

## **A Visual Programming Environment for Machine Vision Engineers**

Paul F Whelan

Vision Systems Group  
School of Electronic Engineering,  
Dublin City University,  
Dublin 9, Ireland.

Ph: +353 1 700 5489  
Fax: +353 1 700 5508  
*paul.whelan@eeng.dcu.ie*

and

Robert Sadleir

Vision Systems Group  
School of Electronic Engineering,  
Dublin City University,  
Dublin 9, Ireland.

Ph: +353 1 700 8592  
Fax: +353 1 700 5508  
*robert.sadleir@dcu.ie*

July 2004

## **A Visual Programming Environment for Machine Vision Engineers**

### **Abstract:**

This paper will detail a free image analysis and software development environment for machine vision application development. The environment provides high-level access to over 300 image manipulation, processing and analysis algorithms through a well-defined and easy to use graphical interface. Users can extend the core library using the developer's interface via a plug-in which features automatic source code generation, compilation with full error feedback and dynamic algorithm updates. The paper will also discuss key issues associated with the environment and outline the advantages in adopting such a system for machine vision application development.

**Keywords:** Machine Vision, Image Analysis, Visual Programming, Development Environment.

### **Introduction**

For many novices to the world of machine vision, the development of automated vision solutions may seem a relatively easy task as it only requires a computer to understand basic elements such as shape, colour and texture? Of course this is not the case. Extracting useful information from images is a difficult task and as such requires a flexible machine vision application development environment. The visual programming environment outlined in this paper (called NeatVision) is one such system (NeatVision, 2004). It aims to provide novice and experienced machine vision engineer's with access to a Java based multi-platform development system. The

NeatVision environment provides an intuitive interface which is achieved using a drag and drop block diagram approach. Each image processing operation is represented by a graphical block with inputs and outputs that can be interconnected, edited and deleted as required. NeatVision (Version 2.0) is available free of charge and can be downloaded directly via the Internet (NeatVision, 2004)

### **NeatVision: An Interactive Development Environment**

NeatVision is just one example of a visual programming development environment for machine vision, other notable examples include commercial programmes such as Khoros (Khoros, 2004) and WiT (Wit, 2004). Visual programming involves defining variables, specifying operations, which are to be performed on these variables and their derivatives in order to perform a specific task. This is achieved by creating a structured flow of data using branching, looping and conditional processing. Traditionally computer programs have been written using textual programming languages. Quiet often such programmes process data in a complex fashion leading to a situation whereby the data paths and the overall structure of the program cannot be easily identified from the textual description. This can make it very difficult to appreciate the relationship between the source code and the functionality that it represents. Although the programmer specifies the data flow in a visual program, the order in which the components execute is defined by the availability of data. Conditional processing concepts are supported in the visual domain by using dedicated flow control components. The main disadvantages of existing visual programming environments includes their cost, lack of cross platform support and the fact that many of them tend to be focused on image processing rather than image

analysis applications (the latter must be considered a key element of any practical machine vision application).

Text based programming languages such as MATLAB (Matlab, 2004) can be a powerful alternative to visual programming. In addition to the disadvantages outlined with respect to the visual programming languages, text based approaches require the user to have a higher level of programming skills when compared to visual programming environments. Text based interactive environments are generally more suitable to experienced users, in fact experienced users can become frustrated by the visual programming environment as complex programmes can take longer to develop. Hence our aim is to produce a suitable environment for those new to machine vision while retaining the flexibility of program design for the more experienced users. Based on our review of existing text and visual programming based machine vision development environments, the key criteria necessary are outlined below:

- **Multi-platform:** The development environment must be able to run on a wide range of computer platforms.
- **Focused on machine vision engineering:** The environment should contain a wide range of image processing and analysis techniques necessary to implement practical machine vision engineering applications.
- **Easy to use:** It should allow users to concentrate on the design of machine vision solutions, as opposed to emphasizing the programming task.
- **Upgradeable:** The environment must contain a mechanism to allow users to develop custom vision modules.

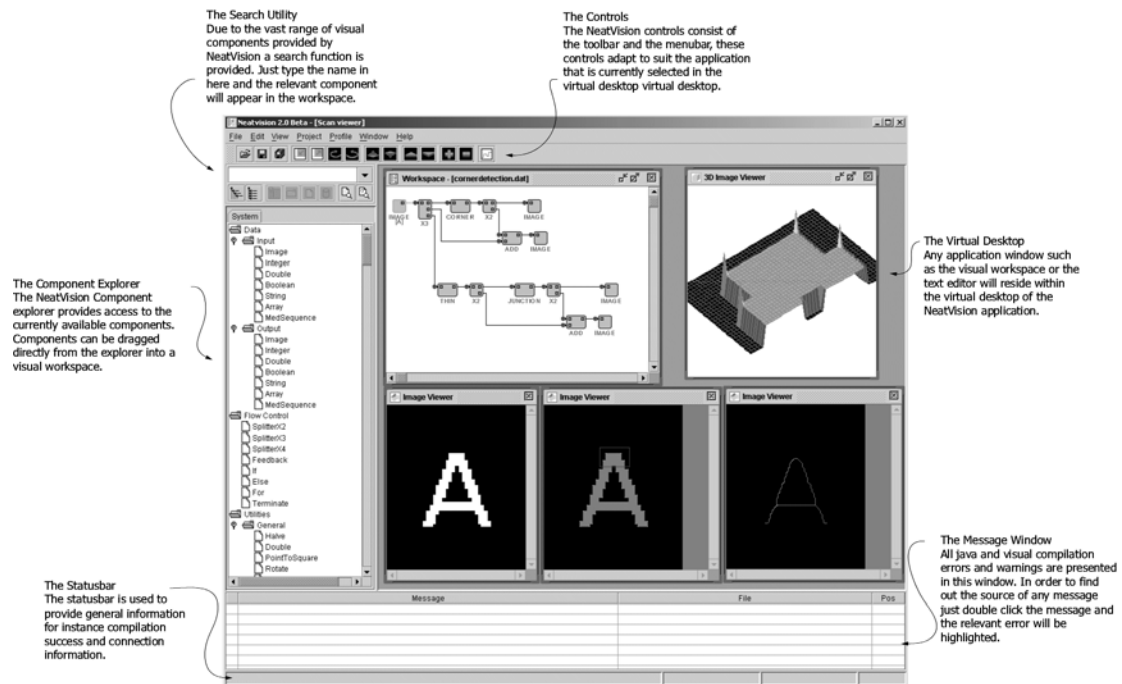
A visual program can be created by defining input data using the input components, then implementing the desired algorithm using the processing and flow control components. The data flow is specified by creating interconnections between the components. The program can be completed by adding output components to view the data resulting from the algorithm execution. Details on the design of the NeatVision development environment appear elsewhere (Whelan and Molloy, 2000).

### **NeatVisions Graphical User Interface (GUI)**

The NeatVision GUI (Figure 1) consists primarily of a workspace where the processing blocks reside. The processing blocks represent the functionality available to the user. Support is provided for the positioning of blocks around the workspace, the creation and registration of interconnections between blocks. The lines connecting each block represent the path of the data through the system. Some of the blocks can generate child windows, which can be used for viewing outputs, setting parameters or selecting areas of interest from an image. If each block is thought of as a function, then the application can be thought of as a visual programming language. The inputs to each block correspond to the arguments of a function and the outputs from a block correspond to the return values. The advantage of this approach is that a block can return more than one value without the added complexity of using C-style pointers. In addition, the path of data through a visual program can be dictated using special flow control components. A visual program can range in complexity from three components upwards and is limited only by the availability of free memory.

As each block is processed it is highlighted (in green) to illustrate that it is active. This allows users to see the relevant speeds of each segment of the parallel data streams within a visual program. This can help identify potential processing bottlenecks within the workspace allowing for a more efficient balanced design. The colour coding of the blocks data connection type and its status also aids in the design process. To aid user operation each data connection has two colour coded properties, namely the block *data type* and *connection status*. NeatVision currently supports eight data types, i.e. Image (red), Integer / Array data (green), Double precision Floating point data (blue), Boolean data (orange), String data (pink), Fourier data (light blue), Coordinate data (purple) and Undefined data (black). The other connection property relates to its status. There are three main states for a connection, *connected (green)*, *disconnected (red)* and *disconnected but using default value (orange)*.

This approach provides a fast and simple alternative to conventional text based programming, while still providing much of the power and flexibility. The visual workspace can be compiled and executed as with a conventional programming language. Errors and warnings are generated depending on the situation. There is currently support for 13 input graphics file formats and 11 output formats. System parameters can be adjusted and the system may be reset and executed again until the desired response is obtained. At any stage blocks may be added or removed from the system. NeatVision also contains a built in web browser to allow easy access to online notes and support tools.



**Figure 1:** Key features of the NeatVision environment.

## Design Details

NeatVision is designed to work at two levels. The *user* level allows the design of imaging solutions within the visual programming environment using NeatVision's core set of functions. NeatVision (Version 2.0) contains 300 image manipulation, processing and analysis functions, ranging from pixel manipulation to colour image analysis to data visualisation. To aid novice users, a full introductory tutorial and some sample programmes can be found on the NeatVision website. A brief description of the main system components is given below:

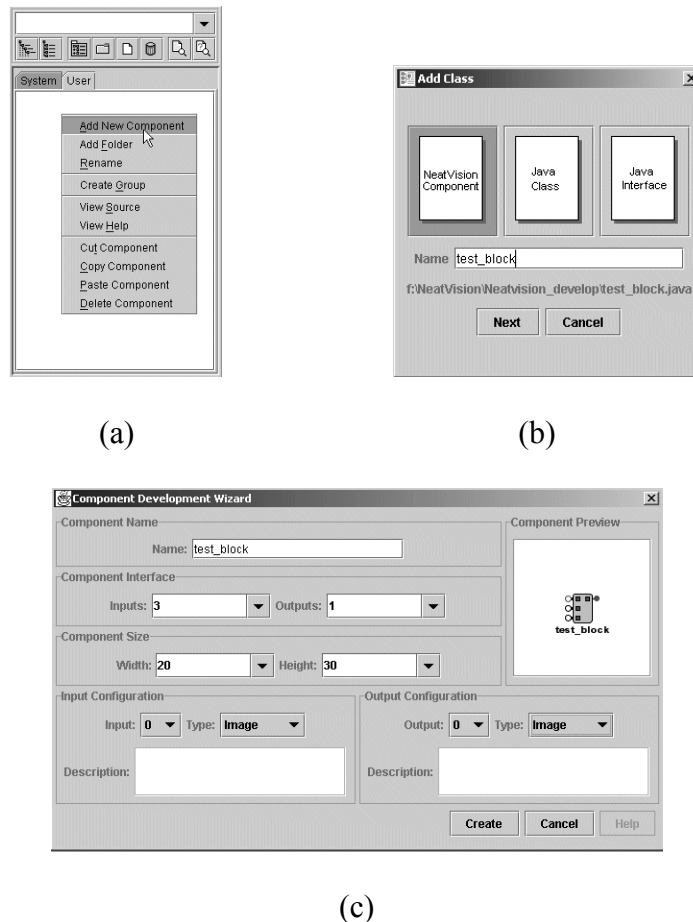
- **Data types:** Image, integer, double, string, Boolean, array, medical image sequences.
- **Flow control:** Path splitting, feedback, if (else), for loop.

- **Utilities:** Rotation, pixel manipulation, resize, URL control, additive noise generators, region of interest, masking operations.
- **Arithmetic operators:** Add, subtract, multiply, divide, logical operators.
- **Histogram:** General histogram analysis algorithms, local equalization.
- **Image Processing:** Look-up tables (LUT), threshold, contrast manipulation.
- **Neighborhood based filtering:** Lowpass, median, sharpen, DOLPS, convolution.
- **Edge detection:** Roberts, Laplacian, Sobel, zero crossing, Canny
- **Edge features:** Line/arc fitting, edge labelling and linking.
- **Analysis:** Thinning, binary detection, blob analysis, labelling, shape feature measures, bounding regions, grey scale corner detectors.
- **Clustering:** K-means (grey scale and colour), unsupervised colour clustering.
- **Image transforms:** Hough (line and circle), Medial Axis, DCT, Cooccurrence, Fourier, distance transforms.
- **Morphology:** Several morphological operators, including erosion, dilation, opening, closing, top-hat, hit-and-miss, watershed.
- **Colour:** Colour space conversion algorithms, RGB, HSI, XYZ, YIQ, Lab.
- **3D Volume:** 3D Operators (thinning, Sobel, threshold, labelling), intensity projections, rendering engine (wire frame, flat, Gouraud, Phong).
- **Low Level:** Pixel level operators; get pixel value, set pixel value and basic shape generation.
- **String:** String operators, object addition, to upper case and to lower case.
- **Maths:** An extensive range of numerical operators and utilities, including constants and random number generation.



- **JAI Colour:** Colour algorithms implemented using JAI (Java Advance Imaging (JAI, 2003)), operators, processing, filters and edge detectors.

At the more advanced *developers* level, NeatVision allows experienced machine vision engineers to develop and integrate their own functionality (Figure 2) through the development and integration of new image processing/analysis modules. Readers should refer to the online NeatVision developers guide for more details on developing and integrating custom machine vision applications (DevGuide, 2004).



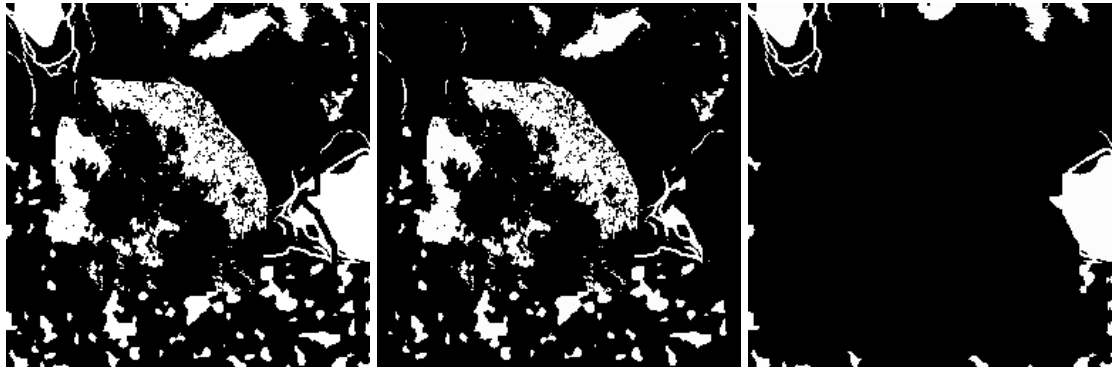
**Figure 2:** NeatVision component development wizard. (a) Select *Add New Component* from the popup menu of the *user* area of the component explorer. (b) Select component to be added. (c) Define the component skeleton (i.e. the number and

type of inputs and outputs for the associate block). This generates the necessary component wrapping code.

## **Sample Applications**

NeatVision provides an image analysis software development environment that can work at several levels. For example, at a relatively low level individual pixels can be manipulated. Alternatively, NeatVision's built in functionality can be used to generate solutions to complex machine vision problems. This section outlines two basic sample applications, beginning with a simple morphological based task.

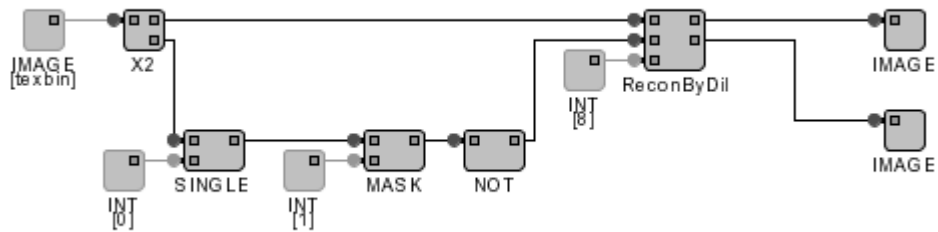
Figure 3 illustrates how the *reconstruction by dilation* morphological operator can be used to remove or detect objects that are touching the binary image border. This would be typically used in microscopic imaging analysis (such as microstructure or cell counting applications) whereby we are only interested in objects within the field of view that do not touch the boundary. Figure 4 illustrates how NeatVision can be used to isolate defects in the center panel of a crown bottle top. The basic approach outlined here involves thresholding the image prior to removing any isolated pixels. The largest blob (white connected region) is then isolated prior to erosion to remove edge artefacts. The convex hulls bay regions are then isolated and eroded (again to remove unwanted edge artefacts) prior to incorporation into the original image.



(a)

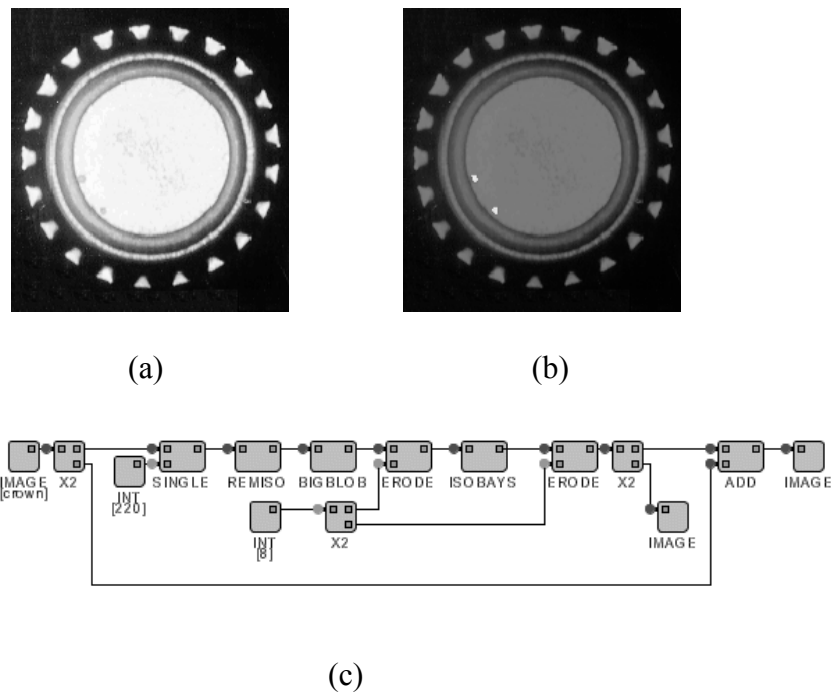
(b)

(c)



(d)

**Figure 3:** Removal of boundary objects using reconstruction by dilation. (a) Original image. (b) Boundary object removal. (c) Detected boundary objects. (d) Associated visual workspace.



**Figure 4:** Bottle top inspection. (a) Input image. (b) Output image in which the defects are highlighted in white. (c) The NeatVision visual program.

## Conclusions

While there is more to the design of a commercial machine vision system than the algorithmic development process (i.e. sensor, lighting and mechanical interface issues are equally important design tasks), a tool such as NeatVision has key advantages during the initial research stages. By using the built in functionality of NeatVision, it removes the requirement for the design engineer to develop custom code just to evaluate a potential algorithmic approach. Similarly the open cross-platform environment presented by this package makes it ideal for the teaching and exchange of ideas in a machine vision educational setting. The cross-platform nature of

NeatVision is especially convenient in the online educational environment, as it does not tie the user to a specific computing platform. One of the authors (PFW) has been successfully using earlier versions of NeatVision since 1995 in such an online educational role, where it has formed the main design platform for an online postgraduate and continuing educational programme in machine vision (EE544 (2004), Whelan (1997)).

In summary, NeatVision was designed to allow novice and experienced users to focus on the machine vision design task rather than concerns about the subtlety of a given programming language. It allows machine vision engineers to implement their ideas in a dynamic and straightforward manner. NeatVision standard and developers versions are freely available via the Internet and are capable of running on a wide range of computer platforms (e.g. Windows, Solaris, Linux).

## References

NeatVision (2004), "NeatVision: Image Analysis and Software Development Environment", Available (MARCH) <http://www.neatvision.com>

Khoros (2004), "Khoros: Khoral Research, Inc", Available (MARCH) <http://www.khoral.com>

WiT (2004), "WiT: Logical Vision", Available (MARCH) <http://www.logicalvision.com/>

Matlab (2004), "MathWorks: Matlab", Available (MARCH) <http://www.mathworks.com/index.shtml>

Whelan, P.F. and Molloy, D (2000), "Machine Vision Algorithms in Java: Techniques and Implementation", Springer-Verlag, London

JAI (2004), "The Java Advanced Imaging (API)", Available (MARCH) <http://java.sun.com/products/java-media/jai>

DevGuide (2004), "NeatVision: Developers Guide", Available (MARCH) <http://neatvision.eeng.dcu.ie/developer.html>

EE544 (2004), "EE544: Computer and Machine Vision", Available (MARCH)  
<http://www.eeng.dcu.ie/courses/post/modules/ee544.html>

Whelan, P.F. (1997), "Remote access to continuing engineering education - RACeE",  
IEE Engineering Science and Education Journal, 6(5), pp 205-211.

## **Diagrams**

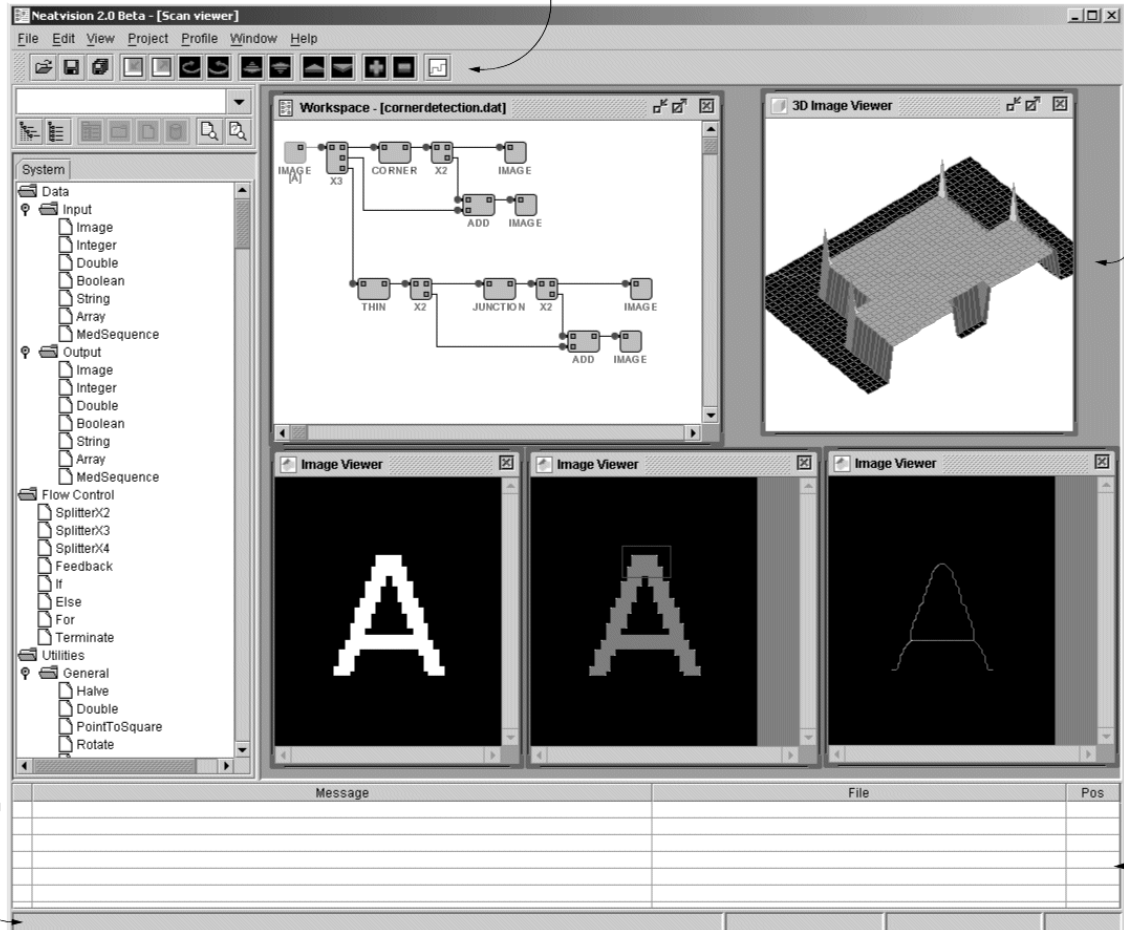


**The Search Utility**  
 Due to the vast range of visual components provided by NeatVision a search function is provided. Just type the name in here and the relevant component will appear in the workspace.

**The Controls**  
 The NeatVision controls consist of the toolbar and the menubar, these controls adapt to suit the application that is currently selected in the virtual desktop virtual desktop.

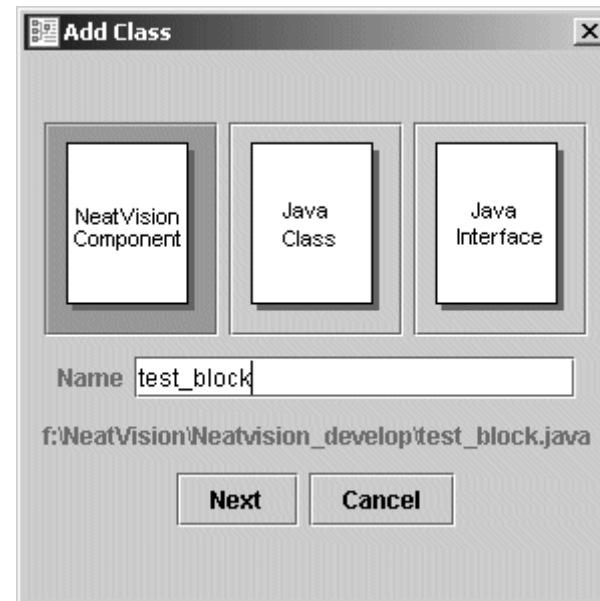
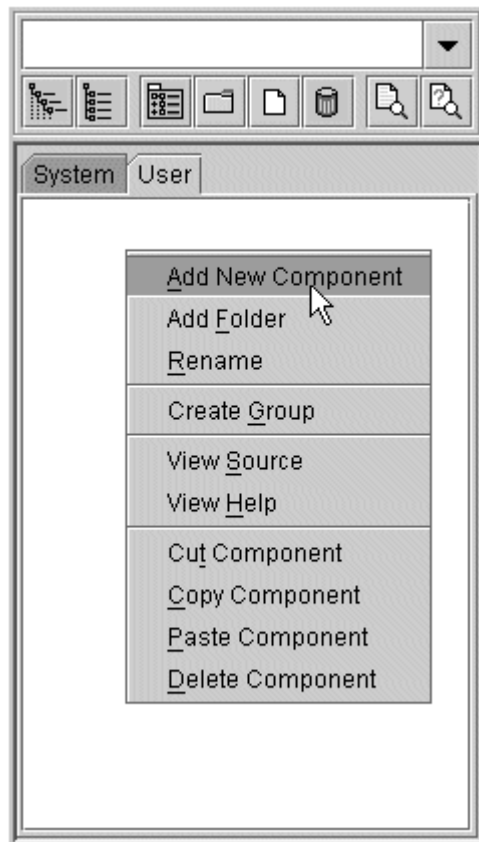
**The Component Explorer**  
 The NeatVision Component explorer provides access to the currently available components. Components can be dragged directly from the explorer into a visual workspace.

**The Statusbar**  
 The statusbar is used to provide general information for instance compilation success and connection information.



**The Virtual Desktop**  
 Any application window such as the visual workspace or the text editor will reside within the virtual desktop of the NeatVision application.

**The Message Window**  
 All java and visual compilation errors and warnings are presented in this window. In order to find out the source of any message just double click the message and the relevant error will be highlighted.

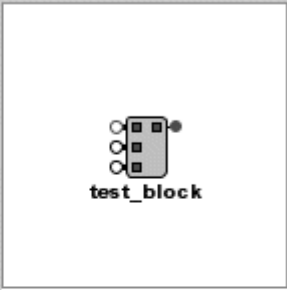


**Component Development Wizard** [X]

**Component Name**  
Name:

**Component Interface**  
Inputs:   Outputs:

**Component Size**  
Width:   Height:

**Component Preview**  
  
test\_block

**Input Configuration**  
Input:   Type:    
Description:

**Output Configuration**  
Output:   Type:    
Description:

