

User's Manual

ACTI✓3BARCODE

Any Position, any Orientation, any Background

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Visual Basic, Microsoft .NET, Visual C++, Visual C#, and ActiveX are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

Copyright © 2000-2008 by MVTec Software GmbH, München, Germany



Edition 1	November 2000	(ActivVisionTools 1.0)
Edition 2	April 2001	(ActivVisionTools 1.3)
Edition 3	September 2001	(ActivVisionTools 2.0)
Edition 4	November 2002	(ActivVisionTools 2.1)
Edition 5	January 2005	(ActivVisionTools 3.0)
Edition 6	February 2006	(ActivVisionTools 3.1)
Edition 7	May 2008	(ActivVisionTools 3.2)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

More information about ActivVisionTools can be found at:

<http://www.activ-vision-tools.com>

How to Read This Manual

This manual explains how to use ActivBarcode to read bar codes. It describes the functionality of ActivBarcode and its cooperation with other ActivVisionTools with Visual Basic examples. Before reading this manual, we recommend to read the manual [Getting Started with ActivVisionTools](#), which introduces the basic concepts of ActivVisionTools and the [User's Manual for ActivView](#) to learn how to load and display images.

To follow the examples actively, first install and configure ActivVisionTools as described in the manual [Getting Started with ActivVisionTools](#). For each example in this manual, there is a corresponding Visual Basic project; these projects can be found in the subdirectory `examples\manuals\activbarcode` of the ActivVisionTools base directory you selected during the installation (default: `C:\Program Files\MVTec\ActivVisionTools`). Of course, you can also create your own Visual Basic projects from scratch.

We recommend to **create a private copy of the example projects** because by experimenting with the projects, you also change their *state*, which is then automatically stored in the so-called description files (extension `.dsc`) by ActivVisionTools. Of course, you can restore the state of a project by retrieving the corresponding description file from the CD.



Contents

1	About ActivBarcode	1
1.1	Introducing ActivBarcode	2
1.2	The Sub-Tools of ActivBarcode	3
2	Using ActivBarcode	5
2.1	Specifying Regions of Interest	6
2.2	Reading Bar Codes	8
2.3	Tuning Image Processing	10
3	Combining ActivBarcode with other ActivVisionTools	13
3.1	Converting Positions to Other Units	14
3.2	Evaluating Results	16
3.3	Output of Results	18
4	Tips & Tricks	21
4.1	Adapting the Display of Results	22
4.2	Configuring the Two Execution Modes	24
4.3	Accessing Results Via the Programming Interface	26

Chapter 1

About ActivBarcode

This chapter will introduce you to the features and the basic concepts of ActivBarcode. It gives an overview about ActivBarcode's *master tool* and its *support tool*, which are described in more detail in [chapter 2](#) on page 5 and [chapter 3](#) on page 13.

1.1	Introducing ActivBarcode	2
1.2	The Sub-Tools of ActivBarcode	3

1.1 Introducing ActivBarcode

With the help of ActivBarcode, you can read various common bar codes: EAN 13, EAN 8, UPC-A, UPC-E (all of them also in the variants Add-On 2 and Add-On 5), 2/5 Industrial, 2/5 Interleaved, Codabar, Code 39, Code 93, Code 128, PharmaCode, RSS-14, RSS-14 Stacked, RSS-14 Stacked Omnidirectional (all RSS-14 codes also as Truncated), RSS Limited, RSS Expanded, RSS Expanded Stacked (all RSS codes also as Composite). You can output the code together with additional results, e.g., the position of the bar code in the image. A bar code can appear in any position and any orientation. Furthermore, ActivBarcode extracts bar codes even from difficult backgrounds and in case of varying illumination. To speed up the extraction, you can specify regions of interest (ROIs), which do not need to fit the bar code tightly and can also contain more than one bar code.

Image Processing Steps

In the standard case, you need not worry about the image processing taking place inside ActivBarcode. However, in more complex situations it might become necessary to tune certain steps of the image processing, which can be accomplished easily using a *support tool* of ActivBarcode. In the following, we therefore briefly describe how ActivBarcode extracts a bar code and introduce the corresponding parameters. How to tune these parameters is described in [section 2.3](#) on page 10.

First, candidate regions have to be found. These are regions where a set of lines or edges exhibit a similar direction. To be accepted as candidate elements, the lines have to be of a certain *size* (thickness). A certain *variation of orientation* along the elements is allowed. Then, for all the candidates a set of scanlines is created. Within each scanline a sequence of bars and spaces is searched for that could be valid for a specific symbology. If a valid sequence is found, the respective bar code symbol region and the decoded string is returned.

Results

The result of ActivBarcode is, of course, the content of the extracted bar code. Furthermore, ActivBarcode returns the position of the bar code in the image (see [figure 1.1](#)). The results of ActivBarcode can be further evaluated using ActivDecision, before you output them via ActivFile, ActivSerial, or ActivDigitalIO; furthermore, you can access results via the programming interface (see [section 4.3](#) on page 26).

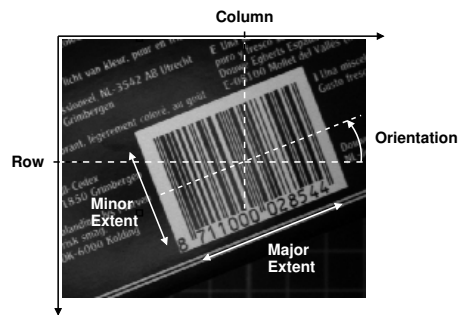


Figure 1.1: The position of a bar code in the image.

1.2 The Sub-Tools of ActivBarcode

Besides its *master tool*, ActivBarcode provides one *support tool*. In [figure 1.2](#), both are depicted together with other ActivVisionTools that you will use in a typical ActivBarcode application.

AVTBarcode1D is the *master tool* of ActivBarcode. In it, you specify the expected type of bar code, which results are to be passed on to other ActivVisionTools, and what is to be displayed by ActivView.



How to use AVTBarcode1D is described in more detail in [section 2.2](#) on page 8.

AVTBarcode1DParameters is a *support tool* of ActivBarcode. With it, you can tune the image processing steps.



How to use AVTBarcode1DParameters is described in [section 2.3](#) on page 10.

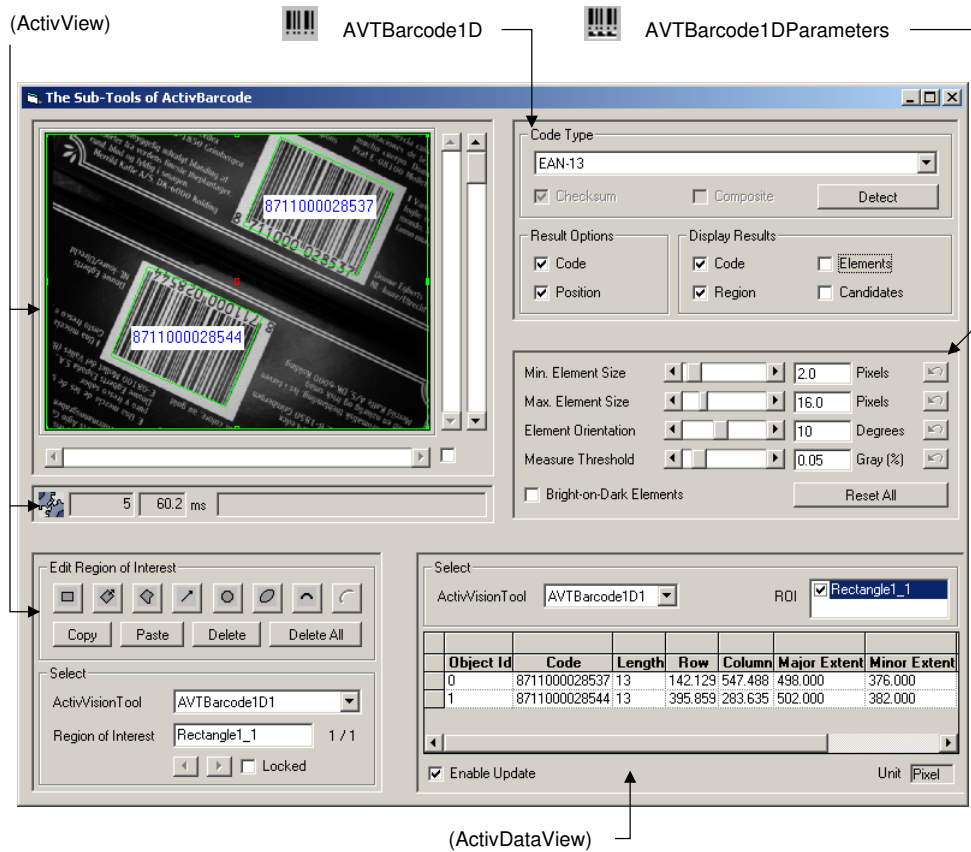


Figure 1.2: The sub-tools of ActivBarcode together with suitable other tools.

Chapter 2

Using ActivBarcode

This chapter will explain how to use ActivBarcode to read various types of bar codes and how to “tune” the underlying image processing.

The corresponding Visual Basic projects show how to successfully read bar codes.




2.1	Specifying Regions of Interest	6
2.2	Reading Bar Codes	8
2.3	Tuning Image Processing	10

2.1 Specifying Regions of Interest Using


ActivBarcode lets you choose between different shapes of ROIs, e.g., arbitrarily oriented rectangles or ellipses. You create an ROI using `AVTViewROI`, which is a *support tool* of `ActivView`.

Visual Basic Example

Preparation for the following example:

- ❑ Open the project `rois\barcode_rois.vbp`. Alternatively, create a new project and place the following tools on the form (in parentheses the icon you have to double-click with the left mouse button): `AVTView` () , `AVTViewROI` () , and `AVTBarcode1D` () .
- ❑ Execute the application (`Run > Start` or via the corresponding button). Open `AVTViewFG` by clicking into `AVTView` with the right mouse button and selecting `Image Acquisition` in the popup menu. Load the image `barcode\code39_01` via the combo box `Input File`.

The following steps are visualized in [figure 2.1](#).

- ① First, you have to tell `AVTViewROI` to create an ROI for `ActivBarcode` by selecting the corresponding entry in the combo box `ActivVisionTool`. In this box, all `ActivVisionTools` are listed that have been placed upon the form. By default, the tools are referenced by the name of the corresponding `ActiveX` control plus a counter to distinguish between multiple instances of a control on the form, e.g. `AVTBarcode1D1`.
- ② Next, select the desired shape of the ROI, for example a rectangle.
- ③ To draw the ROI, move the mouse in the image while keeping the left mouse button pressed. Please experiment at this point with the different shapes. The creation of a polygonal ROI () is more complex: By the first mouse movement, the first side of the polygon is created. You can add a new corner point by clicking on the line with the left mouse button; when you drag it by keeping the mouse button pressed, new polygon sides are created, where you can again add new corner points. To delete a corner point, drag it onto a neighboring corner point.

Note that if you use a line- or arc-shaped ROI, the bar code is searched for only along this ROI. If the bar code moves from image to image, we recommend to use `ActivAlignment` (see the [User's Manual for ActivAlignment](#)) to move the ROI accordingly.

- ④ You can now move the ROI by dragging its pick point in the middle. By dragging the outer pick points you modify its shape. Again, please experiment to get familiar with the ROIs. Note that for a polygonal ROI the “middle point” appears at the center of gravity of the ROI and therefore changes whenever the polygon is modified; besides, it

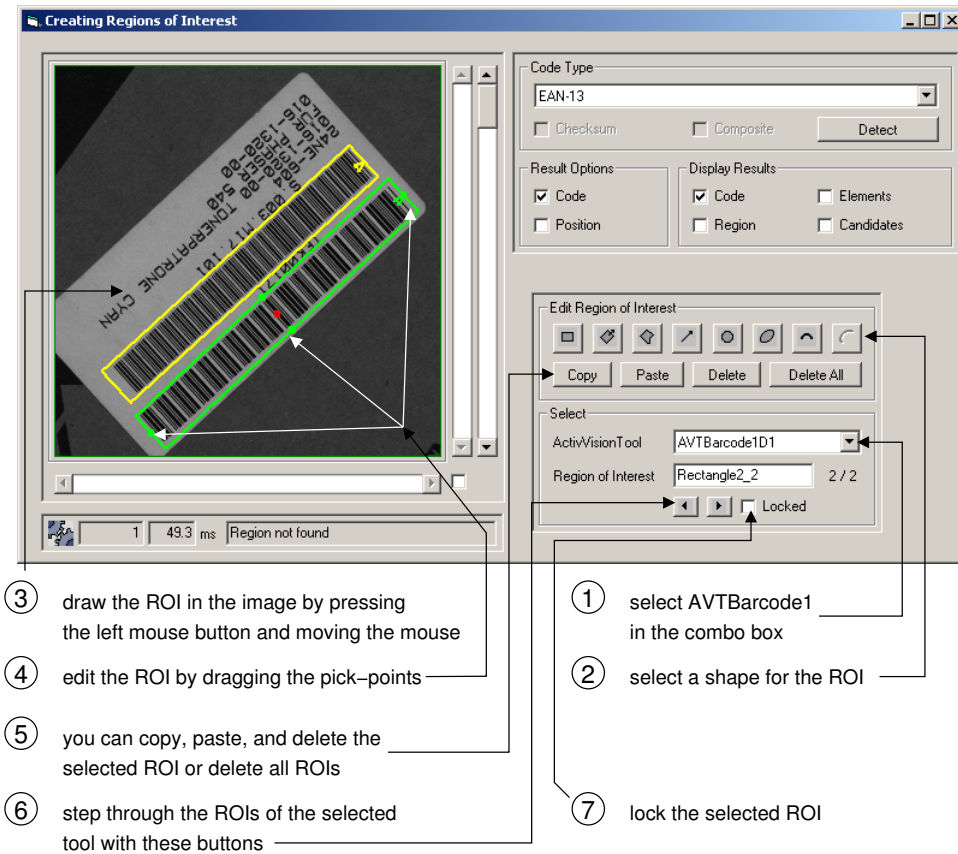




Figure 2.1: Creating a region of interest.

may even lie outside the ROI for concave polygons.

- ⑤ To create a second ROI, repeat step ②. You can also copy, paste, and delete the selected ROI or delete all ROIs.
- ⑥ ROIs can be selected by clicking next to it in the image. Alternatively, use the two arrow buttons   to step through all the ROIs of the selected tool.
- ⑦ By checking Locked you can lock the selected ROI and thus prevent it from accidental editing.



If there are multiple bar codes in an image, you can either create one ROI that contains them all, or multiple ROIs containing one or more bar codes.

2.2 Reading Bar Codes Using

While you are positioning an ROI on the image, ActivBarcode already tries to find a bar code inside it. Using AVTBarcode1D, you can specify the expected type of the bar code and configure the output behavior of ActivBarcode.

Visual Basic Example

Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTViewStatus and AVTDataView by double-clicking  and . You may delete AVTViewROI, as it can be opened at run time by clicking into AVTView with the right mouse button and selecting Region of Interest. Otherwise, open the project reading\barcode_reading.vbp.
- Execute the application (Run > Start or via the corresponding button) and open AVTViewFG by clicking into AVTView with the right mouse button and selecting Image Acquisition in the popup menu. Select the image barcode\code39_01 in the combo box Input File.

The following steps are visualized in [figure 2.2](#).

- ① If you know the type of the bar code in the image, you can specify it in the combo box Code Type. In the example image, both codes are *Code 39* bar codes. If you do not know the type of the bar code, you can press `Detect`. ActivBarcode will then try to determine the type of the bar code by itself.
- ② If the ROI is correctly placed around the bar code, the read code should now be displayed in AVTView. If AVTBarcode1D fails to extract a bar code, AVTViewStatus displays a corresponding message. To decode inverted bar codes (bright elements on a dark background), see [section 2.3](#) on page 10.
- ③ With the help of check boxes, you can specify what is to be displayed on the image: the decoded *text* of the bar code, the image *region* corresponding to the bar code, or the edges extracted along the bar code *elements*. [Section 4.1](#) on page 22 explains how to change the appearance of the displayed results.
- ④ Besides the extracted bar code, ActivBarcode can output additional information (also called *features*), e.g., the position of the bar code.
- ⑤ To view the calculated features, enable the update of results in AVTDataView and select the tool whose results to display in the combo box ActivVisionTool. If there is only one tool, it is selected automatically.

① select the bar code type or let ActivBarcode detect the bar code type

② extracted codes are displayed, otherwise an error message appears

③ display certain results in the image

④ choose additional result types

⑤ enable the update of results and select AVTBarcode1 in the combo box

⑥ select the ROIs, whose results are then displayed

Reading Bar Codes and Selecting Results

Code Type: Code 39

Checksum Composite

Result Options: Code Position

Display Results: Code Elements Region Candidates

1 | 101.8 ms

Select: ActivVisionTool AVTBarcode1D1 ROI: Rectangle2_1 Rectangle2_2

ROI Id	Object Id	Code	Length	Row	Column	Major Extent	Minor Extent	Orientation
Rectangle2_1_0		S.003.M17.101	19	337.418	309.037	1066.000	180.000	-138.505
Rectangle2_2_0		TEK00171	16	408.445	369.261	1112.000	180.000	-138.280

Enable Update Unit Pixel

ActivBarcode

Figure 2.2: Reading bar codes and selecting results.

- ⑥ Automatically, a list of the ROIs of AVTBarcode1D1 appears below. Select the ROIs whose results you want to examine by checking their box. Their results are then displayed in a table, the columns corresponding to the selected features.


2.3 Tuning Image Processing Using

The image processing methods inside ActivBarcode are parameterized in such a way that they work well for standard images of bar codes. If you encounter problems, e.g., when working with uneven or highly reflective surfaces, you can tune the image processing steps described in [section 1.1](#) on page 2 using `AVTBarcode1DParameters`.


It is required to use `AVTBarcode1DParameters` if the bar codes appear inverted (bright elements on a dark background). Make sure `Bright-on-Dark Elements` is checked in this case.

Visual Basic Example

Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add `AVTBarcode1DParameters` by double-clicking  .
Otherwise, open the project `tuning\barcode_tuning.vbp`.
- Execute the application (Run > Start or via the corresponding button) and load the image `barcode\rss14truncated_01`.

The following steps are visualized in [figure 2.3](#).

- ① You can change the value of a parameter via the corresponding slider.
- ② To reset a parameter to its default value click  ; note that only the buttons of those parameters are enabled, whose values were changed. You can reset all parameters at once by clicking `Reset All`.
- ③ `Min Element Size` and `Max Element Size`: These parameters influence candidate estimation. Only regions with elements in the specified range will be considered as possible candidates. Click `Candidates` in `AVTBarcode1D` to visualize the effect. Here, the value for `Min Element Size` has to be decreased for the bar code to be read.
- ④ Decrease `Max Element Size` to exclude edges that happen to be close to the bar code. Here, the large candidate region becomes smaller and is finally split in two separate regions. A benefit of tuning these parameters is an increase in performance.
- ⑤ `Element Orientation`: Determines the accepted orientation range of bar code candidates with respect to the reading direction, i.e., when using directed ROIs. Also specifies the allowed “curvature” of the elements of a bar code with respect to each other.
- ⑥ `Measure Threshold`: Sets the threshold for the detection of edges in the bar code region. For bar codes with very low contrast or noisy images this value has to be increased.

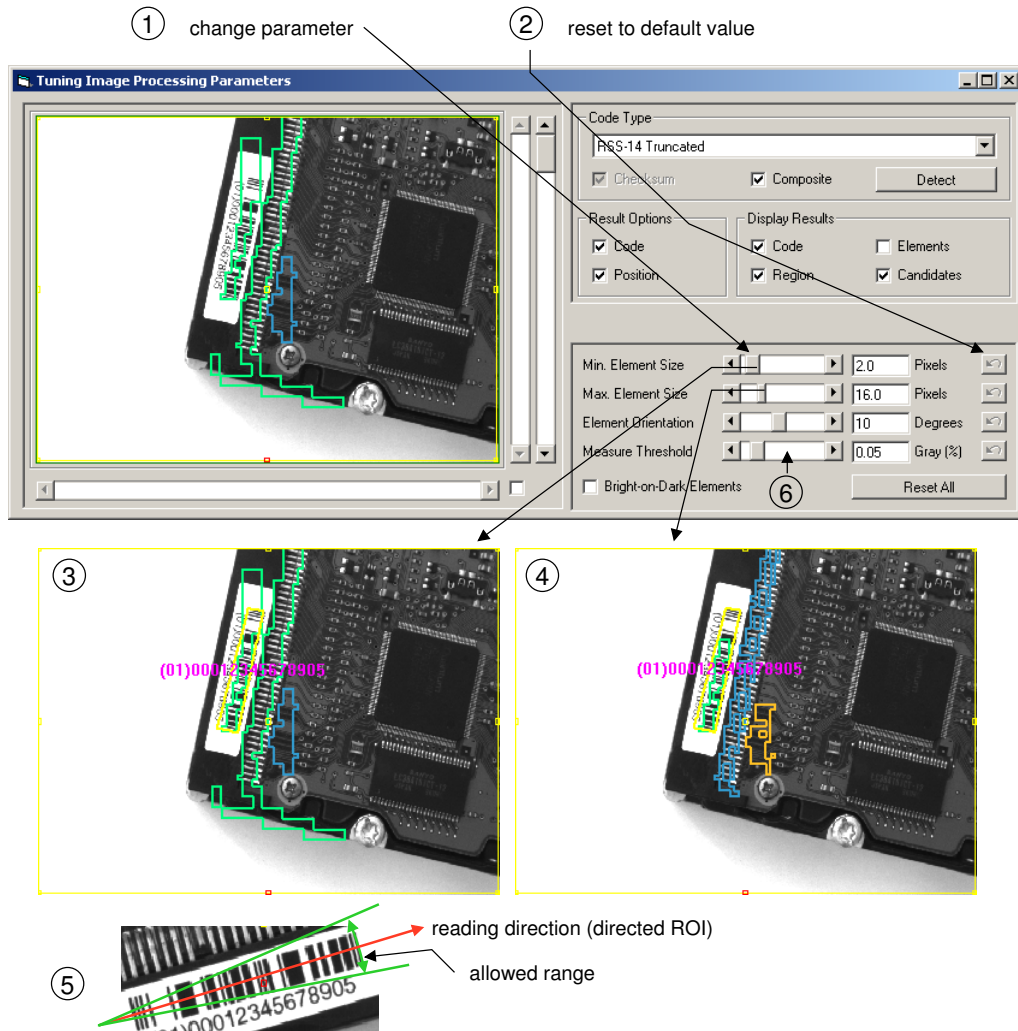


Figure 2.3: Tuning image processing parameters.

Chapter 3

Combining ActivBarcode with other ActivVisionTools

While the previous chapter explained how to read bar codes, this chapter focuses on how to further evaluate and output the results using other ActivVisionTools. How to access results and evaluations via the programming interface is described in [chapter 4](#) on page 21.

In the corresponding Visual Basic projects, the task is to read and evaluate a sequence of bar codes.

3.1	Converting Positions to Other Units	14
3.2	Evaluating Results	16
3.3	Output of Results	18




3.1 Converting Results to Other Units Using

Up to now, the position of a bar code was measured in image coordinates, i.e., pixels. Using `AVTViewCalibration`, you can convert this information into other units. Note that for a more accurate calibration you should employ `ActivGeoCalib`.


The main idea behind `AVTViewCalibration` is that the user draws a line in the image and tells the `ActivVisionTools` the length of this line is in a certain unit. From this information, `AVTViewCalibration` calculates the size of a pixel (i.e., its height as square pixels are assumed) in this unit, which in its turn can be used to convert measurements from pixels into the new unit. Note that this conversion only works if the observed objects lie in the same plane, i.e., at the same distance from the camera, as the line whose length was specified.

Visual Basic Example

Preparation for the following example:

- If you worked on the example in the previous chapter, you may continue using this project. At design time, add `AVTViewROI`, `AVTViewCalibration`, and `AVTDataView` to the form by double-clicking , , and  with the left mouse button. You may delete `AVTBarcode1DParameters`, which can be opened at run time by clicking into `AVTBarcode1D` with the right mouse button and selecting `Parameters`.
Otherwise, open the project `units\barcode_units.vbp`.
- Execute the application and open `AVTViewFG` by clicking into `AVTView` with the right mouse button and selecting `Image Acquisition`. Load the image `barcode\ean13_04`.

The following steps are visualized in [figure 3.1](#) (not shown: `AVTBarcode1D`).

- ① You create the line of known length using `AVTViewROI`. First, select `AVTViewCalibration1` in the combo box `ActivVisionTool`.
- ② To start creating the line, click  .
- ③ To draw the line, move the mouse in the image while keeping the left mouse button pressed.
- ④ You can edit the line by dragging its pick points. For fine positioning, zoom the image and move the displayed part using the scrollbars to the right and below the image.
- ⑤ Select a unit in the combo box `Unit` of `AVTViewCalibration`.

① select AVTViewCalibration1

② click to create a line-shaped ROI

③ to draw the line in the image, move the mouse while pressing the left button

④ edit the line by dragging the pick points

⑤ select a unit in the combo box

⑥ specify the length of the line
the computed pixel size is displayed

⑦ results are converted into the selected unit

Object Id	Code	Length	Row	Column	Major Extent	Minor Extent	Orienta
0	8711000028544	13	2.221	6.376	5.632	3.961	21.379

Figure 3.1: Converting positions into other units.


- ⑥ Specify the length of your line (in the example: 4.1cm) in the text box Length Line and press **Enter**. AVTViewCalibration now calculates the height of a pixel in the selected unit and displays it in the text box Pixel Height.
- ⑦ Automatically, the bar code position is converted into the selected unit.

3.2 Evaluating Results Using

In the former examples, you have employed ActivVisionTools to extract and display bar codes and additional features. Using ActivDecision, you can evaluate these results by formulating *conditions* the results have to meet in order to be “okay”. For a detailed description of ActivDecision please consult the [User’s Manual for ActivDecision](#).

Visual Basic Example

Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTDecision to the form by double-clicking  with the left mouse button; delete AVTDataView, AVTViewCalibration, and AVTViewROI. Otherwise, open the project decisions\barcode_decisions.vbp.
- Execute the application and load the image sequence barcode\barcode1.seq.

The following steps are visualized in [figure 3.2](#) (not shown: AVTView and AVTBarcode1D).

- ① The main functionality of ActivDecision is presented by its *support tools*, which can be opened via clicking on AVTDecision with the right mouse button. The *master tool* displays the overall evaluation of the application.
- ② To view the current feature and evaluation results check Enable Update in AVT-DecisionViewResults.
- ③ ActivDecision lets you compare the value of an individual object, ROI, or tool feature with two boundary values, a minimum and a maximum value. In the image sequence, two bar codes appear, 8 711000 028544 and 8 711000 028537; only the lower one is to be evaluated as “okay”. In the columns Min and Max, change the minimum and maximum values for the feature Code accordingly. Now, select a condition that the result has to meet in the column Operation. In our example, the Code has to be Inside the boundaries. If you select None, the result is not evaluated.
- ④ Those features which meet their condition appear in green, the others in red. If at least one feature is “not okay”, the whole object, ROI, or tool is evaluated as “not okay” as well. Analogously, the overall evaluation of application, which is visualized by AVT-Decision, depends on the tool evaluations.

[Figure 3.2](#) shows suitable conditions for the example task. Step through the image sequence by clicking [\[Single\]](#) in AVTViewFG and examine the evaluations. Experiment by formulating additional conditions, e.g., regarding the position or orientation of the bar codes.

① open the support tools via the context menu (right mouse click)

② first, enable the update of results

③ formulate conditions for objects, ROIs or tools

④ the evaluations are displayed immediately

⑤ specify default conditions

⑥ check this box to show the used parameters

Results ActivDecision

Path: AVTBarcode1D1

Tool ROI Object	Name	Value	Min	Max	Interpretation	Operation
AVTBarcode1D1	Objects Tool	1	0	10000	Number	None
	Good Objects Tool	0	0	10000	Number	None
	Bad Objects Tool	1	0	10000	Number	None
	Good ROIs	0	0	10000	Number	None
	Bad ROIs	1	0	0	Number	= Max
Rectangle1_1	Objects ROI	1	Default	Default	Default	Default
	Good Objects ROI	0	Default	Default	Default	Default
	Bad Objects ROI	1	Default	Default	Default	Default
Object 0	Code	8711000028544	Default	Default	Default	Default
	Length	13	Default	Default	Default	Default

Enable Update Fit Cell Size Substitute Default Unit: lcm

Defaults ActivDecision

Path: AVTBarcode1D1

Tool ROI Object	Name	Min	Max	Interpretation	Operation
AVTBarcode1D1	Objects ROI	1	1	Number	= Max
	Good Objects ROI	0	10000	Number	None
	Bad Objects ROI	0	0	Number	= Max
	Code	8711000028530	8711000028540	String	Inside
	Length	1	30	Number	None
Rectangle1_1	Code	Default	Default	Default	Default
	Length	Default	Default	Default	Default

Fit Cell Size Substitute Default Unit: lcm

Figure 3.2: Formulating conditions to evaluate results.


- ⑤ If many similar objects are extracted which all should meet the same conditions, you can specify default conditions using `AVTDecisionViewDefaults`. Defaults can be set per tool or per ROI; ROI defaults override tool defaults, and individual conditions override defaults.
- ⑥ If you check `Substitute Default`, the entries marked `Default` are substituted by their actual content.

3.3 Output of Results Using

Using ActivFile, you can write the results and the evaluations to a log file. How to access results via the programming interface is described in [section 4.3](#) on page 26, how to output them via a serial interface or a digital I/O board in the [User's Manual for ActivSerial](#) and the [User's Manual for ActivDigitalIO](#), respectively.

Visual Basic Example

Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTOutputFile by double-clicking .
- Otherwise, open the project output\barcode_output.vbp.
- Execute the application and load the image sequence barcode\barcode1.seq.

The following steps are visualized in [figure 3.3](#) (not shown: AVTView, AVTBarcode1D, AVTDecision).

- ① By clicking on **Select**, you can open a file selector box to choose a file name for the log file, which will then appear in the text field beside the button. By pressing **Clear File**, you can clear the content of the selected file.
- ② By checking **Enable Writing** you enable the writing mode.
- ③ You can open the ActivFile's two dialogs **DialogFileOptions** and **DialogOutputDataSelect** by clicking **File Options** and **Data Selection**, respectively.
- ④ In **DialogFileOptions**, you can choose between two file formats: Standard text files (suffix .txt) and the so-called *comma-separated values* (suffix .csv) which can be used as an input to Microsoft Excel. Furthermore, you can select a delimiter.
- ⑤ In the same dialog you can limit the size of the log file in form of the number of *cycles* that are to be recorded. A cycle corresponds to one processing cycle from image input to the evaluation and output of results. If you use this option, ActivFile creates two log files and switches between them, thus assuring that you can always access (at least) the results of the last N cycles, N being the specified number of cycles.
- ⑥ By pressing **Estimate**, you can let ActivFile estimate the size of one cycle. Note that you must first select the output data in order to get meaningful results!
- ⑦ In the left part of **DialogOutputDataSelect**, you can navigate through the result hierarchy similarly to **ActivDecision**.
- ⑧ In the right part of **DialogOutputDataSelect**, choose the output data by checking the

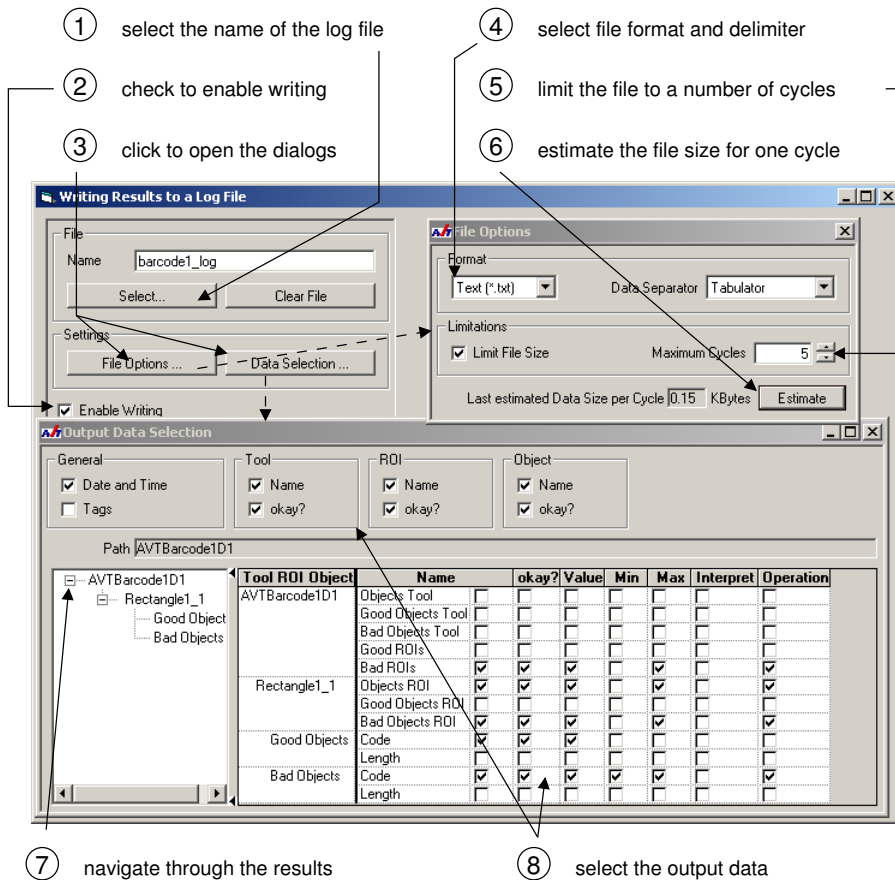


Figure 3.3: Customizing log files.

corresponding boxes. You may output different items depending on the evaluation of an object. By clicking on the column labels with the right mouse button you can check or uncheck all boxes in the column; similarly, you can check or uncheck whole rows or all rows of a certain tool.

If you now step through the image sequence by clicking `Single` in AVTViewFG, the log file is created. Figure 3.4 shows part of an example log file.

```

09/11/04 19:16:25
AVTBarcode1D1 yes
  Bad ROIs          yes  0  0  = Max
  Rectangle1_1     yes
    Objects ROI     yes  1  1  = Max
    Bad Objects ROI yes  0  0  = Max
  0 yes
    Code yes      8711000028537

09/11/04 19:16:26
AVTBarcode1D1 no
  Bad ROIs          no  1  0  = Max
  Rectangle1_1     no
    Objects ROI     yes  1  1  = Max
    Bad Objects ROI no  1  0  = Max
  0 no
    Code no      8711000028544  8711000028530  8711000028540  Inside

09/11/04 19:16:27
AVTBarcode1D1 no
  Bad ROIs          no  1  0  = Max
  Rectangle1_1     no
    Objects ROI     no  2  1  = Max
    Bad Objects ROI no  1  0  = Max
  0 yes
    Code yes      8711000028537
  1 no
    Code no      8711000028544  8711000028530  8711000028540  Inside

```

Figure 3.4: Part of an example log file.

Chapter 4

Tips & Tricks

This chapter contains additional information that facilitates working with ActivBarcode, e.g., how to modify the graphical display of results and how to customize the appearance of an ActivBarcode application in the two execution modes. Furthermore, it shows how to access bar code extraction and evaluations results directly via the programming interface of the Activ-VisionTools.

4.1	Adapting the Display of Results	22
4.2	Configuring the Two Execution Modes	24
4.3	Accessing Results Via the Programming Interface	26
4.3.1	Results of Bar Code Extraction	27
4.3.2	Evaluation Results	32

4.1 Adapting the Display of Results Using

You can adapt the way results and ROIs are displayed using `AVTViewDisplayModes`, which is a *support tool* of `ActivView`.

Visual Basic Example

Preparation for the following example:

- ❑ If you worked on the example in the previous chapter, you may continue using this project. Otherwise, open the project `display\barcode_display.vbp` and execute it (Run ▸ Start or via the corresponding button).
- ❑ Load the image sequence `barcode\barcode1.seq` and create some additional ROIs; `AVTViewFG`, `AVTViewROI`, and `AVTViewCalibration` can be opened at run time via a right mouse button click on `AVTView`.

The following steps are visualized in [figure 4.1](#). Experiment by choosing different settings for the display parameters and watching the result.

- ① Open `AVTViewDisplayModes` by clicking on `AVTView` with the right mouse button and selecting `Display Modes` in the popup menu.
- ② Change the color of the selected pick point. Select another pick point by clicking into its vicinity.
- ③ Change the color of the currently selected ROI. Select another ROI by clicking into its vicinity. Change the color of the other, not selected ROIs of the currently selected tool. Change the color of the ROIs of the other, not selected tools. Select another tool in the combo box `ActivVisionTool` of `AVTViewROI` (if this box contains more than one item). If you use `ActivDecision` to evaluate results, it can mark ROIs evaluated as “not okay” in a special color.
- ④ Change the LUT (look-up table) that is used to display the image.
- ⑤ Change the line width or the ROIs or of the elements (or the region margin, see below).
- ⑥ Change the color of the elements (in the text box `Group I`) or the region (text box `Group II`).
- ⑦ `ActivDecision` uses a separate color to mark objects evaluated as “not okay”. Besides this, a further color is used for highlighting objects if you click on them in `ActivDataView` or in the tree of `ActivDecision`. To test the effect, open `ActivDataView` by clicking on `AVTBarcode1D` with the right mouse button and selecting `Data View` in the appearing context menu; then, click on a object, which is automatically highlighted in `AVTView`.

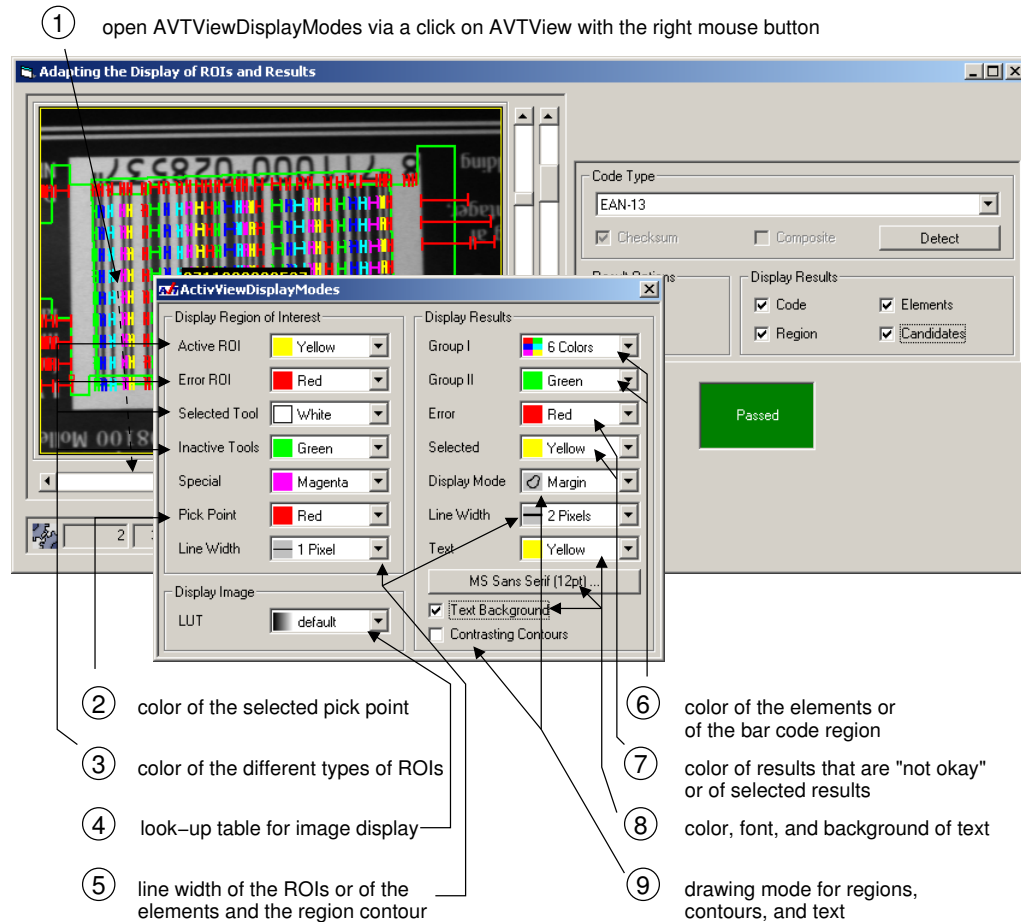


Figure 4.1: Adapting the display of ROIs and results.



- ⑧ Change the color and the font of the text and whether it is displayed on a contrasting background.
- ⑨ Change the way the bar code region is displayed (filled or margin only) and whether contours and text are framed with a contrasting color to make them more visible against busy backgrounds.

4.2 Configuring the Two Execution Modes Via and


In an ActivVisionTools application you can switch between two execution modes: the *configuration mode* and the *application mode*. The former should be used to setup and configure an application, the latter to run it. ActivView's *support tools* AVTViewExecute and AVTViewConfigExec allow you to switch between the two modes and to customize the behavior of an ActivVisionTools application in the two execution modes, e.g., display live images only in the *configuration mode* to setup your application, but then switch it off in the *application mode* to speed up the application. A third sub-tool, AVTViewExecuteSimple, provides a single button to start/stop the application. In this section we also describe AVTViewStatus in more detail.

Visual Basic Example

Preparation for the following example:

- If you worked on the example in the previous chapter, you may continue using this project. At design time, add AVTViewExecuteSimple and AVTViewStatus to the form by double-clicking the icons  and  with the left mouse button. Otherwise, open the project usermodes\barcode_usermodes.vbp.
- Execute the application and load the image sequence barcode\barcode1.seq.

The following steps are visualized in [figure 4.2](#) (not shown: BarcodeControl).

- ① Open AVTViewExecute and AVTViewConfigExec by clicking on AVTView with the right mouse button and selecting Execution and Execution Parameters.
- ② Switch between the two execution modes via AVTViewExecute's combo box Mode.
- ③ To execute one cycle, press `Single`. With the other two buttons you can let the application run continuously and stop it again. By default, AVTViewExecuteSimple starts and stops an application; how to change its behavior to a single-step button is described in the User's Manual for ActivView, [section 3.4](#) on page 34.
- ④ For each of the two execution modes, you can choose what is to be displayed by checking the corresponding boxes in AVTViewConfigExec. You can also specify if images can be dragged to the image window and whether ROIs can be modified in the two modes.
- ⑤ In AVTViewStatus, an icon indicates the current execution mode of the application. In the mode , the application does not perform any processing and waits for your interaction. If you start the continuous mode the cogwheels rotate; any interaction on your part is stored in the event queue and processed after the current cycle is finished. If the cursor gets "busy" the ActivVisionTools have started a particularly time-consuming

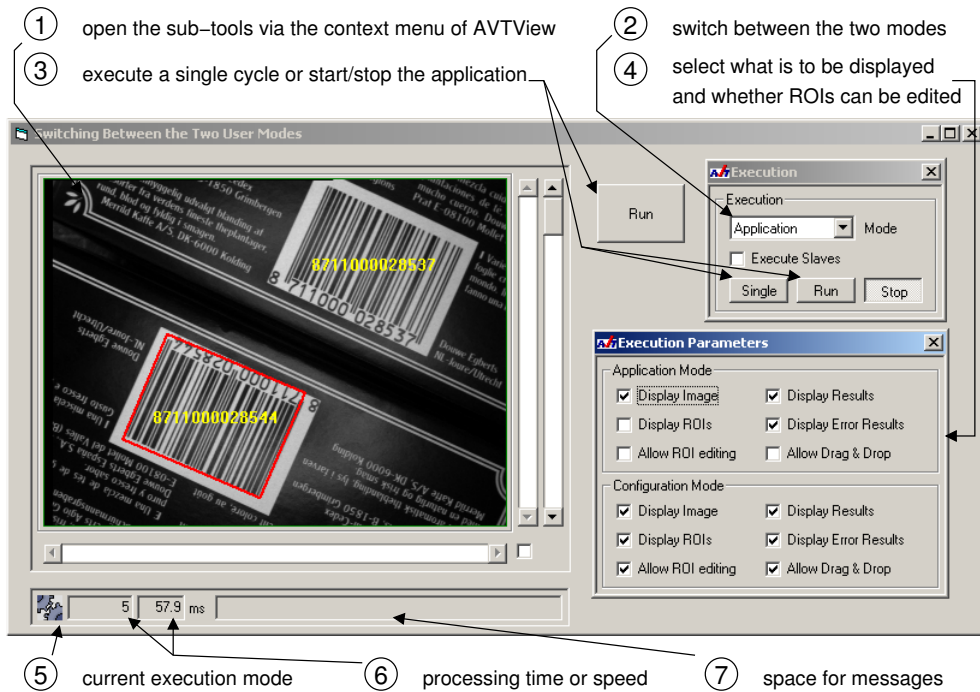


Figure 4.2: Customizing and switching between the two execution modes.

operation, e.g., connecting to an image acquisition device. Any interaction on your part is then deferred to the end of this operation.

- ⑥ AVTViewStatus also shows the number of processed cycles and the time needed for the last processing cycle.
- ⑦ AVTViewStatus display two types of messages: Informative messages describe, e.g., what the application is doing while it is “busy”, while error messages indicate errors that prevent the application from working correctly, e.g., the failure to detect the bar code type automatically. Note that the failure to find a bar code ranks as an information and not as an error, i.e., the application is not interrupted. You can use ActivDecision to mark such cases as “not okay”.

If AVTViewStatus is not added to an application, error messages are displayed in popup dialogs.

More information about AVTViewStatus, e.g., how to modify its appearance, can be found in the User’s Manual for ActivView, [section 3.3](#) on page 32.

4.3 Accessing Results Via the Programming Interface

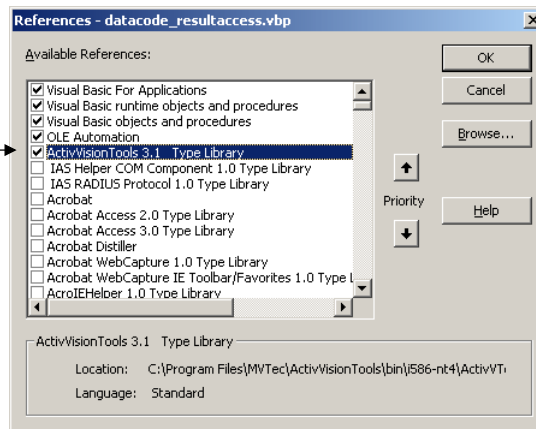
The previous chapters and sections showed how to use ActivVisionTools interactively, i.e., via the graphical user interfaces presented by the underlying ActiveX controls. In this mode, you can develop the image processing part of your machine vision application rapidly and easily, without any programming. However, there is more to ActivVisionTools than the graphical user interfaces: Because ActivVisionTools comes as a set of ActiveX controls, it provides you with an open programming interface, thereby offering full flexibility.

In this section, we show how to access the results of the bar code extraction and of their evaluation via the programming interface. With this, you can, e.g., realize an application-specific graphical user interface, perform additional processing on the results, or send results to a special output device. Detailed information about the programming interface can be found in the **Reference Manual**.

As in the previous sections, the examples stem from Visual Basic 6.0; if the (ActivVisionTools-specific) code differs in Visual Basic .NET, the corresponding lines are also shown (for the first appearance only). For other .NET languages or C++, please refer to the Advanced User's Guide for ActivVisionTools, [section 1.2.3](#) on page 5 and [section 1.3.4](#) on page 28, respectively. Please note that we assume that readers of this part have at least a basic knowledge of Visual Basic.



To work with the programming interface, in VB 6.0 you must first **add the ActivVisionTools type library to the project's references** by checking the box labeled `ActivVisionTools` Type Library in the menu dialog `Project > References`. In Visual Basic .NET, the reference is added automatically.



4.3.1 Accessing the Results of Bar Code Extraction

The principal idea behind accessing the results of an `ActivVisionTool` is quite simple: When a tool has finished its execution, it raises an event called `Finish`, sending its results as a parameter. If you want to access the results, all you have to do, therefore, is to create a corresponding event procedure which handles the event.

Within the Visual Basic environment, you can create event procedures very easily as shown in [figure 4.3](#): In the header of the form's code window there are two combo boxes. Select the instance of `AVTBarcode1D` (by default called `AVTBarcode1D1`) in the left combo box. The right combo box then lists all events available for this object; when you select `Finish`, the event procedure is created automatically. Within this procedure, the results of the bar code extraction are now accessible via the object variable `atToolResults`.

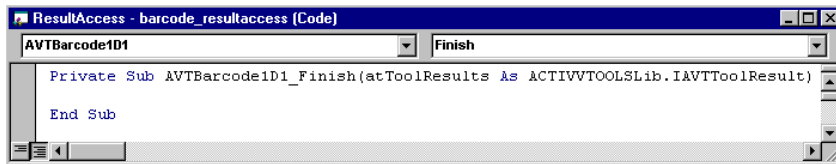


Figure 4.3: Creating a procedure to handle the event `Finish`.

`atToolResults` contains the result data for all ROIs of your instance of `AVTBarcode1D`. The current number of ROIs can be queried via

```
Dim iNumROIs As Integer

iNumROIs = atToolResults.ROINum
```

In Visual Basic .NET, the event handler has a different signature:

```
Private Sub AxAVTBarcode1D1_Finish(ByVal sender As System.Object, _
    ByVal e As _
        AxActivVTools.__AVTBarcode1D_FinishEvent) _
    Handles AxAVTBarcode1D1.Finish
```

A first difference is that tool names start the prefix `Ax`, i.e., `AVTBarcode1D` becomes `AxAVTBarcode1D`. The main difference, however, is that the tool results are not directly passed; instead, they are encapsulated in the parameter `e`. From there, they can be extracted with the following lines:

```
Dim atToolResults As AVTToolResult
atToolResults = e.atToolResults
```

To use classes like `ACTIVVTOOLSlib.AVTToolResult` without the namespace `ACTIVVTOOL-`

SLib as in the code above, you must import this namespace by inserting the following line at the very beginning of the code (more information about importing namespaces can be found in the Advanced User's Guide for ActivVisionTools in [section 1.2.4.5](#) on page 12):

```
Imports ACTIVVTOOLSlib
```

The results of a certain ROI can be accessed by specifying its name in a call to the method `ROIResult`, or by specifying its index in a call to the method `ROIResults`. The following code uses the latter method to access the first ROI of `AVTBarcode1D1`:

```
Dim atROIResult As AVTROIRResult

Set atROIResult = atToolResults.ROIResults(0)
```

Now, we can, e.g., query the number of objects extracted in the ROI via

```
Dim iNumObjects As Integer

iNumObjects = atROIResult.ObjectNum
```

Actual results for an object, i.e., the extracted code or the calculated position can be accessed by specifying the feature of interest and the ID of the object in a call to the method `ObjectValue` of `ACTIVVTOOLSlib.AVTROIResult`. The feature handles are available as methods of the corresponding tool, e.g., `AVTBarcode1D.FeatureHandleCode` being the handle for the extracted code.

The following code fragment uses another method of `ACTIVVTOOLSlib.AVTROIResult`, `ObjectValues`, which returns the values of all objects for the specified feature in an array, to calculate the mean width (i.e. major extent) of the bar code regions:

```
Dim handleWidth As Integer, i As Integer
Dim vWidthArray As Variant
Dim dSumWidth As Double, dMeanWidth As Double

handleWidth = AVTBarcode1D1.FeatureHandleMajorExtent
vWidthArray = atROIResult.ObjectValues(handleWidth)
dSumWidth = 0
For i = 0 To iNumObjects - 1
    dSumWidth = dSumWidth + vWidthArray(i)
Next
dMeanWidth = dSumWidth / iNumObjects
```

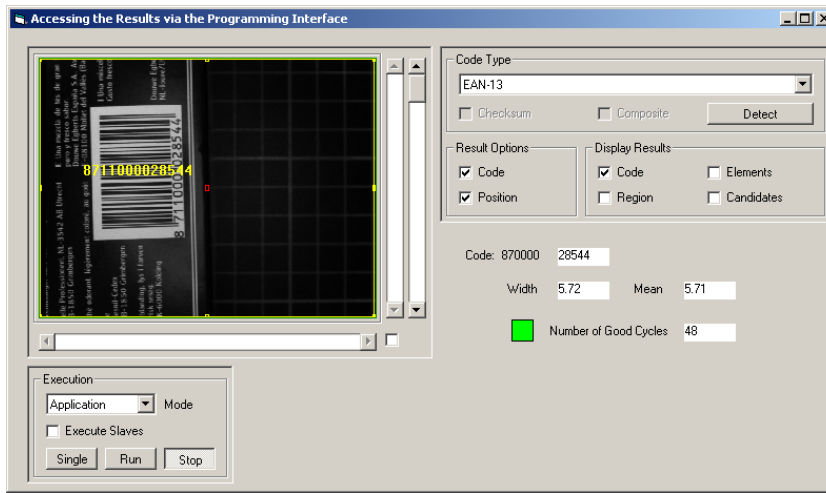


Figure 4.4: Accessing and displaying the results of bar code extraction.

A general difference **in Visual Basic .NET** is that instead of the type `Variant` you must use `Object` when accessing multiple values:

```
Dim vWidthArray As Object
```

The `ActivisionTools` distribution includes the example Visual Basic project `resultaccess\barcode_resultaccess.vbp`, which uses the methods described above to extend the application for reading a sequence of bar codes introduced in the previous chapter. The task is to display the last 5 digits of the extracted code, the width of the region and its mean value (see [figure 4.4](#)). The example project is already configured, just start it and click the button `Run` in `AVTViewExecute`.

Besides accessing the results of bar code extraction, the project code contains additional functionality which is explained briefly in the following. Note that the code is only shown for Visual Basic 6.0; a Visual Basic .NET application with result access can be found in the directory `examples\dotnet\vb\barcode_results`.

First of all, the extracted code and the width is only to displayed if exactly one object has been extracted:

```

If iNumObjects = 1 Then
    sCode = atROIResult.ObjectValue(handleCode, 0)
    TextCode.Caption = Right(sCode, 5)
    dWidth = atROIResult.ObjectValue(handleWidth, 0)
    TextCurrentWidth.Caption = Format(dWidth, "Fixed")

    If bIsError = True Then
        Call ClearAlarm
    End If

Else
    TextCode.Caption = "----"
    TextCurrentWidth.Caption = "----"
    Call SetAlarm
End If

```

If not exactly one object is extracted, the function `SetAlarm` stops the application by setting `AVTView`'s property `RunState` to 'False' and switches the color of the element beside the number of good cycles to red. The function `ClearAlarm` resets the color to green. To test this behavior step through the images sequence to an image containing two bar codes.

```

Private bIsError As Boolean

Private Function SetAlarm()
    AVTView1.RunState = False
    Light.BackColor = vbRed
    bIsError = True
End Function

Private Function ClearAlarm()
    Light.BackColor = vbGreen
    bIsError = False
End Function

```

When calculating the mean width, one has to keep in mind that `AVTBarcode1D` is executed not only when the next image is grabbed but also whenever you modify its ROI (s) or parameters. To distinguish the two cases an event raised by `AVTView` at the start of each execution cycle can be used to set a variable called `bIsNewCycle`:

```

Private bIsNewCycle As Boolean

Private Sub AVTView1_CycleStart()
    bIsNewCycle = True
End Sub

```

Before calculating the mean values within the handler for `AVTBarcode1D`'s event `Finish`, this variable is checked (and immediately reset).

```

If bIsNewCycle = True Then
    iNumGoodCycles = iNumGoodCycles + 1
    bIsNewCycle = False
    dSumWidth = dSumWidth + dWidth
    dMeanWidth = dSumWidth / iNumGoodCycles
    TextMeanWidth.Caption = Format(dMeanWidth, "Fixed")
End If

```

You can test this behavior by changing parameters in AVTBarcode1DParameters: The new width is displayed, while the mean value remains constant.

When using the programming interface of ActivVisionTools, you leave the safe world of the graphical user interfaces where all input is checked for validity automatically. In contrast, if you try to access a non-existent object or result via the programming interface, a run-time error is caused which terminates your application! To avoid this, you can use the Visual Basic error handling mechanisms, i.e., set up an error handler which examines any occurring error and reacts in a suitable way. In the example project, if an error is caused by the result access, a dialog with the error description pops up and the function SetAlarm is called. To view the effect of the error handler, uncheck Position in AVTBarcode1D so that the position is not calculated.

```

Private Sub AVTBarcode1D1_Finish(atToolResults As _
                                ACTIVVTOOLSlib.IAVTToolResult)

    ' variable declarations

On Error GoTo ErrorHandler

    ' procedure body

Exit Sub

ErrorHandler:
    Dim sTitle As String
    If Left(Err.Source, 11) = "ActivVTools" Then
        sTitle = "ActivVisionTools Error"
    Else
        sTitle = "Runtime Error " & CStr(Err.Number)
    End If
    Call MsgBox(Err.Description, vbExclamation, sTitle)
    Call SetAlarm
End Sub

```

By placing the following code at the beginning of the event handler, the actual result access is restricted to the *application mode*. With this mechanism you can setup the vision part of your application in the configuration mode without having to worry about run-time errors.

```
If Not AVTView1.ExecutionMode = eApplicationMode Then
    TextCode.Caption = "----"
    TextCurrentWidth.Caption = "----"
    Exit Sub
End If
```

4.3.2 Accessing Evaluation Results

The evaluation results can be accessed similarly to the results of bar code extraction; in fact, they are even stored in the same object. However, to access the evaluation results you now have to wait for *ActivDecision* to finish, i.e., create the event procedure

```
Private Sub AVTDecision1_Finish(atToolResults As Collection)

End Sub
```



Note that **you will get a run-time error if you try to access evaluation results before *ActivDecision* has finished** (e.g., in the handler for *AVTBarcode1D*'s event *Finish*!

Because *ActivDecision* can evaluate the results of more than one tool, the event handler provides you with a *Collection* of tool results. The following code fragment searches the collection for the results of *AVTBarcode1D1* and “stores” them in *atBarcodeResult*, or exits if no results are found:

```
Dim atToolResult As AVTToolResult
Dim atBarcodeResult As AVTToolResult
Dim bBarcodeResultsFound As Boolean

bBarcodeResultsFound = False
For Each atToolResult In atToolResults
    If atToolResult.Name = "AVTBarcode1D1" Then
        Set atBarcodeResult = atToolResult
        bBarcodeResultsFound = True
    End If
Next
If bBarcodeResultsFound = False Then
    Exit Sub
End If
```

In Visual Basic .NET, the event procedure has the following signature:

```
Private Sub AxAVTDecision1_Finish(ByVal sender As System.Object, _
                                ByVal e As _
                                AxActivVTools._AVTDecision_FinishEvent) _
    Handles AxAVTDecision1.Finish
```

Again, the tool results are encapsulated in the parameter `e`. They can be extracted as follows; note the use of `VBA.Collection` instead of `Collection`!

```
Dim atToolResults As VBA.Collection
atToolResults = e.atToolResults
```

As already remarked in the previous section, tool names are prefixed with `Ax`, thus you must search for the results of `AxAVTBarcode1D1`:

```
If atToolResult.name = "AxAVTBarcode1D1" Then
```

You can query the overall evaluations at different levels, tool, ROI, or object:

```
Dim atROIResult As AVTROIRResult
Dim bToolIsOK As Boolean, bROIIsOK As Boolean, bObjIsOK As Boolean

Set atROIResult = atBarcodeResult.ROIResults(0)

bToolIsOK = atBarcodeResult.Evaluation
bROIIsOK = atROIResult.Evaluation
bObjIsOK = atROIResult.ObjEvaluation(0)
```

Furthermore, you can access the evaluation of individual features like the bar code of the object with the ID 0, but also of tool features (e.g., the evaluation of the number of “bad” ROIs) or of ROI features (e.g., the evaluation of the number of objects) via the corresponding feature handle. In contrast to object features, the handles for tool and ROI features are available as properties of `ACTIVVTOOLSlib.AVTToolResult`.

```
Dim handleCode As Integer, handleBadROIs As Integer
Dim handleObjectsInROI As Integer
Dim bCodeIsOK As Boolean, bBadROIsIsOK As Boolean
Dim bObjectsInROIIsOK As Boolean

handleCode = AVTBarcode1D1.FeatureHandleCode
handleBadROIs = atBarcodeResult.FeatureHandleBadROIs
handleObjectsInROI = atBarcodeResult.FeatureHandleObjectsROI

bCodeIsOK = atROIResult.ObjFeatureEvaluation(handleCode, 0)
bBadROIsIsOK = atBarcodeResult.FeatureEvaluation(handleBadROIs)
bObjectsInROIIsOK = atROIResult.FeatureEvaluation(handleObjectsInROI)
```

The `ActivVisionTools` distribution includes the example Visual Basic project `evalac-`

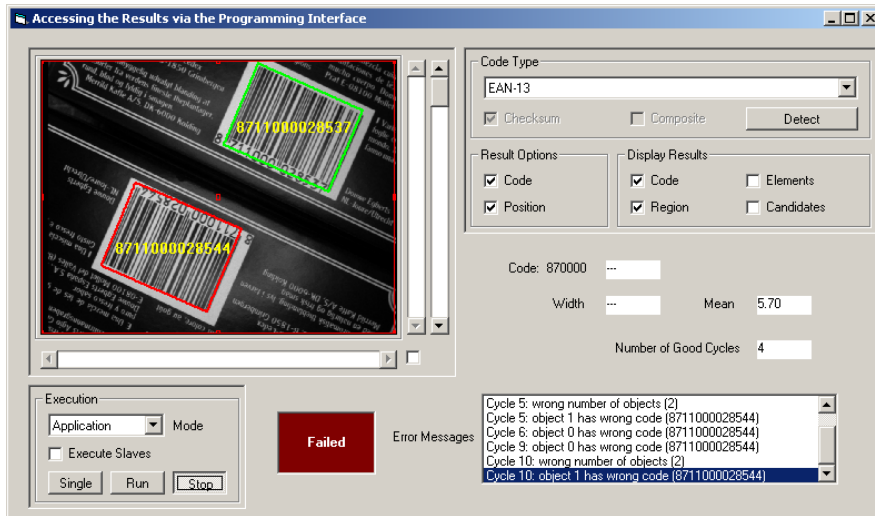


Figure 4.5: Accessing and displaying the evaluation results.

cess\barcode_evalaccess.vbp, which uses these methods to extend the application described in the previous section. Now, `ActivDecision` is used to check the extracted bar code as described in [section 3.2](#) on page 16; in case the overall evaluation shows an error, the cause of the error is to be displayed (see [figure 4.5](#)). The example project is already configured, just start it and click the button `Run` in `AVTViewExecute`.

Besides accessing the evaluation results, the project code contains additional functionality which is explained briefly in the following (again only for Visual Basic 6.0!). First of all, the main `If - Then - Else` clause around the display of results of the bar code extraction now tests the overall evaluation of `AVTBarcode1D1`:

```

If atBarcodeResult.Evaluation = True Then
    ' access and display bar code extraction results
Else
    ' examine cause of error more closely
End If

```

In the example application, the only cause for `AVTBarcode1D1` being evaluated as “not okay” is that its (one) ROI has been evaluated as “not okay”. This in turn can have two possible reasons: First, that the wrong number of objects has been extracted (i.e., less or more than one). In this case, the following code fragment displays a corresponding message in a list box which indicates how many objects have been extracted:

```
If atROIResult.FeatureEvaluation(handleObjectsInROI) = False Then
    sErrMsg = "Cycle " & iNumCycles & ": wrong number of objects (" & _
        atROIResult.Value(handleObjectsInROI) & ")"
    Call DisplayMessage(sErrMsg)
End If
```

The error message also contains the number of the cycle in which the error occurred. The corresponding counter is incremented in the handler for AVTView's event CycleStart which was introduced already in the previous section:

```
Private iNumCycles As Integer

Private Sub AVTView1_CycleStart()
    iNumCycles = iNumCycles + 1
End Sub
```

A second cause of error is that an object has been evaluated as “not okay”. Again, a corresponding error message is created, this time containing the extracted bar code. The following code shows how to access the indices of all objects evaluated as “not okay”:

```
Dim badObject As Variant

If atROIResult.FeatureEvaluation(handleBadObjectsInROI) = False Then
    ' get all bad objects (cause: bad code)
    For Each badObject In atROIResult.BadObjectIndices
        sErrMsg = "Cycle " & iNumCycles & ": object " & badObject & _
            " has wrong code (" & _
                atROIResult.ObjectValue(handleCode, badObject) & ")"
        Call DisplayMessage(sErrMsg)
    Next
End If
```