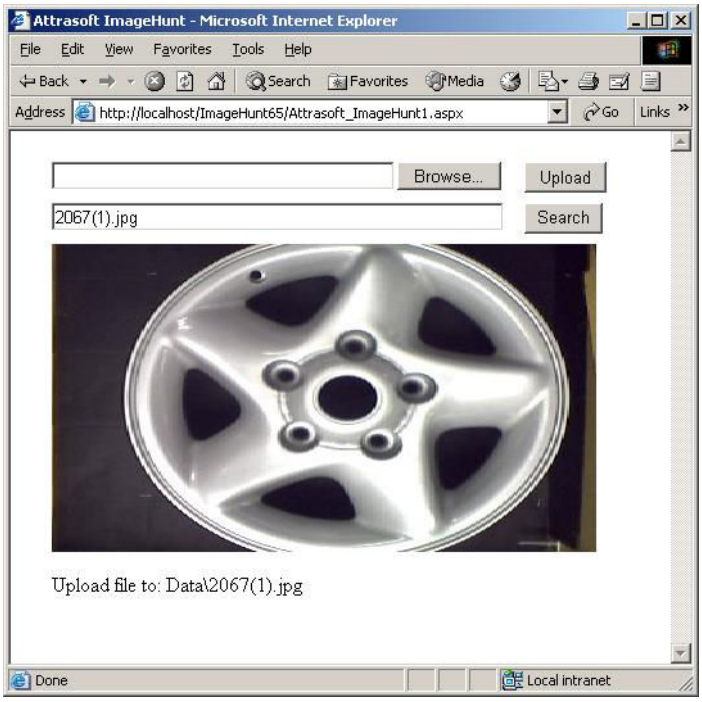

Figure 27.3 Upload image.

(4) Click the “Search” button to search (Figure 27.4).

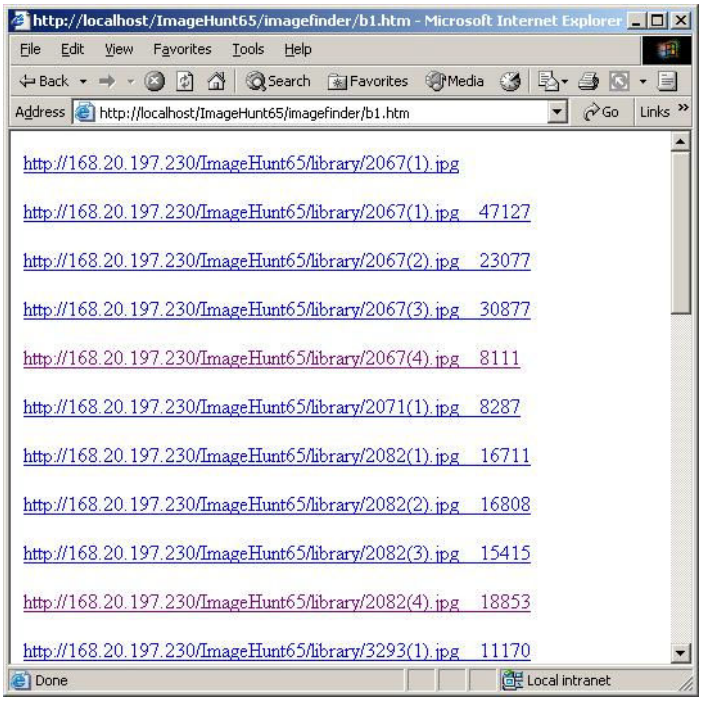

Figure 27.4 Display the results.

27.3 Introduction to Web Server

Web applications require a server and a client. The client requests a page from the server and the server returns the page to the client; the page is displayed inside the Internet Explorer.

To develop a web application, you must have a server. The **ImageHunt** requires Microsoft IIS (Internet Information Services) and the Internet Explorer. You must install IIS before you install C#.

Internet Browsers display pages written in http language, web page. The web pages are stateless, i.e. a page does not store any information about its contents from one page to the next. The equivalent to the Windows version of the ImageFinder is this: when you start the ImageFinder, you are only allowed 1 click; if you want to make another click, close the ImageFinder, restart, and then make your next click.

There are several ways to address this limitation, including the "Get" method and the "Post" method.

- The "Post" method stores cookies on the local machine. If you choose the "Post" method, you must inform your clients to accept cookies.
- The "Get" method sends state information back to the server via the web address.

Later in this chapter, you will need to choose one of the two methods.

27.4 Install ImageHunt

The software requirement for the **ImageHunt** is Microsoft .Net Framework.

To load the **ImageHunt**:

1. IIS Folder or Virtual Folder.
- Copy "CD:\ImageHunt65" to "C:\InetPub\wwwroot\"; **OR**
- Create a virtual folder in IIS, "C:\…\ ImageHunt65\".

2. IIS Application
- Click Start\ Setting\ Control Panel \Administrative Tools\Internet Service Manager;
- Right click the "ImageHunt65" folder and select Property;
- Click the "Application" button to create application;
- Click "OK".

3. Web shared.
- Make sure the "ImageHunt65" folder is web shared.

27.5 Create Web Project

Now we start to create an **ImageHunt**. To create a new C# ASP.NET Application project:

- Start the Visual Studio .Net.
- Click File ➔ New Project command. The New Project dialog box is displayed.
- Highlight the Visual C# project folder in the Project Type list to display the templates that are available for C#. Then, highlight the **ASP.NET Application** template.
- Enter a name for the project and select the location for the project. A folder with the same name as the project is automatically added to the location you specify.
- Click the "OK" button to start the new project.

The implementation is fairly straightforward. The algorithm is:

Step 1. Open an image via the Internet Explorer.

Step 2. Upload the Image.

Step 3. Create a batch file on the server based on the submitted image.

Step 4. Run the batch file.

Step 5. Send the results back.

We will make the following assumption for simplicity: all library images are stored in a single folder.

27.6 Step 1. Open Image File

27.6.1 Create the Data Directory

After you create the application, you create the Data directory that will accept uploaded files. After you create this directory, you must also set write permission for the ASPNET worker account. In the Solution Explorer window of Visual Studio .NET, right-click the ASP.Net project, point to Add, and then click New Folder to create a folder, "Data".

27.6.2 Modify the WebForm1.aspx Page

To modify the HTML code of the WebForm1.aspx file to permit users to upload files, follow these steps:

1. In the WebForm1.aspx Designer window, right-click WebForm1.aspx and select View HTML Source.

2. Locate the following HTML code, which contains the:

```
<form>
…
</form>
```

3. Replace it as follows:

```
<form id="Form1" method="post" enctype="multipart/form-data" runat="server">
        <INPUT type=file id=File1 name=File1 runat="server" />
</form>
```

4. Run and test it.

This html code simply opens an Open File Dialog so Users can select a key image.

27.7 Step 2. Upload Image

Add the following to the form so it will look like Figure 27.1:

- Upload button
- Search button
- TextBox
- Error message label

Double click the “Submit” button and enter:

```
private void Submit1_ServerClick(object sender, System.EventArgs e)
        {
        if( ( File1.PostedFile != null ) && ( File1.PostedFile.ContentLength > 0 ) )
                {
                string fn = System.IO.Path.GetFileName(File1.PostedFile.FileName);
                SaveLocation = Server.MapPath("Data") + "\\" +  fn;

                Session [ "theFileName" ] = SaveLocation;
                string displayLocation =  dataPath + "\\" +  fn;
                try
                {
                        File1.PostedFile.SaveAs(SaveLocation);
                        Label2.Text = "Upload file to: " + displayLocation;

                        Image2.ImageUrl = displayLocation;
                        TextBox2.Text =
                        System.IO.Path.GetFileName(File1.PostedFile.FileName);
                }
                catch ( Exception ex )
                {
                        Label2.Text = "Error: " + ex.Message;
                }
                }
        else
                {
                        Response.Write("Please select a file to upload.");
                }
```

```
}
```

First of all, the following code makes sure a valid image has been selected:

```
if( ( File1.PostedFile != null ) && ( File1.PostedFile.ContentLength > 0 ) )
        {
        …
        }
else
        {
                Response.Write("Please select a file to upload.");
        }
```

The following code finds the image name from the client's computer and creates the path for this image on the server:

```
string fn = System.IO.Path.GetFileName(File1.PostedFile.FileName);
SaveLocation = Server.MapPath("Data") + "\\" +  fn;
```

The following code uploads the file:

```
try
{
        File1.PostedFile.SaveAs(SaveLocation);
}
catch ( Exception ex )
{
        Label2.Text = "Error: " + ex.Message;
}
```

27.8 Step 3. Create Batch File

27.8.1 Cookies or URL

The User will make three clicks:

- Select File;
- Upload File;
- Search.

The third click is the "Search" button, which will lead to a new web form. The code for the "Search" button is:

```
private void Button1_Click(object sender, System.EventArgs e)
{
string fn = TextBox2.Text;
SaveLocation = Server.MapPath("Data") + "\\" +  fn;
```

```
Response.Redirect ("Attrasoft_ImageHunt2.aspx" + "?p="+ SaveLocation);
}
```

The uploaded image is passed to the new web form, “Attrasoft_ImageHunt2.aspx”. You could choose cookies (Post Method):

```
Session [ "theFileName" ] = SaveLocation;
```

Or URL (Get Method):

```
Response.Redirect ("Attrasoft_ImageHunt2.aspx" + "?p="+ SaveLocation);
```

Obviously, we have chosen the URL to pass the location of the uploaded image. If you choose the Cookies, you must inform your client to accept cookies.

27.8.2 Create Batch File

At this point, we are redirected to web form, “Attrasoft_ImageHunt2.aspx”. Upon starting this web form, the following code will call function, “start()”:

```
private void Page_Load(object sender, System.EventArgs e)
        {
        start();
        }
```

The first thing “start()” will do is to create a batch file used by the **ImageFinder for Dos**:

```
string master = "master.txt";
string parameterFile = "abm70.txt";
bool createFile ()
        {
        StreamWriter sw;
        StreamReader sr;
        string sRead;
        int i = 0;

        try
        {
                sr = new StreamReader ( matchLocation + master );
                sw = new StreamWriter ( matchLocation + parameterFile );

                i = 0;
                while ( sr.Peek () != -1 )
                {
                sRead = sr.ReadLine ();
                if ( i == 3 )
                {
```

```
                //sw.WriteLine ( "trainFileName=" + Session [ "theFileName" ] );
                sw.WriteLine
                        ( "trainFileName=" +Request.QueryString ["p"].ToString () );
                }
                else
                        sw.WriteLine (sRead);
                        i++;
                }
                sw.Close ();
                sr.Close ();
                }
                catch ( Exception em )
                {
                        Label1.Text =  em.ToString () +"\n";
                        return false;
                }
                return true;
        }//create file
```

This function basically copies "master.txt" to "abm70.txt" line by line, with the exception of line 3:

```
        if ( i == 3 )
                {
                //sw.WriteLine ( "trainFileName=" + Session [ "theFileName" ] );
                sw.WriteLine
                        ( "trainFileName=" +Request.QueryString ["p"].ToString () );
                }
```

This is where the **ImageHunt** learns where the new image is. There are several other minor things; we will omit them here.

27.9 Dos Class

We will briefly review the Dos70 library introduced in the last chapter. We will use the "Attrasoft.Transapplet70.Dos70" library; please read the last chapter to become familiar with this library.

The main class in this library will be:

Attrasoft.TransApplet70.Dos70.Dos70.

The interface, which is implemented by Dos70, is:

```
public interface I_Dos70
{
//Parameters
```

```
  string getFilePath ();
  void setFilePath (string s);

// Command
  bool save (int i);
  bool open (int i );
  bool load ();
  bool run();
  bool ImageFinderForDos (int i);
}
```

To set the batch file, use:

```
 string getFilePath ();
 void setFilePath (string s).
```

There is only one command:

```
bool ImageFinderForDos (int x).
```

which is the same as:

```
bool open (int i );
bool run();
```

The parameter, x, can have the following values: 1, 2, 3, 4, 5.

- Value 1 corresponds to the ImageFinder command: “Batch/Open” + “Batch/Run”;
- Value 2 corresponds to the ImageFinder command: “Batch/Open 2” + “Batch/Run”; …

To include the class library in the project,

- In the Solution Explorer, right click References and select Add Reference;
- Browse to find “Dos70.dll” in “c:\transapplet70\”;
- Highlight it and click the “OK” button.

To declare an object, add:

```
Attrasoft.TransApplet70.Dos70.Dos70 d70
= new Attrasoft.TransApplet70.Dos70.Dos70 ();
```

27.10 Step 4. Batch Run

Now the batch file is created, and we simply make a batch run:

```
void start()
        {
```

```
                matchLocation = Server.MapPath(filePath) + "\\" ;

                bool b = createFile ();
                if (!b)
                {
                        Label1.Text = "Creating parameter file fails!" ;
                        return ;
                }

                …


                d70 = new Attrasoft.TransApplet70.Dos70.Dos70 ();

                d70.setFilePath (matchLocation);
                d70.open (1);
                …

                70.run ();

                Response.Redirect ("imagefinder/b1.htm");
        }
```

The following code creates a batch file:

```
        bool b = createFile ();
                if (!b)
                {
                        Label1.Text = "Creating parameter file fails!" ;
                        return ;
                }
```

The following code creates a Dos70 object:

```
d70 = new Attrasoft.TransApplet70.Dos70.Dos70 ();
```

The following code sets up the parameters:

```
d70.setFilePath (matchLocation);
d70.open (1);
…
```

The following code runs the batch file:

```
70.run ();
```

This statement will create an ImageFinder output file. The last step is to show the results:

Response.Redirect ("imagefinder/b1.htm").

28. ImageFinder Support Service Packages

ImageFinder for Windows is an off-the-shelf Application Software that enables System Integrators, Solution Developers, and Individuals to quickly test their own Image Recognition ideas.

ImageFinder for Dos is command-line software that enables System Integrators, Solution Developers, and Individuals to make a quick-and-dirty system integration to test their product prototypes and services.

TransApplet is .Net Class Library that enables System Integrators, Solution Developers, and Individuals to quickly add Image Recognition capability to their products and services.

Attrasoft **Transapplet Services** are designed to accelerate a company's path to deploy Image Recognition Solutions. Our structured Service Offerings help you to develop your products/services with the **ImageFinder** as a component.

28.1 What is Support Service?

The **TransApplet Annual Support rate** is the 20% of the project.

ImageFinder Support Service Packages are designed to help a person/company to understand the process of integrating Image Recognition technology into their product/service. From a large set of possible paths, a person/company must plan the most realistic execution to ensure the success of the project.

- **The focus of Support Service is to follow the right development path and shorten the learning curve for developers.**
- **Based on dozens of past projects, Attrasoft's development experience will specify the required work and map the shortest path to the system integration;**
- **Most importantly, Attrasoft Services might prevent you from following a wrong path, thus saving you from an unnecessary waste of resources, or even failure of the project.**

28.2 Process

Image Recognition Solution Development Procedure is:

(1) Set up the system (you can do that);
(2) Collect the data (you can do that);
(3) Preliminary Assessment via our off-the-shelf software, the **ImageFinder**.

- You should get Identification Rates ranging from 60% to 89%.
- The best rate, one of our customers (without any customization) was able to obtain, was an 89% Identification Rate.

- The off-the-shelf ImageFinder has 70 open parameters for users to adjust, which is the reason customers are only able to achieve Identification Rates ranging from 60% to 89%.
- The ImageFinder itself has 3,000+ internal parameters which users have no access to. Customization is the process of adjusting these 3,000+ internal parameters for your specific image type. This is where the high degree of accuracy (95% to 99.9%) for your specific image type is obtained.

(4) Feasibility Project via Customized Stand-alone Software.

- ATTRASOFT will develop a stand-alone software, which will address the special needs / requirements of your application.
- This will allow you to show your upper management the practicality of a larger project.

(5) Programming Library in .Net;

(6) System Integration to your Production Line;

(7) Licensing & Annual Support.

One-time License:	2% of the project.
Annual License:	0.6% of the project
Annual Support:	20% of the project. (First year will be required.)

28.3 What is a Feasibility Study?

Attrasoft Image Recognition technology offers a suite of products,

- **ImageFinder** (off-the-shelf),
- **ImageFinder for Dos,**
- **TransApplet** (Library version of the ImageFinder),
- **Customized-ImageFinder,**
- **Customized-TransApplet**),

that enables the success of projects a person/company will develop with Attrasoft's software components.

In general, a **Feasibility Study** is very valuable because experience from many past projects will be naturally deployed into your project. The very first question we encountered, and you will be asked when you justify your project to your company, is "Can this be done?" Very often, the answer is more than a simple "Yes"; a Feasibility Study will be necessary to respond this question, which is further divided into a:

- **Preliminary Assessment** with 200 of your images (i.e., 100 image pairs), and a
- **Feasibility Study** with 2,000 of your images (i.e., 1,000 image pairs).

28.4 TransApplet Support

If the off-the-shelf product satisfies your needs, your path will be:

(1) Set up the system (you can do that);
(2) Collect the data (you can do that);
(3) Purchase the **ImageFinder** and make a Preliminary Assessment;
(4) Purchase **TransApplet**;
(5) System Integration Support;
(6) License and Annual Support (20% of the project).

The most likely path you will go through, which requires some customization, is:

(1) Set up the system (you can do that);
(2) Collect the data (you can do that);
(3) Purchase the **ImageFinder** and make a Preliminary Assessment;
(4) Purchase Customization of the standalone software;
(5) Purchase Customized **TransApplet**;
(6) System Integration;
(7) License and Annual Support (20% of the project).

Step (4), Customization deals with problems like:

- Reducing the operation Complexity via Attrasoft tuning the 3000+ internal parameters to one specific image type;
- Speed Optimization;
- Internal Structure Optimization;
- Graphical User Interface Customization;
- Database other that Microsoft Access;
- Database Interface;
- Video Formats other than .avi files;
- New Image Preprocessing Filters;
- Customized Filters;
- Programming Library;
- Specific Symmetries or Combination of Symmetries;
- Attrasoft can implement any symmetry (or combination of symmetries) which can be described by mathematics;
- Further refinement Tuning for small image segments;
- Fine Tuning of the Neural Parameters;
- Digital Image Database (Combine **ImageFinder** with Database);
- Image Formats other than jpg and gif;
- Counting objects which are NOT physically separated;
- Reducing all images by the same amount without distortion to 100x100;
- Internet Image Search Engines;
- Multi-layers of image matching;
- Web Interface (solutions that will provide users with a searchable database using a web interface);

- Other Specific Needs.