GenePix® Pro Software

Scripting Reference Guide

5000552 B
February 2011

Molecular Devices

This document is provided to customers who have purchased Molecular Devices, Inc. ("Molecular Devices") equipment, software, reagents, and consumables to use in the operation of such Molecular Devices equipment, software, reagents, and consumables. This document is copyright protected and any reproduction of this document, in whole or any part, is strictly prohibited, except as Molecular Devices may authorize in writing.

Software that may be described in this document is furnished under a license agreement. It is against the law to copy, modify, or distribute the software on any medium, except as specifically allowed in the license agreement. Furthermore, the license agreement may prohibit the software from being disassembled, reverse engineered, or decompiled for any purpose.

Portions of this document may make reference to other manufacturers and/or their products, which may contain parts whose names are registered as trademarks and/or function as trademarks of their respective owners. Any such usage is intended only to designate those manufacturers' products as supplied by Molecular Devices for incorporation into its equipment and does not imply any right and/or license to use or permit others to use such manufacturers' and/or their product names as trademarks.

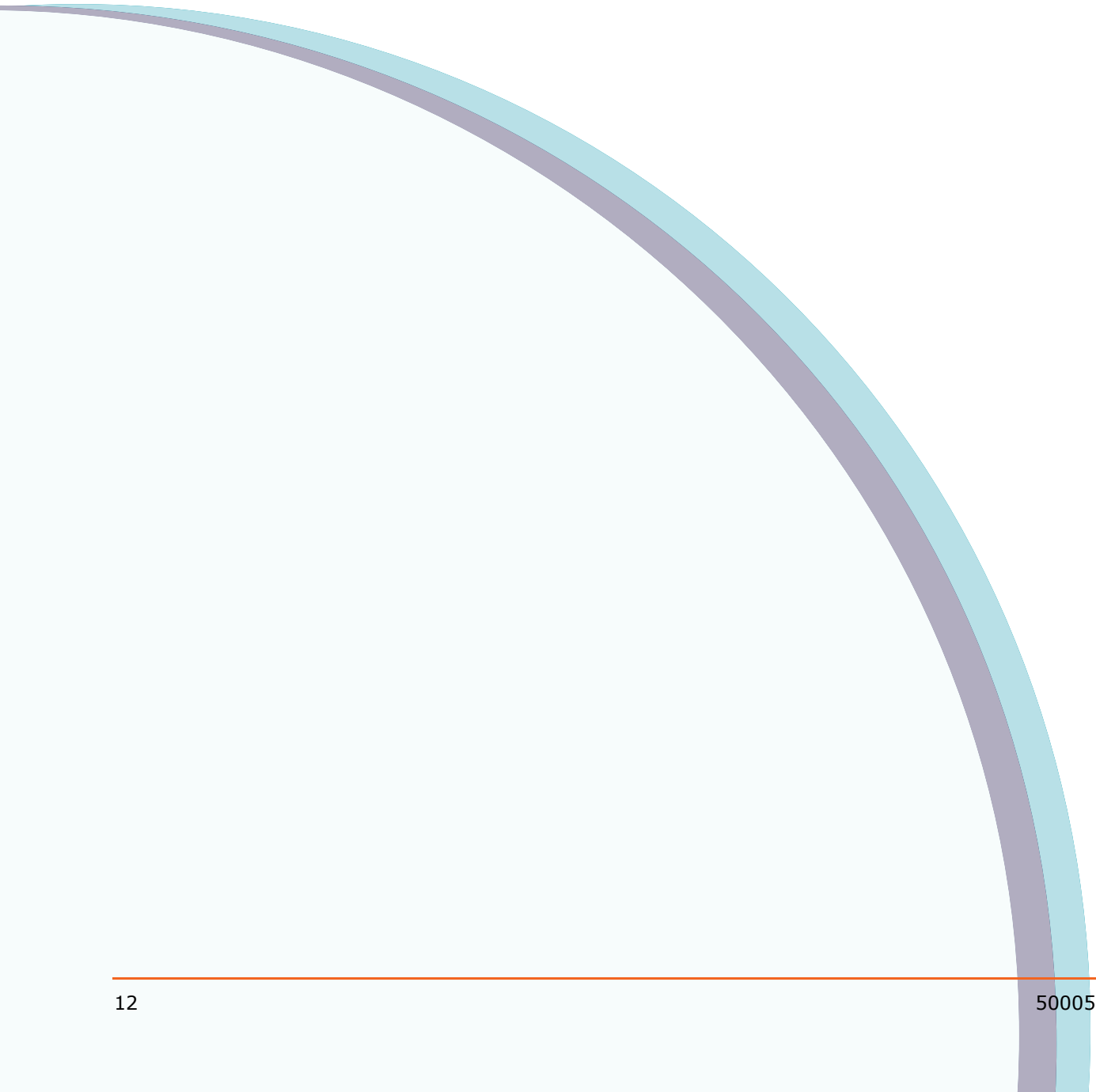
Molecular Devices makes no warranties or representations as to the fitness of this equipment for any particular purpose and assumes no responsibility or contingent liability, including indirect or consequential damages, for any use to which the purchaser may put the equipment described herein, or for any adverse circumstances arising therefrom.

For research use only. Not for use in diagnostic procedures.

The trademarks mentioned herein are the property of Molecular Devices, Inc. or their respective owners. These trademarks may not be used in any type of promotion or advertising without the prior written permission of Molecular Devices, Inc.

Product manufactured by Molecular Devices, Inc.
1311 Orleans Drive, Sunnyvale, California, United States of America 94089.
Molecular Devices, Inc. is ISO 9001 registered.
© 2011 Molecular Devices, Inc.
All rights reserved.
Printed in the USA.

Contents



Preface

This preface describes the intended audience for this guide and provides a brief description of the GenePix® Pro application scripting capability. It also defines the typographical conventions used in this guide.

Note: The information in the guide applies to the GenePix Pro software up to and including GenePix 6 software.

Who This Guide Is For

This guide is written for those who wish to use the GenePix Pro HTML interface to format and display data in conjunction with a scripting language, to manipulate the data, to automate common tasks and to add interactivity to reports.

About this Guide

This guide contains the following chapters:

Title	Content Overview
Chapter 1: Introduction To Scripting on page 15	Describes which scripting languages can be used and how to use the tutorial contained in this guide.
Chapter 2: Scripting Tools on page 17	Contains setup information needed before you being the tutorial, HTML, VBScript and JavaScript reference information.
Chapter 3: HTML Basics on page 19	Provides some basic HTML guidelines.
Chapter 4: VBScript on page 33	Describes the VBScript language and provides details of how to use it.
Chapter 5: Results Tab Scripting on page 49	Describes how to manipulate data from GenePix® Pro software Results tab.
Chapter 6: Access to Other Tabs on page 59	Describes how to use the Image tab, Histogram and Scatter Plot tabs, Lab Book tab and the Report tab.
Chapter 7: Automated Acquisition on page 63	Describes how to script acquisitions.
Chapter 8: GenePix® Pro Software Scripting in Python on page 69	Describes how to manipulate GenePix Pro data and acquisition directly with Python.
Chapter 9: GenePix® Pro Software Scripting in PerlScript on page 73	Describes how to manipulate GenePix Pro data and acquisition directly with PerlScript.
Appendix A: GenePix® Pro Object References on page 77	Contains reference information for the GenePix Pro object.
Appendix B: Scanner Object Reference on page 109	Contains reference information for the Scanner object.
Appendix C: Image Tab Object Reference on page 115	Contains reference information for the Image Tab object.
Appendix D: Histogram Object Reference on page 129	Contains reference information for the Histogram object.

Title	Content Overview
Appendix E: Results Object Reference on page 133	Contains reference information for the Results object.
Appendix F: Report Object Reference on page 157	Contains reference information for the Report object.
Appendix G: ScatterPlot Object Reference on page 163	Contains reference information for the Scatter Plot object.
Appendix H: Solutions To Exercises on page 165	Contains the answers to the exercises used throughout the tutorial.
Appendix I: Standard Web Color Names on page 215	Contains a list of web colors and their associated RGB values that can be used in HTML.

Conventions

Within the scope of this guide, the following typographical conventions are used.

WARNING! A warning indicates an operation that may cause personal injury if precautions are not followed.

CAUTION! Indicates an operation that may cause damage to the instrument, device, or data, if the precautions are not followed.

Note: Provides essential information for the completion of a procedure.

Tip! Provides useful information that helps apply the techniques and procedures in the text to your specific needs, and provides shortcuts, but is not essential to the completion of a procedure.

1 Introduction To Scripting

This chapter contains the following topics:

The Report tab in GenePix Pro software contains an embedded Internet Explorer browser. A report is a *.htm (or *.html) document that contains both HTML and a script in a scripting language such as VBScript, JavaScript, PerlScript, or Python.

For example, a Quality Control report might take data from the Results tab and construct a number of different graphs that display at a glance whether or not a scan is successful.

By contrast, an Automation script scans a test area of an array, analyzes the data, and if the data passes a number of tests, it then scans the whole array and analyzes the data.

Note: An Automation script and a Quality Control report are installed by default with GenePix Pro software.

Which Scripting Languages Can I Use?

The GenePix Pro software natively supports scripting in VBScript and JavaScript.

The scripting interface for GenePix Pro software has been designed to be highly flexible. The parameters passed in GenePix Pro scripts are fairly simple types (BSTR strings, integers, doubles, and arrays of these types) so they are accessible from any scripting language such as PerlScript or Python that can run in Internet Explorer and that understands COM dispatch interfaces.

To script GenePix Pro software in a non-native scripting language such as Python or PerlScript, you must install a third-party scripting engine. Scripting engines can be downloaded for free from a distribution site such as www.activestate.com.

The Default GenePix Pro Software Language

This scripting guide will concentrate almost exclusively on VBScript, for several reasons. First, VBScript is more accessible for beginners to scripting. Second, for simplicity it is better to concentrate on one scripting language. Third, there are several implementations of JavaScript (namely JavaScript itself and Microsoft's JScript), whereas there is only a single VBScript implementation. There is, therefore, more opportunity for confusion when using JavaScript than when using VBScript. In addition, VBScript is a Microsoft product and can claim more compatibility with the embedded Internet Explorer browser than can JavaScript.

Because different versions of Internet Explorer support different HTML and VBScript features, and because GenePix Pro software uses the version of Internet Explorer installed on your computer, MDS Analytical Technologies recommends that you install the most recent version of Internet Explorer that is available (5.0 or higher).

If you have a background in C/C++ programming or JavaScript, you may wish to write your own scripts using JavaScript. The GenePix Pro Object Model is entirely consistent with JavaScript.

The Scripting Tutorial

The scripting tutorial takes you through HTML, VBScript, and the GenePix Pro Object Model. After each new concept is introduced, there is a small exercise to perform. If your objective in reading the tutorial is to learn how to write scripts, then you should attempt every exercise.

Scripting Tools

2

This chapter contains the following topics:

Before You Begin

Before you begin scripting, you should have the following software installed on your computer:

- GenePix® Pro software, version 3.0 or later;
- Internet Explorer 5.0 or 5.5;
- Microsoft Script Debugger (not essential, but very useful), free from http://www.microsoft.com/downloads/details.aspx?familyid=2f465be0-94fd-4569-b3c4-dffdf19ccd99&displaylang=en. Go to Script Debugger / Downloads. Alternatively, Microsoft Visual InterDev 6.0, part of Visual Studio 6.0, is an excellent debugger.
- An HTML editor such as HomeSite from Allaire software, or a plain text editor such as Notepad, for editing scripts. GenePix Pro software also has a simple, built-in text editor for editing reports.

Note: WYSIWYG HTML editors (such as Front Page or Dreamweaver) are not very useful when writing scripts that interact with GenePix Pro software. However, they can be quite useful for learning basic HTML.

- GenePix Pro software, version 6.0 and later, HTML files need to be created in this software rather than in the Microsoft Windows Notepad or other HTML editors.

HTML References

Although this tutorial discusses some of the basics of HTML and VBScript, the following references may be useful, both for novices and more advanced users:

- Danny Goodman 2006, *Dynamic HTML: The Definitive Reference*, O'Reilly & Associates Inc., Sebastopol CA. [An indispensable reference work on all aspects of HTML and DHTML.]
- Chuck Musciano & Bill Kennedy 2006, *HTML: The Definitive Guide*, 6th ed., O'Reilly & Associates Inc., Sebastopol CA. [A good introduction to HTML.]

VBScript References

The following are useful Visual Basic Scripting references:

- *Visual Basic Scripting, a complete electronic language reference for VBScript*, free from http://msdn.microsoft.com/en-us/library/t0aew7h6(VS.85).aspx. [Go to VBScript / Downloads and click on **32-bit VBScript Documentation Download (VBSDOC.exe 473KB)**.]
- Matt Childs, Paul Lomax & Ron Petrusha 2000, *VBScript in a Nutshell: A Desktop Quick Reference*, O'Reilly & Associates Inc., Sebastopol CA. [A complete reference to the VBScript language.]
- Paul Lomax 1997, *Learning VBScript*, O'Reilly & Associates Inc., Sebastopol CA. [An invaluable resource for learning VBScript.]
- Adrian Kingsley-Huges, Kathy Kingsley-Huges, Daniel Read 2007, *VBScript Programmer's Reference*, Wrox Press, Birmingham. [The most up-to-date printed reference on VBScript.]

JavaScript References

The following are useful JavaScript references:

- *JScript, a complete electronic language reference for JScript*, free from http://msdn.microsoft.com/en-us/library/hbxc2t98.aspx. [Go to JScript / Downloads and click on **32-bit JScript Documentation Download (JSDOC.exe 473KB)**.]
- David Flanagan 2006, *JavaScript: The Definitive Guide*, 3rd ed., O'Reilly & Associates Inc., Sebastopol CA. [A good introduction to JavaScript.]

In addition, the sample scripts installed with GenePix Pro (and always available from the Home button on the Report tab) should prove invaluable resources on the specific use and implementation of the GenePix Pro Object Model.

- Microsoft also has introductory scripting tutorials on their web site: http://msdn.microsoft.com/en-us/library/ms950396.aspx. Go to VBScript / Documentation / Tutorial or JScript / Documentation / Tutorial.

There are many other resources on the web, particularly for HTML, Dynamic HTML and JavaScript.

HTML Basics

3

This chapter contains the following topics:

GenePix ®Pro Reports uses HTML to display data on the Report tab, so in this very brief introduction to HTML you will concentrate on basic document structure, and some HTML's more commonly-used display features.

Browsers interpret HTML to present documents as they see fit. While this can be troublesome when coding for the Internet where there are various versions of a number of different browsers, it is less so within the GenePix Pro environment since GenePix Pro software uses the Internet Explorer browser only. Nevertheless, you will have to be prepared for some trial and error to produce the effects you want when using HTML.

HTML Files

HTML files are plain text files that have been saved with a *.htm or *.html file extension. Because they are plain text they can be edited in any plain text editor, such as Notepad, or the Report editor in the GenePix Pro program (click the Edit button on the Report tab). Dedicated HTML editors such as HomeSite provide a great deal of extra functionality, but fundamentally they are text editors.

Tags

HTML documents are instructions to a browser telling it what to display, and how to display it. These instructions are known as tags, which consist of the tag name, and optional tag attributes, all placed between opening and closing angled brackets < >. For example, to tell the browser to bold some text, you put the text between the and tags. The backslash (/) followed by the tag name is always used as the end tag. Some end tags are optional, but it is good coding practice to include optional end tags.

Attributes are additional properties that can be assigned to the tag. For example, if you wish to define a table with a width of 600 pixels, you can type:

```
<table width="600"></table>
```

'width' is the attribute that is assigned to the table, and it is inserted in the opening tag.

More than one attribute can be used in a single tag, separated by spaces:

```
<table width="600" cols=3 bgcolor="tan"></table>
```

The values of attributes, for example, 600 or tan, are typically enclosed in quotation marks, but these are also optional. It is good practice to use quotation marks.

Case

HTML (and for that matter VBScript) is case-insensitive, which means that tags can be in upper case, lower case, or mixed case. However, it is good coding practice to always use one case. Some HTML editors, such as HomeSite, can re-write all your tags as upper-case or lower-case, as you choose.

White space

Spaces, tabs, and carriage returns are ignored by the browser when they are outside tags. For example, you can write:

```
<b>This text is bold</b>
```

or

```
<i>
```

This text is italic

```
</i>
```

In addition, you can use arbitrary spaces within tags, for example to separate attributes. It is good coding practice to use lots of white space to delineate different elements in a script. If you examine the example scripts, you will see that they are formatted using many tabs and carriage returns to make them easier to read.

HTML Document Structure

HTML documents have a common structure that contains a number of important signposts for the browser.

First, the entire document must be enclosed in the <html> tag.

Second, immediately after the <html> tag comes the <head> tag, which includes instructions that apply to the whole document, such as <title> and <style>.

Finally, the main body of the document is placed within the <body> tag:

```
<html>

<head>
        <title>
                    Sample GenePix Report
        </title>
</head>
<body>
        GenePix Report
</body>

</html>
```

Note: To display the text **GenePix Report**, include it in the <body> of the document without any additional tags.

Exercise 1: Adding a Report to the Report Tab

Open the previous HTML document on the GenePix Pro Report tab (copy and paste it into a new Report, and click Apply). Make sure the **GenePix Report text** is bold. Add some italicized text. Add more text and make it bold italic.

For the solution to this exercise see Exercise 1 on page 165.

Adding a Script to the Report Tab

To add a script to the Report Tab:

1. Click the Report Tab in the GenePix Pro window.

Figure 3-1 GenePix® Pro Application Window

2. To open the HTML editor for the Report tab file, click **Edit** in the File section in the upper left corner of the window.

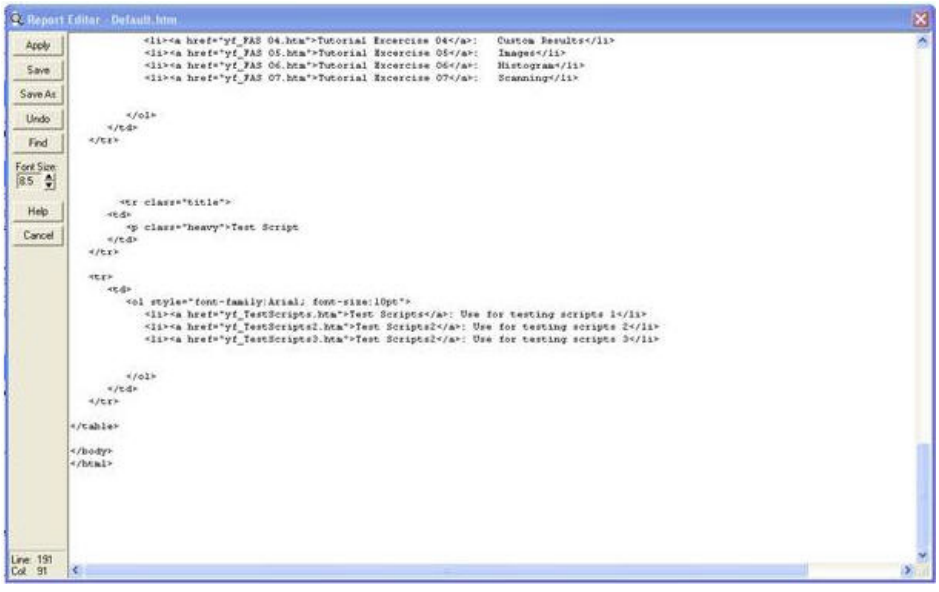


Figure 3-2 Default Edit Screen

3. New HTML text can be typed into this file using standard HTML syntax. In this example, the new HTML text shown in Figure 3-3 is included at the end of the file.

```
<tr class="title">
  <td>
    <p class="heavy">Test Script</p>
  </td>
</tr>

<tr>
  <td>
    <ol style="font-family:Arial; font-size:10pt">
      <li><a href="yf_TestScripts1.htm">Test Scripts 01</a>: Use for
Testing scripts 1</li>
      <li><a href="yf_TestScripts2.htm">Test Scripts 02</a>: Use for
testing scripts 2</li>
      <li><a href="yf_TestScripts3.htm">Test Scripts 03</a>: Use for
testing scripts 3</li>
    </ol>
  </td>
</tr>
```

Figure 3-3 Example HTML Text

4. Click **Save** to save the new text and return to the updated Report Tab.

Figure 3-4 Updated Report Tab

The file is now shown in on the Reports Home page.

5. Click the new link to open another window.

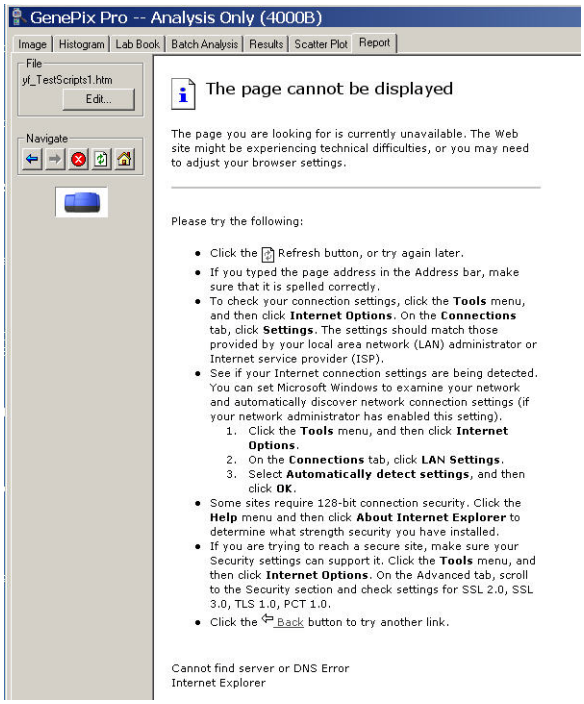


Figure 3-5 New Window

6. Click **Edit** to open the editor for the selected link window.

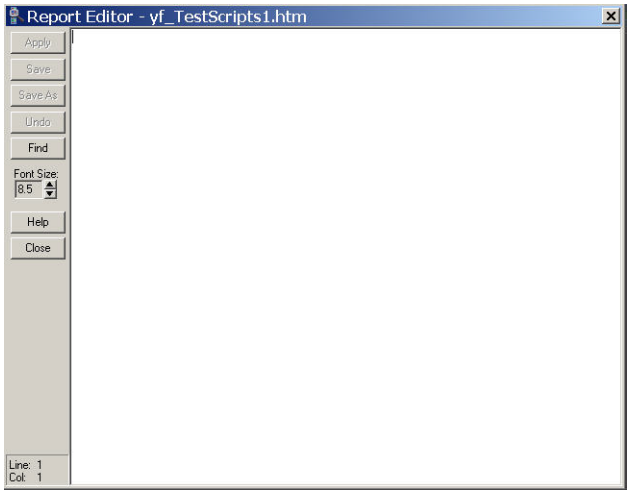


Figure 3-6 Editor Window

7. Copy and paste or type the HTML/VBscript content into the form and click the **Save** button.

The script is now linked to the Report tab and a new HTML file containing the script is created. Script files are saved to the **Genepix/Axon** folder on your system.

Comments

Comments in HTML are enclosed within <!-- and --> tags. There must be a space after the opening tag and before the final tag. Comments are not displayed by the browser.

Comments can be as long as you like, and can include any text whatsoever:

```
<html>
      <!-- This is a sample GenePix report -->
<!-- This is another comment: it is being used for
instruction.-->

<head>
    <title>
         Sample GenePix Report
    </title>
</head>

<body>
    GenePix Report - Exercise 01 - Adding Custom Reports to the
Report Tab.
</body>

</html>
```

Exercise 2: Adding Comments to the Report

Using the code from your previous exercise, add some comments to it explaining what you have done to the code.

```
<html>
<!-- This is a sample GenePix report -->
<!--
    This is another comment: it is being used for instruction.
-->
<head>
    <title>
         Sample GenePix Report: Exercise 2
    </title>
</head>

<body>
    GenePix Report Adding Comments
</body>

</html>
```

For the solution to this exercise see Exercise 2 on page 165

Formatting Text

The web began as a plain text medium, and the programmer's control over text in HTML documents is still not very sophisticated. Document formatting is complicated by the fact that users can themselves change the way documents appear by changing settings within their browsers (such as preferred font faces and sizes etc). For example, in Internet Explorer 5.0 go to the View / Text Size menu, and note the different text size settings. Users' settings can play havoc with document formatting, so it is important to maintain tight control on format by specifying as many properties as are necessary to maintain a document's appearance.

Cascading Style Sheets

The preferred way of specifying text styles within an HTML document is using a cascading style sheet. This is a style specification that is placed within the <head> tag at the beginning of a document, and which can then be used throughout the document. It uses the following syntax:

```
<html>
<head>

    <style type="text/css">
         h2 {font-size: 12 pt; font-family: Arial}
         h2 {font-size: 11 pt; font-family: Arial}
         p {page-break-after: always; font-size: 10 pt; font-
family: Arial}
    </style>

    <title>
         Sample GenePix Report
    </title>
</head>

<body>
    <h2>GenePix Report.</h2>
</body>
</html>
```

The style sheet is specified within the <style> tag, and must include the attribute type="text/css". Each style, such as heading 1 (h1), heading 2 (h2), paragraph (p), is then specified on a separate line, using the above syntax. These particular styles (h1, h2, and p) have a number of built in features; for example h1 and h2 are bold. For a complete list of style attributes, consult a reference on HTML.

Table 3-1 shows more commonly used text styles.

Table 3-1 Commonly Used Text Styles

Attribute	Description	Values	Example
font-size	Sets size of text.	points (pt) inches (in) centimeters (cm) pixels (px)	{font-size: 12pt}
font-family	Sets typeface.	typeface name font family name	{font-family: courier}

Table 3-1 Commonly Used Text Styles (cont'd)

Attribute	Description	Values	Example
font-weight	Sets thickness of type.	extra-light light demi-light medium demi-bold bold extra-bold	{font-weight: bold}
font-style	Italicizes text.	normal italic	font-style: italic}
line-height	Sets the distance between baselines.	points (pt) inches (in) centimeters (cm) pixels (px) percentage (%)	{line-height: 24pt
color	Sets color of text.	color-name RGB triplet{	{color: blue}
text-decoration	Underlines or otherwise highlights text.	none underline italic line-through	{text-decoration: underline}
margin-left	Sets distance from left edge of page.	points (pt) inches (in) centimeters (cm) pixels (px){	{margin-left: 1in}
margin-right	Sets distance from right edge of page.	points (pt) inches (in) centimeters (cm) pixels (px){	{margin-right: 1in}
margin-top	Sets distance from top edge of page.	points (pt) inches (in) centimeters (cm) pixels (px){	{margin-top: -20px}
text-align	Sets justification.	left center right	{text-align: right}
text-indent	Sets distance from left margin.	points (pt){ inches (in) centimeters (cm) pixels (px)	text-indent: 0.5in}
background	Sets background images or colors.	URL color-name RGB triplet	{background: #33CC00}

Once a style has been specified, it is applied to text within the body of the document wherever the associated tag is used, as in the above example where <h2> has been applied.

Note: In the previous CSS, the paragraph specification includes the page-break-after attribute. This applies only to printed pages, not to pages displayed on the screen.

Exercise 3: Adding some of the commonly used CSS tags

Using the code from the previous exercise, remove your formatting with the and tags, and add the <h1> and <h2> tags for each. You might also want to update the text and comments.

For the solution to this exercise see Exercise 3 on page 166.

Some Useful Formatting Tags

The
 tag inserts a line break without a following space. Add it to the end of a line of text. It does not have an end tag.

<hr color="blue" align="center" width="600">

The <hr /> tag inserts a horizontal rule in the document. Some of the more useful attributes are included above, but you can always insert a plain horizontal rule by using just the <hr /> tag alone. It does not have an end tag.

Bold. Place the tags around text to be bolded.

Italic. Place the tags around text to be italicized.

Unordered list. This is a bulleted list. Lists can be nested. To add bullets, each item in the list must be surrounded by the (that is, the list item) tag:

```
<p>Features in this table have:
     <ul>
          <li><p>at least one ratio greater than 2.5 or less
than 0.4;</p></li>
          <li><p>all ratios within 10 per cent of each
other;</p></li>
          <li><p>flag 'good' or no flag ('not found' and 'bad'
features excluded).</p></li>
     </ul>
</p>
```

<blockquote></blockquote>

Indents a section of text within a document.

Use the tag for dividing a document into distinct block elements. For example, you can apply text formatting to all the elements in the tag. Another use for is to declare a variable that can be referenced in a script:

```
<div id=graph2></div>
```

In this example the variable graph2, which may be assigned to a graph constructed in a script, is placed in this document division.

Exercise 4: Adding an Unordered List and Indented Text

Add the previous <ul> code to your test document to see how it is rendered. Remove one or more of the <li> tags to see what happens. Remove one or more of the <p> tags to see what happens. Remove all the <ul> and <li> tags and use <blockquote> and
 instead to indent the list.

For the solution to this exercise see Exercise 4 on page 166.

Tables: Large-scale Document Formatting

So far, this tutorial has examined text formatting, and has not explained how to arrange different pieces of text in a document. While there are several sophisticated ways of doing this using Dynamic HTML, the following section uses the simple but effective method of tables.

A table consists of a number of cells, organized horizontally into rows and vertically into columns. Tables provide a very powerful method of formatting large amounts of data, and they are easy to manipulate by a script.

Note that GenePix Pro software has a built-in Table Builder that automates much of the work that is required to build a table (see the Example Table Builder script). However, if you do not understand the basic structure of HTML tables, you will not be able to use the Table Builder to its full potential. Furthermore, the Table Builder is not suitable for all tables, you will occasionally need to hand code part or all of a table.

Table Tags

<table></table>

The <table> tag is used to create a table. Table tag has a large number of possible attributes listed in Table 3-2 Useful Table Attributes:

Table 3-2 Useful Table Attributes

Attribute	Description	Values	Example
align	Aligns the table within the next outermost container.	center left right	<table align="left">
gcolor	Sets the background color.	color-name RGB triplet	<table bgcolor=”antiquewhite”>
border	Sets the thickness of the table's border.	pixels	<table border=1>
bordercolor	Sets the color of the table's border.	color-name RGB triplet	<table bordercolor=”lemonchiffon”>
cellpadding	Sets the cell padding, which is the amount of empty space between the border of the table cell and the content of the cell.	pixels (px) percentage (%)	<table cellpadding=1>

Table 3-2 Useful Table Attributes

Attribute	Description	Values	Example
cellspacing	Sets the cell spacing, which is the amount of empty space between the outer edges of each table cell.	pixels (px) percentage (%)	<table cellspacing=2>
cols	Sets the number of columns in the table. This is an optional attribute, as the browser interprets the number of columns from other tags (for example, from the number of cells in a row). However, it helps with the legibility of code.	integer	<table cols=4>
height, width	Sets the width and height of the table. Browsers often ignore the height attribute, and ignore the width attribute if it is less than the minimum required to render the content of the table.	pixels (px) percentage (%)	<table height="600" width="600">

<tr></tr>

The <tr> tag is used to define each row in the table. It can use some of the same attributes as the <table> tag, such as bgcolor and height. It also has the align attribute, but this aligns the contents of the row.

<td></td>

The <td> tag is used to define each element in the cell. It can use some of the same attributes as the <table> tag, such as bgcolor, height, and width. It also has the align attribute, but this aligns the contents of the cell. In addition, it has the following useful attributes:

Table 3-3 Attributes

Attribute	Description	Values	Example
colspan	Specifies the number of columns in the table that the cell should span.	Integer	<td colspan=3>
rowspan	Specifies how many rows in a table the cell should span.	Integer	<td rowspan=2>
valign	Vertically aligns the text within the cell, as opposed to align, which positions the text horizontally within the table.	Middle Baseline Bottom, Top	<td valign="top">

id

The id attribute has a number of uses, but in the context of scripting in GenePix Pro software you use it for referencing objects from scripts. If you type:

```
<td id=Graph2></td>
```

then you can use Graph2 as an object reference in a script on the same page.

Table Styles

One further attribute can be used in each of the <table>, <tr>, and <td> tags, and it is the style attribute. You can implement text styles within tables using style, as in the following example for the <tr> tag:

```
<tr style='font-size: 11pt; font-weight: bold'>
```

This style tag could be used in the same way in a <table> or a <td> tag.

An example

The following code fragment describes a table with three rows and two columns:

```
<table width=600 cols=2>

    <tr>
         <td>
             Press the Start Scan button to begin.
         </td>

         <td align="right" valign="top">
             <input type=button value="Start Scan"
onclick="ButtonTest()">
         </td>
    </tr>

    <tr>
         <td colspan=2>
             <hr width=600 color="blue" align=left>
         </td>
    </tr>

    <tr>
         <td colspan=2 align="center">
             <h2>Intensity Distribution Statistics</h2>
         </td>
         </tr>

</table>
```

Exercise 5: Adding Columns, Borders, and Buttons

Copy the example table into a report and load it on the Report tab. Edit the table to make the following changes: add a third column between the two existing columns, but leave it empty so that the table does not change appearance; add a border to see what the third column looks like; swap the first row and the third row; give each row a different background color; add a style attribute to a cell.

For the solution to this exercise see Exercise 5 on page 168.

Forms

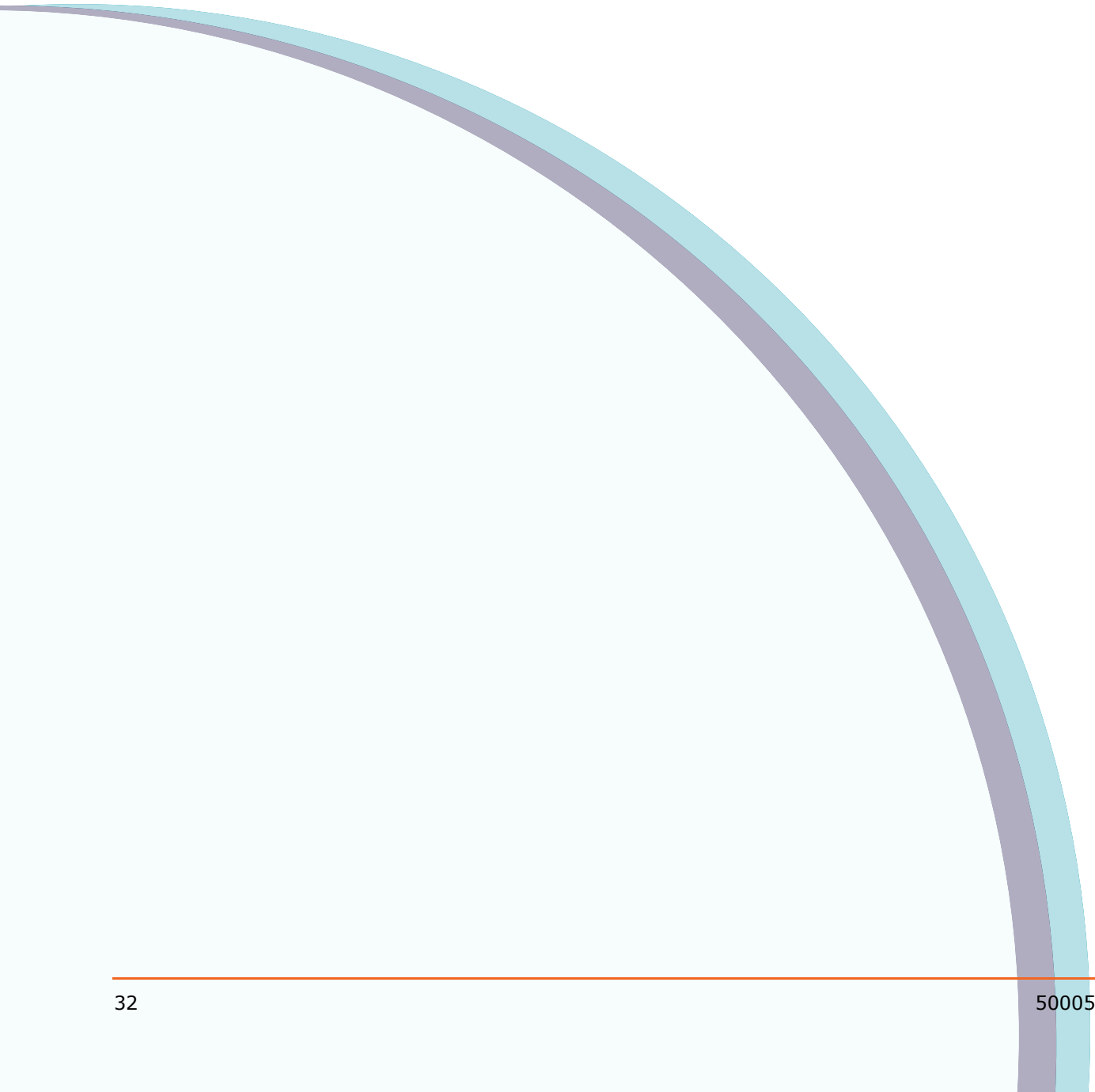
The term “form” is used to cover the various elements that are used when filling out forms in a web browser: text fields, buttons, check boxes, etc. You will have a quick look at buttons, as the lessons learned from buttons generalizes to the other form elements:

```
<input type="button" value="Test" onclick="ButtonTest()">
```

The above line of code (which comes from the table example above) contains most of the attributes:

- To insert a button you use an <input> tag; each of the form elements uses this tag.
- You specify the type of form element using the type attribute; in this case, a button.
- You specify the text on the button using the value attribute; in this case, **Test**.
- Onclick is an “event handler” that calls a subroutine from a script, telling the browser what to do when the button is clicked; in this case, call the ButtonTest() subroutine. There is a large number of event handlers; among others, onclick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseOver, onMouseUp, onLoad, onResize. These can be used in <input> tags, or applied to other tags such as <table>, <tr>, <td> or <body>.

It is now time to turn to scripting.



VBScript

4

This chapter contains the following topics:

Introduction

VBScript is a programming language designed specifically for use in HTML documents on the web. So while it contains many of the commands and capabilities common to other programming languages, it has a number of limitations. For example, for security reasons VBScript file manipulation abilities are limited (otherwise, you could write a malicious script to delete the contents of a user's hard drive). Apart from this, VBScript is a fully-fledged language, and so you are going to have to master a number of standard programming techniques.

For more complete treatments, see the references listed in Scripting Tools on page 17.

The <script> Tag

In HTML documents, scripts are placed within the <script> tag. The script can be anywhere within the <head> or <body> tags, and you can have as many scripts as you want. You can also use scripts in different scripting languages in the same document, as you will see later. In order to tell the browser which language you are using, always use the language attribute in your <script> tag:

```
<script language="vbscript">
```

Case

Like HTML, VBScript is case-insensitive, which means that the browser ignores the case of scripts:

```
Dim GenePix
Set genepix = window.external
```

In this example, you have declared a new variable with the name 'GenePix', and in the next line you have used the variable without the capitalization. However, it is good coding practice to use one case or another.

White space

There are no restrictions on white space within lines of code in VBScript, so you can use spaces and tabs.

```
Dim GenePix
                                    Set GenePix = window.external


Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
```

You can also use as many carriage returns as you like to separate lines, as in the above example. It is good coding practice to use lots of white space to delineate different elements in a script.

Comments

Comments in VBScript are preceded by a single quotation mark:

```
' Declare a variable
' Then set it to window.external
Dim GenePix
Set GenePix = window.external
```

Comments can be as long as you like, and can include any text whatsoever. However, unlike in HTML comments, you need to include another single quote after each carriage return in the comment, as in the above example where the comment spans two lines.

The Internet Explorer scripting engine also lets you use JavaScript comments, which are

```
//:
' this is a VBScript comment.
// this is a JavaScript comment.
```

Running Scripts

As mentioned above in the Forms section, one can apply event handlers to many different HTML tags. For example, to run a script automatically when a report is opened in the Report tab, you can put the event handler onload in the <body> tag:

```
<body language="VBscript" onLoad="HandleLoad()">
```

In this example, the subroutine HandleLoad() is run as soon as the HTML document is loaded. If HandleLoad() calls the rest of the script, then the whole script can be run as soon as the document is loaded. However, to use this method you must specify the scripting language that is being used, in this case VBScript. In the default New GenePix Pro script, the language is set to VBScript in the <body> tag. You can always open the default New GenePix Pro script by selecting “New Report..." from the file menu button.

A second common event handler is onclick, which is often used with buttons:

```
<input type="button" value="Test" onclick="ButtonTest()">
```

In this case, the subroutine **ButtonTest()** is called when the button is clicked.

Exercise 6: Creating An Alert Box

The following script brings up an alert box when the button is clicked:

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 06
     </title>
</head>

<body language="vbscript" onload="ButtonTest()">
     GenePix Report
         <p>
         <input type="button" value="Not Test"
onclick="ButtonTest()">

     <script language="vbscript">

         ' This sub brings up an alert box
         sub ButtonTest()
              alert("Oh no! The alert message is different!")
         end sub

     </script>

</body>
</html>
```

Copy the above code into the GenePix Pro Report editor. Modify the script so that: the alert message is different; the text on the button is different; the alert box is also brought up when the document is first loaded; the button is below the **GenePix Report** text.

For the solution to this exercise see Exercise 6 on page 169.

The Microsoft Script Debugger

If you have the Microsoft Script Debugger installed, now is a good time to become a little more familiar with it. The Debugger has its own documentation that you can access through the Debugger's Help menu, and this is well worth reading if you intend to do a substantial amount of scripting. However, this section provides some introductory points.

There are two types of common errors: syntax errors and runtime errors.

A syntax error occurs when you use the wrong syntax in the script: That is, you misspell a word, or fail to close a multi-line statement. When the script is first opened in the GenePix® Pro program, it is compiled, and if there are any syntax errors, they will be picked up immediately. In this case, the script will not run at all, and you will immediately get an error message.

A runtime error occurs when a command attempts to perform an action that is invalid. In this case, you will get an error that looks like Figure 4-1 Runtime Error Message.

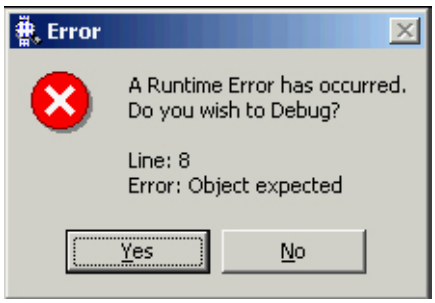


Figure 4-1 Runtime Error Message

The error message has the option to break to the debugger. Click **Yes** and the debugger opens at the line that caused the error, and you will usually be able to see the cause of the error. If in the script for Exercise 6 you change **end sub** to **end su**, you will generate an error like the one above. Click **Yes** to open the debugger.

When your scripts become more complex, you can introduce a third kind of error, the logic error. Logic errors occur when the script does not behave as you expect it to. In such a case, you might know the general area of the script where problems are occurring, but not be able to pin them down. In such cases the Script Debugger becomes very useful indeed. You can insert the command Stop into a script, and the script will break to the Debugger, where you can step through the script's execution a line at a time. For example, in the Exercise 6 script, insert the Stop command immediately above the sub ButtonTest() line, and run the script. The script executes until it reaches the Stop command, and then it launches the Debugger. Press F8 (Step Into); the Debugger then executes the script one line at a time, so that you can see the effects of each line of code.

One of the more useful features of the Debugger is the Command window, which you can open by clicking View >Command Window. In the Command window you can execute commands while the script you are debugging is itself executing. You can use this functionality to check the values of various properties, etc. For example, to find out the value of a variable, just type a ‘ followed by the variable name, press return, and the value is printed in the Command window.

See the Debugger's documentation for more details on stepping through scripts, and see On Error Resume Next on page 47 for how to write error-handling routines.

Variables and Constants

Variables are placeholders for data stored in memory. Variables come in many different types: for example, a variable might be an integer, or a decimal number, or a text string, or a boolean (that is, it has only two possible values, true or false):

```
var1 = 10
var2 = "test"
var3 = 3.141
```

Exercise 7: Using Variables

Modify the script from the last exercise so that it has three buttons called 'var1', 'var2', and 'var3', and when you press a button, it brings up an alert box displaying the value of the associated variable.

For the solution to this exercise see Exercise 7 on page 170.

Option Explicit and Dim

In your script from the above exercise, put option explicit on its own in the first line after the <script> tag and run the script again. It should generate an error 'variable is undefined'. This is because option explicit requires that variables are declared before they are used for the first time. You do this using the dim statement:

```
Dim var1, var2, var3
var1 = 10
var2 = "test"
var3 = 3.1415
```

Option Explicit and Dim are essential debugging tools, as they catch misspellings of variable names: always use them. Whenever you use a new variable, always Dim it first, and always make the first line of a script Option Explicit.

Constants

Occasionally you may want to give a constant value a placeholder name. In such a case, you use a constant. The syntax for declaring a constant is similar to declaring a variable, except that you declare and assign the value in the same statement:

```
const MyConstant = 10
```

In the above example, the value of 10 is assigned to the constant MyConstant. You cannot use variables, user-defined functions, or intrinsic VBScript functions (such as Chr) in constant declarations. By definition, they can't be constants. You also cannot create a constant from any expression that involves an operator; that is, only simple constants are allowed.

Naming Restrictions

Variable and constant names follow the standard rules for naming anything in VBScript. A variable name:

- Must begin with an alphabetic character.
- Cannot contain an embedded period.
- Must not exceed 255 characters.
- Must be unique in the scope in which it is declared.

Scope and Lifetime of Variables

A variable's scope is determined by where you declare it. When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable. It has local scope and is called a procedure-level variable. If you declare a variable outside a procedure, you make it recognizable to all the procedures in your script. This is a script-level variable, and it has script-level scope.

How long a variable exists is its lifetime. The lifetime of a script-level variable extends from the time it is declared until the time the script is finished running. At procedure level, a variable exists only as long as you are in the procedure. When the procedure exits, the variable is destroyed. Local variables are ideal as temporary storage space when a procedure is executing. You can have local variables of the same name in several different procedures because each is recognized only by the procedure in which it is declared.

Types

VBScript has only one data type called a variant. A variant is a special kind of data type that can contain different kinds of information, depending on how it's used (hence VBScript is known as a loosely typed language). In the above example, var1 is an integer, var2 is a string, and var3 is a single-precision floating-point number, and these subtypes are assigned automatically. Strings are always enclosed in double quotation marks. For example, if you type:

```
var4 = "10"
```

then var4 is a string, and the result of var4 + var4 is "1010". If you write:

```
var4 = 10
```

then var4 is an integer, and the result of var4 + var4 is 20.

Table 4-1 VBScript Subtypes is a list of VBScript subtypes (from Microsoft's VBScript Language Reference):

Table 4-1 VBScript Subtypes

Subtype	Description
Empty	Variant is uninitialized. Value is 0 for numeric variables or a zero-length string (“”) for string variables.
Null	Variant intentionally contains no valid data.
Boolean	Contains either True or False.
Byte	Contains an integer in the range 0 to 255.
Integer	Contains an integer in the range -32,768 to 32,767.
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Long	Contains an integer in the range -2,147,483,648 to 2,147,483,647.
Single	Contains a single-precision, floating-point number in the range -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
Double	Contains a double-precision, floating-point number in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
Date (Time)	Contains a number that represents a date between January 1, 100 to December 31, 9999.
String	Contains a variable-length string that can be up to approximately 2 billion characters in length.
Object	Contains an object.
Error	Contains an error number.

You can use conversion functions to convert data from one subtype to another; for example, CBool, CInt, CDbl, CSng, CLng, CStr converts a subtype to a boolean, integer, double, single, long and string, respectively:

```
var4 = "10"
var4 = CInt(var4)
```

The above example begins by setting var4 to the string **10**, and then converting it to the integer **10**.

Arrays

So far in this tutorial, variables have had single values; they are scalar variables. Sometimes, you may want to hold more than one value in the same variable, and in such a case you use an array. Arrays have to be declared using Dim, but you also need to state how many elements the array is to contain; you do this in brackets following the array name in the Dim statement. This is a zero-based number; that is, counting array elements begins from zero. The following array contains four elements:

```
Dim newArray(3)
newArray(0) = 10
newArray(1) = "test"
newArray(2) = 3.1415
newArray(3) = "10"
```

Exercise 8: Working With Arrays

Modify the script from Exercise 7: Using Variables on page 36, so that instead of defining a number of variables and displaying their values with alerts, you use the above array.

For the solution to this exercise see Exercise 8 on page 171.

Dynamic Arrays

Sometimes you do not know the size of an array in advance; in this case you use a dynamic array and declare the size of the array dynamically in the script. To declare a dynamic array, use empty brackets:

```
Dim dynArray()
```

When you find out how large the array has to be, use ReDim:

```
ReDim dynArray(5)
```

ReDim erases any information that is in the array. To ReDim while preserving the information, use Preserve:

```
ReDim Preserve dynArray(5)
```

UBound

Use UBound to determine the upper index number of an array. Because arrays are zero-based, the number returned by UBound is one less than the number of elements in the array:

```
NumElements = UBound(newArray) + 1
```

Exercise 9: Adding the UBound Function

Use the UBound function and an alert to display the number of elements in an array.

For the solution to this exercise see Exercise 9 on page 172.

Multidimensional Arrays

Arrays are not limited to holding a single element of data at each index location. For example, you can think of a matrix of data as an array in two dimensions: to reference a data point, you must use two numbers. Arrays in VBScript can have up to 60 dimensions, however, GenePix® Pro software allows a maximum of two dimensions (that is, 2-d matrices).

You can use Dim and ReDim with multidimensional arrays, but one needs to specify the number of elements in both dimensions, for example:

```
Dim TwoDArray(5, 6)
ReDim TwoDArray(7, 6)
```

In the above example, the TwoDArray is created with 5 rows and 6 columns, and then re-dimensioned with 7 rows and 6 columns.

Similarly, you can use UBound with multidimensional arrays, but one must specify the dimension:

```
alert(UBound(TwoDArray, 1))
```

In the above example, an alert displays the upper index of the first dimension of the array TwoDArray. Note that dimensions are NOT numbered with a zero-based list.

Operators

When numbers and strings are defined, you can combine and compare them using the following operators:

Arithmetic

Arithmetic operators are shown in Table 4-2 Arithmetic Operators.

Table 4-2 Arithmetic Operators

Description	Symbol
Exponentiation	^
Unary negation	-
Multiplication	*
Division	/
Integer division	\
Modulus arithmetic	Mod
Addition	+
Subtraction	-
String concatenation	&

Comparison

Comparison operators are shown in Table 4-3 Comparison Operators.

Table 4-3 Comparison Operators

Description	Symbol
Equality	=
Inequality	<>
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Object equivalence	Is

Logical

Logical operators are shown in Table 4-4 Logical Operators.

Table 4-4 Logical Operators

Description	Symbol
Logical negation	Not
Logical conjunction	And
Logical disjunction	Or
Logical exclusion	Xor
Logical equivalence	Eqv
Logical implication	Imp

Note: WYSIWYG HTML editors (such as Front Page or Dreamweaver) are not very useful when writing scripts that interact with the GenePix Pro program. However, they can be quite useful for learning basic HTML

Exercise 10: Use of Concatenation in String Arrays

Define an array with the elements 'this', 'is', 'string', 'concatenation'. Use string concatenation to join the strings to form the statement 'this is string concatenation', and then use the text in an alert.

For the solution to this exercise see Exercise 10 on page 173.

Conditional Decision-Making

Decision-making is one of the most powerful techniques in the programmer's armory. If some condition obtains, then you can execute one piece of code, and if another condition obtains, you can execute another piece.

If...Then...End If

Conditional decision-making begins with the If...Then...End If statement:

```
Dim x, y
x = 5
y = 3
If x > y then
 alert("greater than")
End If
```

This fragment of code checks to see if one number is greater than another, and if it is, it brings up the alert.

If the condition statement fails (that is, if x <= y) then the program takes no action. That can be changed by adding an Else clause:

```
Dim x, y
x = 5
y = 3

If x > y then
 alert("greater than")
Else
 alert("not greater than")
End If
```

You can check the operation of this code by changing the values of x and y. If ...Then statements can be nested for more complex decisions.

Note: The syntax of the If...Then statement must be closely adhered to: The Then must be at the end of the line, and the whole statement must end with End If. If using an Else, it must be placed on a line by itself, as in the above example. The indenting used above is optional, but highly recommended for legibility.

Exercise 11: Determining String Length

The Len function returns the length of a string (that is, the number of characters). For example, try alert(Len("test")). Write a script that takes an array with two strings in it, checks the length of the two strings and alerts "longer than" or "not longer than" depending on the outcome. You might also like to construct naturally worded alerts using string concatenation (for example, 'dollar is longer than doll').

For the solution to this exercise see Exercise 11 on page 173.

Subs and Functions

A sub procedure is a series of VBScript statements, enclosed by Sub and End Sub statements, which perform actions but don't return a value. A sub procedure can take arguments (constants, variables, or expressions that are passed by a calling procedure). If a sub procedure has no arguments, its sub statement must include an empty set of parentheses ().

For example, in Exercise 12: Using Sub Command on page 44, the ButtonTest sub is called by the onClick event handler. It performs an action (the alert) but does not return any value.

Sometimes you may want to call a sub explicitly, and not in response to an event like an onClick. In such a case, you use the call command:

```
call MySub
```

The call command runs the named sub, and once the sub has finished, program execution returns to the next line after the call.

A function procedure is a series of VBScript statements enclosed by the Function and End Function statements. A function procedure is similar to a sub procedure, but can also return a value. A function procedure can take arguments (constants, variables, or expressions that are passed to it by a calling procedure). If a function procedure has no arguments, its function statement must include an empty set of parentheses. A function returns a value by assigning a value to its name in one or more statements of the procedure.

Consider the following code, which you can append to your exercise code:

```
Dim sum
sum = addition(var1, var3)
alert(sum)
Function addition(one, two)
 addition = one + two
End Function
```

This is a very simple function that adds two numbers together. The function is called by passing it two variables with the sum = addition(var1, var3) construct.

Note: In the function definition, you use two arbitrarily named arguments that do not need to be declared with a dim; that is, the variables one and two are declared by being in the function name. The number of variables passed to the function must match the number of arguments used in the function definition.

To obtain the value of the function, assign the value that you want your function to return to a variable whose name is the same as the name of the function. Where you have “addition = one + two,” the function is called addition, and the value you assign to addition is the value returned by the function.

Exercise 12: Using Sub Command

Repeat the Exercise 11: Determining String Length on page 42 using a sub.

For the solution to this exercise see Exercise 12 on page 174.

Exiting a sub or a function

Sometimes you may not wish a sub or a function to proceed if some condition obtains. In such a case, you can exit the sub or function with the Exit Sub or Exit Function statements, respectively:

```
Function addition(one, two)
 addition = one + two
 If addition > 50 then
         Exit Function
 End If
 ...other code to execute
End Function
```

In the above incomplete code fragment, if the value of addition is greater than 50, the script exits the function; otherwise, it proceeds with the rest of the code in the function.

Looping Statements

Loops are extremely important in executing a piece of code a specified number of times. The most common looping statement is For...Next.

```
Dim i
For i = 0 to 4
 Document.writeln "index"
 Document.writeln i
Next
```

For each number in the sequence 0, 1, 2, 3, 4, the code between for and next is executed. The index variable i increments by one each time the loop is executed, and it stops after it reaches the final value and executes the code.

You can use For...Next loops to execute the same operation a specified number of times, or to generate a sequence of numbers that you then use for some other purpose. In the above code example, the code prints 'index' five times, but it also prints the index number.

Exercise 13: Working With Looping Statements

Copy the above code into a script and verify its operation. Note: the Document.writeln command (pronounced 'write line') writes directly to the browser without needing any HTML.

For the solution to this exercise see Exercise 13 on page 175.

Other Looping Statements

Sometimes For...Next is not quite appropriate for the looping statement that you require. For example, suppose you want a loop to continue for as long as some condition obtains, but you don't know in advance for how long that condition will obtain. In such a case you can use a Do Until...Loop statement, or a Do While...Loop statement.

Using variables for the counter

The counter values in a For...Next loop can be variables; for example, you can type:

```
Dim i, someVariable
someVariable = 10
For i = 0 to someVariable - 1
 document.writeln someVariable
Next
```

Exercise 14a: Using For..Next Loop

Write a report that uses a For...Next loop to produce the following series of alerts: 2 + 2 = 4, 4 + 4 = 8, 8 + 8 = 16, 16 + 16 = 32.

Exercise 14b: Using Do..Until Loop

When you have Exercise 14a: Using For..Next Loop working, change it so that it uses a Do Until...Loop statement.

For the solution to this exercise see Exercise 14 on page 175.

Including External Scripts

When writing complex scripts, you may find yourself re-using code from previous scripts. While cutting and pasting code can work for small projects, for larger projects you may find it more convenient to save frequently used code in separate files and then import those files into a script.

To open existing script files into another script, use the src attribute in a <script> tag. In the following code fragment, two separate script files are included by using the src attribute:

```
<script language="vbscript" src=" first_include.htm ">
</script>
<script language="vbscript" src="second_include.htm">
</script>
<script language="vbscript">
 call firsttest()
 call secondtest()
</script>
```

The included files 'first_include.htm' and 'second_include.htm' can be any plain text files, but in this case they have been saved with a *.htm extension. The code in first_include looks like this:

```
Sub firsttest()
 alert("first included")
End Sub
```

Notice that it has no HTML tags: it is plain script. The main script can call a sub that is wholly defined within the included file, as it does with the firsttest() statement.

Exercise 15: Creating External Scripting Calls

Get the above script working. You will need to save the first code fragment within an HTML document with all the usual HTML tags. Next, save the second code fragment as a *.htm file and call it 'first_include.htm'. Finally, you will need to create 'second_include.htm' with a function titled 'secondtest()'.

For the solution to this exercise see Exercise 15 on page 176.

Reading and Writing Text Files

To output data from GenePix Pro software in custom file formats, you can use the FileSystemObject (a standard Microsoft scripting object) to perform all standard file operations, including writing text files.

The FileSystemObject is in Microsoft's scrrun.dll, which is usually installed in the Windows directory. In addition to FileSystemObject, scrrun.dll includes four other objects available for File I/O and other tasks. These objects include the File object, the TextStreamObject object, the Folder object, and the Drive object. All of these objects have properties and methods that are detailed in the Microsoft Visual Basic Scripting Help file.

If you are running Windows 2000, Windows ME or Windows XP, you already have scrrun.dll installed. Otherwise, you can obtain scrrun.dll by installing one of the following packages from Microsoft: Windows Script Host (this is a free download: go to Windows Script Host > Downloads), Windows NT Option Pack, Microsoft Internet Information Server 3.0, Scripting 3.1 upgrade, Visual Studio 6.0, Visual Basic 6.0.

Once you have scrrun.dll installed, you can begin scripting the FileSystemObject. The two basic file operations that you will need to understand are file writing and file reading. The following function creates a text file and writes some data into it:

```
Function Write()
 Dim fso, file
 Set fso = CreateObject("Scripting.FileSystemObject")
 Set file = fso.CreateTextFile("c:\testfile.txt", True)
 file.WriteLine "Test output file."
 file.Close
End Function
```

This function creates a text file called ‘Testfile.txt’ in the root directory on the C: drive. The ‘True’ parameter allows overwriting of an existing file with the same name.

You can also use the GenePix Pro scripting interface to return a reference to the file system object; however, it also requires that you have scrrun.dll installed: see FileSystem on page 83.

On Error Resume Next

There will be occasions in your scripts when forces outside your control can cause run-time errors, for example when taking user input, or when interacting with a file system or a database. Fortunately, VBScript contains the On Error Resume Next statement, which you can use to catch errors when they occur. On Error Resume Next allows the script to continue executing even after an error has occurred. On Error Resume Next has to be used in every procedure in which you have an error-handling routine. The trick in using On Error Resume Next is to know how to call an error-handling routine only when an error has occurred. This is done through the Err object.

The Err object has a number of properties that tell you about the current error state of a procedure. The most important of these is Number. If no error has occurred, the Err.Number property returns a value of zero; otherwise, it returns the error's numerical value. This behavior allows you to write the following error-handling routine:

```
Function Division(numerator, denominator)

      On Error Resume Next

         Dim fResult
         fResult = numerator/denominator

         If Err.Number <> 0 then
              alert("There has been an error.")
              Exit Function
         End If

         Division = fResult

End Function
```

Any time you use division, there is the potential for a divide by zero error. If you try to send in zero in the denominator in the above function, the error will be raised, but the script will not crash because you have used On Error Resume Next. The next line will be executed, and because Err.Number will be non-zero, the error-handling procedure will be executed.

The Err object has a number of other properties. Probably the most useful of these is Description: this returns as a string the error text that would normally be displayed in the VBScript error dialog box when the script crashes. However, you can use the Description yourself in an error-handling procedure. In the error-handling procedure in Division above, you could replace the existing alert with the following:

```
alert("Error: " & Err.Description)
```

Adding Error Recovery

On Error Resume Next

Add the above function to a script and test On Error Resume Next.
For the solution to this exercise see Exercise 15 on page 176.

Using Err. Description

Replace the error message in Division with a message using Err. Description.
For the solution to this exercise see Exercise 15 on page 176.

Results Tab Scripting

5

This chapter contains the following topics:

You are now at a point where you can begin to manipulate data from GenePix® Pro software. Begin by getting some data from the Results tab. If you don't have any Results files, now is the time to create one. The smaller the Results file you use, the faster a script will run.

The GenePix® Pro Object Model

Begin with a new Report document: on the Report tab, click New. At the beginning of the script you will see a number of statements looking like this:

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
```

The Set statement assigns an object reference (in the above code, reference to the GenePix Pro object and to the Results tab object) to a variable. In this way, you can use Results instead of window.external.Results. Each of these set statements provides variables with which you can talk to different GenePix Pro tabs. The complete GenePix Pro Object Model is described in Appendix A: GenePix® Pro Object References on page 77.

In this script you are only going to get data from the Results tab, so you can delete or comment out the Set statements for the other tabs, and for the Array Statistics and Graph objects if you like. Commenting out (that is, prefixing each undesired line by a comment marker) is a useful technique, as you can use the code by deleting the comment marker.

A Note On Results Data Types

Data from the Results tab are returned as double precision floats, except for the Name and ID columns, where they are returned as strings.

The Log Ratio column, which displays 'Error' in the Results tab when the ratio value is negative, returns an Error type for any error values. When manipulating individual Log Ratio values in script, you should always check the type name of each value to make sure that it is not an error. You can do this using the TypeName function or the VarType function.

Some Results Tab Properties

```
Results.ColumnName(column number), Results.Width
```

Use the document.writeln method to produce a list of column names and numbers, as these will be useful:

```
Dim i, nColumns
nColumns = Results.Width-1
For i = 0 to nColumns
     Document.writeln(Results.ColumnName(i))
     Document.writeln(i)
Next
```

Notice that the columns are numbered with a zero-based list.

Exercise 16: Adding Result Tab Properties

Add the above code to the New script and open it in GenePix Pro.

For the solution to this exercise Exercise 16 on page 179.

Scripting Tables

Document.writeln is a crude way of placing text on the page, so let's construct a table instead by combining scripting and html:

```
Dim sTmp, i, nColumns
nColumns = Results.Width-1
sTmp = "<table border=0 style='font-size: 11pt; font-family:
Arial'>"
For i = 0 to nColumns
 sTmp = sTmp & "<tr><td>" & Results.ColumnName(i) & "</td>"
 sTmp = sTmp & "<td>" & CStr(i) & "</td></tr>"
Next
sTmp = sTmp & "</table>"
Table1.InnerHTML = sTmp
```

How does this work? You construct an HTML table as a string, and then assign this HTML to be the Inner HTML of the Table1 object. You will use this technique extensively, so study it carefully.

Exercise 17: Using the OnLoad Event Handler

Replace the previous Document.writeln code with the above code in your script. To get it running, you will need to put it in a subroutine, call the subroutine using an onload event handler in the <body> tag, and add to the body of the document.

After you have it working (it gives a vertical list of column names followed by numbers), try experimenting with the formatting: make the text bold, or italic, or change the font size or other table properties.

For the solution to this exercise Exercise 17 on page 180.

Results.Column(ColumnNumber), Results.Height

Your next task is to get a column of data using the Results.Column method, which returns a whole column of data in an array. You will also use the Results.Height method, which tells you how many rows there are (again, in a zero-based list) in the Results tab. Get the data from the first main data column, F635 Median, which is in column 8 (you can check this with your previous ColumnName script):

```
Dim column8, nRows, i

column8 = Results.Column(8)

nRows = Results.Height

For i = 0 to nRows - 1
         Document.writeln(column8(i))
Next
```

Exercise 18: Adding Tables To The Results Data

Re-write this script so that the output is in an HTML table (use the method employed above for ColumnName).

For the solution to this exercise Exercise 18 on page 181.

Results.Column("ColumnName")

The Results.Column method can also take column names as arguments. If using column names, they must be spelled exactly as they are in the Results file column titles, and enclosed in quotation marks.

Exercise 19: Using Column Names From Results

Modify the script from the previous exercise to use the column 8 column name.

For the solution to this exercise Exercise 19 on page 182.

Results.HideItems "Flag", boolean

HideItems allows you to hide and show results entries based on their flag value. It is similar to the functionality implemented in the Display dialog box, except that the changes are not displayed in the Results tab. This is very useful and powerful for performing analyses where you want to exclude, for example, Not Found features or Bad Features. The following example demonstrates the syntax: you must specify the flag (either by number or name) and whether or not it is hidden (true, corresponds to hidden, false corresponds to shown; by default, if the second parameter is left out, true is assumed).

```
Results.HideItems -50, true
Results.HideItems "absent", true
```

After applying the above code, any column array extracted from the Results data will have all absent and not found features removed. The numerical codes for the Flags are documented in the Flags topic.

Exercise 20: Removing Features With Bad or Not Found Flags

Modify the script from the previous exercise so that features with Not Found and Bad flags are excluded from the output.

For the solution to this exercise Exercise 20 on page 183.

Results.HideMatching "column", "text"

HideMatching allows you to hide results entries based on matching a text string in a named column. This is very useful and powerful for performing analyses where you want to exclude, for example, all features with a particular substance name: you might name all control spots 'control', and then exclude them from an analysis. The following example demonstrates the syntax: you must specify the column in which to search for the matching text (either by number or name) and the text for which to search:

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.HideMatching "Name", "control"
```

There is no provision for undoing a HideMatching call. However, there are several workaround solutions. One is to use the HideItems call. By passing the value true for some flag using HideItems, the Results data is refreshed and all data hidden using HideMatching are shown. A second workaround is simply to get a second instance of the Results data, for example, Results2 = GenePix.Results, and then continue to work with that data object.

Exercise 21: Removing Specific Features

Modify the script from the previous exercise so that a particular feature name and all features from with an certain id are removed from the list. Hint: Select an id name from Expercise 19.

For the solution to this exercise Exercise 21 on page 184.

Results Header Properties

Apart from the data columns in the Results tab, you also have access to the file header information. Each of the properties shown in Table 5-1 is accessed using the Results.property syntax (for example, Results.Version, Results.DateTime, etc.):

Table 5-1 Properties

Property	Description
Version	Returns the version number of the results file (only if the results have been saved)
DateTime	Returns the Date and Time when the images were scanned
Model	Returns the Scanner Model
Comment	Returns comment set when saving results
Barcode	Returns the barcode scanned in from the array
SettingsFile	Returns the filename of the settings file active during scanning
GALFile	Returns the name of the array list used
Creator	Returns the version number of the GenePix® Pro software used to create the results
Wavelength	Returns the laser wavelengths
PMT	Returns the PMT gain
NormalizationFactor	Returns the normalization factor for the given ratio column
Temperature	Returns the temperature voltage measured prior to scanning
LaserPower	Returns the laser power measured prior to scanning

For a complete list of Results tab properties, see Results on page 104 for the Results Object in the GenePix Object Model Reference.

Exercise 22: Outputting Header Data to a table

Construct a table that outputs all of the header data to a table.

For the solution to this exercise see Exercise 22 on page 185.

The Table Builder Object

One of the more processor-intensive operations when scripting is building long tables for outputting large amounts of data. For example, try the following exercise.

Exercise 23: Building Tables For Large Amounts of Data

Write a script to create a table with the columns: block, column, row, name, ID, F532 Mean, Index. Use it on a Results file that has one 24/24 block of data, then on one that has sixteen 24/24 blocks (this second computation will take quite some time; if you have a slow computer, don't attempt it).

For the solution to this exercise see Exercise 23 on page 188.

You will have found from the above exercise that outputting large tables by concatenating character strings can be very slow. **The Table Builder object in GenePix Pro speeds up this process, so it can be useful when building large tables and when speed is a requirement of a script.** The Example Table Builder script on the default GenePix Pro reports page demonstrates how to use this object.

To create a table with the Table Builder, first you need to create an instance of the object:

```
Dim TableBuilder
Set TableBuilder =
CreateObject("AxonWebUtilities.TableBuilder.1")
Then set the table's properties:
TableBuilder.Clear  ' clears the object to build a new table
TableBuilder.Width =  ' sets the width of the table in pixels
TableBuilder.Style = "font-size: 8pt; font-family: Arial"
      ' sets the style for the table
TableBuilder.Color = RGB(255, 255, 255)
      ' sets table background color
TableBuilder.Border = boolean
      ' true for a border, false for none
TableBuilder.NoWrap = Boolean
' true for nowrap applied to body cells, false for wrapping
```

TableBuilder.MaxRows = ' sets the maximum number of rows in the table.

Use MaxRows when you wish to use only a certain row range from the input arrays. Now add the content to the table:

```
TableBuilder.Headings =
      "<td><b>Tag</b></td><td nowrap><b>Value</b></td>"
      ' HTML text defining headings and their styles
TableBuilder.AddColumn(array)
' adds an array as a column of data to the table
TableBuilder.Sort(column number, direction)
' sorts a numbered column; sorting direction is 0 ascending; -1
is descending. The sort also ' works as a toggle, so that
subsequent sorts will toggle direction.
TableBuilder.ColumnURL(column number) = "url"
' adds a hyperlink to every cell in a column array. Can be used
in combination with
' GenePix.WebAddress, which returns the currently selected URL
in Options / Analysis,
' so that one does not have to insert a URL by hand.
TableBuilder.Row(row number)
' returns a numbered row as an array (this is zero-based).
```

And the table is ready to be built:

```
Dim Table
Table = TableBuilder.Build
TableTag.InnerHTML = Table
```

As in other uses of Inner HTML, the TableTag variable has been defined earlier in the Report, usually as an id element of a or <td> tag.

Exercise 24: Using the Table Building Objects

Repeat the previous exercise, this time using the Table Builder object. You might want to add the following functionality (which is demonstrated in the Example_TableBuilder report): after clicking on a column heading, the column sorts; in one column (for example, the name column), each entry is hyperlinked to a web database.

For the solution to this exercise see Exercise 24 on page 190.

An Interesting Features Report

You can create a report that lists all the features (names and IDs) with Ratio of Medians or Median of Ratios or Regression Ratio greater than 2.5 or less than 0.4. In addition, you can exclude features that are flagged Not Found or Bad. In addition you can create a color-coded table, so that table cells reporting ratio values are approximately the same color as their features on the image.

The Statistics Object

GenePix Pro software has a built in Statistics object which takes an array as its argument and which can return the median, mean, standard deviation, max, and min of the values in the array. In addition, it has the ability to bin the values in an array. The Example Results script on the default GenePix Pro Reports page demonstrates how to use this object.

If you open a new Report (using the New command in the Report tab) notice that the variable ArrayStats is assigned to the ArrayStatistics object:

```
Dim ArrayStats
Set ArrayStats = CreateObject
("AxonWebUtilities.ArrayStatistics.1")
To extract statistics from the array yourArray, set:
ArrayStats.array = yourArray
Then after declaring some new variants, we extract statistics
from the object using the following syntax:
MeanyourArray = ArrayStats.Mean
MedianyourArray = ArrayStats.Median
StdDevyourArray = ArrayStats.StdDev
MinyourArray = ArrayStats.Min
MaxyourArray = ArrayStats.Max
```

So in the MeanyourArray variable, you now have the mean value of all the elements of yourArray.

Exercise 25: Calculate Key Statistic Results

Calculate the mean, median, standard deviation, min and max of the F635 median intensity from a Results file, and output the Results using either alerts, or document.writeln, or in a table.

For the solution to this exercise see Exercise 25 on page 191.

Binning

The binning method of the Statistics object takes an array and a bin width as arguments, and returns two arrays, one of bins, and one of counts.

```
Dim afBins, afCounts
call ArrayStats.Bin(yourArray, binWidth, afBins, afCounts)
```

The bins array contains the values of the center of the bins, and the counts array contains the numbers in each bin.

Exercise 26: Binning Statistic Data

Write a script that bins the F635 Median data column, using a bin width of 50, and outputs a list of bins and a list of counts, either with document.writeln or in a table.

For the solution to this exercise see Exercise 26 on page 192.

The Graph Object

GenePix Pro software has a Graph object with which users can graph data. The Example Graph script on the default GenePix Pro Reports page demonstrates how to use this object.

To create a graph with the Graph object, first you need to create an instance of the object:

```
Dim Graph
Set Graph = CreateObject("AxonWebUtilities.Graph")
```

(This definition is already in the New script.)

Then define the graph's properties. Before you add data to the graph object, you should always clear the object:

```
Graph.clear ' clears the current graph settings
```

Then you must provide data in two arrays, one for the X axis and one for the Y axis, and you must give a Data Name. Note: this Data Name must be the same for both axes! That way, you can draw a number of traces on the same graph, identifying them by name:

```
Dim YDataArray, XDataArray
Graph.YData("Data Name") = YDataArray
Graph.XData("Data Name") = XDataArray
Then, give the graph its various properties:
Graph.MajorTitle = "Major Title" ' optional; adds a major title
to the graph
Graph.MinorTitle = "Minor Title" ' optional; adds a minor title
to the graph
Graph.SetXRange xmin, xmax ' optional; set the range for the X
axis, between xmin and xmax
Graph.SetYRange ymin, ymax ' optional; set the range for the Y
axis, between ymin and ymax
Graph.Legend = boolean ' true or false to include or exclude a
Legend
Graph.SetXAxisTitleAndUnits "title", "units" ' optional; adds
title (units) to the X axis
Graph.SetYAxisTitleAndUnits "title", "units" ' optional; adds
title (units) to the Y axis
Graph.XAxisMode =  ' 0 gives linear axis; 1 gives log
Graph.YAxisMode =  ' 0 gives linear axis; 1 gives log
Graph.BackgroundColor = RGB(redvalue, greenvalue, bluevalue) '
each value from 0 to 255
Graph.TraceColor("Data Name") = RGB(redvalue, greenvalue,
bluevalue) ' each value from 0 to 255
Graph.TraceStyle("Data Name") =  ' 1 gives no trace; 0 gives a
trace; 2 gives a histogram

Graph.SymbolStyle("Data Name") =  ' 1 gives ELLIPSE
 ' 2 gives UPTRIANGLE
 ' 3 gives RECTANGLE
 ' 4 gives DIAMOND
 ' 5 gives DOWNTRIANGLE
 ' 6 gives ELLIPSE_OUTLINE
 ' 7 gives UPTRIANGLE_OUTLINE
       ' 8 gives RECTANGLE_OUTLINE
 ' 9 gives DIAMOND_OUTLINE
       ' 10 gives DOWNTRIANGLE_OUTLINE
```

```
        ' 11 gives CROSS
        ' 12 gives PLUS
        ' 13 gives MINUS
 ' 14 gives VERTICALBAR

Graph.SymbolSize("Data Name") =  ' numerical diameter of the
symbols
Graph.SymbolColor("Data Name") = RGB(redvalue, greenvalue,
bluevalue) ' set the color of the symbols
Graph.PointSymbolSize("Data Name", pointnumber) =  ' numerical
diameter for a particular pointnumber symbol
Graph.PointSymbolColor("Data Name", pointnumber) = RGB( , , ) '
RGB color for a particular pointnumber symbol
Graph.PointSymbolStyle("Data Name", pointnumber) = stylenumber
' style for a particular pointnumber symbol
Graph.Symbols("Data Name") = boolean ' true to include symbols,
false to exclude
Graph.AutoScaleX ' include to scale the graph automatically in
the X direction
Graph.AutoScaleY ' include to scale the graph automatically in
the Y direction
Graph.FullScaleX ' include to draw the graph with its full X
axis range
Graph.FullScaleY ' include to draw the graph with its full Y
axis range
Once the graph is defined, one needs to obtain its image. This
is done as follows:
Dim nWidth, nHeight, sTmp, sName
nWidth = 300
nHeight = 300
sName = graph.Image(nWidth, nHeight)
If sName<>"" then
sTmp = "<img src=" & sName & ">"
End If
GraphObj.InnerHTML= sTmp
```

You extract the image using the graph.image construction, while specifying the image width and height. You then create an HTML text string using the <img> tag, which is used in HTML for displaying images. Finally, you assign this HTML to an object, which in this case you call GraphObj, and which you display using the construction in the <body> tag.

Exercise 27: Using the Graph Object To Plot Data

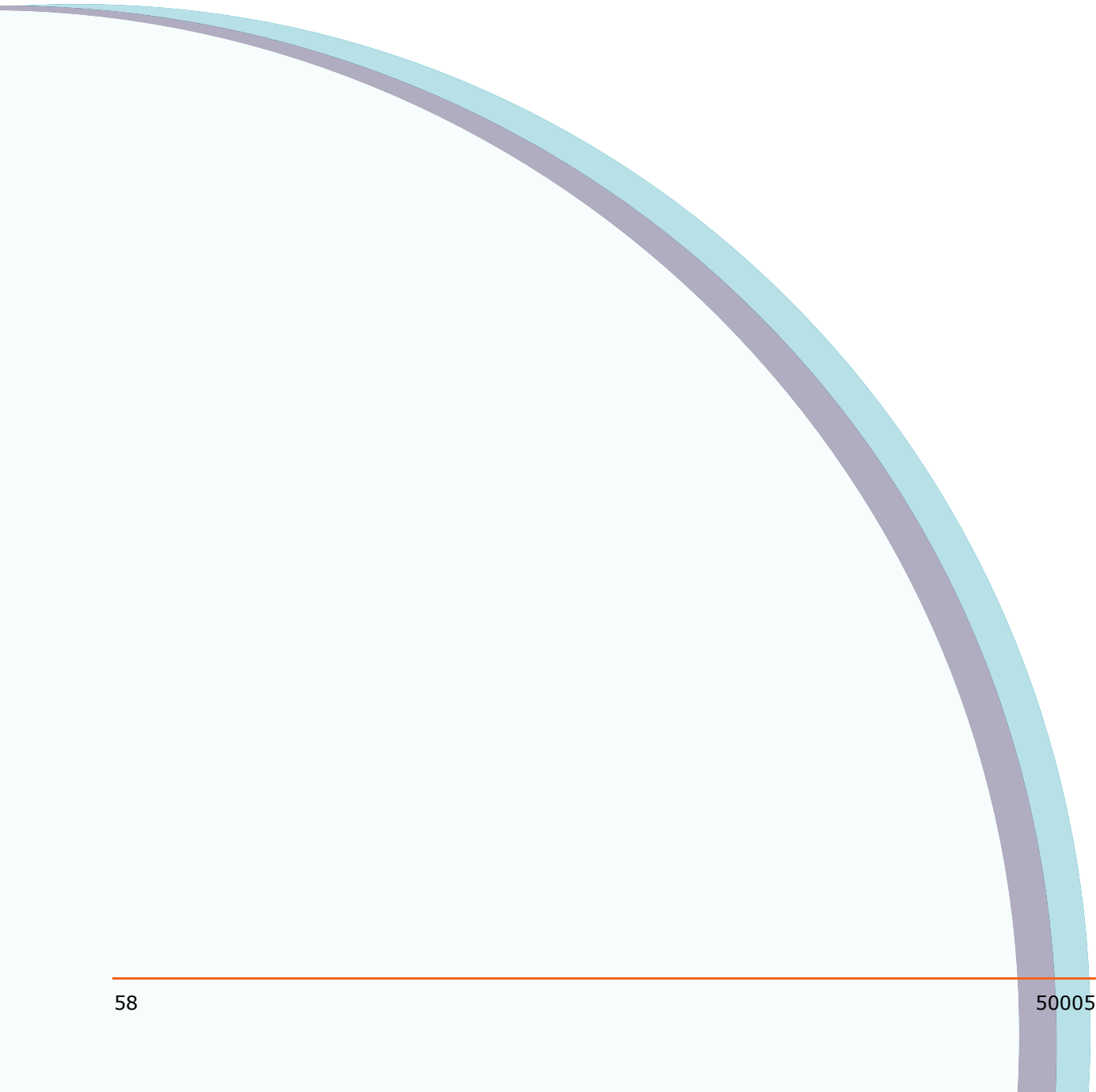
Plot and display the values of F635 Median versus F532 Median. You can copy the code directly from the above example and customize it.

For the solution to this exercise see Exercise 27 on page 193.

Exercise 28: Create A Histogram Of Binned Data

Building on the binning exercise, plot a histogram of the binned data of F635 Median.

For the solution to this exercise see Exercise 28 on page 195.



Access to Other Tabs

6

This chapter contains the following topics:

Image Tab

ImageTab.image(number), ImageTab.ImageWidth, ImageTab.ImageHeight

Returns the file name for a given image, which is typically placed in a temporary directory.

```
0        preview 1 image;
1        preview 2 image;
2        preview 3 image;
3        wavelength 1 image;
4        wavelength 2 image;
5        wavelength 3 image;
6        wavelength 4 image;
7        ratio 1 image;
8        ratio 2 image;
9        ratio 3 image.
```

To get all three images and to set their width and height, consider the following code:

```
 Dim nImageWidth, nImageHeight, fAspectRatio
 nImageWidth = ImageTab.ImageWidth()
 nImageHeight = ImageTab.ImageHeight()
 fAspectRatio = nImageHeight / nImageWidth
 Dim i, sName
Dim i, Name

      // ratio 1
      Name = ImageTab.Image(7)
      document.images(0).src = Name
      document.images(0).width = 180
      document.images(0).height = (180 * AspectRatio)

     // Wavelength 1
      Name = ImageTab.Image(3)
      document.images(1).src = Name
      document.images(1).width = 180
      document.images(1).height = (180 * AspectRatio)
```

```
// Wavelength 2
Name = ImageTab.Image(4)
document.images(2).src = Name
document.images(2).width = 180
document.images(2).height = (180 * AspectRatio)
```

In the new code in this example, the document.images construction is an array of all the <img> objects contained by the document. In the example above, the src suffix to the document.images() construction sets the file source for the images, while the width and height suffixes are self-explanatory.

To display the images, you must place one <img> (image) tag in the HTML layout section of the document for each image in the document.images() array. For example, if you are using a table for layout, you might use the following code to display two images:

```
<tr>
     <td><img></td>
     <td><img></td>
</tr>
```

Exercise 29: Display Images In A Report

Write a script that displays the three images from the Image tab. Arrange them horizontally across the Report tab.

For the solution to this exercise see Exercise 29 on page 197.

ImageTab.CurrentImage, ImageTab.CurrentImageWidth, ImageTab.CurrentImageHeight

Returns the current Image tab display. This method allows you to obtain, for example, zoomed images.

Exercise 30: Display Zoomed Images

Modify the script from the previous exercise to replace the ratio image with a zoomed ratio image.

ImageTab.AutoScaleDisplaySettings

Auto scales the brightness and contrast to display the image optimally. Use this command before exporting images to a Report so that the image is set with the correct brightness and contrast. This is the same as Auto Scale Display Settings.

For the solution to this exercise see Exercise 30 on page 198.

Exercise 31: Display Auto Scaled Images

Modify the script from the previous exercise to AutoScale the display settings of each image.

Histogram Tab and Scatter Plot Tab

The GenePix® Pro Object Model provides access to the current Histogram image and ScatterPlot image.

For the solution to this exercise see Exercise 31 on page 200.

Histogram.SaveImage(width, height), ScatterPlot.SaveImage(width, height)

Returns the currently displayed images from the Histogram and Scatter Plot tabs, respectively.

Exercise 32: Displaying Images

Write a Report that displays the current Histogram and Scatter Plot images.

For the solution to this exercise see Exercise 32 on page 201.

Histogram.FullScale, ScatterPlot.AutoScale

FullScale in the Histogram tab sets both the X and the Y axes to full scale. AutoScale in the Scatter Plot tab autoscales the display.

Exercise 33: Mixing Auto Scale and Full Scale Images

Modify the script from the previous exercise so that the Histogram is displayed at full scale and the Scatter Plot is autoscaled.

For the solution to this exercise see Exercise 33 on page 202.

Lab Book Tab

Unlike the other tabs, the Lab Book tab does not have an associated Object, but there is a single method in the GenePix Object model that applies to the Lab Book. You can add a comment to the Lab Book using the LogComment method:

```
Dim GenePix
GenePix = window.external
GenePix.LogComment("Insert your comment here")
```

Exercise 34: Adding Lab Book Results To Custom Report

Add the above code to your Report and verify that it adds a comment to the Lab Book.

For the solution to this exercise see Exercise 34 on page 204.

Report Tab

A number of Report tab functions are themselves scriptable.

Report.ExportReport("file path and name")

The Export HTML method is scriptable. Once a report has been generated, you can automatically save it as HTML. Any image files, such as bitmaps of graphs and images, are also saved in the destination directory. If there is a file with the same name in the destination directory, it will be overwritten without prompting:

```
Report.ExportReport("c:/temp/file_name.htm")
```

Exercise 35: Export Report Tab Information

Add the above code to the end of one of your scripts and check that Report export works.

For the solution to this exercise see Exercise 35 on page 205.

Report.Busy = boolean

Setting Report.Busy to true sends a 'busy' message to the GenePix Pro application and sets in motion the GenePix Pro icon on the Report tab; setting Report.Busy to false stops the icon. Use Report.Busy in computation-intensive reports when you want to inform the user that computations are proceeding.

Report.Refresh

Reloads the current report.

Note: It is very easy to create an infinite loop with this command! If you write a simple report that includes the line Report.Refresh without any qualification, you will create an infinite loop.

Report.Print

Prints the current report, using the current default system printer. Unless there is a configuration problem with the printer, the print job goes ahead without any further prompting. For this reason, make sure that your printer settings are correct before using the Report.Print command.

Report.Status = "status message", Report.StatusColor = RGB(, ,)

Writes status messages at the bottom of the Report tab:

```
Report.StatusColor = RGB(255,0,0)
Report.Status = "Calculating"
To remove a status message, you need to write:
Report.Status = ""
```

Automated Acquisition

7

This chapter contains the following topics:

So far you have concentrated on reports for manipulating data that has already been acquired. This chapter explains how to use script acquisitions in GenePix® Pro software.

GenePix.PreviewScan(), GenePix.DataScan(), GenePix.DualScan(), GenePix.StopScan()

These four commands start and stop GenePix Pro acquisitions. A Dual scan is a Preview scan followed by a Data Scan.

You might be tempted to think that the following code:

```
GenePix.DualScan()
```

is equivalent to:

```
GenePix.PreviewScan()
GenePix.DataScan()
```

but this is not correct, and the lesson to be learned is a general one that applies to all acquisition scripting.

Exercise 36: Scripted Data Acquisitions

Verify that the above code examples are not equivalent; that is, show that they give different behavior in the GenePix Pro program (you might want to use a small scan area to speed up the execution).

For the solution to this exercise see Exercise 36 a on page 207, Exercise 36 b on page 207, and Exercise 36 c on page 208.

GenePix® Pro Callbacks

This next section discusses scripted processes. Scripting a process is potentially much more complicated than performing calculations on existing data, because one process can interfere with another. The script does not know to wait before beginning the second process. When scripting two acquisition processes like a Preview scan and a Data scan, you should call the second process only when the first is complete. If you call the second immediately after the first, the second overrides the first. Fortunately, there is a way of determining when a process is finished.

GenePix.OnScanDone(), GenePix.OnPreviewDone(), GenePix.OnAnalyzeDone(), GenePix.OnAutoAlignDone(), GenePix.OnScanAbort()

A callback is a message from the GenePix Pro program informing you that a process is finished. There are five messages, each indicating the completion of:

- data scan
- preview scan
- completed analysis
- auto alignment
- a scan stopped manually by the user

OnPreviewDone can be used for checking a scanned barcode immediately after a Preview Scan, while OnAnalyzeDone can be used for opening a Quality Control Report after an analysis. Callbacks become powerful when used in conjunction with Workflow options.

Unfortunately, these callbacks only work within JavaScript. You can get around this shortcoming by defining some callback functions in JavaScript that can themselves be referenced in VBScript:

```
<script language="VBScript">
Function OnScanDoneVB
      'code to execute after the data scan
End Function
Function OnPreviewDoneVB
      'code to execute after the preview scan
End Function
Function OnAnalyzeDoneVB
      'code to execute after the analysis
End Function
Function OnAutoAlignDoneVB
      'code to execute after the auto align
End Function
Function OnScanAbort
      'code to execute after a data scan has been stopped by
the user.
End Function
</script>
<script language="JavaScript">
GenePix.OnScanDone = function () { OnScanDoneVB(); }
GenePix.OnPreviewDone = function () { OnPreviewDoneVB(); }
GenePix.OnAnalyzeDone = function () { OnAnalyzeDoneVB(); }
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
GenePix.OnScanAbort = function () { OnScanAbortVB(); }
</script>
```

In the code above, there are both a VBScript section and a JavaScript section. The VBScript section uses callback functions defined in the JavaScript section. All you have to do is copy the JavaScript code into your HTML document, and then use the correct callback names in your VBScript. See also OnFeatureValues on page 100, OnBackgroundValues on page 99.

Because the JavaScript functions are not themselves wrapped in a function, they will be registered as soon as the script is loaded. If you wish to delay their registration, the call can wrap them in a function, and then call that JavaScript function from within the main VBScript code:

```
<script language="VBScript">
call initializecallbacks()
</script>
<script language="JavaScript">
function initializecallbacks () {
GenePix.OnScanDone = function () { OnScanDoneVB(); }
GenePix.OnPreviewDone = function () { OnPreviewDoneVB(); }
GenePix.OnAnalyzeDone = function () { OnAnalyzeDoneVB(); }
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
GenePix.OnScanAbort = function () { OnScanAbortVB(); }
}
</script>
```

Finally, once your automated acquisition is complete, you may wish to unregister these callback functions. You can do this using the ClearCallbacks method.

Exercise 37: Code To Alert Script Data Scan Is Finished

Copy the above code into a script. Run a dual scan, and once it is finished use the OnScanDoneVB sub to alert "Data Scan Finished."

For the solution to this exercise see Exercise 37 on page 209.

SwitchToTab

GenePix.SwitchToTab(n)

You may have noticed when running a scan from the Report tab that the focus does not change to the Image tab. However, switching between tabs is scriptable. The tabs are numbered in a zero-based list from left to right, so that the Image tab is numbered 0.

Exercise 38: Set Focus On Image Tab When Scan Begins

Modify the script from the previous exercise so that the focus switches to the Image tab when a scan begins.

For the solution to this exercise see Exercise 38 on page 210.

Workflow Options

GenePix.Option(n) = 0 or 1

If you open the GenePix Pro Options dialog box and look at the Workflow tab, you will see a number of interesting options listed under ‘Automatically’. With some appropriate choices here, you can script a combined acquisition and analysis very simply, because feature alignment and analysis can be done automatically.

Select the third option, 'Align features after a data scan'. Notice that the fourth option changes. Select the fourth option. Notice that this makes the fifth option accessible.

The selecting and clearing of these options is scriptable. To select an option, set its value to 1; to clear an option, set it to 0.

Note: The options form a zero based list, so that 'Save images after a data scan' is number 0.

Exercise 39: Automatically Acquire and Analysis Data

Building on the previous exercise, write a script that automatically does a Preview scan, and a Data scan, aligns features, and analyzes them.

For the solution to this exercise see Exercise 39 on page 211.

Opening A Settings File

GenePix.LoadFile("file path and name")

The script you wrote in the above exercise goes some of the way towards fully automating acquisition and analysis. One further refinement is to load a custom settings file before the acquisition begins. The LoadFile command allows you to do that.

Exercise 40: Load Custom Settings

Building on the previous exercise, load a custom settings file to work with the demo data that does the following: defines a scan area that covers just the first block of features, and defines just one block that is roughly placed over the first block.

For the solution to this exercise see Exercise 40 on page 212.

Cleaning Up

GenePix.DiscardImages(), GenePix.DiscardResults(), GenePix.DiscardSettings()

While writing and testing the code for Exercise 40, you may have noticed that the GenePix Pro program prompts the user to save images, settings and results before acquiring new data. Since this may get in the way of an automated acquisition, you can use the Discard commands to avoid these message boxes.

Exercise 41: Remove Message Boxes

Use the Discard commands to eliminate any message boxes from the automated acquisition script you wrote in Exercise 40.

For the solution to this exercise see Exercise 41 on page 213.

PMT Settings

You can set the values of the PMT Gain before scanning a script, using the Scanner object:

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
' This is a zero-based list!
Scanner.PMT(0) = 500
Scanner.PMT(1) = 600
```

The PMTs are numbered with a zero-based list, so that the first PMT is 0, and the second is 1.

Exercise 42: Setting PMT to Defined Values

Modify your automation script so that the PMTs are set to 600 and 650 before scanning.

Write an automation script that:

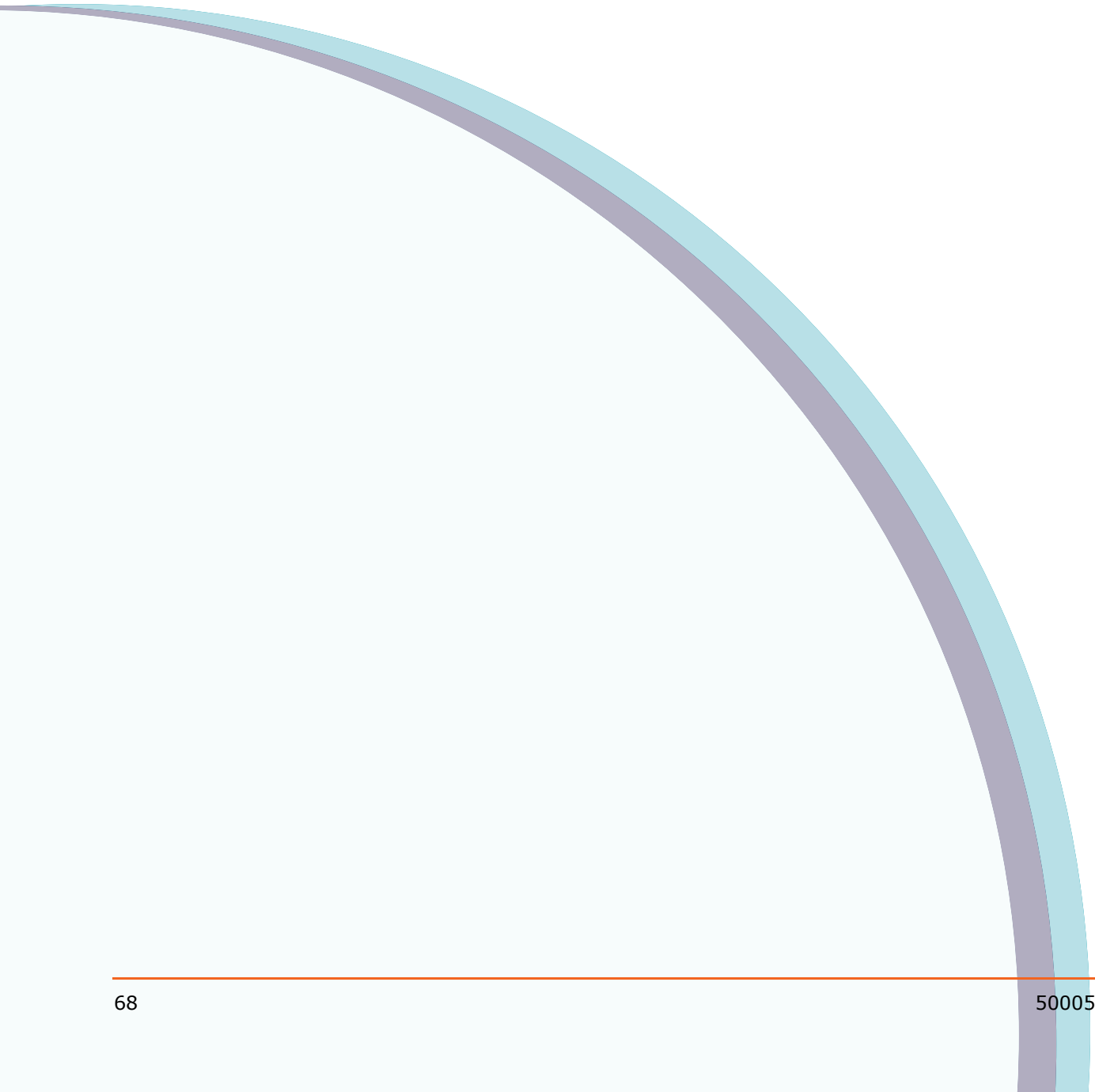
- scans a single block and analyzes the data
- runs the data through a Quality Control report

If the data passes the Quality Control tests it then:

- scans the whole image
- analyzes the data
- presents a Quality Control report of the whole data slide

This is essentially the same as the Automation script, listed in the 'Advanced Scripts' section of the Report Tab's Home page.

For the solution to this exercise see Exercise 42 on page 214.



GenePix® Pro Software Scripting in Python

8

If you already know Python, you can apply your knowledge to script GenePix® Pro software.

First, you must install a recent build of Python that includes support for ActiveX scripting. You can obtain this for free from www.activestate.com.

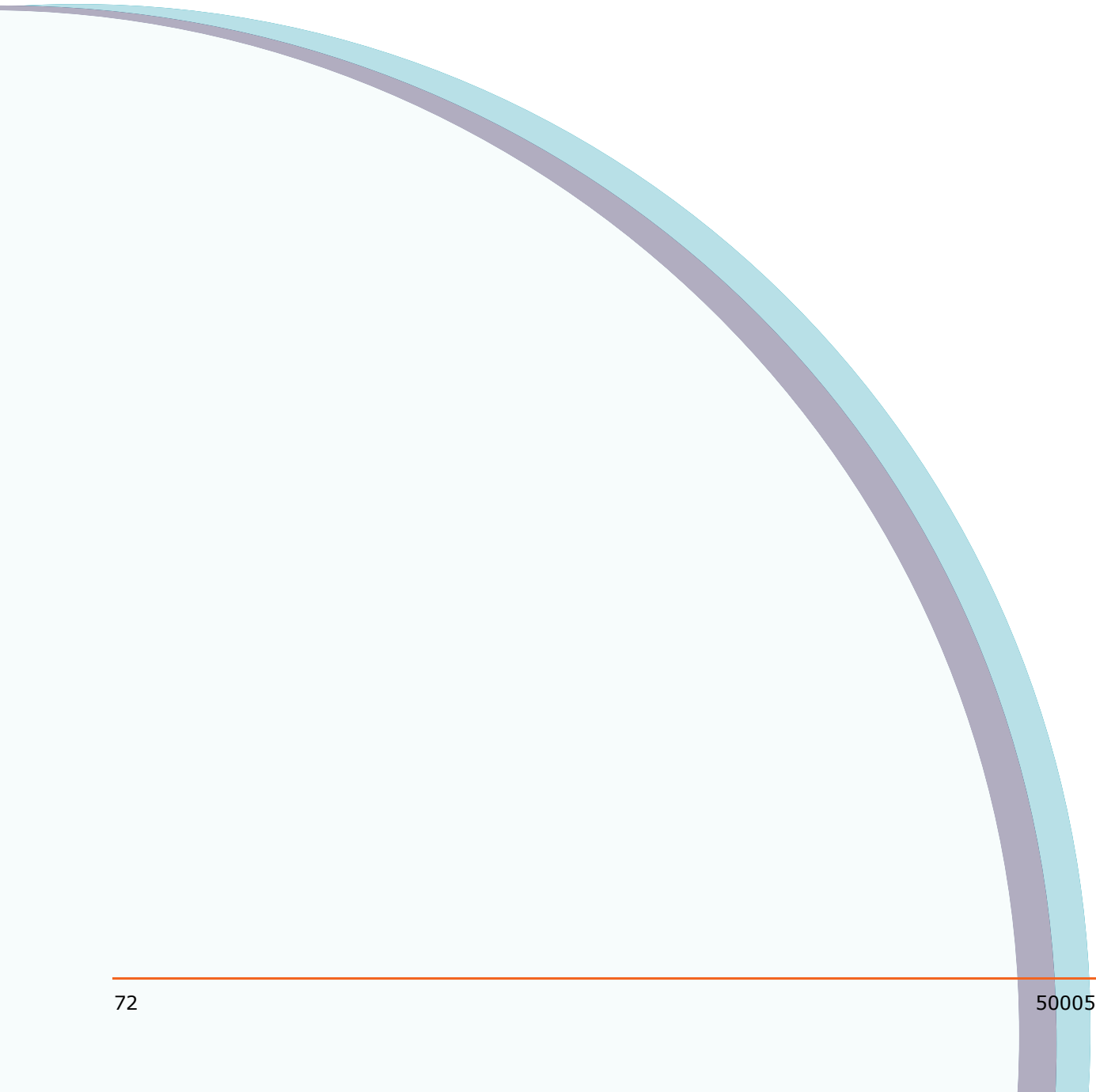
Once Python is installed, you will be able to manipulate GenePix Pro data and acquisition directly with Python.

The following sample HTML page uses Python to demonstrate some of the main parts of the GenePix Pro Object Model. Copy and paste the code into a text file and save it as Python_Test.htm in your Axon/Params directory. Then you can open it in the GenePix Pro Report tab and run it.

```
<head>
     <style type="text/css">
         @import url(GenePix_Style_Base.css);
     </style>

     <title>Python Test</title>
</head>

<body language="python" marginheight="0" marginwidth="0"
topmargin="0" leftmargin="0">

<!-- HTML Layout portion --><form id="GeneForm">

<table width=600 border=0 cellspacing=0 cellpadding=5>
     <tr class="title">
         <td>
              <p class="heavy">Python Test
         </td>
     </tr>
     <tr>
         <td class="underline">
              This Report tests the operation of the scripting
language Python in GenePix.<br>
              It gets data from the Results tab, does a Preview
Scan and tests a callback function.<br>
              Open a GenePix Results file that contains an
image and press <input type="button" value="Start"
onclick="fnResults()" style="font-size:9pt">.<br>
              <b>Note:</b> for this script to work, you need to
have a third-party Python scripting engine installed.
         </td>
     </tr>
     <tr>
         <td>
              <b>Results Test</b><br><br>
              <b>Results file name:</b> <span
id="spnResultsFileName"></span>
              <br>
```

```
                <b>Number of rows in the current Results
file:</b> <span id="spnRows"></span>
                <br>
                <b>First entry in Ratio of Medians column:</b>
<span id="spnRM"></span>
                <br>
            </td>
        </tr>
        <tr>
            <td>
                <b>Results Tab Image: </b>
                <div id="divImage"></div>
            </td>
        </tr>
</table>

</form>
<script language="python">
# get GenePix references
GenePix = window.external
Results = GenePix.Results

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
# FUNCTION: fnResults()
# This function works through some Results tab functionality
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++

def fnResults():
   # get Results file name
   sResultsFileName = Results.FileName
   window.spnResultsFileName.innerText = sResultsFileName

   # get number of rows in Results
   nHeight = Results.Height
   window.spnRows.innerText = nHeight

   # get Results column and display a value
   # note, this is returned as a tuple
   sColumnName = "Ratio of Medians"
   asTmp = Results.Column(sColumnName)
   nValue0 = asTmp[0]
   window.spnRM.innerText = nValue0

   # call the image test
   fnImage()

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
# FUNCTION: fnImage()
# This function displays the current image from the Results tab
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
```

```
def fnImage():
   # get the image
   sImageSource = Results.Image(0)

   # display the image
   window.divImage.innerHTML = "<img src=" + sImageSource + ">"

   # start a preview scan
   GenePix.PreviewScan()
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
# FUNCTION: OnPreviewDonePython()
# This function is a callback defined in javascript: it runs
when a preview scan is finished
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++

def OnPreviewDonePython():
   # get the current image
   alert("Preview finished: callback successful.")

</script>
<script language="JavaScript">
//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
// We need to use JavaScript to register callback functions.
// The following function registers our Python handlers.
//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
GenePix.OnPreviewDone = function () { OnPreviewDonePython(); }
</script>
</body>
</html>
```



GenePix® Pro Software Scripting in PerlScript

9

If you already know Perl, you can apply your knowledge to script GenePix® Pro software.

First, you must install a recent build of Perl that includes support for ActiveX scripting. You can obtain this for free from www.activestate.com.

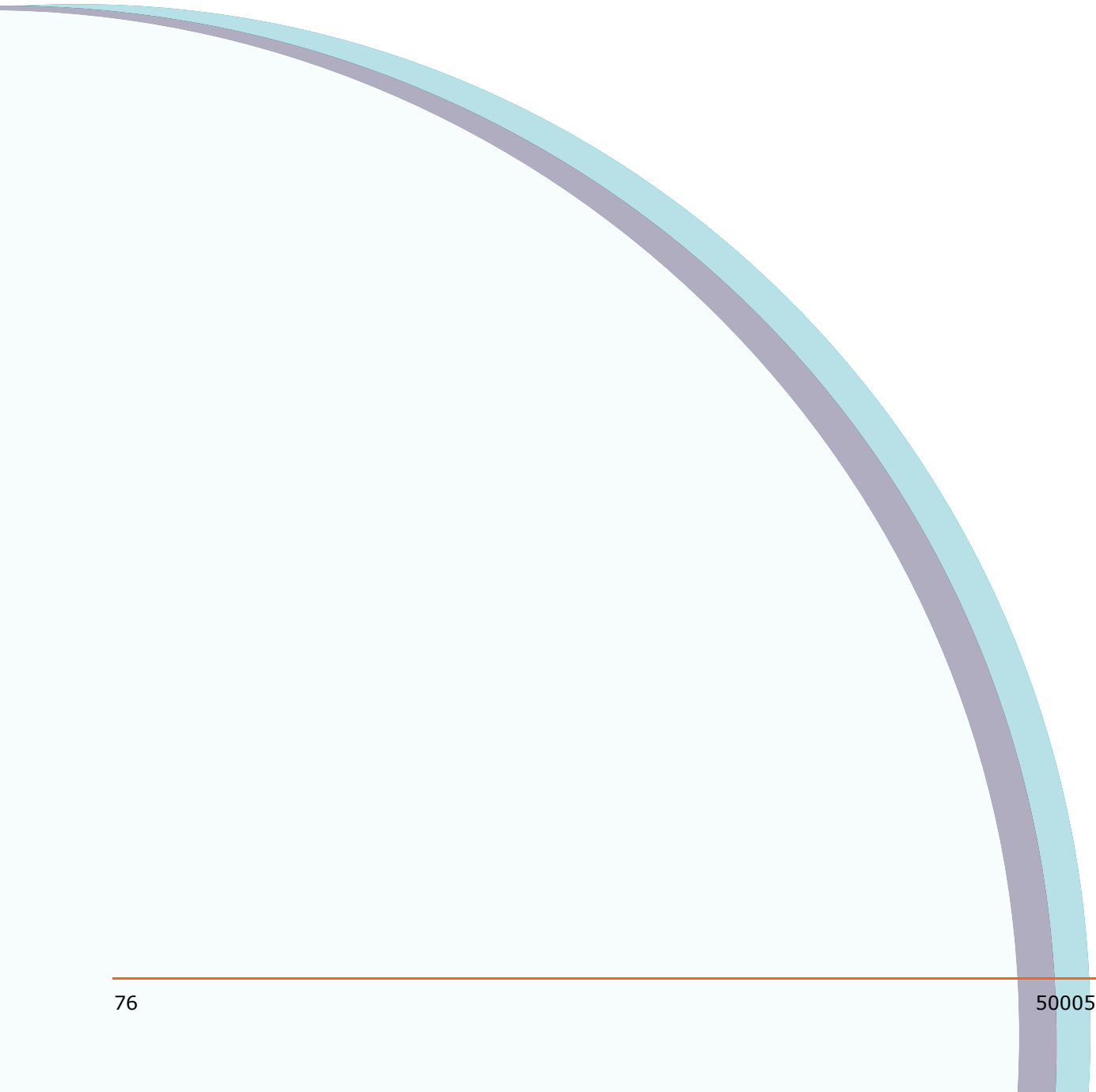
Once Perl is installed, you will be able to manipulate GenePix Pro data and acquisition directly with PerlScript.

The following sample HTML page uses PerlScript to demonstrate some of the main parts of the GenePix Pro Object Model. Copy and paste the code into a text file and save it as PerlScript_Test.htm in your Axon/Params directory. Then you can open it in the GenePix Pro Report tab and run it.

```
<head>
     <style type="text/css">
         @import url(GenePix_Style_Base.css);
     </style>

     <title>PerlScript Test</title>
</head>

<body language="perlscript" marginheight="0" marginwidth="0"
topmargin="0" leftmargin="0">

<!-- HTML Layout portion -->
<form id="GeneForm">

<table width=600 border=0 cellspacing=0 cellpadding=5>
     <tr class="title">
         <td>
              <p class="heavy">PerlScript Test
         </td>
     </tr>
     <tr>
         <td class="underline">
              This Report tests the operation of the scripting
language PerlScript in GenePix Pro.<br>
              It gets data from the Results tab, does a Preview
Scan and tests a callback function.<br>
              Open a GenePix Results file that contains an
image and press <input type="button" value="Start"
onclick="Start()" style="font-size:9pt">.<br>
              <b>Note:</b> for this script to work, you need to
have a third-party PerlScript scripting engine installed.
         </td>
     </tr>
     <tr>
         <td>
              <b>Results Test</b><br><br>
              <b>Results file name:</b> <span
id="spnResultsFileName"></span>
```

```
                <br>
                <b>Number of rows in the current Results
file:</b> <span id="spnRows"></span>
                <br>
                <b>First entry in Ratio of Medians column:</b>
<span id="spnRM"></span>
                <br>
            </td>
        </tr>
        <tr>
            <td>
                <b>Current Results Tab Image: </b>
                <div id="divImage"></div>
            </td>
        </tr>
</table>

</form>
<script language="perlscript">

$GenePix = $window->external;
$Results = $GenePix->Results;

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
# FUNCTION: Start()
# This function runs when the Start button is clicked.
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++

sub Start() {

   # get the Results file name and display it
   $FileNameValue = $Results->FileName;
   $window->{spnResultsFileName}->{innerText} = $FileNameValue;

   # get the number of rows in the Results tab and display it
   $numRows = $Results->Height;
   $window->{spnRows}->{innerText} = $numRows;

   # get first entry in Ratio of Medians column
   @RM = $Results->Column(26);
   foreach $RM (@RM) {
      $RatioMedians0 = $$RM[0];
   }
   $window->{spnRM}->{innerText} = $RatioMedians0;

   # put the current Results image in the Report window
   $ImageSource = $Results->Image(0);
   $window->{divImage}->{innerHTML} = ("<img
src=$ImageSource>");

   # do a Preview Scan
   $GenePix->PreviewScan;
```

```
}

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
# FUNCTION: OnPreviewDonePerl()
# this sub is called from the JavaScript callback when the
Preview Scan is done
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++

sub OnPreviewDonePerl() {
   $window->alert("Preview finished: callback successful.");
}

</script>

<script language="JavaScript">

//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
// We need to use JavaScript to register callback functions.
// The following function registers our PerlScript handlers.
//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++

GenePix = window.external;
GenePix.OnPreviewDone = function () { OnPreviewDonePerl(); }

</script>
</body>
</html>
```



GenePix® Pro Object References

A

This appendix contains the following GenePix® Pro object references:

AlignFeatures

AlignFeatures(bPreCorrelate)

Finds blocks and aligns features (that is, equivalent to Shift+F6). You can use the optional Boolean parameter to specify whether to find blocks before features. Set this parameter to True to find blocks.

Use FeatureShape on page 82 to choose circular, square or irregular feature-indicators, and use FillFeatureShape on page 83 to choose whether to fill irregular feature-indicators.

You can also use the individual commands FindArray, FindBlocks and FindFeatures.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.AlignFeatures ' finds blocks and features
GenePix.AlignFeatures(false) ' finds features only
```

Analyze

Analyzes the current images with the current Block positions.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.Analyze
```

ArrayListName

Returns the file path and name of the current loaded GenePix ArraytListName file.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.ArrayListName
```

Barcode

Barcode[= sBarcode]

Gets and sets the currently scanned barcode that is displayed in the 'B:' field of the Status Bar. (Barcode on page 135 returns the barcode from a loaded Results file.)

Example

```
Dim GenePix
Set GenePix = window.external
Dim sBarcode
' get barcode
sBarcode = GenePix.Barcode
alert(sBarcode)
' set barcode
GenePix.Barcode = "123456"
```

CheckVersion

CheckVersion(sVersion)

Compares a given version string with the version of the application. The current version of GenePix Pro software is reported in the Hardware Diagnostics dialog box, or is accessible by script using the GenePix Pro program. See Version on page 107.

If a positive number is returned, the current version is greater than the submitted string; if a negative number is returned, the current version is less than the submitted string.

"4" means the difference is in the major release number, "3" is a difference in the minor revision, "2" is a difference in the bugfix release number; "1" is a difference in the build.

```
For example, if GenePix Pro is at version 4.0.1.64:
GenePix.CheckVersion("3.0.6.70") returns 4;
GenePix.CheckVersion("4.0.1.64") returns 0;
GenePix.CheckVersion("4.0.0.64") returns 2.
```

ClearCallbacks

Clears all registered callback functions in a script. It is particularly important to use this function after the OnBackgroundValues on page 99 and OnFeatureValues on page 100 callbacks, otherwise these functions will be called after each feature is analyzed in an analysis.See Automated Acquisition on page 63.

DataScan

Starts a Data Scan.

See Automated Acquisition on page 63 for an account of scanning and using callbacks in script.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DataScan
```

DiscardImages

Discards all currently held images in the Image tab.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DiscardImages
```

DiscardResults

Discards all currently held results in the Results tab.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DiscardResults
```

DiscardSettings

Discards any changes made to the current settings and initializes new settings.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DiscardSettings
```

DualScan

Method of: GenePix Object, Scanner Object

DualScan

Executes a Dual Scan: a Preview Scan followed by a Data Scan.

See Automated Acquisition on page 63 for a description of scanning and using callbacks in script.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DualScan
```

Eject

Ejects the slide (returns the slide carriage from the laser scanning position to the slide loading position).

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.Eject
```

ExportImages

ExportImages sFilePath, sComment, bFlag

Exports the currently loaded images to disk as JPEGs. To save files as TIFFs, use SaveImages on page 104.

The file path string can be a relative path or an absolute path; if no file name string is included then the file is saved with the automatically assigned name. The comment string can be any comment. The flag is a hex number corresponding to various Save Image options:

```
&h001000&save the preview 1 image;
&h002000&save the preview 2 image;
&h004000&save the preview 3 image;
&h008000&save the wavelength 1 image;
&h010000&save the wavelength 2 image;
&h020000&save the wavelength 3 image;
&h040000&save the wavelength 4 image;
&h080000&save the ratio 1 image;
&h200000&save the ratio 2 image;
&h200000&save the ratio 3 image.
```

To perform a combination of these options, you need to concatenate the hex numbers, either by addition or by bitwise. (Hex numbers in VBScript are prefixed by &h.) For example, to save wavelength 1 and wavelength 2, use &h008000& or &h010000& for the flag value. (Adding an & at the end of each hex number forces VBScript to convert to a long variable type.)

Alternatively, you can use GenePix.SaveImages on page 104 as an unqualified statement and the images will be saved according to the currently-set options in the Save Images dialog box:

```
Dim GenePix
Set GenePix = window.external
' To export with default values:
GenePix.ExportImages
' To export with custom values:
GenePix.ExportImages "c:\Axon\Data\myImages.tif", "my images",
&h008000& or &h010000&
```

ExtraHeaders

Returns any user-defined header information read from the currently-loaded GAL file.

Example

```
Dim GenePix
Set GenePix = window.external
Dim UserDefinedString
UserDefinedString = GenePix.ExtraHeaders
alert(UserString)
```

FeatureShape

FeatureShape [= nValue]

Gets and sets the feature-indicator shape to find during AlignFeatures or FindFeatures:

```
* 0 is circular
* 1 is square
* 2 is irregular and filled
* 3 is irregular and not filled.
```

Example

```
Dim GenePix
Set GenePix = window.external
alert(GenePix.FeatureShape) ' alerts the current feature-
indicator setting
GenePix.FeatureShape = 2  ' sets the feature-indicator shape to
irregular, filled
```

FileNamingOptions

FileNamingOptions [=sOptions]

Gets and sets the various file-naming options in the Save Images and Save Results dialog boxes, that is, the use of user name, barcode, date and numeric prefix/suffix, as a comma-delimited string.

```
Example
Dim GenePix
Set GenePix = window.external
' set options
GenePix.FileNamingOptions = "username,barcode,date,numeric"
' get options
alert GenePix.FileNamingOptions
```

FileSystem

Returns a VBScript file system object.

You must have the scripting run-time library (scrrun.dll) installed: see Reading and Writing Text Files on page 46.

Example

```
Dim GenePix, fso
Set GenePix = window.external
Set fso = GenePix.FileSystem
```

FillFeatureShape

This function has now been superseded by FeatureShape on page 82.

FindAll

Performs an <F8> command: FindArray, Find Blocks, Align Features.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.FindAll
```

FindArray

Calls the Find Array command to position the whole set of blocks on an image over the array.

When the FindArray process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindArray, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindArray
```

Example 2

This example script demonstrates how to use FindArray, FindBlocks, FindFeatures and a callback to automate spot-finding. (You can just use FindAll to perform an <F8> command.)

```
<html>
<head>
<title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
Option Explicit

' create GenePix object
Dim GenePix
```

```
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
</body>
</html>
```

FindBlocks

Calls the Find Blocks command to position each block individually on an image.

When the FindBlocks process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindBlocks, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindBlocks
```

Example 2

This example script demonstrates how to use FindArray, FindBlocks, FindFeatures and a callback to automate spot-finding.

```
<html>
<head>
<title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
Option Explicit

' create GenePix object
Dim GenePix
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
```

```
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
</body>
</html>
```

FindFeatures

Calls the Align Features in All Blocks command.

To set the feature-indicator type use FeatureShape.

When the FindFeatures on page 95 process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindFeatures, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindFeatures
```

Example 2

This example script demonstrates how to use FindArray, FindBlocks, FindFeatures and a callback to automate spot-finding.

```
<html>
<head>
<title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
Option Explicit
```

```
' create GenePix object
Dim GenePix
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
</body>
</html>
```

HelpFile

Returns the full path to the GenePix Pro Help file.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.HelpFile
```

Histogram

Returns the Histogram automation interface, with which you can access properties and methods of the Histogram tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
```

ImageFileNames

Returns the file names of the currently-loaded images as a comma-delimited string.

Example

```
Dim GenePix
Set GenePix = window.external
Dim sName
sName = GenePix.ImageFileNames
alert(sName)
```

ImagePath

Gets and sets the save path for image files (that is, the current directory where images are saved).

Example

```
Dim GenePix
Set GenePix = window.external
' set image path
GenePix.ImagePath = "C:\Axon\Data\"
' get image path
Alert GenePix.ImagePath
```

ImageTab

Returns the ImageTab automation interface, with which you can access properties and methods of the Image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
```

LoadFile(sFilePath)

Loads the named file. You can load image files (*.tif, *.jpg), Settings files (*.gps), Results files (*.gpr), Array Lists (*.gal), and scripts (*.htm, *.html).

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.LoadFile("C:\Axon\Params\demo.gps")
```

To load more than one file at once, include each full path and file name in the string, separated by a | character. For example:

```
GenePix.LoadFile
"\Axon\Data\Demo_532_nm.tif|\Axon\Data\Demo_635_nm.tif"
```

LogComment(sComment)

Records a comment in the Lab Book.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.LogComment("Insert your comment here")
```

LogPerformanceData

Logs a string to the last column of the performance log. This column is reserved for calibration intensities generated by the Sensitivity Calibration script. See PerformanceLog on page 111.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.LogPerformanceData(“6574,5487”)
```

NextImageName

Returns the next image name that will be generated for image saving, together with all the file prefixes and suffixes.

See also FileNamingOptions on page 82.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.NextImageName
```

ExportImages

ExportImages sFilePath, sComment, bFlag

Exports the currently-loaded images to disk as JPEGs. To save files as TIFFs, use SaveImages on page 104.

The file path string can be a relative path or an absolute path; if no file name string is included then the file is saved with the automatically-assigned name. The comment string can be any comment. The flag is a hex number corresponding to various Save Image options:

```
&h001000&save the preview 1 image;
&h002000&save the preview 2 image;
&h004000&save the preview 3 image;
&h008000&save the wavelength 1 image;
&h010000&save the wavelength 2 image;
&h020000&save the wavelength 3 image;
&h040000&save the wavelength 4 image;
&h080000&save the ratio 1 image;
&h200000&save the ratio 2 image;
&h200000&save the ratio 3 image.
```

To perform a combination of these options, you need to concatenate the hex numbers, either by addition or by bitwise or. (Hex numbers in VBScript are prefixed by &h.) For example, to save wavelength 1 and wavelength 2, use &h008000& or &h010000& for the flag value. (Adding an & at the end of each hex number forces VBScript to convert to a long variable type.)

Alternatively, you can use SaveImages on page 104 as an unqualified statement and the images will be saved according to the currently-set options in the Save Images dialog box:

```
Dim GenePix
Set GenePix = window.external
' To export with default values:
GenePix.ExportImages
' To export with custom values:
GenePix.ExportImages "c:\Axon\Data\myImages.tif", "my images",
&h008000& or &h010000&
```

ExtraHeaders

Returns any user-defined header information read from the currently-loaded GAL file.

Example

```
Dim GenePix
Set GenePix = window.external
Dim UserDefinedString
UserDefinedString = GenePix.ExtraHeaders
alert(UserString)
```

FeatureShape

FeatureShape [= nValue]

Gets and sets the feature-indicator shape to find during AlignFeatures or FindFeatures:

```
* 0 is circular
* 1 is square
* 2 is irregular and filled
* 3 is irregular and not filled.
```

Example

```
Dim GenePix
Set GenePix = window.external
alert(GenePix.FeatureShape) ' alerts the current feature-
indicator setting
GenePix.FeatureShape = 2  ' sets the feature-indicator shape to
irregular, filled
```

FileNamingOptions

FileNamingOptions [=sOptions]

Gets and sets the various file-naming options in the Save Images and Save Results dialog boxes, that is, the use of user name, barcode, date and numeric prefix/suffix, as a comma-delimited string.

Example

```
Dim GenePix
Set GenePix = window.external
' set options
GenePix.FileNamingOptions = "username,barcode,date,numeric"
' get options
alert GenePix.FileNamingOptions
```

FileSystem

Returns a VBScript file system object.

You must have the scripting run-time library (scrrun.dll) installed: see Reading and Writing Text Files on page 46.

Example

```
Dim GenePix, fso
Set GenePix = window.external
Set fso = GenePix.FileSystem
```

FillFeatureShape

This function has now been superseded by FeatureShape on page 82.

FindAll

Performs an <F8> command: FindArray on page 83, Find FindBlocks on page 85, AlignFeatures on page 78.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.FindAll
```

FindArray

Calls the Find Array command to position the whole set of blocks on an image over the array.

When the FindArray process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindArray, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindArray
```

Example 2

This example script demonstrates how to use FindArray, FindBlocks, FindFeatures and a callback to automate spot-finding. (You can just use FindAll on page 83 to perform an <F8> command.)

```
<html>
<head>
          <title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
Option Explicit

' create GenePix object
```

```
Dim GenePix
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
</body>
</html>
```

FindBlocks

Calls the Find Blocks command to position each block individually on an image.

When the FindBlocks process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindBlocks, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindBlocks
```

Example 2

This example script demonstrates how to use FindArray, FindBlocks, FindFeatures and a callback to automate spot-finding.

```
<html>
<head>
<title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
Option Explicit

' create GenePix object
Dim GenePix
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
```

```
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
</body>
</html>
```

FindFeatures

Calls the Align Features in All Blocks command.

To set the feature-indicator type use FeatureShape.

When the FindFeatures process is finished it executes the callback OnAutoAlignDone on page 99. If you need to execute any commands after FindFeatures, then you have to use this callback. See Example 2 below on how to use callbacks.

Example 1

```
Dim GenePix
Set GenePix = window.external
GenePix.FindFeatures
```

Example 2

This example script demonstrates how to use FindArray on page 83, FindFeatures on page 86, FindFeatures on page 95 and a callback to automate spot-finding.

```
<html>
<head>
<title>Automated Spot-Finding Example</title>
</head>
<body language="vbscript">
<input type="button" value="Start" onclick="Start()">

<script language="vbscript">
```

```
Option Explicit

' create GenePix object
Dim GenePix
Set GenePix = window.external

' declare global counter variable
Dim count

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Start()

' register the callback function
call InitializeCallbacks()

' initialize counter
count = 0

' find array
GenePix.FindArray

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function OnAutoAlignDoneVB()

' increment counter
count = count + 1

' decide which command to execute next
Select Case count
Case 1
GenePix.FindBlocks
Case 2
GenePix.FindFeatures
End Select

End Function

</script>

<script language="JavaScript">

// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
// We need to use JavaScript to register callback functions.
// The following function registers our VBScript handlers.
// ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

function InitializeCallbacks()
{
GenePix.OnAutoAlignDone = function () { OnAutoAlignDoneVB(); }
}

</script>
```

```
</body>
</html>
```

HelpFile

Returns the full path to the GenePix Pro Help file.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.HelpFile
```

Histogram

Returns the Histogram automation interface, with which you can access properties and methods of the Histogram tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
```

ImageFileNames

Returns the file names of the currently loaded images as a comma-delimited string.

Example

```
Dim GenePix
Set GenePix = window.external
Dim sName
sName = GenePix.ImageFileNames
alert(sName)
```

ImagePath

Gets and sets the save path for image files (that is, the current directory where images are saved).

Example

```
Dim GenePix
Set GenePix = window.external
' set image path
GenePix.ImagePath = "C:\Axon\Data\"
' get image path
Alert GenePix.ImagePath
```

ImageTab

Returns the ImageTab automation interface, with which you can access properties and methods of the Image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
```

LoadFile

LoadFile(sFilePath)

Loads the named file. You can load image files (*.tif, *.jpg), Settings files (*.gps), Results files (*.gpr), Array Lists (*.gal), and scripts (*.htm, *.html).

Example

Dim GenePix

```
Set GenePix = window.external
GenePix.LoadFile("C:\Axon\Params\demo.gps")
To load more than one file at once, include each full path and
file name in the string, separated by a | character. For
example:
GenePix.LoadFile?Axon\Data\Demo_532_nm.tif|\Axon\Data\Demo_635_
nm.tif"
```

LogComment(sComment)

Records a comment in the Lab Book.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.LogComment("Insert your comment here")
```

LogPerformanceData

Logs a string to the last column of the performance log. This column is reserved for calibration intensities generated by the Sensitivity Calibration script. See PerformanceLog on page 111.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.LogPerformanceData(?574,5487?
```

NextImageName

Returns the next image name that will be generated for image saving, together with all the file prefixes and suffixes.

See also FileNamingOptions on page 82.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.NextImageName
```

OnAnalyzeDone

Callback when an analysis is completed. See Automated Acquisition on page 63 for how to use this command.

OnAutoAlignDone

Callback when an Auto Align is completed. See Automated Acquisition on page 63 for how to use this command.

OnBackgroundValues

Callback during an analysis that returns an array of background pixel values for a feature. As with all GenePix Pro callbacks, you need to register the callback in JavaScript and implement it in VBScript. To use the VBScript function, start an analysis, either from within the script or from the Analyze button, and OnBackgroundValues is called after each feature is analyzed.

The VBScript function returns five parameters: the block, column and row to which the feature belongs, the image from which the pixel values come (1 or 2) and an array of background pixel values:

```
<script language="vbscript">
Function OnBackgroundInfoVB(nBlock, nRow, nCol, nImage, anData)
 document.writeln("Feature:" & CStr(nBlock) & "," & CStr(nRow)
& "," & CStr(nCol) & " Image:" & CStr(nImage))
End Function
</script>
<script language="JavaScript">
GenePix.OnBackgroundValues = function (nBlock, nRow, nCol,
nImage, anData) { OnBackgroundInfoVB(nBlock, nRow, nCol,
nImage, anData); }
</script>
```

After the analysis has finished, you should use ClearCallbacks on page 79 to clear the callback function, otherwise performing a normal analysis will call OnBackgroundValues in your script. OnBackgroundValues is used in the Feature Quality Control report in GenePix.

See Automated Acquisition on page 63.

OnFeatureValues

Callback during an analysis that returns an array of feature pixel values for a feature. As with all GenePix Pro callbacks, you need to register the callback in JavaScript and implement it in VBScript. To use the VBScript function, start an analysis, either from within the script or from the Analyze button, and OnFeatureValues is called after each feature is analyzed.

The VBScript function returns five parameters: the block, column and row to which the feature belongs, the image from which the pixel values come (1 or 2) and an array of feature pixel values:

```
<script language="vbscript">
     Function OnFeatureInfoVB(nBlock, nRow, nCol, nImage,
anData)
          document.writeln("Feature:" & CStr(nBlock) & "," &
CStr(nRow) & "," & CStr(nCol) & " Image:" & CStr(nImage))
     End Function
</script>

<script language="JavaScript">
     GenePix.OnFeatureValues = function (nBlock, nRow, nCol,
nImage, anData) { OnFeatureInfoVB(nBlock, nRow, nCol, nImage,
anData); }
</script>
```

After the analysis has finished, you should use ClearCallbacks on page 79 to clear the callback function, otherwise performing a normal analysis will call OnFeatureValues in your script. OnFeatureValues is used in the Feature Quality Control report in GenePix software.

See Automated Acquisition on page 63.

OnFlagFeaturesDone

Use Callback when a Flag Features query, called with Results.FlagFeatures on page 138, is completed.

Use Results.FlagFeaturesQueries on page 138 to obtain an array of strings containing the names of all saved queries from the from the Flag Features dialog box.

See Automated Acquisition on page 63 in the scripting tutorial on how to use this command.

OnPreviewDone

Callback when a Preview Scan is completed. See Automated Acquisition on page 63 for how to use this command.

OnScanAbort

Callback when a scan is stopped by pressing the Stop button. See Automated Acquisition on page 63 for how to use this command.

OnScanDone

Callback when a DataScan is completed.

See Automated Acquisition on page 63 for how to use this command.

OnScanStart

Callback when a DataScan is started.

See Automated Acquisition on page 63 for how to use this command.

Option

Option(nOption)[= bValue]

Gets and sets Workflow options from the "Automatically" list on the Options>Workflow tab. The list is zero-based, which means that the first option is option 0. Set an option to 0 to clear it, and to 1 to select it. In the following code fragment, the third option in the Workflow list is selected.

```
Dim GenePix
Set GenePix = window.external
' This is a zero-based list!
GenePix.Option(2) = 1
```

GenePix.Option also allows you to specify that only the names and IDs from a GAL file are applied to the blocks when loading using LoadFile. Use the index:

```
Dim GenePix
Set GenePix = window.external
' Save the existing option setting
Dim nSaveOption
nSaveOption = GenePix.Option(-1)
GenePix.Option(-1) = TRUE
GenePix.LoadFile("C:\Axon\Params\demo.gal")
' Restore the option that we changed
GenePix.Option(-1) = nSaveOption
```

PresetImageSlots

PresetImageSlots(slotnumber1, slotnumber2,...)

When importing images from files not stored in the GenePix program, use PresetImageSlots to specify the image channels in which to put the images and bypass the Assign Image dialog box.

The slot numbers are 1-based.

PresetImageSlots must be used together with PresetImageWavelengths below

Example

```
Dim GenePix
Set GenePix = window.external
// set the image wavelengths to 635 and 532
call GenePix.PresetImageWavelengths(635, 532)
call GenePix.PresetImageSlots(1, 2)
```

ScatterPlot

Returns the Scatter Plot automation interface, with which you can access properties and methods of the Scatter Plot tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
```

PresetImageWavelengths

PresetImageWavelengths(imagewavelength1, imagewavelength2,...)

When importing images from files not stored in the GenePix program, use PresetImageWavelengths to specify the wavelengths of the images and bypass the Assign Image dialog box.

PresetImageWavelengths must be used together with PresetImageSlots above.

Example

```
Dim GenePix
Set GenePix = window.external
// set the image wavelengths to 635 and 532
call GenePix.PresetImageWavelengths(635, 532)
call GenePix.PresetImageSlots(1, 2)
```

PreviewScan

Starts a Preview Scan.

See Automated Acquisition on page 63 for a description of scanning and using callbacks in scripts.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.PreviewScan
```

RatioCount

Returns the number of ratios currently defined. This is a 1-based number.

If you want the number of ratios defined in the Results file, use Results.RatioCount on page 148.

Example

```
Dim GenePix
Set GenePix = window.external
     alert(GenePix.RatioCount)
```

RatioFormulationByChannel

Returns the currently-defined ratios in terms of wavelength channels as an array, for example, (0/1, 0/2, 0/3).

For ratio formulations in terms of laser wavelengths, for example, (635/532, 635/470, 635/400), see Ratios on page 103.

For the ratios in the current Results file, use Results.RatioFormulationByChannel on page 149.

Example

```
Dim GenePix
Set GenePix = window.external

Dim asRatioFormulationByChannel, i

asRatioFormulationByChannel = GenePix.RatioFormulationByChannel

For i = 0 to UBound(asRatioFormulationByChannel)
     alert asRatioFormulationByChannel(i)
Next
```

Ratios

Returns the currently-defined ratios in terms of laser wavelengths, for example, (635/532, 635/490, 635/445).

For ratio formulations in terms of wavelength channel numbers for example, (0/1, 0/2, 0/3), see GenePix.RatioFormulationByChannel on page 103.

For the ratios in terms of laser wavelengths in the current Results file, see Results.Ratios on page 148.

Example

```
Dim GenePix
Set GenePix = window.external

Dim asRatios, i

asRatios = GenePix.Ratios

For i = 0 to UBound(asRatios)
     alert asRatios(i)
Next
```

Report

Returns the Report Tab on page 62 automation interface, with which you can access properties and methods of the Report

Example

```
Dim GenePix
Set GenePix = window.external
Dim Report
Set Report = GenePix.Report
```

Results

Returns the Results automation interface, with which you can access properties and methods of the Results

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
```

ResultsPath

Gets and sets the current Results path (that is, where the user is saving Results).

Example

```
Dim GenePix
Set GenePix = window.external
' set Results path
GenePix.ResultsPath = "C:\Axon\Data\"
' get Results path
Alert GenePix.ResultsPath
```

SaveImages

SaveImages sFilePath, sComment, bFlag

Saves the currently-loaded images to disk as TIFF files. To export images as JPEGs, use ExportImages on page 81.

The file path string can be a relative path or an absolute path; if no file name string is included then the file is saved with the automatically assigned name. The comment string can be any comment. The flag is a hex number corresponding to various Save Image options:

```
&h000001 save images compressed;
&h000002 save all images as separate files;
&h001000 save the preview 1 image;
&h002000 save the preview 2 image;
&h004000 save the preview 3 image;
&h008000 save the wavelength 1 image;
&h010000 save the wavelength 2 image;
&h020000 save the wavelength 3 image;
&h040000 save the wavelength 4 image;
&h080000 save the ratio 1 image;
&h200000 save the ratio 2 image;
&h200000 save the ratio 3 image.
```

To perform a combination of these options, you need to concatenate the hex numbers, either by addition or by bitwise. (Hex numbers in VBScript are prefixed by &h.) For example, to save wavelength 1 and wavelength 2, use &h008000 or &h010000 for the flag value.

Alternatively, you can use GenePix.ExportImages on page 90 as an unqualified statement and the images are saved according to the currently-set options in the ExportImages dialog box:

```
Dim GenePix
Set GenePix = window.external
' To save with default values:
GenePix.SaveImages
' To save with custom values:
GenePix.SaveImages "c:\Axon\Data\myImages.tif", "my images",
&h008000 or &h010000
```

The &h000002 flag should always be used in conjunction with one or more other flags. &h000002 says "save images as separate single image files", but it has to know which images to save. So to save Wavelength 1 and Wavelength 2 as separate files, write:

```
Dim bSaveFlags
bSaveFlags = &h000002 + &h008000 + &h0010000
GenePix.SaveImages "c:\Axon\Data\myImages.tif", "comment me no
comments", bSaveFlags
GenePix automatically appends the appropriate suffix to the
individual file names, in this case "_635nm" and "_532nm".
```

SaveSettings(sFileName)

Saves the current settings to file.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.SaveSettings("mySettings.gps")
```

SessionTime

Returns the number of seconds elapsed since GenePix.exe was started.

```
Dim GenePix
Set GenePix = window.external
alert GenePix.SessionTime
```

SettingsName

Returns the full path to the current settings file.

```
Dim GenePix
Set GenePix = window.external
Dim name
name = GenePix.SettingsName
alert(name)
```

Settings

Returns a semi-colon-delimited string of settings from the Hardware Settings dialog box. Each field:value pair is delimited by a colon.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.Settings
```

STD2

StdDev2[=bValue]

Use to change the standard deviation calculation method (done manually in Options / Analysis).

Set to true to use StdDev2; set to false to use the normal standard deviation.

There is a similar property in the Results Object which returns the standard deviation method that was used in the analysis.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.StdDev2 = True
```

StopScan

Stops the current scan. This triggers the OnScanAbort callback.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.StopScan
```

SwitchToTab(nTab)

Switches the display to the numbered tab. The tabs are numbered from left to right in a zero-based list.

The example below switches the display to the Image tab.

Example

```
Dim GenePix
Set GenePix = window.external
' This is a zero-based list!
GenePix.SwitchToTab(0)
```

UserName

Returns a string containing the GenePix Pro software user name of the current user.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.UserName
```

Version

Returns a string containing the version of the GenePix Pro application. This version is reported in the Application field in the Hardware Diagnostics tab.

You can use the CheckVersion on page 79 function to check an arbitrary version number against the current version.

Example

```
Dim GenePix
Set GenePix = window.external
alert GenePix.Version
```

WavelengthChannels

Returns the currently-defined wavelength channels as an array, for example, (0, 1, 2, 3).

If you want the laser wavelengths from the scanner, use GenePix.Wavelengths.

If you want the wavelength channels from the Results file, use Results.WavelengthChannels.

Example

```
Dim GenePix
Set GenePix = window.external
Dim anWavelengths, i
anWavelengths = GenePix.WavelengthChannels
For i = 0 to UBound(anWavelengths)
alert( anWavelengths(i) )
Next
```

Wavelengths

Returns the laser wavelengths from the scanner, for example, (635, 532).

For the wavelength channels currently defined (for example, 0, 1), use GenePix.WavelengthChannels on page 107.

For the laser wavelengths in the current Results file, use Results.Wavelengths.

Example

```
Dim GenePix
Set GenePix = window.external
Dim anWavelengths, i
anWavelengths = GenePix.Wavelengths
For i = 0 to UBound(anWavelengths)
Alert( anWavelengths(i) )
Next
```

WebAddress

Returns the current web address for feature information. The web address is set in the Options > Analysis dialog box.

Example

```
Dim GenePix, sURL
Set GenePix = window.external
sURL = GenePix.WebAddress
alert sURL
```

Scanner Object Reference

B

This appendix contains the following Scanner Object references:

DataScan

Starts a Data Scan.

See Automated Acquisition on page 63 for an account of scanning and using callbacks in script.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DataScan
```

DualScan

Method of: GenePix Object, Scanner Object

Does a Dual Scan: a Preview Scan followed by a Data Scan.

See Automated Acquisition on page 63 for an account of scanning and using callbacks in script.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.DualScan
```

FocusPosition

Returns the focus position from the Results file header.

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim FPosition
FPosition = Results.FPosition
```

Info

Returns a comma-delimited string of properties and their current measured values from the Hardware Diagnostics dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.Info
```

InfoMax

Returns a comma-delimited string of properties and their maximum allowed values from the Hardware Diagnostics dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.InfoMax
```

InfoMin

Returns a comma-delimited string of properties and their minimum allowed values from the Hardware Diagnostics dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.InfoMin
```

LinesAveraged

Returns the number of lines averaged from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert(Results.LinesAveraged)
```

Name

Returns the version string of the current scanner (reported in the Scanner field in the Hardware Diagnostics dialog box).

If using the demo driver, the string is:

```
GenePix type (Emulated)
```

where type is either 4000, 4100 or 4200;

If using a scanner, the string is:

```
GenePix scanner type (#serial number)
```

where scanner type is 4000A, 4000B, 4100A or 4200A.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.Name
```

NumLasers

Returns the number of lasers.

PerformanceLog

PerformanceLog(nRecord)

Returns a record from the GenePix® scanner performance log as a string. Use PerformanceLogCount on page 112 to determine the number of records that can be returned:

Example

```
Dim GenePix, nRecords, i
Set GenePix = window.external

Dim Scanner
Set Scanner = GenePix.Scanner

nRecords = Scanner.PerformanceLogCount-1

For i = 0 to nRecords
         alert Scanner.PerformanceLog(i)
Next
```

PerformanceLogCount

Returns the number of records (that is, the number of rows of data) in the GenePix performance log. This is a 1-based number.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.PerformanceLogCount
```

PixelSize

Returns the pixel size from the Results file header. The pixel size is the ratio of the pixel size in the source image to the pixel size in the saved JPEG image.

To get the pixel size from the current image in the Image tab, use ImageTab.PixelSize on page 127.

To get and set the pixel size for Data Scans, use Scanner.PixelSize.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.PixelSize)
```

PMT

PMT(nPMT)

Returns the PMT settings from the Results file header.

The following example gets each PMT setting and displays the value in a message box on the screen.

To read or set the PMT Gain of the scanner, use Scanner.PMT.

Example

```
Dim GenePix
Set GenePix = window.external

Dim Results
Set Results = GenePix.Results

Dim anPMT(1), i

For i = 0 to 1
          anPMT(i) = Results.PMT(i)

          Alert(anPMT(i))
Next
```

Power

Power(nLaser)[= nPowerValue]

Gets and sets the percentage of full power at which to operate. This is a zero-based list, so the first laser is 0 and the second is 1. The following code fragment reads the power setting of the first laser and sets the power of the second laser to 10%.

To get the Power values from a Results file, use Results.ScanPower on page 151.

Example

```
Dim GenePix, nFirstLaser
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
' get power
nFirstLaser = Scanner.Power(0)
' set power
Scanner.Power(1) = 10
```

PreviewScan

Starts a Preview Scan.

See Automated Acquisition on page 63 for an account of scanning and using callbacks in script.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.PreviewScan
```

Settings

Returns a semi-colon-delimited string of settings from the Hardware Settings dialog box. Each field:value pair is delimited by a colon.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
alert Scanner.Settings
```

SlotEnable

SlotEnable [= bStatus]

Gets and sets the status of the laser slot in the Hardware Settings dialog box, as a Boolean.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
' get the status
Dim bStatus
bStatus = Scanner.SlotEnable(1)
alert(bStatus)
```

SlotLaser

SlotLaser [= nLaser]

Gets and sets the laser for the slot in the Hardware Settings dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Scanner
Set Scanner = GenePix.Scanner
' get the laser
Dim nLaser
nLaser = Scanner.SlotLaser(1)
alert(nLaser)
```

StopScan

Stops the current scan. This triggers the OnScanAbort on page 100 callback.

Example

```
Dim GenePix
Set GenePix = window.external
GenePix.StopScan
```

Image Tab Object Reference

C

This appendix contains the following Image Tab Object references:

AddBlock

AddBlock(nXOrigin, nYOrigin, nCols, nColSpacing, nRows, nRowSpacing, nDiameter, nRotation, FeatureLayout)

Adds a new block with the specified dimensions and properties, as in the Block Properties dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a block
Call ImageTab.AddBlock( 1000, 2000, 24, 200, 21, 200, 100, 0, 0
)
```

AddMeasuringTool

AddMeasuringTool(nMTType, anVertices, nNumVertices, pvIndex)

Adds a new measuring tool region to the Image tab with the specified properties, as in the Measuring Tools button.

nMTType is an integer specifying the type of measuring tool:

Table C-1 nMTType Integer

0	Simple Line Profile
1	Polyline Profile
2	Rectangular Averaged Line
3	Rectangular Region
4	Polygonal Region
5	Elliptical Region

anVertices is an array of integers specifying the x and y coordinates (in pixels, NOT microns) for the vertices of the measuring tool region, in the form:

```
[x1, y1, x2, y2, ... xnNumVertices, ynNumVertices]
```

nNumVertices is the number of vertices specified for the region. This is a 1-based number.

pvIndex is a pointer to type a Variant; returns a handle for the new measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
```

AutoScaleDisplaySettings

Adjusts the brightness and contrast controls on the Image tab to display the image optimally. This is the same as Auto Scale Display Settings.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
ImageTab.AutoscaleDisplaySettings
```

CurrentImage

Returns the file name for the current image, which is a temporary file. In effect, This as being a reference to a JPEG of the current image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim sFileName
sFileName = ImageTab.CurrentImage
alert sFileName
```

CurrentImageHeight

Returns the height of the current image in the ImageTab, in terms of source pixels.

If you have zoomed in on an image, even though the image as it is displayed looks very large, the CurrentImageHeight may be quite small, because it is measured in source pixels. If you want to get the height of the Current Image as it is displayed, use document.body.clientHeight - document.body.topMargin - document.body.bottomMargin.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim nHeight
nHeight = ImageTab.CurrentImageHeight
```

CurrentImageNumber

CurrentImageNumber[=nImage]

Gets and sets the currently-displayed image type in the Image tab. The parameter nImage takes the following values:

0 preview 1 image;

1preview 2 image;

2preview 3 image;

3 wavelength 1 image;

4wavelength 2 image;

5wavelength 3 image;

6wavelength 4 image;

7ratio 1 image;

8ratio 2 image;

9ratio 3 image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
' set the display
ImageTab.CurrentImageNumber = 2
' get the display
alert ImageTab.CurrentImageNumber
```

CurrentImageWidth

Returns the width of the current image in the ImageTab, in terms of source pixels.

If you have zoomed in on an image, even though the image as it is displayed looks very large, the CurrentImageWidth may be quite small, because it is measured in source pixels. If you want to get the width of the Current Image as it is displayed, use document.body.clientWidth - document.body.leftMargin - document.body.rightMargin.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim Width
Width = ImageTab.CurrentImageWidth
```

DeleteMeasuringTool

DeleteMeasuringTool(index)

Deletes the measuring tool region specified by index from the Image tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// delete the line profile tool
call ImageTab.DeleteMeasuringTool(pvLineIndex)
```

GetBlockVertices

GetBlockVertices(nBlock, aX, aY)

Returns the vertices of a block, by block number:

- nBlock is the block number, 0-based
- aX and aY are arrays of X- and Y-coordinates that are returned by the function

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
' dimension arrays to hold the vertices
Dim aX, aY
' get the vertices of the first block
Call ImageTab.GetBlockVertices( 0, aX, aY )
```

Image(nImage)

Returns the file name for a given image, which is a temporary JPEG file. In effect, this is a reference to a JPEG of the image. The image number nImage takes the following values:

0 preview 1 image;

1 preview 2 image;

2 preview 3 image;

3 wavelength 1 image;

4 wavelength 2 image;

5 wavelength 3 image;

6 wavelength 4 image;

7 ratio 1 image;

8 ratio 2 image;

9 ratio 3 image.

To get the currently displayed image (that is, the contents of the Image tab), use ImageTab.CurrentImage on page 117.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim sFileName
' get the first ratio image
sFileName = ImageTab.Image(11)
```

ImageHeight

Returns the height of the loaded images, in pixels.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim nHeight
nHeight = ImageTab.ImageHeight
```

ImageInfo

ImageInfo(nImage)

Returns the Image Information for a given image. The parameter nImage can take the following values:

?current image

0preview 1 image;

1 preview 2 image;

2 preview 3 image;

3 wavelength 1 image;

4 wavelength 2 image;

5 wavelength 3 image;

6wavelength 4 image;

7ratio 1 image;

8ratio 2 image;

9ratio 3 image.

The Image Information is returned as a text string, with the different fields separated by carriage return-linefeed (which in VBScript has the constant value vbCrLf). To display the Image Information you will have to parse the returned string (as is done in the Print Image Report).

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim sInfo
sInfo = ImageTab.ImageInfo(0)
```

ImageWidth

Returns the width of the loaded images, in pixels.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim nWidth
nWidth = ImageTab.ImageWidth
```

IsValidMeasuringTool

IsValidMeasuringTool(index)

Returns the status of the specified measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// alert status of the tool: returns true or false
alert( ImageTab.IsValidMeasuringTool(pvLineIndex) )
```

Measuring Tools

AddMeasuringTool

AddMeasuringTool(nMTType, anVertices, nNumVertices, pvIndex)

Adds a new measuring tool region to the Image tab with the specified properties, as in the Measuring Tools tool-button.

nMTType is an integer specifying the type of measuring tool:

Table C-2 nMTType Integer Values

Integer	Description
0	Simple Line Profile
1	Polyline Profile
2	Rectangular Averaged Line
3	Rectangular Region
4	Polygonal Region
5	Elliptical Region

anVertices

is an array of integers specifying the x and y coordinates (in pixels, NOT microns) for the vertices of the measuring tool region, in the form:

```
[x1, y1, x2, y2, ... xnNumVertices, ynNumVertices]
```

nNumVertices is the number of vertices specified for the region. This is a 1-based number.

pvIndex is a pointer to type Variant; returns a handle for the new measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
```

DeleteMeasuringTool

DeleteMeasuringTool(index)

Deletes the measuring tool region specified by index from the Image tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// delete the line profile tool
call ImageTab.DeleteMeasuringTool(pvLineIndex)
```

IsValidMeasuringTool

IsValidMeasuringTool(index)

Returns the status of the specified measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// alert status of the tool: returns true or false
alert( ImageTab.IsValidMeasuringTool(pvLineIndex) )
```

LastMeasuringToolID

Returns the index of the last-created measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim nLastTool
nLastTool = ImageTab.LastMeasuringToolID
```

MTArea

MTArea(index)

Returns the area of the specified measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a rectangular region from (200,200) to (400,500)
Dim anVertices(4), pvBoxIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 500
Call ImageTab.AddMeasuringTool(3, anVertices, 2, pvBoxIndex)
// return the area of the tool specified by pvBoxIndex
alert( "Area: " & ImageTab.MTArea(pvBoxIndex) )
```

MTLength

MTLength(index)

Returns the length of the specified measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile from (200,200) to (400,500)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 500
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// return the length of the tool specified by pvLineIndex
alert( "Length: " & ImageTab.MTLength(pvLineIndex) )
```

MTMaxIntensity

Returns the Measuring Tool Maximum Intensity found.

MTMean

MTMean(index, nWavelength)

Returns the mean of all the pixels inside the specified measuring tool, from the specified wavelength image. Wavelength is 0-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// get the mean intensity in the tool at wavelength 0
alert( "Mean: " & ImageTab.MTMean(pvLineIndex, 0) )
```

MTMedian

MTMedian(index, nWavelength)

Returns the median of all the pixels inside the specified measuring tool, from the specified wavelength image. Wavelength is 0-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// get the median intensity in the tool at wavelength 0
alert( "Median: " & ImageTab.MTMedian(pvLineIndex, 0) )
```

MTMinIntensity

Returns the Measuring Tool Minimum Intensity found.

MTStdDev

MTStdDev(index, nWavelength)

Returns the standard deviation of all the pixels inside the specified measuring tool, from the specified wavelength image. Wavelength is 0-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// get the standard deviation in the tool at wavelength 0
alert( "SD: " & ImageTab.MTStdDev(pvLineIndex, 0) )
```

MTType

MTType(index)

Returns the type of the specified measuring tool.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// add a simple line profile tool from (200,200) to (400,200)
Dim anVertices(4), pvLineIndex, nToolType
anVertices(0) = 200
anVertices(1) = 200
anVertices(2) = 400
anVertices(3) = 200
Call ImageTab.AddMeasuringTool(0, anVertices, 2, pvLineIndex)
// return the type of the tool specified by pvLineIndex - that
is, 0 for a line profile tool
nToolType = ImageTab.MTType(pvLineIndex)
```

NumMeasuringTools

Returns the number of measuring tool regions currently drawn on the Image tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim nNumberOfTools
nNumberOfTools = ImageTab.NumMeasuringTools
```

PixelSize

PixelSize(nChannel)

Returns the pixel size (that is, the resolution) from the specified channel of the currently loaded image. See ImageTab.Image(nImage) on page 120 for a list of channel numbers.

To get and set the pixel size for Data Scans, use Scanner.PixelSize on page 112.

To get the pixel size from a Results file, use Results.PixelSize on page 147.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// alert the pixel size of the first single-wavelength image
alert( ImageTab.PixelSize(3) )
```

ScanRegion

ScanRegion(left, top, right, bottom)

Sets a new scan region with the specified dimensions in pixels, NOT in microns.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
// set a scan region that is 50 by 50 pixels
Call ImageTab.ScanRegion(0,0,50,50)
// clear the current scan region
Call ImageTab.ScanRegion(0,0,0,0)
```

SubImage

SubImage(nImage, x, y, nWidth, nHeight)

Saves a specified region of the currently-loaded image. The parameter nImage can have the following values:

0 preview 1 image;

1 preview 2 image;

2 preview 3 image;

3wavelength 1 image;

4wavelength 2 image;

5 wavelength 3 image;

6wavelength 4 image;

7ratio 1 image;

8ratio 2 image;

9ratio 3 image.

x and y are the coordinates of the top left of the sub-image, nWidth and nHeight are the width and height of the sub-image in microns. The example below gets a sub-image of the first ratio image, with top left at (200, 300) and width and height of 500 each.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
Dim sImageSource
sImageSource = ImageTab.SubImage(7, 200, 300, 500, 500)
```

ZoomOut

The scripting equivalent of Zoom Full Scale.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ImageTab
Set ImageTab = GenePix.ImageTab
GenePix.SwitchToTab(0)
ImageTab.ZoomOut
```

Histogram Object Reference

D

This appendix contains the following Histogram Object references:

Bins

Bins(nWavelength)

Returns a variant array of bin counts from the histogram for a particular wavelength.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim afBins
afBins = Histogram.Bins(1)
alert afBins(0)
```

BinWidth

Returns the bin width of the Histogram. This is set on the Histogram tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
alert Histogram.BinWidth
```

FullScale

Sets the X & Y axes of the Histogram to full scale.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Histogram.FullScale
```

HalfWidth

HalfWidth(nWavelength)

Returns the half-width of the histogram at the specified wavelength, that is, the width of the histogram at the 50% point.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim fHalfRed, fhalfGreen
fHalfRed = Histogram.HalfWidth(0)
fHalfGreen = Histogram.HalfWidth(1)
```

ImageHeight

Returns the height of the current Histogram image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim Height
Height = Histogram.ImageHeight
```

ImageWidth

Returns the width of the current Histogram image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim Width
Width = Histogram.ImageWidth
```

IntensityRatio

IntensityRatio(nRatio)

Returns the current value of the Count Ratio, where nRatio can take a value between 0 and 2.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim fRatio
fRatio = Histogram.IntensityRatio(0)
```

Peak

Peak(nImage)

Returns the center bin value for the peak of the histogram for the specified image, that is, if the histogram peaks at 3225, this method returns 3225.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Dim fPeakRed, fPeakGreen
fPeakRed = Histogram.Peak(0)
fPeakGreen = Historam.Peak(1)
```

SaveImage

SaveImage(nWidth, nHeight)

Saves an image of the current Histogram display at a given size in a temporary directory.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
document.images(0).src = Histogram.SaveImage(400, 300)
```

SetXRange

SetXRange(nMin, nMax)

Sets the minimum and maximum X-axis values for the Histogram.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
call Histogram.SetXRange(0, 100)
```

SetYRange

SetYRange(nMin, nMax)

Sets the minimum and maximum Y-axis values for the Histogram.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Histogram
Set Histogram = GenePix.Histogram
Call Histogram.SetYRange(0, 0.7)
```

Results Object Reference

E

This appendix contains the following Results Object references:

ApplyNormalization

Applies the current normalization configuration to the Results. This is the same as the Apply Normalization button on the Results tab.

See NormalizationFactor on page 146; NormalizationMethod on page 146; RemoveNormalization on page 149.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.ApplyNormalization
```

BackgroundSubtractionMethod

Returns the background subtraction method from the current Results file. This value is stored in the Results file header.

If the method is User (that is, Global/User-specified constant), use BackgroundSubtractionUserValue on page 135 to get the user-specified values.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert(Results.BackgroundSubtractionMethod)
```

BackgroundSubtractionUserValue

Returns the user-specified background subtraction values from the current Results file. This property is only available if the background subtraction method used in the Results file is User (that is, Global/User-specified constant).

This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert(Results.BackgroundSubtractionUserValue)
```

Barcode

Returns the barcode from the current Results file. This value is stored in the Results file header.

```
(GenePix.Barcode returns the barcode from the 'B:' field in the
Status Bar.)
```

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sBarcode
sBarcode = Results.Barcode
```

Column

Column(nColumn)[sColumnName]

Returns a complete column of data as a variant array. You can specify either a column number, or a column name. If you are using a column name, the name must be spelled as it appears in the Results tab column titles (case-insensitive), and must be enclosed in quotation marks.

Except for the Name and ID columns, which are returned as strings, data from the Results tab is returned as double precision floats. 'Error' values from Log Ratio columns are returned as the VBScript vbError data type.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim asID, afBlock
asID = Results.Column("ID")
afBlock = Results.Column(0)
```

ColumnName

ColumnName(nColumn)

Returns the title of a given column. This value is zero-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sColumnName
sColumnName = Results.ColumnName(5)
```

Comment

Returns comment from the current Results file, which is set when saving results from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sComment
sComment = Results.Comment
```

Creator

Returns the version number of the GenePix® Pro software used to create the current Results file. This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sCreator
sCreator = Results.Creator
```

DateTime

Returns the Date and Time when the images were scanned for the current Results file. This value is stored in the Results file header.

Example

Dim GenePix

```
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sDateTime
```

DeleteSubImage

DeleteSubImage(sFileName)

Deletes a sub-image file, given the file name.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results, sFileName, sHTML
Set Results = GenePix.Results
sFileName = Results.GetFeatureImage (0, 4)
sHTML = "<img src=" & sFileName & ">"
call Results.DeleteSubImage(sFileName)
```

FileName

Returns the file name of the currently loaded Results file (if the results have been saved).

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results, sName
Set Results = GenePix.Results
sName = Results.Filename
```

Flag

Flag(nRow)[= nFlagValue]

Gets and sets the flag value of a given feature (that is, row). Features are specified by index number. The following example sets the flag value of feature 5 to -50, then returns the value of the flag.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nTmp
' set flag value
Results.Flag(5) = -50
' get flag value
nTmp = Results.Flag(5)
```

FlagFeatures

FlagFeatures(sQueryName, sFlag)

Executes a saved Flag Features query on the current results. sQueryName contains the name of the query to execute, while sFlag contains the name of the flag to apply to the features that satisfy the query (for example, "good", "bad"). This has exactly the same properties as using the Flag Features dialog directly for example, features flagged good or bad obtain an Autoflag flag.

Use GenePix.OnFlagFeaturesDone on page 100 to continue your script once the query has been executed.

Use Results.FlagFeaturesQueries on page 138 to obtain an array of strings containing the names of all saved queries from the Flag Features dialog box.

Example

Dim GenePix

```
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim asQueries
' get query names
asQueries = Results.FlagFeaturesQueries
' flag features good with the first query
call Results.FlagFeatures(asQueries(0), "good")
```

FlagFeaturesQueries

Returns an array of strings consisting of the names of all saved queries from the Flag Features dialog box.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim asQueries
' get query names
asQueries = Results.FlagFeaturesQueries
' alert a query name
alert (asQueries(0))
```

FocusPosition

Returns the focus position used to acquire the images for the current Results file. This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nFPosition
nFPosition = Results.FocusPosition
```

GALFile

Returns the name of the array list used to create the current Results file. This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sGALFile
sGALFile = Results.GALFile
```

GetFeatureImage

GetFeatureImage(nImageNumber, nFeatureNumber)

Gets the image of a feature from the Results JPEG file. This is equivalent to getting a Feature Viewer image.

To get a feature image you must specify the image number and the feature number (0-based; note that the Results Index column is 1-based). The possible image numbers are:

```
0 Results ratio 1 image;
1 Results ratio 2 image;
2 Results ratio 3 image;
3 Results wavelength 1 image;
4 Results wavelength 2 image;
5 Results wavelength 3 image;
6 Results wavelength 4 image.
```

The following example gets the ratio image of the fifth feature in the Results tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results, sFileName
Set Results = GenePix.Results
sFileName = Results.GetFeatureImage (0, 4)
alert(sFileName)
```

GetSubImage

GetSubImage(nImageNumber, nImageOriginX, nImageOriginY, nWidth, nHeight)

Gets a sub-image from the Results JPEG file. To get an arbitrary sub-image you must specify the image number, the origin of the sub-image, its width, and its height.

The possible image numbers are:

```
0        Results ratio 1 image;
1        Results ratio 2 image;
2        Results ratio 3 image;
3        Results wavelength 1 image;
4        Results wavelength 2 image;
5        Results wavelength 3 image;
6        Results wavelength 4 image;
```

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results, sFileName
Set Results = GenePix.Results
sFileName = Results.GetSubImage(0, 0, 0, 50, 50)
divImage.innerHTML = "<img src=" & sFileName & ">"
```

GroupRows

Groups all selected rows together at the top of the Results spreadsheet.

This does the same as the Group Rows command.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.GroupRows
```

Height

Returns the number of rows in the Results tab. This is not zero-based, that is, if the rows are numbered zero to 461, then Results.Height returns 462.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nHeight
nHeight = Results.Height
alert(nHeight)
```

HideItems

HideItems nFlag, bNormalize, bHide

Allows you to hide and show results entries based on their flag values. It is similar to the functionality implemented in the Display dialog box, except that the changes are not displayed in the Results tab. This is very useful and powerful for performing analyses where you want to exclude, for example, Not Found features or Bad features.

The following example demonstrates the syntax: you must specify the flag (either by number or name), whether or not it is a Normalize feature (true is Included in Normalization, false is Not Included), and whether or not it is hidden (true, corresponds to hidden, false corresponds to shown):

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.HideItems -50, true, true' hides not found, norm.
Results.HideItems "absent", false, true' hides absent, not
norm.
```

HideMatching

HideMatching nColumn, sHideText

Allows you to hide results entries based on matching a text string in a named column. This is very useful and powerful for performing analyses where you want to exclude, for example, all features with a particular substance name: you might name all control spots 'control', and then exclude them from an analysis.

The following example demonstrates the syntax: you must specify the column in which to search for the matching text (either by number or name) and the text (case-insensitive) for which to search.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.HideMatching 3, "control"
```

There is no provision for undoing a HideMatching call. However, there are several workaround solutions. You can use the HideItems call. By passing the value true for some flag using HideItems, the Results data is refreshed and all data hidden using HideMatching is shown.

A second workaround is to get a second instance of the Results data, for example, Set Results2 = GenePix.Results, and then continue to work with that data object.

Image

Image(nImage)

Returns the file name for the Results tab's JPEG image. The image number nImage takes the following values:

```
0        Results ratio 1 image;
1        Results ratio 2 image;
2        Results ratio 3 image;
3        Results wavelength 1 image;
4        Results wavelength 2 image;
5        Results wavelength 3 image;
6        Results wavelength 4 image;
```

The following example returns the first Results ratio image:

```
Example
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sImage
sImage = Results.Image(0)
```

ImageHeight

Returns the height of the Results tab's JPEG image in pixels.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nImageHeight
nImageHeight = Results.ImageHeight
```

ImageOriginX

Returns the X coordinate of the top left of the TIFF image that was analyzed to produce this Results file.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert(Results.ImageOriginX)
```

ImageOriginY

Returns the Y coordinate of the top left of the TIFF image that was analyzed to produce this Results file.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert(Results.ImageOriginY)
```

ImageWidth

Returns the width of the Results tab's JPEG image in pixels.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nImageWidth
nImageWidth = Results.ImageWidth
```

Index

Index(nBlock, nRow, nColumn)

Returns the index number of a feature given block, row and column.

The following example returns the index of the feature in block 1, row 2, column 3.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nTmp
nTmp = Results.Index (1,2,3)
alert(nTmp)
```

InNormalization

InNormalization(nRow)[= nFlagValue]

Gets and sets the normalization flag value of a given feature (that is, row). Features are specified by index number. Set the value to 1 to Include in Normalization, 0 to Exclude from Normalization.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nTmp
' set flag value
Results.InNormalization(5) = 1
' get flag value
nTmp = Results.InNormalization(5)
alert (nTmp)
```

IsValidColumn

IsValidColumn(nColumn)

Returns TRUE if the column is valid, that is, if it contains data.

See the Columns dialog box, where columns that are black are valid, while columns that are grayed out are not valid.

Example

```
Dim GenePix
Set GenePix = window.external

Dim Results
Set Results = GenePix.Results

' find out if the first wavelength column has data
If Results.IsValidColumn(8) then
         alert("Column F1 Median has data")
End If
```

JPEGOriginX

Returns the X coordinate of the JPEG origin for the Results file, in microns. This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.JPEGOriginX)
```

JPEGOriginY

Returns the Y coordinate of the JPEG origin for the Results file, in microns. This value is stored in the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.JPEGOriginY)
```

LaserOnTime

Returns the cumulative on time for the laser in hours from the Results file header. The following code fragment gets each laser on-time setting and displays the value in a message box on the screen.

```
Dim GenePix
Set GenePix = window.external

Dim Results
Set Results = GenePix.Results

Dim LaserOnTime(2), i

For i = 0 to 1
         LaserOnTime(i) = Results. LaserOnTime(i)
         Alert(LaserOnTime(i))
Next
```

LaserPower

LaserPower(nLaser)

Returns the laser power from each laser measured prior to scanning from the Results file header.

The following example gets each laser power setting and displays the value in a message box on the screen.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim anLaserPower(2), i
For i = 0 to 1
anLaserPower(i) = Results.LaserPower(i)
Alert(anLaserPower(i))
Next
```

Model

Returns the scanner model from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sModel
sModel = Results.Model
```

NormalizationFactor

NormalizationFactor(nWavelength)

Returns the normalization factor for the given wavelength from the Results file header, where the wavelengths are numbered 0.

See ApplyNormalization on page 134; NormalizationMethod on page 146; RemoveNormalization on page 149.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim fNormalizationFactor
fNormalizationFactor = Results.NormalizationFactor(0)
```

NormalizationMethod

NormalizationMethod[= sNormalizationMethod]

Gets and sets the normalization method applied to the current Results file, that is, the one specified in the Results header and the one specified in the ConfigureNormalization dialog box.

When getting the normalization method, if you have set the mean of the Ratio of Medians of all features to be equal to 1, for example, this method returns the comma-delimited string "RatioOfMedians,Allfeatures,MeanValue=1.000000". It only returns the normalization that has been applied to the current results. If normalization has not been applied to the results, it returns an empty string.

When setting the normalization method, set it to a comma-delimited string containing the ratio column, the features to apply it to, and the mean value, for example, "RatioOfMedians,Normalizationfeatures,MeanValue=1.000000". This is case-insensitive, and you do not have to set all three properties at the same time. For example, you could set only "RatioOfMedians", and leave the other settings as they are.

See ApplyNormalization on page 134; NormalizationFactor on page 146; RemoveNormalization on page 149.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
' set
Results.NormalizationMethod =
"RegressionRatio,Allfeatures,MeanValue=1"
' get
alert(Results.NormalizationMethod)
```

PixelSize

Returns the pixel size from the Results file header. The pixel size is the ratio of the pixel size in the source image to the pixel size in the saved JPEG image.

To get the pixel size from the current image in the Image tab, use ImageTab.PixelSize on page 127.

To get and set the pixel size for Data Scans, use Scanner.PixelSize on page 112.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.PixelSize)
```

PMT

PMT(nPMT)

Returns the PMT settings from the Results file header.

The following example gets each PMT setting and displays the value in a message box on the screen.

To read or set the PMT Gain of the scanner, use Scanner.PMT on page 112.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim anPMT(1), i
For i = 0 to 1
anPMT(i) = Results.PMT(i)
Alert(anPMT(i))
Next
```

RatioCount

Returns the number of ratios defined in the current Results file. This is a 1-based number.

For the number of ratios defined at the application level (for example, from the Options/Analysis dialog box), use GenePix.RatioCount on page 102.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert Results.RatioCount
```

Ratios

Returns the ratios in terms of laser wavelengths from the Results file header as an array, for example, (635/532, 635/490, 635/445). If there is no Results file open this returns an empty array.

For ratio formulations in the current Results file in terms of wavelength channel numbers, use Results.RatioFormulationByChannel on page 149.

For ratios in terms of laser wavelengths defined at the application level, use GenePix.Ratios on page 103.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim asRatios, i
asRatios = Results.Ratios
For i = 0 to UBound(asRatios)
alert asRatios(i)
Next
```

RatioFormulationByChannel

Returns the ratios in terms of wavelength channels from the Results file header as an array, for example, (0/1, 0/2, 0/3). If there is no Results file open this returns an empty array.

For ratio formulations in the current Results file in terms of laser wavelengths, for example, (635/532, 635/470, 635/400), use Results.Ratios on page 148.

For ratio formulations by channel currently defined at the application level (that is, on the Image tab), use GenePix.RatioFormulationByChannel on page 103.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim asRatioFormulationByChannel, i
asRatioFormulationByChannel = Results.RatioFormulationByChannel
For i = 0 to UBound(asRatioFormulationByChannel)
alert asRatioFormulationByChannel(i)
Next
```

RemoveNormalization

Removes any normalization from the Results. This is the same as the Remove Normalization button on the Results tab.

See ApplyNormalization on page 134; NormalizationFactor on page 146; NormalizationMethod on page 146.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.RemoveNormalization
```

RowID

Returns the substance ID from a given row. This value is zero-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sRowID
sRowID = Results.RowID(5)
```

RowName

RowName(nRow)

Returns the substance name from a given row. This value is zero-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sRowName
sRowName = Results.RowName(5)
```

Save

Save sFilePath, sComment, JPEGFlag, PromptFlag, VersionFlag

Saves the currently displayed Results to disk.

JPEGFlag can be 0 (do not save JPEG of images with Results), 1 (save JPEG of images with Results) or -1 (use the current setting in the Save Results dialog box).

PromptFlag can be 0 (prompt if some features do not have IDs associated with them) or 1 (do not prompt if some features do not have IDs associated with them).

VersionFlag can be 0 (save as current GPR version) or 1 (save as GenePix Pro 3.0 compatible).

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.Save "C:\Axon\Data\myResults.gpr", "Comment", 1, 0
```

In addition, you can use Results.Save without any other parameters, and the Results are saved to the current active directory using a file name generated from the current options selected in the Save Results dialog box. If a file with that name already exists, you are prompted to supply a different file name.

ScanPower

ScanPower(nLaser)

Returns the scan power settings (that is, the percentage power setting of the lasers) from the Results file header. The following example gets each scan power value and displays the value in a message box on the screen.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim anScanPower(1), i
For i = 0 to 1
anScanPower(i) = Results.ScanPower(i)
Alert(anScanPower(i))
Next
```

To read or set the scanner power currently used by the scanner, see Scanner.Power on page 113.

ScanRegion

Returns the scan region from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.ScanRegion)
```

Select

Select(nRow)[= nSelectStatus]

Gets and sets the selection status (that is, if it is highlighted) of a given feature (that is, row). Features are specified by index number. The following example selects the first 100 features in the Results file, then gets the selection status of the first feature.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim i, bTmp
' selects the first 100 features
For i = 0 to 100
Results.Select(i) = True
Next
' get selection status of first feature
bTmp = Results.Select(0)
alert (bTmp)
```

SettingsFile

Returns the file name of the settings file active during scanning from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sSettingsFile
sSettingsFile = Results.SettingsFile
```

Sort

Sort nColumn, nDirection

Sorts the Results data source (but not the display), on a given column and in a given direction. The column number is a zero-based list, while for the sort direction 1 is ascending and 0 is descending. You can also use column names, which must be enclosed in quotation marks.

```
Example
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Results.Sort 5, 0
Results.Sort "Block", 0
```

SortColumn

Returns the column that the Results are sorted on.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nSortColumn
nSortColumn = Results.SortColumn
```

SortDirection

Returns the sort direction for the file (1 = ascending, 0 = descending).

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nSortDirection
nSortDirection = Results.SortDirection
alert(Results.LinesAveraged)
```

StdDev2

Returns the type of standard deviation used in the analysis: true for SD2, and false for Normal. There is a similar method in the GenePix Object, see STD2 on page 106, that can be used to Set the type of SD used before an analysis.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results, SDname
Set Results = GenePix.Results
If Results.StdDev2 then
 SDname = "SD2"
Else
 SDname = "SD"
End If
```

Supplier

Returns the slide supplier from the Results file header.

```
Example
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
alert (Results.Supplier)
```

Temperature

Returns the temperature voltage measured prior to scanning from the Results file header.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sTemperature
sTemperature = Results.Temperature
```

Value

Value(nRow, nColumn)

Returns the value at a given row-column, where the column can be referred to by number or name. These values are zero-based.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nValue1, nValue 2
nValue1 = Results.Value(5,3)
nValue2 = Results.Value(4, "Name")
```

Version

Returns the version number of the results file from the Results file header (only if the results have been saved).

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim sVersion
sVersion = Results.Version
```

Wavelengths

Returns the laser wavelengths from the Results file header as an array, for example, (635, 532, 490, 445). If there is no Results file open this returns an empty array.

For the wavelength channels in the Results files, for example, (0, 1, 2), use Results.WavelengthChannels.

For the laser wavelengths at the application level (that is, in the Image tab), use GenePix.Wavelengths on page 108.

Example

```
Dim GenePix
Set GenePix = window.external

Dim Results
Set Results = GenePix.Results

Dim anWavelengths, i

anWavelengths = Results.Wavelengths

For i = 0 to UBound(anWavelengths)
         Alert( anWavelengths(i) )
Next
```

WavelengthChannels

Returns the wavelength channels from the Results file header as an array, for example, (0, 1, 2, 3). If there is no Results file open this returns an empty array.

For the laser wavelengths in the Results files, use Results.Wavelengths on page 154.

For the wavelength channel numbers currently defined at the application level (for example, in the Image tab), use GenePix.WavelengthChannels on page 107.

Example

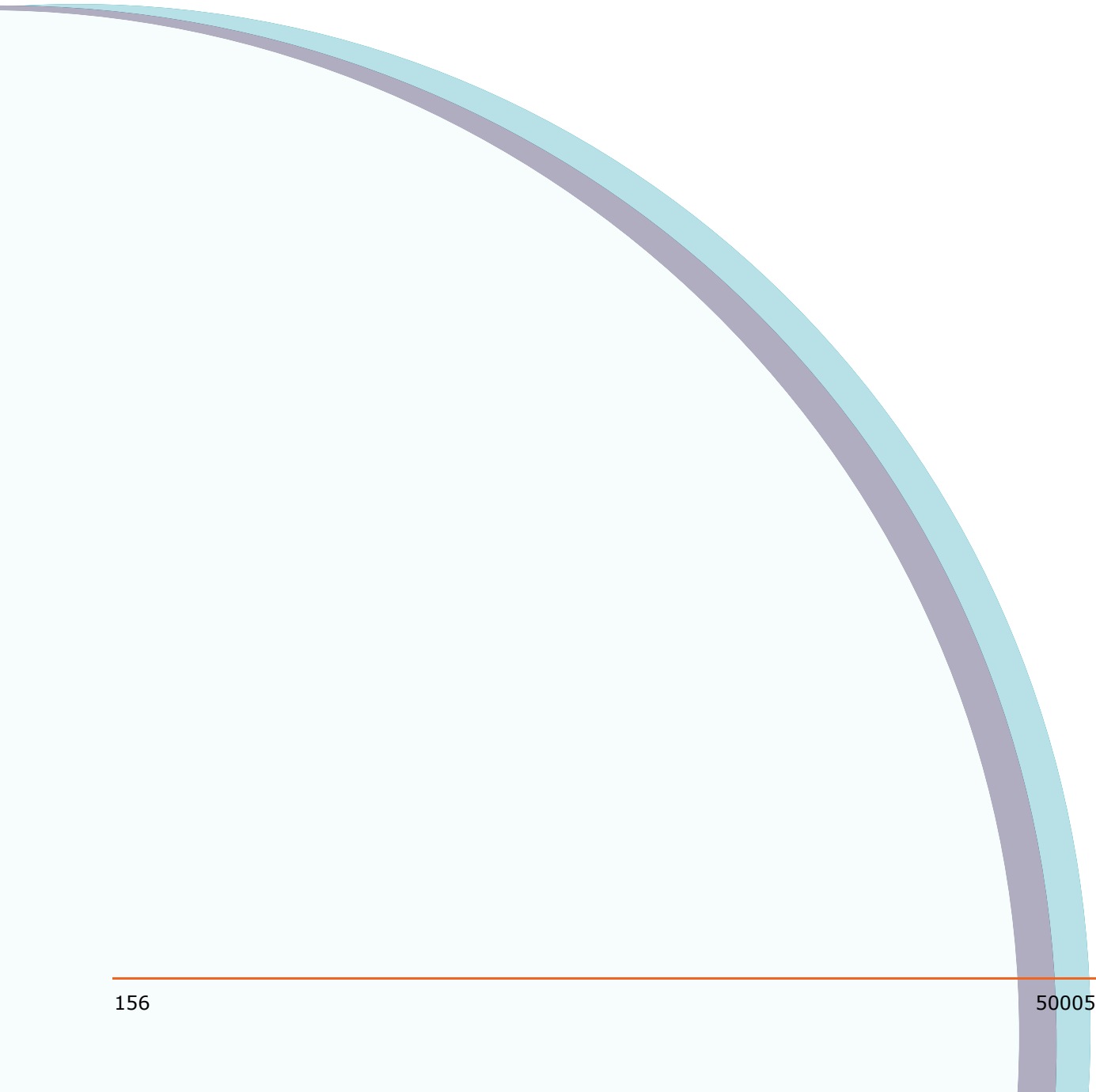
```
Dim GenePix
Set GenePix = window.external

Dim Results
Set Results = GenePix.Results

Dim anWavelengths, i

anWavelengths = Results.WavelengthChannels

For i = 0 to UBound(anWavelengths)
         Alert( anWavelengths(i) )
Next
```

Width

Returns the number of columns in the Results tab. This is a 1-based number.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results
Dim nWidth
nWidth = Results.Width
```



Report Object Reference

F

This appendix contains the following Report Object references:

Busy

Sets the busy state, which is useful to indicate activity for numerically intensive scripts. By setting the busy state to true when a procedure begins, you ensure that the 'rotating GenePix' icon actually rotates. Once the procedure is finished, set the busy state to false.

Example

```
Dim Report
Set Report = GenePix.Report
Report.Busy = true
...code to execute
Report.Busy = false
```

ExportReport

ExportReport(sFileName)

Exports the current report and dependent files (such as bitmaps of graphs and images) to a new destination. If there is a file with the same name in the target location, it will be overwritten without prompting. See Export for details on how the Report parser works.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Report
Set Report = GenePix.Report
Report.ExportReport("c:\temp\file_name.htm")
```

Print

Prints the current report.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Report
Set Report = GenePix.Report
Report.Print
```

Refresh

Reloads the current report.

Note: This command has the potential to create an infinite loop, as in the following example!

Example

```
Dim GenePix
Set GenePix = window.external
Dim Report
Set Report = GenePix.Report
Report.Refresh
```

Status

Status, StatusColor

Writes status messages at the bottom of the Report tab.

Example

```
Dim GenePix
Set GenePix = window.external
Dim Report
Set Report = GenePix.Report
Report.StatusColor = RGB(255,0,0)
Report.Status = "Calculating"
To remove a status message, you need to write:
Report.Status = ""
```

StatisticsObject

Users have access to part of GenePix® Pro's statistics calculator, which takes as its argument an array, and which returns statistical quantities. The Example Results script on the default GenePix reports page demonstrates how to use this object.

Begin by creating the ArrayStatistics object:

```
Dim ArrayStats
Set ArrayStats =
CreateObject("AxonWebUtilities.ArrayStatistics")
To extract statistics from yourArray, define:
Dim yourArray
ArrayStats.array = yourArray
```

Then:

```
Dim fMean, fMedian, fStdDev, fMin, fMax
fMean = ArrayStats.Mean
fMedian = ArrayStats.Median
fStdDev = ArrayStats.StdDev
fMin = ArrayStats.Min
fMax = ArrayStats.Max
```

There is also a binning method, which takes an array and a bin width as arguments, and returns two arrays, one of bins, and one of counts:

```
Dim afBins, afCounts
call ArrayStats.Bin(yourArray, binWidth, afBins, afCounts)
```

Finally, there is a useful method that returns the number of numbers in the array. This can be used to check if there are any non-numeric entries in an array that is being processed numerically:

```
alert(ArrayStats.Count)
```

GraphWindowObject

The GenePix Pro software graph window is used for the Pixel Plot, Histogram and Scatter Plot. You can use it to plot any data you like. The Example Graph script on the default GenePix reports page demonstrates how to use this object.

First:

```
Dim Graph
Set Graph = CreateObject("AxonWebUtilities.Graph")
```

(This definition is already in the New script.)

For a number of reasons, the graph variable should be declared globally (for example, outside any sub or function). If it is declared inside a function, the graph's image (which is a temporary file) is deleted by the GenePix Pro program as soon as the sub or function goes out of scope, and so you will not be able to print it.

Define the graph's properties. Before you add data to the graph object, you should always clear the object:

```
Graph.clear' clears the current graph settings
```

Provide data in two arrays, one for the X axis and one for the Y axis, and you must give a Data Name.

Note: This Data Name must be the same for both axes. That way, you can draw a number of traces on the same graph, identifying them by name

```
Dim YDataArray, XDataArray
Graph.YData("Data Name") = YDataArray
Graph.XData("Data Name") = XDataArray
Then, give the graph its various properties:
Graph.MajorTitle = "Major Title"
' optional; adds a major title to the graph
Graph.MinorTitle = "Minor Title"
' optional; adds a minor title to the graph
Graph.SetXRange xmin, xmax
' optional; sets the range for the X axis, between xmin and
xmax
Graph.SetYRange ymin, ymax
' optional; sets the range for the Y axis, between ymin and
ymax
Graph.Legend = boolean
' true or false to include or exclude a Legend
Graph.SetXAxisTitleAndUnits "title", "units"
' optional; adds title (units) to the X axis
Graph.SetYAxisTitleAndUnits "title", "units"
' optional; adds title (units) to the Y axis
Graph.XAxisMode = mode
' 0 gives linear axis; 1 gives log
Graph.YAxisMode = mode
' 0 gives linear axis; 1 gives log
Graph.BackgroundColor = RGB(redvalue, greenvalue, bluevalue)
' each value from 0 to 255
Graph.TraceColor("Data Name") = RGB(redvalue, greenvalue,
bluevalue)
' each value from 0 to 255
Graph.TraceStyle("Data Name") =
' 1 gives no trace; 0 gives a trace; 2 gives a histogram
Graph.SymbolStyle("Data Name") =
' 1 gives ELLIPSE
' 2 gives UPTRIANGLE
' 3 gives RECTANGLE
' 4 gives DIAMOND
' 5 gives DOWNTRIANGLE
' 6 gives ELLIPSE_OUTLINE
' 7 gives UPTRIANGLE_OUTLINE
' 8 gives RECTANGLE_OUTLINE
' 9 gives DIAMOND_OUTLINE
' 10 gives DOWNTRIANGLE_OUTLINE
' 11 gives CROSS
' 12 gives PLUS
' 13 gives MINUS
' 14 gives VERTICALBAR
Graph.SymbolSize("Data Name") =
' numerical diameter of the symbols
Graph.SymbolColor("Data Name") = RGB(redvalue, greenvalue,
bluevalue)
' Set the color of the symbols
Graph.PointSymbolSize("Data Name", pointnumber) =
' numerical diameter for a particular pointnumber symbol
Graph.Symbols("Data Name") = boolean
```

```
' true to include symbols, false to exclude
Graph.AutoScaleX
' scales the graph automatically in the X direction
Graph.AutoScaleY
' scales the graph automatically in the Y direction
Graph.FullScaleX
' draws the graph with its full X axis range
Graph.FullScaleY
' draws the graph with its full Y axis range
```

Graph.image(width, height) returns the file name of the graph's image, which you use to construct HTML. For example:

```
Dim sGraphImage, sGraphHTML
sGraphImage - Graph.Image(300, 300)
sGraphHTML = "<img src=" & sGraphImage & ">"
```

TableBuilderObject

One of the more processor-intensive operations when scripting is building long tables for outputting large amounts of data. The Table Builder object speeds up this process, so it can be useful when speed is a requirement of a script. The Example Table Builder script on the default GenePix reports page demonstrates how to use this object.

First:

```
Dim TableBuilder
Set TableBuilder =
CreateObject("AxonWebUtilities.TableBuilder")
Then Set the table's properties:
TableBuilder.Clear
' clears the object to build a new table
TableBuilder.Width =
' sets the width of the table in pixels
TableBuilder.Style = "style sheet"
' specify styles for the table
TableBuilder.Color = RGB(redvalue, greenvalue, bluevalue)
' sets table background color
TableBuilder.Border = Boolean
'true for a border, false for none
TableBuilder.NoWrap = Boolean
' true for nowrap applied to body cells, false for wrapping
TableBuilder.Attributes = attributes
' a string containing additional table-level attributes
TableBuilder.MaxRows =
' sets the maximum number of rows in the table
```

Now add the content to the table:

```
TableBuilder.Headings = "HTML text string"
' HTML text defining headings and their styles
TableBuilder.AddColumn(array)
' adds an array as a column of data to the table
TableBuilder.AddColAttributes ColumnNumber, array
' takes an array of HTML table cell attribute strings, and
applies each entry in the array to the corresponding table cell
in the ColumnNumber column.
TableBuilder.AddMatrixAttributes matrix, ColumnNumber
```

```
' takes a matrix of HTML table cell attribute strings, and
applies each entry in the matrix to the corresponding table
cell. Specifying a ColumnNumber from which to begin adding the
attributes is optional.
TableBuilder.AddMatrix(matrix)
' adds a matrix as columns of data to the table
TableBuilder.Sort column number, direction
' sorts a numbered column; sorting direction is 0 ascending; -1
is descending. The sort also works as a toggle, so that
subsequent sorts will toggle direction.
TableBuilder.ColumnURL(column number) = "url"
' adds a hyperlink to every cell in a column array. Can be used
in combination with GenePix.WebAddress, which returns the
currently selected URL in Options / Analysis, so that one does
not have to insert a URL by hand.
TableBuilder.ColumnURLTarget(column number) = "target"
' adds a target window for the URL, such as "_blank". The
default is "_self".
TableBuilder.Row(row number)
' returns a numbered row as an array (this is zero-based).
TableBuilder.Column(column number)
' returns a numbered column as an array (this is zero-based).
TableBuilder.Matrix
' returns the complete table of values as a 2D matrix
TableBuilder.Transpose
' transposes the rows and columns of the table
And the table is ready to be built:
TableBuilder.Build
' builds the table and returns the generated HTML as a string
```

Open Batch Scan Log

This command opens the batch scan log file, which contains information about each slide that was scanned in the batch, such as:

```
GenePix Pro Autoloader Results Log          V1.0
Batch Scan
Date: 2005/02/02
Time: 10:22:10


Slide number: 01
         Barcode:                SPOTGEN
         Settings File loaded:   Current
         Scan started at:        10:22:17
         Scan finished at:       10:24:07
         Image File saved:       C:\Axon\Data\1.tif


Slide number: 02
```

ScatterPlot Object Reference

G

This appendix contains the following ScatterPlot Object references:

AutoScale

Automatically scales the X and Y axes.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
ScatterPlot.AutoScale
```

ImageHeight

Returns the height of the current Scatter Plot image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
Dim nHeight
nHeight = ScatterPlot.ImageHeight
```

ImageWidth

Returns the width of the current Scatter Plot image.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
Dim nWidth
nWidth = ScatterPlot.ImageWidth
```

SaveImage

SaveImage(nWidth, nHeight)

Saves an image in the system temp directory of the current Scatter Plot display at a given size.

Example

```
Dim GenePix
Set GenePix = window.external
Dim ScatterPlot
Set ScatterPlot = GenePix.ScatterPlot
Dim sScatterPlotImage
sScatterPlotImage = ScatterPlot.SaveImage(300,300)
```

Solutions To Exercises

H

Exercise 1

```
<html>
     <!-- This is a sample GenePix report -->
     <!-- This is another comment: it is being used for
instruction.-->

<head>
     <title>
         Sample GenePix Report
     </title>
</head>

<body>
     GenePix Report - Exercise 01 - Adding Custom Reports to
Report Tab.

</body>
</html>
```

Exercise 2

```
<html>
<!-- This is a sample GenePix report -->
<!--
     This is another comment: it is being used for instruction.
-->

<head>
     <title>
         Sample GenePix Report: Exercise 2
     </title>
</head>

<body>
     GenePix Report Adding Comments
</body>
</html>
```

Exercise 3

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 3
     </title>
</head>

<body>
     <h2>Heading 1 style</h2>
     <!-- The above is formatted using the h2 style -->

     <h2>Heading 2 style</h2>
     <!-- The above is formatted using the h2 style -->

     <p>Default paragraph style</p>
     <!-- The above is formatted using the default paragraph
style -->

</body>
</html>
```

Exercise 4

Using <ul> and

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 4a
     </title>
</head>

<body>
     <h2>Heading 1 style</h2>
         <!-- The above is formatted using the h2 style -->
     <h2>Heading 2 style</h2>
         <!-- The above is formatted using the h2 style -->
     <p>Default paragraph style</p>
         <!-- The above is formatted using the default
paragraph style -->
     <p>Features in this table have: using LI
         <ul>
              <li><p>at least one ratio greater than 2.5 or
less than 0.4;</p></li>
              <li><p>all ratios within 10 per cent of each
other;</p></li>
              <li><p>flag 'good' or no flag ('not found' and
'bad' features excluded).</p></li>
         </ul>
     </p>
</body>
</html>
```

Using blockquote and br

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 4b
     </title>
</head>

<body>
     <h2>Heading 1 style</h2>
     <!-- The above is formatted using the h2 style -->

     <h2>Heading 2 style</h2>
     <!-- The above is formatted using the h2 style -->

     <p>Default paragraph style</p>
     <!-- The above is formatted using the default paragraph
style -->

     Features in this table have: Using Blockquote and BR

     <blockquote>
         <br>at least one ratio greater than 2.5 or less than
0.4;
         <br>all ratios within 10 per cent of each other;
         <br>flag 'good' or no flag ('not found' and 'bad'
features excluded).
     </blockquote>

</body>
</html>
```

Exercise 5

A third column, a border, each row a different background color

```
<html>
<head>
     <title>
         Sample GenePix Report
     </title>
</head>

// <body>
<body language="vbscript" onload="ButtonTest()">

<table width=600 cols=3 border=1>
     <tr bgcolor="red">
         <td>
              <p>Press the Start Scan button to begin.<br><br>
         </td>

         <td>
              <p>
              <!-- The trick is to put in a third cell in a row
-->
         </td>
         <td align="right" valign="top">
              <input type=button value="Test Button"
onclick="ButtonTest()">
              <script language="vbscript">
                     ' This sub brings up an alert box
                          sub ButtonTest()
                                 alert("Button Worked")
                          end sub
              </script>
         </td>
     </tr>
     <tr bgcolor="yellow">
         <td colspan=2>
              <hr width=600 color="blue" align=left>
         </td>
     </tr>
     <tr bgcolor="blue">
         <td colspan=2 align="center">
              <h2>Intensity Distribution Statistics</h2>
         </td>
     </tr>
</table>

</body>
</html>
```

Exercise 6

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 06
     </title>
</head>

<body language="vbscript" onload="ButtonTest()">
     GenePix Report
     <p>
     <input type="button" value="Not Test"
onclick="ButtonTest()">

     <script language="vbscript">
         ' This sub brings up an alert box
         sub ButtonTest()
              alert("Oh no! The alert message is different!")
         end sub
     </script>

</body>
</html>
```

Exercise 7

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 07
     </title>
</head>

<body language="vbscript">
     GenePix Report
     <p>
     <input type="button" value="var1" onclick="ButtonTest1()">
     <input type="button" value="var2" onclick="ButtonTest2()">
     <input type="button" value="var3" onclick="ButtonTest3()">

     <script language="vbscript">
         var1 = 10
         var2 = "test"
         var3 = 3.1415

         ' This sub brings up an alert box
         Sub ButtonTest1()
              alert(var1)
         End Sub

         Sub ButtonTest2()
              alert(var2)
         End Sub

         Sub ButtonTest3()
              alert(var3)
         End Sub

     </script>

</body>
</html>
```

Exercise 8

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 08
     </title>
</head>

<body language="vbscript">
     GenePix Report
     <p>
     <input type="button" value="var1" onclick="ButtonTest(0)">
     <input type="button" value="var2" onclick="ButtonTest(1)">
     <input type="button" value="var3" onclick="ButtonTest(2)">
     <input type="button" value="var4" onclick="ButtonTest(3)">

     <script language="vbscript">
         Option Explicit
         Dim newArray(3)
         newArray(0) = 10
         newArray(1) = "test"
         newArray(2) = 3.1415
         newArray(3) = "10"

         ' This sub brings up an alert box
         Sub ButtonTest(nButton)
              alert(newArray(nButton))
         End Sub

     </script>

</body>
</html>
```

Exercise 9

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 09
     </title>
</head>

<body language="vbscript">
     GenePix Report
     <p>
     <input type="button" value="var1" onclick="ButtonTest(0)">
     <input type="button" value="var2" onclick="ButtonTest(1)">
     <input type="button" value="var3" onclick="ButtonTest(2)">
     <input type="button" value="var4" onclick="ButtonTest(3)">

     <script language="vbscript">
         Option Explicit
         Dim newArray(3)
         newArray(0) = 10
         newArray(1)  = "test"
         newArray(2)  = 3.1415
         newArray(3) = "10"

         ' This sub brings up an alert box
         Sub ButtonTest(nButton)
              alert(newArray(nButton))
         End Sub

         alert("The number of elements in newArray is " &
UBound(newArray)+1)

     </script>

</body>
</html>
```

Exercise 10

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 10
     </title>
</head>

<body language="vbscript">

     <script language="vbscript">
         Option Explicit
         Dim newArray(3)
         newArray(0) = "This"
         newArray(1)  = "is"
         newArray(2)  = "string"
         newArray(3) = "concatenation"

         ' This sub brings up an alert box
         alert newArray(0) & " " & newArray(1) & " " &
newArray(2) & " " & newArray(3)

     </script>

</body>
</html>
```

Exercise 11

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 11
     </title>
</head>

<body language="vbscript">

     <script language="vbscript">
         Option Explicit
         Dim var1, var2
         var1 = "dollar"
         var2 = "dolleeeee"

         If len(var1) > len(var2) then
              alert var1 & " is longer than " & var2
         Else
              alert var1 & " is not longer than " & var2
         End If
     </script>

</body>
</html>
```

Exercise 12

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 12
     </title>
</head>

<body language="vbscript">

     <script language="vbscript">

         Option Explicit
         Dim var1, var2
         var1 = "dollar"
         var2 = "doll"
         call length(var1, var2)

         Sub length(v1, v2)
              If len(v1) > len(v2) then
                     alert v1 & " is longer than " & v2
              Else
                     alert v1 & " is not longer than " & v2
              End If
         End Sub

     </script>

</body>
</html>
```

Exercise 13

```
<html>
     <!-- This is a sample GenePix report -->
     <!-- This is another comment: it is being used for
instruction.-->
<head>
     <title>
         Sample GenePix Report: Exercise 13
     </title>
</head>

<body language="vbscript">
     GenePix Report - Looping
     <br>
     <br>

     <script language="vbscript">
         Dim i
         For i = 0 to 4
              Document.writeln "index"
              Document.writeln i
         Next
     </script>

</body>
</html>
```

Exercise 14

a. using for...next

```
<html>
<head>
     <title>
         Sample GenePix Report:Exercise 14a
     </title>
</head>

<body language="vbscript">

     <script language="vbscript">
         Option Explicit
         Dim i, n
         n = 2

         For i = 0 to 3
              alert n & " + " & n & " = " & n+n
              n = n + n
         Next

     </script>
</body>
</html>
```

b. using do untilLoop

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 14b
     </title>
</head>

<body language="VBscript">

     <script language="vbscript">

         Option Explicit
         Dim i, n
         n = 2

         Do Until n > 16
              alert n & " + " & n & " = " & n+n
              n = n + n
         Loop

     </script>

</body>
</html>
```

Exercise 15

a. main script

```
<html>
<head>
      <title>Include Script: Exercise 15a </title>
</head>

<body>

     <script language="vbscript" src="first_include.htm">
     </script>

     <script language="vbscript" src="second_include.htm">
     </script>

     <script language="vbscript">
         call firsttest()
         call secondtest()
     </script>

</body>
</html>
```

a-1. first_include

```
Sub firsttest()
    alert ("first included")
End Sub
```

a-2. second_inlcude

```
Sub firsttest()
    alert ("second included")
End Sub
```

15b. adding error recovery: on error resume next

```
<html>
<head>
     <title>Include Script: Exercise 15a </title>
</head>

<body>

     <script language="vbscript" src="first_include.htm">
     </script>

     <script language="vbscript" src="second_include.htm">
     </script>

     <script language="vbscript">

          call firsttest()
          call secondtest()

          Function Division(numerator, denominator)
               On Error Resume Next
               Dim fResult
               fResult = numerator/denominator

               If Err.Number <> 0 then
                      alert("Error: " & Err.Description)
                      Exit Function
               End If

               Division = fResult

          End Function

     </script>

</body>
</html>
```

15c. first_include

```
<html>
<head>
     <title>Include Script: Exercise 15a </title>
</head>

<body>
     <script language="vbscript" src="first_include.htm">
     </script>

     <script language="vbscript" src="second_include.htm">
     </script>

     <script language="vbscript">
         call firsttest()
         call secondtest()

         Function Division(numerator, denominator)
              On Error Resume Next
              Dim fResult
              fResult = numerator/denominator

              If Err.Number <> 0 then
                     alert("Error: " & Err.Description)
                     Exit Function
              End If

              Division = fResult

         End Function

     </script>

</body>
</html>
```

Exercise 16

You should see a number of lines of text listing column title and number, like this:

Block 0 Column 1 Row 2 Name 3 ID 4 X 5 Y 6 Dia. 7 F635 Median 8 F635 Mean 9 F635 SD 10...

```
<html>
<head>
     <title>
         Sample GenePix Report:Exercise 16
     </title>
</head>

<body language="vbscript" onload="table()">

     <div id=table1></div>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()
              Dim sTmp, i,nColumns
              nColumns = Results.Width-1
              sTmp = "<table border=0 style='font-size: 11pt;
font-family: Arial'>"

              For i = 0 to nColumns
                     sTmp = sTmp & "<tr><td>" &
Results.ColumnName(i) & "</td>"
                     sTmp = sTmp & "<td>" & CStr(i) &
"</td></tr>"
              Next

              sTmp = sTmp & "</table>"
              Table1.InnerHTML=sTmp

         End Sub

     </script>

</body>
</html>
```

Exercise 17

```
<html>
<head>
     <title>
         Sample GenePix Report
     </title>
</head>

<body language="vbscript" onload="table()">
     <div id=table1></div>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()
              Dim sTmp, i
              sTmp = "<table border=0 style='font-size: 11pt;
font-family: Arial'>"

              For i = 0 to 43

                   sTmp = sTmp & "<tr><td>" &
Results.ColumnName(i) & "</td>"
                   sTmp = sTmp & "<td>" & CStr(i) &
"</td></tr>"

              Next

              sTmp = sTmp & "</table>"
              Table1.InnerHTML=sTmp

         End Sub

     </script>

</body>
</html>
```

Exercise 18

```
<html>
<head>
     <title>
         Sample GenePix Report
     </title>
</head>

<body language="vbscript" onload="table()">

     <div id=table1></div>

     <script language="vbscript">

         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()
              Dim sTmp, i, nRows, column4,column8
              sTmp = "<table border=0 style='font-size: 14pt;
font-family: Arial'>"
              column4 = Results.Column(4)
              column8 = Results.Column(8)
              nRows = Results.Height

              For i = 0 to nRows - 1
                     sTmp = sTmp & "<tr><td>" & column4(i) &
"<td>" & column8(i) & "</td>"
              Next

              sTmp = sTmp & "</table>"

              Table1.InnerHTML=sTmp
         End Sub

     </script>

</body>
</html>
```

Exercise 19

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 19
     </title>
</head>

<body language="vbscript" onload="table()">

     <div id=table1></div>

     <script language="vbscript">

         Option Explicit

         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()

              Dim sTmp, i, nRows, column8, column4
              sTmp = "<table border=0 style='font-size: 14pt;
font-family: Arial'>"
              column4 = Results.Column("ID")
              column8 = Results.Column("F635 Median")
              nRows = Results.Height

              For i = 0 to nRows - 1
                     sTmp = sTmp & "<tr><td>" & column4(i) &
"<td>" & column8(i) & "</td>"
              Next

              sTmp = sTmp & "</table>"
              Table1.InnerHTML= sTmp

         End Sub

     </script>

</body>
</html>
```

Exercise 20

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 20
     </title>
</head>

<body language="vbscript" onload="table()">
     <div id=table1></div>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         set GenePix = window.external
         Dim Results
         set Results = GenePix.Results

         sub table()
              Dim sTmp, i, nRows, column8, column4
              sTmp = "<table border=0 style='font-size: 11pt;
font-family: Arial'>"
              Results.HideItems "absent", true
              Results.HideItems "not found", true

              ' Note that we must get column data after we have
hidden Results items

              column4 = Results.Column("ID")
              column8 = Results.Column("F635 Median")
              nRows = Results.Height

              for i = 0 to nRows - 1
                     sTmp = sTmp & "<tr><td>" & column4(i) &
"<td>" & column8(i) & "</td>"
              next

              sTmp = sTmp & "</table>"
              Table1.InnerHTML=sTmp

         end sub

     </script>

</body>
</html>
```

Exercise 21

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 21
     </title>
</head>

<body language="vbscript" onload="table()">
     <div id=table1></div>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()
              Dim sTmp, i, nRows, column8, column4
              sTmp = "<table border=0 style='font-size: 11pt;
font-family: Arial'>"
              Results.HideMatching "ID", "empty"
              //Results.HideMatching "block", "1"

              ' Note that we must get column data after we have
hidden Results items

              column4 = Results.Column("ID")
              column8 = Results.Column("F635 Median")
              nRows = Results.Height

              For i = 0 to nRows - 1
                     sTmp = sTmp & "<tr><td>" & column4(i) &
"<td>" & column8(i) & "</td>"
              Next

              sTmp = sTmp & "</table>"
              Table1.InnerHTML=sTmp
         End Sub

     </script>

</body>
</html>
```

Exercise 22

The following script is the default Example Results Header script, so the HTML section uses some slightly more complex formatting than some of the other exercise scripts:

```
<!-- This page demonstrates access to the GenePix Results
Header -->
<html>
<head>
     <style type="text/css">

         h2 {font-size: 12 pt; font-family: Arial}
         h2 {font-size: 11 pt; font-family: Arial}
         p {page-break-after: always; font-size: 10 pt; font-
family: Arial}

     </style>

     <title>Results Header Example: Exercise 22</title>

</head>

<body language="vbscript" onLoad="HandleLoad">
     <!-- HTML Layout portion -->
     <table width=600 border=0 cellspacing=0 cellpadding=0>
         <tr>
              <h2>Results Header Example</h2>
              <p><hr color="blue">
         </tr>

         <tr>
              <p>The following table contains information from
the header of the Results file that you have open in the
Results tab.
         </tr>

         <tr>
              <td>
                     <hr color="blue">
              </td>
         </tr>

         <tr>
              <td id="header">
              </td>
         </tr>

         <tr>
              <td>
                     <hr color="blue">
              </td>
         </tr>
```

```
<script language=VBScript>
Option Explicit
' Assign some GenePix Object Model references to variables

Dim GenePix
Set GenePix = window.external
Dim Results
Set Results = GenePix.Results

'
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
' This sub is called immediately after the Report is opened
'
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Sub HandleLoad
     ' Exit if there is no data in the Results tab

     If Results.Height = 0 then
         alert("Open a Results file and then press Refresh")
         exit sub
     End If
     call FillHeader

End Sub
'
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
' This sub gets Results file header info and builds a table
'
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Sub FillHeader
     ' Build the table of header info

     Dim sTmp, Model, Creator, DateTime, Comment, Barcode,
Settings, GalFile
     Model = Results.Model
     Creator = Results.Creator
     DateTime = Results.DateTime
     Comment  = Results.Comment
     If Comment = "" then
         Comment = "none"
     End If

     ' Limit comment length to 75 characters
     If Len(Comment) > 75 then
         Comment = Left(Comment, 75)
     End If

     Barcode  = Results.Barcode
     If Barcode = "" then
         Barcode = "none"
     End If

     Settings = Results.SettingsFile
     GalFile  = Results.GalFile
```

```
     Dim W1, W2, PMT1, PMT2, Temperature, LaserPower1,
LaserPower2

     W1 = Results.Wavelength(0)
     W2 = Results.Wavelength(1)
     PMT1 = Results.PMT(0)
     PMT2 = Results.PMT(1)
     Temperature = Results.Temperature
     LaserPower1 = Results.LaserPower(0)
     LaserPower2 = Results.LaserPower(1)

     sTmp = "<table width=600 border=0 style='font-size: 9pt;
font-family: Arial'>"

     sTmp = sTmp & "<tr>"
     sTmp = sTmp & "<td bgcolor=#ececff colspan=7 height=36
valign=middle align=center><b>Header Info</b></td>"
     sTmp = sTmp & "</tr>"

     sTmp = sTmp & "<tr><td><b>Scanned by:</b> " & Model &
"</td>"
     sTmp = sTmp & "<td><b>Analyzed by:</b> " & Creator &
"</td>"

     sTmp = sTmp & "<tr><td><b>When scanned:</b> " &
CStr(DateTime) & "</td>"
     sTmp = sTmp & "<td><b>GPS file:</b> " & Settings & "</td>"

     sTmp = sTmp & "<tr><td><b>Image wavelengths:</b> " & W1 &
", " & W2 & "</td>"
     sTmp = sTmp & "<td><b>GAL file:</b> " & GalFile &
"</td></tr>"

     sTmp = sTmp & "<tr><td><b>PMT Gain:</b> " & CStr(PMT1) &
", " & CStr(PMT2) & "</td>"
     sTmp = sTmp & "<td><b>Temperature (V):</b> " &
CSTR(Temperature) & "</td></tr>"

     sTmp = sTmp & "<tr><td><b>Laser Power (V):</b> " &
CStr(LaserPower1) & ", " & CStr(LaserPower2) & "</td></tr>"

     sTmp = sTmp & "<tr><td><b>Barcode:</b> " & Barcode &
"</td></tr>"

     sTmp = sTmp & "<tr><td colspan=2><b>Comment:</b> " &
Comment & "</td></tr>"

     sTmp = sTmp & "</table>"
     header.InnerHTML=sTmp

End Sub
</script>
</body>
</html>
```

Exercise 23

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 23
     </title>
</head>

<body language="vbscript" onload="table()">

     <div id=table1></div>

     <script language="vbscript">

         Option Explicit

         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results

         Sub table()

              Dim sTmp, i, nRows, block, column, row, name, ID,
F532Mean, Index
              sTmp = "<table border=0 style='font-size: 11pt;
font-family: Arial'>"
              block = Results.Column("block")
              column = Results.Column("column")
              row = Results.Column("row")
              name = Results.Column("name")
              ID = Results.Column("ID")
              F532Mean = Results.Column("F532 Mean")
              Index = Results.Column("Index")
              nRows = Results.Height
```

```
                For i = 0 to nRows - 1
                        sTmp = sTmp & "<tr><td>" & block(i) &
"</td>"
                        sTmp = sTmp & "<td>" & column(i) & "</td>"
                        sTmp = sTmp & "<td>" & row(i) & "</td>"
                        sTmp = sTmp & "<td>" & name(i) & "</td>"
                        sTmp = sTmp & "<td>" & ID(i) & "</td>"
                        sTmp = sTmp & "<td>" & F532Mean(i) &
"</td>"
                        sTmp = sTmp & "<td>" & Index(i) &
"</td></tr>"
                Next

                sTmp = sTmp & "</table>"
                Table1.InnerHTML=sTmp

          End Sub

     </script>

</body>
</html>
```

Exercise 24

For the additional suggested features in this script, look at the Example Table Builder default script.

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 24
     </title>
</head>

<body language="vbscript" onload="table()">
     <div id=tabletag></div>
     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results
         Dim TableBuilder
         Set TableBuilder=CreateObject
("AxonWebUtilities.TableBuilder.1")
         Sub table()
              Dim heading
              TableBuilder.Clear
              TableBuilder.Width = 600
              TableBuilder.Style = "font-size: 8pt; font-
family: Arial"
              TableBuilder.Color = RGB(255, 255, 255)
              TableBuilder.Border = false
              TableBuilder.NoWrap = true
              ' TableBuilder.MaxRows =      ' sets the maximum
number of rows in the table.
              ' create a string for the heading
              heading = "<td><b>Block</b></td>"
              heading = heading & "<td
nowrap><b>Column</b></td>"
              heading = heading & "<td nowrap><b>Row</b></td>"
              heading = heading & "<td nowrap><b>Name</b></td>"
              heading = heading & "<td nowrap><b>ID</b></td>"
              heading = heading & "<td nowrap><b>Ratio of
Medians</b></td>"
              heading = heading & "<td
nowrap><b>Index</b></td>"
              TableBuilder.Headings = heading
              TableBuilder.AddColumn(Results.Column("block"))
              TableBuilder.AddColumn(Results.Column("column"))
              TableBuilder.AddColumn(Results.Column("row"))
              TableBuilder.AddColumn(Results.Column("name"))
              TableBuilder.AddColumn(Results.Column("ID"))
              TableBuilder.AddColumn(Results.Column("F532
Mean"))
```

```
                TableBuilder.AddColumn(Results.Column("Index"))
                Dim Table
                Table = TableBuilder.Build
                TableTag.InnerHTML = Table
            End Sub

        </script>
</body>
</html>
```

Exercise 25

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 25
     </title>
</head>

<body language="vbscript" onload="stats()">
     <div id=tabletag></div>

     <script language="vbscript">

         Option Explicit
         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results
         Dim ArrayStats
         Set ArrayStats = CreateObject
("AxonWebUtilities.ArrayStatistics.1")

         Sub stats()
              Dim meanval, medianval, SDval, minval, maxval
              ArrayStats.array = Results.Column(8)
              meanval= ArrayStats.mean
              alert ("The mean is " & meanval)
              medianval = ArrayStats.median
              alert ("The median is " & medianval)
              SDval = ArrayStats.StdDev
              alert ("The standard deviation is " & SDval)
              minval = ArrayStats.min
              alert ("The minimum is " & minval)
              maxval = ArrayStats.max
              alert ("The maximum is " & maxval)
         End Sub

     </script>
</body>
</html>
```

Exercise 26

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 26
     </title>
</head>

<body language="vbscript" onload="stats()">
     <div id=tabletag></div>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results
         Dim ArrayStats
         Set ArrayStats = CreateObject
("AxonWebUtilities.ArrayStatistics.1")
         Dim TableBuilder
         Set TableBuilder =
CreateObject("AxonWebUtilities.TableBuilder.1")

         Sub stats()
              Dim Bins, Counts
              call ArrayStats.Bin(Results.Column(8), 50, Bins,
Counts)
              Dim heading
              TableBuilder.Clear
              TableBuilder.Width = 600
              TableBuilder.Style = "font-size: 8pt; font-
family:  Arial"
              TableBuilder.Color = RGB(255, 255, 255)
              TableBuilder.Border = false
              TableBuilder.NoWrap = true
              ' create a string for the heading
              heading = "<td><b>Bins</b></td>"
              heading = heading & "<td
nowrap><b>Counts</b></td>"
              TableBuilder.Headings = heading
              TableBuilder.AddColumn(Bins)
              TableBuilder.AddColumn(Counts)
              Dim Table
              Table = TableBuilder.Build
              TableTag.InnerHTML = Table
         End Sub

     </script>
</body>
</html>
```

Exercise 27

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 27
     </title>
</head>

<body language="vbscript" onload="mygraph()">

     <div id=graphobj></div>

     <script language="vbscript">
         Option Explicit

         ' Assign some GenePix Object Model references to
variables
         Dim GenePix
         Set GenePix = window.external
         Dim Results
         Set Results = GenePix.Results
         Dim graph
         Set graph = CreateObject("AxonWebUtilities.Graph.1")

         Sub mygraph()
              graph.YData("Data Name") = Results.Column(8)
              graph.XData("Data Name") = Results.Column(17)

              ' Then, give the graph its various properties:
              graph.MajorTitle = "Test graph"     ' optional;
adds a major title to the graph
              graph.MinorTitle = "F635 v F532"    ' optional;
adds a minor title to the graph
              graph.Legend = false                ' true or
false to include or exclude a Legend
              graph.SetXAxisTitleAndUnits "F532 Median", ""   '
optional; adds title (units) to the X axis
              graph.SetYAxisTitleAndUnits "F635 Median", ""   '
optional; adds title (units) to the Y axis
              graph.XAxisMode = 0                 ' 0 gives
linear axis; 1 gives log
              graph.YAxisMode = 0                 ' 0 gives
linear axis; 1 gives log
              graph.TraceColor("Data Name") = RGB(0, 0, 255)  '
each value from 0 to 255
              graph.TraceStyle("Data Name") = 1   ' 1 gives no
trace; 0 gives a trace; 2 gives a histogram
              graph.SymbolStyle("Data Name") = 1
```

```
                ' 1 gives ELLIPSE
                ' 2 gives UPTRIANGLE
                ' 3 gives RECTANGLE
                ' 4 gives DIAMOND
                ' 5 gives DOWNTRIANGLE
                ' 6 gives ELLIPSE_OUTLINE
                ' 7 gives UPTRIANGLE_OUTLINE
                ' 8 gives RECTANGLE_OUTLINE
                ' 9 gives DIAMOND_OUTLINE
                ' 10 gives DOWNTRIANGLE_OUTLINE
                ' 11 gives CROSS
                ' 12 gives PLUS
                ' 13 gives MINUS
                ' 14 gives VERTICALBAR

                graph.SymbolSize("Data Name") = 2   ' numerical
diameter of the symbols
                graph.SymbolColor("Data Name") = RGB(0, 0, 255) '
set the color of the symbols
                graph.Symbols("Data Name") = true   ' true to
include symbols, false to exclude
                graph.AutoScaleX  ' include to scale the graph
automatically in the X direction
                graph.AutoScaleY  ' include to scale the graph
automatically in the Y direction
                Dim nWidth, nHeight, sTmp, Name
                nWidth = 300
                nHeight = 300
                Name = graph.Image(nWidth, nHeight)

                If Name<>"" then
                       sTmp = "<img src=" + Name + ">"
                End If

                GraphObj.InnerHTML=sTmp

            End Sub

        </script>

</body>
</html>
```

Exercise 28

One of the keys in doing this exercise is setting the correct graph options: make sure that graph.tracestyle is set to histogram.

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 28
     </title>
</head>

<body language="vbscript" onload="mygraph()">

     <div id=graphobj></div>

     <script language="vbscript">
         Option Explicit

          ' Assign some GenePix Object Model references to
variables

          Dim GenePix
          Set GenePix = window.external
          Dim Results
          Set Results = GenePix.Results
          Dim ArrayStats
          Set ArrayStats = CreateObject
("AxonWebUtilities.ArrayStatistics.1")
          Dim graph
          Set graph = CreateObject("AxonWebUtilities.Graph.1")

          Sub mygraph()
               Dim Bins, Counts
               call ArrayStats.Bin(Results.Column(8), 50, Bins,
Counts)
               graph.YData("Data Name") = Counts
               graph.XData("Data Name") = Bins

               ' Then, give the graph its various properties:

               graph.MajorTitle = "Test histogram"
               ' optional; adds a major title to the graph

               graph.MinorTitle = "F635 Median"
               ' optional; adds a minor title to the graph

               graph.Legend = false
               ' true or false to include or exclude a Legend

               graph.SetXAxisTitleAndUnits "Bins", ""
               ' optional; adds title (units) to the X axis

               graph.SetYAxisTitleAndUnits "Counts", ""
               ' optional; adds title (units) to the Y axis
```

```
                graph.XAxisMode = 0
                ' 0 gives linear axis; 1 gives log

                graph.YAxisMode = 0
                ' 0 gives linear axis; 1 gives log

                graph.TraceColor("Data Name") = RGB(0, 0, 255)
                ' each value from 0 to 255

                graph.TraceStyle("Data Name") = 2
                ' 1 gives no trace; 0 gives a trace; 2 gives a
histogram

                graph.SymbolStyle("Data Name") = 1
                ' 1 gives ELLIPSE
                ' 2 gives UPTRIANGLE
                ' 3 gives RECTANGLE
                ' 4 gives DIAMOND
                ' 5 gives DOWNTRIANGLE
                ' 6 gives ELLIPSE_OUTLINE
                ' 7 gives UPTRIANGLE_OUTLINE
                ' 8 gives RECTANGLE_OUTLINE
                ' 9 gives DIAMOND_OUTLINE
                ' 10 gives DOWNTRIANGLE_OUTLINE
                ' 11 gives CROSS
                ' 12 gives PLUS
                ' 13 gives MINUS
                ' 14 gives VERTICALBAR

                graph.SymbolSize("Data Name") = 2
                ' numerical diameter of the symbols

                graph.SymbolColor("Data Name") = RGB(0, 0, 255)
                ' set the color of the symbols

                graph.Symbols("Data Name") = false
                ' true to include symbols, false to exclude

                graph.AutoScaleX
                ' include to scale the graph automatically in the
X direction

                graph.AutoScaleY
                ' include to scale the graph automatically in the
Y direction

                Dim nWidth, nHeight, sTmp, Name
                nWidth = 300
                nHeight = 300
                Name = graph.Image(nWidth, nHeight)
```

```
                If Name<>"" then
                       sTmp = "<img src=" + Name + ">"
                End If

                GraphObj.InnerHTML=sTmp

            End Sub

        </script>

</body>
</html>
```

Exercise 29

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 29
     </title>
</head>

<body language="VBscript" onload="loadimages()">

     <table>
         <tr>
              <td><img></td>
              <td><img></td>
              <td><img></td>
         </tr>
     </table>

     <script language="vbscript">
         Option Explicit

         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim ImageTab
         Set ImageTab = GenePix.ImageTab

         Sub loadimages()
              Dim ImageWidth, ImageHeight, AspectRatio
              ImageWidth  = ImageTab.CurrentImageWidth
              ImageHeight = ImageTab.CurrentImageHeight

              If ImageWidth <> 0 then
                     AspectRatio = ImageHeight / ImageWidth
              End If
```

```
                Dim i, Name

                // ratio 1
                Name = ImageTab.Image(7)
                document.images(0).src = Name
                document.images(0).width = 180
                document.images(0).height = (180 * AspectRatio)

                // Wavelength 1
                Name = ImageTab.Image(3)
                document.images(1).src = Name
                document.images(1).width = 180
                document.images(1).height = (180 * AspectRatio)

                // Wavelength 2
                Name = ImageTab.Image(4)
                document.images(2).src = Name
                document.images(2).width = 180
                document.images(2).height = (180 * AspectRatio)
            End Sub

        </script>

    </body>
    </html>
```

Exercise 30

```
<html>
<head>
    <title>
        Sample GenePix Report: Exercise 30
    </title>
</head>

<body language="VBscript" onload="loadimages()">
    <table>
        <tr>
            <td><img></td>
            <td><img></td>
            <td><img></td>
        </tr>
    </table>

    <script language="vbscript">
        Option Explicit

        ' Assign some GenePix Object Model references to
variables

        Dim GenePix
        Set GenePix = window.external
        Dim ImageTab
```

```
            Set ImageTab = GenePix.ImageTab

            Sub loadimages()

                 Dim currentHeight, currentWidth
                 currentHeight = ImageTab.CurrentImageHeight
                 currentWidth = ImageTab.CurrentImageWidth
                 document.images(0).src = ImageTab.CurrentImage
                 document.images(0).width = 180
                 document.images(0).height = 180 * (CurrentHeight
/ CurrentWidth)
                 Dim ImageWidth, ImageHeight, AspectRatio
                 ImageWidth  = ImageTab.ImageWidth
                 ImageHeight = ImageTab.ImageHeight

                 If ImageWidth <> 0 then
                        AspectRatio = ImageHeight / ImageWidth
                 End If

                 Dim i, Name

                 // ratio 1
                 Name = ImageTab.Image(7)
                 document.images(0).src = Name
                 document.images(0).width = 180
                 document.images(0).height = (180 * AspectRatio)

                 // Wavelength 1
                 Name = ImageTab.Image(3)
                 document.images(1).src = Name
                 document.images(1).width = 180
                 document.images(1).height = (180 * AspectRatio)

                 // Wavelength 2
                 Name = ImageTab.Image(4)
                 document.images(2).src = Name
                 document.images(2).width = 180
                 document.images(2).height = (180 * AspectRatio)

            End Sub

        </script>
</body>
</html>
```

Exercise 31

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 31
     </title>
</head>

<body language="VBscript" onload="loadimages">

     <table>
         <tr>
              <td><img></td>
              <td><img></td>
              <td><img></td>
         </tr>
     </table>

     <script language="vbscript">
         Option Explicit

         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim ImageTab
         Set ImageTab = GenePix.ImageTab

         Sub loadimages
              ImageTab.AutoScaleDisplaySettings
              Dim currentHeight, currentWidth
              currentHeight = ImageTab.CurrentImageHeight
              currentWidth = ImageTab.CurrentImageWidth
              document.images(0).src = ImageTab.CurrentImage
              document.images(0).width = 180
              document.images(0).height = 180 * (CurrentHeight
/ CurrentWidth)
              Dim ImageWidth, ImageHeight, AspectRatio
              ImageWidth  = ImageTab.ImageWidth
              ImageHeight = ImageTab.ImageHeight
              ImageTab.AutoScaleDisplaySettings

              If ImageWidth <> 0 then
                     AspectRatio = ImageHeight / ImageWidth
              End If
```

```
        Dim i, Name

             // ratio 1
             Name = ImageTab.Image(7)
             document.images(0).src = Name
             document.images(0).width = 180
             document.images(0).height = (180 * AspectRatio)

             // Wavelength 1
             Name = ImageTab.Image(3)
             document.images(1).src = Name
             document.images(1).width = 180
             document.images(1).height = (180 * AspectRatio)

             // Wavelength 2
             Name = ImageTab.Image(4)
             document.images(2).src = Name
             document.images(2).width = 180
             document.images(2).height = (180 * AspectRatio)

        End Sub

    </script>
</body>
</html>
```

Exercise 32

```
<html>
<head>
    <title>
        Sample GenePix Report: Exercise 32
    </title>
</head>

<body language="VBscript" onload="loadimages()">

    <table>
        <tr>
             <td><img></td>
             <td><img></td>
        </tr>
    </table>

    <script language="vbscript">
        Option Explicit

        ' Assign some GenePix Object Model references to
variables

        Dim GenePix
        Set GenePix = window.external
        Dim Histogram
```

```
          Set Histogram = GenePix.Histogram
          Dim ScatterPlot
          Set ScatterPlot = GenePix.ScatterPlot

          Sub loadimages()
               ' Get dimensions and image of Histogram
               Dim ImageWidth, ImageHeight, AspectRatio
               ImageWidth  = Histogram.ImageWidth
               ImageHeight = Histogram.ImageHeight

               If ImageWidth <> 0 then
                      AspectRatio = ImageHeight / ImageWidth
               End If

               document.images(0).src = Histogram.SaveImage(280,
(280 * AspectRatio))
               ' Get dimensions and image of Scatter Plot
               Dim ImageWidth2, ImageHeight2, AspectRatio2

               ImageWidth2  = ScatterPlot.ImageWidth
               ImageHeight2 = ScatterPlot.ImageHeight

               If ImageWidth2 <> 0 then
                      AspectRatio2 = ImageHeight2 / ImageWidth2
               End if

               document.images(1).src =
ScatterPlot.SaveImage(280, (280 * AspectRatio2))
          End Sub

     </script>

</body>
</html>
```

Exercise 33

```
<html>
<head>
     <title>
          Sample GenePix Report: Exercise 33
     </title>
</head>

<body language="VBscript" onload="loadimages">

     <table>
          <tr>
               <td><img></td>
               <td><img></td>
          </tr>
     </table>
```

```
     <script language="vbscript">
         Option Explicit

         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Histogram
         Set Histogram = GenePix.Histogram
         Dim ScatterPlot
         Set ScatterPlot = GenePix.ScatterPlot

         Sub loadimages
              Histogram.FullScale

              ' Get dimensions and image of Histogram
              Dim ImageWidth, ImageHeight, AspectRatio
              ImageWidth  = Histogram.ImageWidth
              ImageHeight = Histogram.ImageHeight

              If ImageWidth <> 0 then
                     AspectRatio = ImageHeight / ImageWidth
              End If

              document.images(0).src = Histogram.SaveImage(280,
(280 * AspectRatio))

              ScatterPlot.AutoScale
              ' Get dimensions and image of Scatter Plot
              Dim ImageWidth2, ImageHeight2, AspectRatio2
              ImageWidth2  = ScatterPlot.ImageWidth
              ImageHeight2 = ScatterPlot.ImageHeight
              If ImageWidth2 <> 0 then
                     AspectRatio2 = ImageHeight2 / ImageWidth2
              End If

              document.images(1).src =
ScatterPlot.SaveImage(280, (280 * AspectRatio2))

         End Sub

     </script>
</body>
</html>
```

Exercise 34

This is simply the previous script with the LogComment line added.

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 34
     </title>
</head>

<body language="VBscript" onload="loadimages">
     <table>
         <tr>
              <td><img></td>
              <td><img></td>
         </tr>
     </table>

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external
         Dim Histogram
         Set Histogram = GenePix.Histogram
         Dim ScatterPlot
         Set ScatterPlot = GenePix.ScatterPlot

         Sub loadimages
              GenePix.LogComment("Getting Histogram and Scatter
Plot images.")
              Histogram.FullScale
              ' Get dimensions and image of Histogram

              Dim ImageWidth, ImageHeight, AspectRatio
              ImageWidth  = Histogram.ImageWidth
              ImageHeight = Histogram.ImageHeight

              If ImageWidth <> 0 then
                     AspectRatio = ImageHeight / ImageWidth
              End if

              document.images(0).src = Histogram.SaveImage(280,
(280 * AspectRatio))
              ScatterPlot.AutoScale
              ' Get dimensions and image of Scatter Plot

              Dim ImageWidth2, ImageHeight2, AspectRatio2
              ImageWidth2  =ScatterPlot.ImageWidth
              ImageHeight2 = ScatterPlot.ImageHeight
```

```
                If ImageWidth2 <> 0 then
                       AspectRatio2 = ImageHeight2 / ImageWidth2
                End If

                document.images(1).src =
ScatterPlot.SaveImage(280, (280 * AspectRatio2))

          End Sub

     </script>

</body>
</html>
```

Exercise 35

This script gets the Histogram and Scatter Plot images, and exports the Report to the temp directory.

```
<html>
<head>
     <title>
          Sample GenePix Report: Exercise 35
     </title>
</head>

<body language="VBscript" onload="loadimages">

     <table>
          <tr>
               <td><img></td>
               <td><img></td>
          </tr>
     </table>

     <script language="vbscript">
          Option Explicit
          ' Assign some GenePix Object Model references to
variables

          Dim GenePix
          Set GenePix = window.external
          Dim Histogram
          Set Histogram = GenePix.Histogram
          Dim ScatterPlot
          Set ScatterPlot = GenePix.ScatterPlot
          Dim Report
          Set Report = GenePix.Report
```

```
          sub loadimages
               Histogram.FullScale
               ' Get dimensions and image of Histogram

               Dim ImageWidth, ImageHeight, AspectRatio
                ImageWidth  = Histogram.ImageWidth
               ImageHeight = Histogram.ImageHeight

               If ImageWidth <> 0 then
                      AspectRatio = ImageHeight / ImageWidth
               End If

               document.images(0).src = Histogram.SaveImage(280,
(280 * AspectRatio))

               ScatterPlot.AutoScale
                ' Get dimensions and image of Scatter Plot

               Dim ImageWidth2, ImageHeight2, AspectRatio2
               ImageWidth2  = ScatterPlot.ImageWidth
               ImageHeight2 = ScatterPlot.ImageHeight

               If ImageWidth2 <> 0 then
                      AspectRatio2 = ImageHeight2 / ImageWidth2
               End If

               document.images(1).src =
ScatterPlot.SaveImage(280, (280 * AspectRatio2))
               Report.ExportReport("c:/temp/images.htm")

          end sub

     </script>

</body>
</html>
```

Exercise 36 a

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 36a
     </title>
</head>

<body language="VBscript" onload="callback()">
     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.DualScan
         End Sub

     </script>

</body>
</html>
```

Exercise 36 b

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 36b
     </title>
</head>

<body language="VBscript" onload="callback()">
     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.PreviewScan
         End Sub

     </script>

</body>
</html>
```

Exercise 36 c

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 36c
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.DataScan
         End Sub

     </script>

</body>
</html>
```

Exercise 37

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 37
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.DualScan
         End Sub

         Sub OnScanDoneVB()
              alert ("Data Scan Finished")
         End Sub

     </script>

     <script language="JavaScript">
         GenePix.OnScanDone    = function () { OnScanDoneVB();
}
         GenePix.OnAnalyzeDone = function () {
OnAnalyzeDoneVB(); }

     </script>

</body>
</html>
```

Exercise 38

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 38
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.SwitchToTab(0)
              GenePix.DualScan
         End Sub

         sub OnScanDoneVB()
              alert ("Data Scan Finished")
         end sub

     </script>

     <script language="JavaScript">
         GenePix.OnScanDone    = function () { OnScanDoneVB();
}
         GenePix.OnAnalyzeDone = function () {
OnAnalyzeDoneVB(); }

     </script>

</body>
</html>
```

Exercise 39

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 39
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.SwitchToTab(0)
              GenePix.Option(0) = 0
              GenePix.Option(1) = 0
              GenePix.Option(2) = 1
              GenePix.Option(3) = 1
              GenePix.Option(4) = 0
              GenePix.DualScan
         End Sub

     </script>

</body>
</html>
```

Exercise 40

This exercise requires two files: the following script, and a settings file that contains one block and a scan area over the first block of demo data (such as ("C:\Axon\Scripts\GenePixPro6\Auto_1.gps"):

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 40
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.SwitchToTab(0)

     GenePix.LoadFile("C:\Axon\Scripts\GenePixPro6\Auto_1.gps")
              GenePix.Option(0) = 0
              GenePix.Option(1) = 0
              GenePix.Option(2) = 1
              GenePix.Option(3) = 1
              GenePix.Option(4) = 0
              GenePix.DualScan
         End Sub

     </script>

</body>
</html>
```

Exercise 41

This exercise requires two files: the following script, and a settings file that contains one block and a scan area over the first block of demo data (such as ("C:\Axon\Scripts\GenePixPro6\Auto_1.gps"):

```
<html>
<head>
     <title>
         Sample GenePix Report: Exercise 41
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         Set GenePix = window.external

         Sub callback()
              GenePix.SwitchToTab(0)
              GenePix.DiscardImages
              GenePix.DiscardSettings
              GenePix.DiscardResults

     GenePix.LoadFile("C:\Axon\Scripts\GenePixPro6\Auto_1.gps")
              GenePix.Option(0) = 0
              GenePix.Option(1) = 0
              GenePix.Option(2) = 1
              GenePix.Option(3) = 1
              GenePix.Option(4) = 0
              GenePix.DualScan
         End Sub

     </script>

</body>
</html>
```

Exercise 42

This exercise requires two files: the following script, and a settings file that contains one block and a scan area over the first block of demo data (such as ("C:\Axon\Scripts\GenePixPro6\Auto_1.gps"):

```
<html>
<head>
     <title>
         Sample GenePix Report Exercise 42
     </title>
</head>

<body language="VBscript" onload="callback()">

     <script language="vbscript">
         Option Explicit
         ' Assign some GenePix Object Model references to
variables

         Dim GenePix
         set GenePix = window.external
         Dim Scanner
         Set Scanner = GenePix.Scanner

         sub callback()
              GenePix.SwitchToTab(0)
              GenePix.DiscardImages
              GenePix.DiscardSettings
              GenePix.DiscardResults

     GenePix.LoadFile("C:\Axon\Scripts\GenePixPro6\Auto_1.gps")
              GenePix.Option(0) = 0
              GenePix.Option(1) = 0
              GenePix.Option(2) = 1
              GenePix.Option(3) = 1
              GenePix.Option(4) = 0
              Scanner.PMT(0) = 600
              Scanner.PMT(1) = 650
              GenePix.DualScan
         end sub

     </script>

</body>
</html>
```

Standard Web Color Names

I

Table I-1 Standard Web Color Names

Color Name	RGB Value	Color Name	RGB Value
aliceblue	#F0F8FF	algae (yellow green)	#9ACD32
antiquewhite	#FAEBD7	aqua	#00FFFF
aquamarine	#7FFFD4	azure	#F0FFFF
beige	#F5F5DC	bisque	#FFE4C4
black	#000000	blanchedalmond	#FFEBCD
blue	#0000FF	blueviolet	#8A2BE2
brown	#A52A2A	burlywood	#DEB887
cadetblue	#5F9EA0	chartreuse	#7FFF00
chocolate	#D2691E	coral	#FF7F50
cornflowerblue	#6495ED	cornsilk	#FFF8DC
crimson	#DC143C	cyan	#00FFFF
darkblue	#00008B	darkcyan	#008B8B
darkgoldenrod	#B8860B	darkgray	#A9A9A9
darkgreen	#006400	darkkhaki	#BDB76B
darkmagenta	#8B008B	darkolivegreen F	#556B2F
darkorange	#FF8C00	darkorchid	#9932CC
darkred	#8B0000	darksalmon	#E9967A
darkseagreen	#8FBC8F	darkslateblue	#483D8B
darkslategray	#2F4F4F	darkturquoise	#00CED1
darkviolet	#9400D3	deeppink	#FF1493
deepskyblue	#00BFFF	dimgray	#696969
dodgerblue	#1E90FF	firebrick	#B22222
floralwhite	#FFFAF0	forestgreen	#228B22
fuchsia	#FF00FF	gainsboro	#DCDCDC
ghostwhite	#F8F8FF	gold	#FFD700
goldenrod	#DAA520	gray	#BEBEBE
green	#008000	greenyellow	#ADFF2F
honeydew	#F0FFF0	hotpink	#FF69B4
indianred	#CD5C5C	indigo	#4B0082
ivory	#FFFFF0	khaki	#F0D58C
lavender	#E6E6FA	lavenderblush	#FFF0F5
lawngreen	#7CFC00	lemonchiffon	#FFFACD
lightblue	#ADD8E6	lightcoral	#F08080
lightcyan	#E0FFFF	lightgoldenrodyellow	#FAFAD2
lightgreen	#90EE90	lightgrey	#D3D3D3

Table I-1 Standard Web Color Names (cont'd)

Color Name	RGB Value	Color Name	RGB Value
lightpink	#FFB6C1	lightsalmon	#FFA07A
lightseagreen	#20B2AA	lightskyblue	#87CEFA
lightslategray	#778899	lightsteelblue	#B0C4DE
lightyellow	#FFFFE0	lime	#00FF00
limegreen	#32CD32	linen	#FAF0E6
magenta	#FF00FF	maroon	#800000
mediumaquamarine	#66CDAA	mediumblue	#0000CD
mediumorchid	#BA55D3	mediumpurple	#9370DB
mediumseagreen	#3CB371	mediumslateblue	#7B68EE
mediumspringgreen	#00FA9A	mediumturquoise	#48D1CC
mediumvioletred	#C71585	midnightblue	#191970
mintcream	#F5FFFA	mistyrose	#FFE4E1
moccasin	#FFE4B5	navajowhite	#FFDEAD
navy	#000080	oldlace	#FDF5E6
olive	#808000	olivedrab	#6B8E23
orange	#FFA500	orangered	#FF4500
orchid	#DA70D6	palegoldenrod	#EEE8AA
palegreen	#98FB98	paleturquoise	#AFEEEE
palevioletred	#DB7093	papayawhip	#FFEFD5
peachpuff	#FFDAB9	peru	#CD853F
pink	#FFC0CB	plum	#DDA0DD
powderblue	#B0E0E6	purple	#800080
red	#FF0000	rosybrown	#BC8F8F
royalblue	#4169E1	saddlebrown	#8B4513
salmon	#FA8072	sandybrown	#F4A460
seagreen	#2E8B57	seashell	#FFF5EE
sienna	#A0522D	silver	#C0C0C0
skyblue	#87CEEB	slateblue	#6A5ACD
slategray	#708090	snow	#FFFAFA
springgreen	#00FF7F	steelblue	#4682B4
tan	#D2B48C	teal	#008080
thistle	#D8BFD8	tomato	#FF6347
turquoise	#40E0D0	violet	#EE82EE
wheat	#F5DEB3	white	#FFFFFF
whitesmoke	#F5F5F5	yellow	#FFFF00

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z