GenePattern Programmer's Guide

Table of Contents

Introduction

This guide assumes that you are a programmer and familiar with GenePattern. If you are new to GenePattern, work through the GenePattern Tutorial before reading the more detailed information in this guide.

As a programmer, you generally work with GenePattern in one of two ways:

- **Creating GenePattern tasks.** Each GenePattern task invokes a program that executes a desired function. You can use any language to write a program that can then be invoked as a GenePattern task. For more information, see:

    - Writing Modules for GenePattern: This section provides tips for writing code that will be invoked as a GenePattern task.

    - Writing MATLAB Modules for GenePattern: This section describes how to address MATLAB licensing and distribution issues. Read this section if you are writing MATLAB code that will be invoked as a GenePattern task.

- **Accessing GenePattern from Java, MATLAB, or R.** GenePattern libraries for these three programming environments make it easy for your applications to run GenePattern tasks and retrieve analysis results. Each library supports arbitrary scripting, access to GenePattern modules via function calls, and development of new methodologies that combine modules in arbitrarily complex combinations. For more information, see:

    - Using GenePattern from Java

    - Using GenePattern from MATLAB

    - Using GenePattern from R

Writing Modules for GenePattern

Creating a GenePattern task is a two-step process:

1. Write a program that executes the desired function. For more information, see Guidelines for Writing Programs for GenePattern Modules.

2. Use the GenePattern Web Client to create a GenePattern task that invokes the program that you have written. For more information, see Creating Tasks.

Guidelines for Writing Programs for GenePattern Modules

When writing a program that will be run as a GenePattern task, keep in mind the following:

- **Use the programming language of your choice.** You can write the program in the language of your choice. You can use a compiled language, such as C, to create an executable or you can use a scripting language, such as Perl, to create a script that is run by an interpreter.

- **Write messages to standard error and standard output.** GenePattern tasks are run on the server. The user provides arguments and retrieves results, but does not interact during task execution. If necessary, write normal output to standard output (stdout) and error messages to standard error (stderr); avoid writing error messages to standard output. GenePattern captures stdout and stderr in log files, which can be retrieved by the user.

- **Write output files to the current working directory.** When a task completes, GenePattern displays the output files that are in the current working directory. Files written to other locations are not displayed as task output files (otherwise known as analysis result files).

- **Read module data files from <libdir>.** If your module needs to read from any data files which are part of the module (rather than user input), it will need to know the directory where the module lives on the server; that is, `<libdir>`.

- **Read and write standard GenePattern file formats.** When reading and writing data files, you generally want to use the standard GenePattern file formats, as described in File Formats. This makes it easier for users to analyze their data using a combination of GenePattern modules. If you choose to use your own unique file formats, be aware that other GenePattern modules will not be able to read those files.

    For Java, MATLAB, and R, GenePattern provides libraries that include methods for reading and writing GenePattern files (such as res, gct, and odf files). These libraries are designed for accessing GenePattern from the Java, MATLAB, and R environments, but are also useful when writing modules to be invoked by GenePattern. For instructions on downloading the libraries, see Using GenePattern from Java, Using GenePattern from MATLAB, or Using GenePattern from R.

4

- **Use parameter flags.** When designing the program and its command line, use parameter flags (for example, -f *input_file*) rather than relying on parameter positions. Parameter flags allow users to build command lines with variable numbers of arguments, which makes it easy to omit optional parameters.

  When writing R code, if you have optional input parameters on the command line, you must use named rather than positional parameters in the command line definition; for example: `input.filename=<input.filename>`. If you name all of the parameters, then you can use ellipses rather than listing the parameter names in the function definition (for example: `myfunc <- function(...)`) at the expense of clarity in documenting input parameters.

- **Process all parameters as strings.** All command line parameters are passed to your code as strings, even if a parameter is apparently numeric. If your code expects a numeric argument, explicitly convert the string argument to a number; for example, as.integer(arg).

- **Avoid absolute pathnames.** When writing code to be used with GenePattern, avoid absolute pathnames. For example, in perl, specify the interpreter on the command line rather than embedding the interpreter in the script; that is, use the command line "`perl myscript`" rather than including "`#!/usr/bin/perl`" as the first line of the `myscript.pl` file.

- **Avoid Windows forbidden filenames.** Machines running Windows cannot accept files with the following names, regardless of the file extension: con, prn, aux, nul, com1, com2, com3, com4, lpt1, lpt2, lpt3. For cross-platform compatibility, avoid files with these names.

- **Adapting R code.** When you create a task in GenePattern, you specify the command line that invokes the program that performs the desired function. Generally, the command line includes arguments, such as the parameters for the algorithm and the data file to analyze. For example, the following command line invokes the `myfunc()` function in the R script named `myscript.R`, passing a single parameter, `input.filename`:

  ```
  <R> <libdir>myscript.R myfunc <input.filename>
  ```

  Calling R script from a command line is possible, but generally not useful because you cannot pass arguments to the script. To pass arguments to your R code, create a function. For example:

  ```
  myfunc <- function ( input.filename )
    ... (your R-code here)
  ```

Creating Tasks

To create a task that invokes the program that you have written, use the GenePattern Web Client:

1. Display the Web Client home page. (If you are unfamiliar with the GenePattern Web Client, see the Web Client Guide.)

2. Open the task definition form in one of the following ways:

   ▪ Click *Create* in the Tasks pane.

   ▪ Click *create task* under Tasks and Pipelines in the right column.

   ▪ View or edit a task and use the *Clone* button to create an identical task.

   The Web Client displays the task definition form:
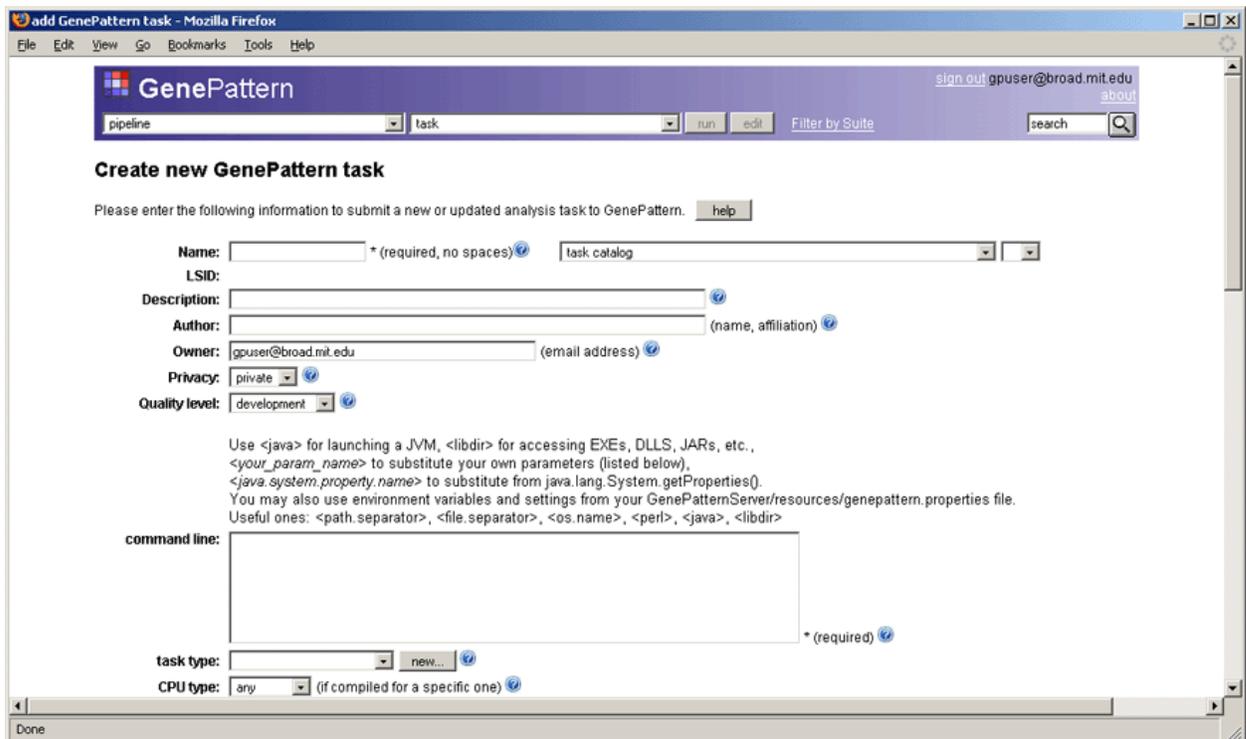
3. Click *Help* for instructions on how to complete the task definition form.

4. Complete the task definition form.

5. Click *Save* to create the task.

Note: By adding a task, a user can execute arbitrary code on the GenePattern server. Because arbitrary code may include malicious code, consider taking precautions to protect your server: for example, employing virus scanner software and restricting access to appropriately privileged (non-root) users. For more information about securing your server, see Securing the Server in the *Web Client Guide*.

Tutorial: Creating a Task

While creating a task is not difficult, it can be intimidating the first time. The following brief tutorial walks you through the process by helping you create a task named `log_transform`. The program invoked by this task is a Perl script, `gp_tutorial_files/log_transform/log_transform.pl`, which log-transforms all positive values in a data set and sets all negative or zero values to zero. This Perl script is part of the GenePattern tutorial data set, which is downloaded during a full installation of GenePattern and is also available on the GenePattern web site.

To create the `log_transform` task:

1. Display the Web Client home page.

2. In the Tasks pane, click *Create*. The Web Client displays the task definition form. (The task definition form, with the tutorial data filled in, appears at the end of the procedure.)

3. Enter the following information in the first few fields:

   ▪ **Name**: LogTransform

   ▪ **Description**: Log transform a data (gct) file.

   ▪ **Author**: Your name and affiliation.

   ▪ **Privacy**: Select private. This means that only you (users logged in using the user name in the *Owner* field) can view and run the task. Public allows all users connected to this server to view and run the task.

   ▪ **Quality level**: Select preproduction. This indicates that you have finished development, but are not yet ready for production.

4. Enter the following text in the *command line* field (if you are working online, cut and paste this text into the field):

   `<perl> <libdir>log_transform.pl -F <input.filename> -o <output.file>`

6

Typically, you enter the command line as a combination of fixed text and variables defined by GenePattern. This allows the command line to be independent of the operating environment and allows different values to be specified at different invocations of the command. This command line uses the following variables:

- <perl> represents the full path to the Perl installation used by GenePattern.

  Similar variables are available for <java> and <R>. If your program was run by another interpreter, such a Python or LISP, you could use an absolute pathname (for example, /usr/bin/python) or create a new GenePattern variable for the interpreter pathname (for example, python=/usr/bin/python) in the `GenePatternServer/resources/genepattern.properties` file. If your program was an executable file, you would not need an interpreter, so would not need this variable.

- <libdir> represents the full path to the directory that contains the files for this task, including the program file.

- The Perl script, `log_transform.pl`, expects two parameters, an input file and an output file name: `-F <input.filename> -o <output.file>`. When your program has parameters, you include them in the command line and also define them in the *Parameters* field, which appears below.

5. Enter the following information in the next few fields:

   - **task type**: Preprocess & Utilities

   - **CPU**: Any

   - **Operating system**: Any

   - **Language**: Perl

   - **Version comment**: A brief note of what is special or interesting about this version.

   - **output description**: Select gct from the list of formats.

     The *output description* field defines all file formats used by your module. To select multiple formats, use Shift+click (Option+click or Apple+click). Include both input and output file formats in this list.

6. Use the *Support files* field to upload your program, any other files needed to execute the task, and documentation for the task (a file with a text extension, such as .pdf, .doc, or .txt):

   1. Click the *Browse* button at the end of the first line in the *Support files* field. The Web Client displays the File Upload window.

   2. Navigate to the `gp_tutorial_files/log_transform` directory, select the file `log_transform.pl`, and click *Open*. This is the script that implements the task.

   3. Click the *Browse* button at the end of the second line in the *Support files* field. The Web Client displays the File Upload window.

   4. Select the file `LogTransform.pdf` and click *Open*. This is the documentation for the task. When a GenePattern user displays your task and clicks the *Help* button, GenePattern displays the support file that has a text extension (such as .pdf .doc, or .txt).

7. Use the *Parameters* field to describe your two program parameters: input.filename and output.file. The parameter names and descriptions that GenePattern displays when a user runs your task are the parameter names and descriptions that you provide here.

   In the first row of the *Parameters* field, enter the following information for input.filename:

   - **name**: input.filename

   - **description**: The dataset to be transformed (gct format).

   - **type**: choose input file

   - **fileFormat**: choose gct

   In the second row, enter the following information for output.file:

   - **name**: output.file

   - **description**: The name of the new transformed file.

   - **type**: choose text (this is a name that the user enters)

8. Click *save*. The Web Client displays a message informing you that the task has been saved.

9. Run the task to confirm that it has been added to the GenePattern server correctly.

GenePattern

pipeline | task | run | edit | Filter by Suite | search

Create new GenePattern task

Please enter the following information to submit a new or updated analysis task to GenePattern. [help]

Name: `LogTransform` * (required, no spaces) | task catalog

LSID:

Description: `Log transform a data (gct) file.`

Author: `Gene P. User` (name, affiliation)

Owner: `gpuser` (email address)

Privacy: private

Quality level: preproduction

Use <java> for launching a JVM, <libdir> for accessing EXEs, DLLS, JARs, etc.,
<*your_param_name*> to substitute your own parameters (listed below),
<*java.system.property.name*> to substitute from java.lang.System.getProperties().
You may also use environment variables and settings from your GenePatternServer/resources/genepattern.properties file.
Useful ones: <path.separator>, <file.separator>, <os.name>, <perl>, <java>, <libdir>

command line:
```
<perl> <libdir>log_transform.pl -F <input.filename> -o
<output.file>
```
* (required)

task type: Preprocess & Utilities | new...

CPU type: any (if compiled for a specific one)

operating system: any (if operating system-dependent)

Language: Perl | min. language version:

Version comment: `New task`

output description: output file format(s): gct / Gene List / gtr | new...

Support files: The actual program plus any required libraries will will be accessible to your command line as <libdir>*filename*
(jar, dll, exe, pl, doc, etc.)
`\\Krypton\hkuehn\GenePattern\TutorialData\gp_tutorial_files\log_transform\` Browse...
`\\Krypton\hkuehn\GenePattern\TutorialData\gp_tutorial_files\log_transform\` Browse...
Browse...
Browse...
Browse...

Current files:

Parameters: The names of these parameters will be available for the command line (above) in the form <name>.
Parameters with "filename" in their name will be treated as input filenames.

	name	description (optional)	choices (optional semicolon-separated list of choices.)	default value	optional	prefix when specified	type	file format
example:	min	values below minimum will be set to this value	2=green, default;0=red;1=blue	2				
	input.filename	The dataset to be transformed (gct format).			☐		input file	eps / gct / Gene List
	output.file	The name of the new transformed file.			☐		text	
					☐		text	
					☐		text	
					☐		text	

save | clear | help

Writing MATLAB Modules for GenePattern

If you are writing MATLAB code to be invoked as a GenePattern task, follow the guidelines in Writing Modules for GenePattern. In addition, for MATLAB code, you must address licensing and distribution issues, as described in this section:

- Two Approaches: Direct and Compiled
- MATLAB Versions
- Adapting Your MATLAB Code
- Compiling Your MATLAB Code

8

Two Approaches: Direct and Compiled

You can invoke a MATLAB executable from a GenePattern module using one of two approaches: the direct approach or the compiled approach. Following are brief descriptions of each approach, including its advantages and disadvantages:

● **Direct approach**. In the direct approach, the GenePattern task directly invokes the MATLAB executable, which executes your M-code. This approach is best suited for use on a standalone GenePattern server, where you already have a MATLAB license and you will not redistribute the MATLAB-based GenePattern modules to other users who do not have their own MATLAB licenses. The advantages to this approach are: it is the simplest way to getting your M-code running on GenePattern, it can be used for any MATLAB and GenePattern supported platform, and it allows for easier modification of the M-code files as you modify your analysis. The disadvantages of this approach are: it requires a MATLAB license for each concurrent user on the GenePattern server machine and, if you change platforms, you must change the command line because different platforms have different methods of passing arguments to MATLAB.

● **Compiled approach**. In the compiled approach, you use the MATLAB Compiler to generate a standalone executable; the GenePattern task then invokes that executable. The advantages to this approach are: it allows redistribution of the module to other GenePattern users who do not have their own MATLAB license and it can be run on a shared server without needing to get a MATLAB license for each concurrent user. The disadvantages are: it requires you to have a MATLAB Compiler license, it can be used only on platforms supported by the MATLAB Compiler, it must be compiled separately for each platform, and the GenePattern server must have the MATLAB Component Runtime (MCR) installed before it can run the compiler-generated executable.

If you are simply using your M-code on your standalone GenePattern server, the direct approach is simpler; however, if you want to give copies of your M-code to other people or deploy your M-code on a shared GenePattern server, the compiled approach is preferred. The compiled approach may provide slightly better performance for fast running tasks since the startup delay will be shorter, but the actual execution time will be approximately the same for either approach.

MATLAB Versions

The instructions in this section are based on the following MATLAB versions:

● MATLAB 7.1 (part of Release 14). Earlier versions of MATLAB use a different (deprecated) mechanism for deployment, which is not compatible with the instructions provided in this guide. For Mac OS X, these instructions were tested using MATLAB 7.2.

● MATLAB Compiler 4.0 or later. The MATLAB Compiler is currently available on Windows, Unix, and Mac OS X; therefore, these are the only platforms on which you may deploy a compiled MATLAB-based task. For Mac OS X, these instructions were tested using the MATLAB Compiler 4.4.

   The MATLAB Compiler generates executables only for the platform on which it is executing. For example, if you create a MATLAB executable on Windows, the MATLAB-based module that invokes that executable can only be deployed on a GenePattern server running under Windows. Therefore, you need a MATLAB (and MATLAB Compiler) license for each platform on which you wish to deploy your executable.

Adapting Your MATLAB Code

When you create a task in GenePattern, you specify the command line that invokes the program that performs the desired function. Generally, the command line includes arguments, such as the parameters for the algorithm and the data file to analyze.

Calling script M-code from a command line is possible, but generally not useful because you cannot pass arguments to the script. To pass arguments to your M-code, create a no-return entry function to serve as the top level call into MATLAB. The following example defines a no-return entry function that accepts two parameters:

```
function analyzeThis ( filename, whatToWrite )
 ... (your M-code here)
```

Writing Modules for GenePattern provides additional guidelines for writing code that will run as a GenePattern task.

Compiling Your MATLAB Code

If you do not plan to use the compiled M-code approach, skip this section and continue with Distributing Your MATLAB Code.

Compiling your MATLAB M-code into a standalone executable is described in the MATLAB Compiler Documentation. Please refer to this documentation to understand all of the options available to you. To summarize the simplest case, from within MATLAB, at the MATLAB prompt, execute the following command:

```
mcc -m analyzeThis
```

where *analyzeThis* is the name of your entry function. This command generates the following files in your `$MATLAB_ROOT/work` directory:

analyzeThis (Linux, Mac OS X) *or* analyzeThis.exe (Windows)	Executable file
analyzeThis.ctf	Component Framework file
analyzeThis.c (Linux, Windows)	C language Source Code
analyzeThis.h (Linux, Windows	C Language Header file
analyzeThis_main.c	C language Source Code
analyzeThis_mcc_component_data.c	C language Source Code

Note: To use the MATLAB compiler on Mac OS X, you must have Xcode 2.2 installed; minimally, the Developer Tools, gcc 4.0, gcc 3.3, Mac OS X SDK, and BSD SDK. These instructions were tested using Xcode 2.2.1.

Distributing Your MATLAB Code

After writing your MATLAB code, use the GenePattern Web Client to create a GenePattern task that invokes the code that you have written. Creating Tasks describes how to create a GenePattern task. This section provides supplemental information for MATLAB:

● Direct Approach Distribution

● Compiled Approach Distribution

Direct Approach Distribution

Creating Tasks describes how to create a GenePattern task that invokes the code that you have written. This section provides additional information that applies when you are directly calling the MATLAB executable from the GenePattern task:

● Windows Command Line

● Preferred Command Line

Windows Command Line

On Windows, your GenePattern task definition form can contain a simple command line that calls MATLAB with the -r flag to execute your function; for example:

```
matlab -nosplash -r "analyzeThis <p1> <p2>"
```

This example invokes MATLAB without the splash screen (`-nosplash`) and directs it to execute the quoted command, where `p1` and `p2` are parameters that you specify in the GenePattern task definition form and that are passed to the MATLAB command line as Strings. MATLAB looks for the function `analyzeThis` on the MATLAB path; therefore, it is not necessary to upload the function as a support file, although it is recommended.

To ensure that the GenePattern server can call the MATLAB executable, you typically add the MATLAB directory to your PATH system environment variable. (Alternatively, you can enter the full path to the MATLAB executable on the command line, but this makes it more difficult to deploy the module on other GenePattern servers.)

To check that MATLAB is on your path:

1. Open a DOS window.

2. Type `matlab` and press Enter.

If the MATLAB application starts, MATLAB is on your path.

If MATLAB is not on your path, add it:

1. Select *Start>Settings>Control Panel*.

2. Double-click *System*.

3. Select the Advanced tab.

4. Click the *Environment Variables* button.

5. Select or create the PATH variable.

6. Add the `$MATLAB_ROOT/bin` directory to the path.

Open a new DOS window and check again that MATLAB is on your path.

Preferred Command Line

On platforms other than Windows, the execution of the command line differs slightly due to variations in the Java Virtual Machines (VMs) that GenePattern is running. If you use the simple `matlab` command, as described for Windows, the Java VMs on these platforms attempt to parse and quote the command line resulting in MATLAB generating errors in its `eval` function.

On these platforms, you must use a wrapper Java class to launch MATLAB. This wrapper class also works on Windows and does not rely on the PATH variable, which makes it the preferred method for implementing the direct approach on any platform.

To use the wrapper Java class:

1. On the GenePattern task definition form, add the `runmatlab.jar` file as a support file. To request a copy of this file, send e-mail to gp-help@broad.mit.edu; alternatively, the java source code for the RunMatlab wrapper class is included here: RunMatlab.java.

2. Write your command line as follows:

   ```
   <java> -cp <libdir>runmatlab.jar RunMatlab analyzeThis <p1> <p2>
   ```

   Where `analyzeThis` is the name of your MATLAB entry function name and <p1> and <p2> are the arguments to the function. The `RunMatlab` class ensures that the arguments are correctly written out and calls MATLAB with the `-nosplash` and `-nodisplay` arguments.

Compiled Approach Distribution

Creating Tasks describes how to create a GenePattern task that invokes the code that you have written. This section provides additional information that applies when you are compiling your M-code into a standalone executable and invoking that executable from the GenePattern task:

● Preparing the GenePattern Server

● Writing the Launcher Script

● Writing the Module Command Line

● Adding Support Files

● Distribution Licensing

Preparing the GenePattern Server

To run a standalone executable generated by the MATLAB Compiler, the GenePattern server must have the MATLAB Component Runtime (MCR) installed. This is a collection of shared libraries, which contains the runtime code for MATLAB, that is used by the standalone application. If the GenePattern server has MATLAB installed, you do not need to install the MCR; it is already installed.

Full details for installing the MCR can be found in the MATLAB Compiler documentation, in the section titled "Deployng Components to Other Machines". To summarize this documentation, on the GenePattern server machine, you need to run the MCRInstaller:

On **Windows**, to run the MCRInstaller:

1. Copy `<matlabroot>\toolbox\compiler\deploy\win32\MRCInstaller.exe` to the server machine.

2. Run `MCRInstaller.exe`.

On **Linux**, to run the MCRInstaller:

1. In MATLAB, at the MATLAB prompt, execute the command `buildmcr`.

2. Copy `<matlabroot>/toolbox/compiler/deploy/MCRInstaller.zip` to the server machine.

3. On the server machine, unzip `MCRInstaller.zip` into a directory (`<mcr_root>`).

4. Update the dynamic library path for the user running the GenePattern server:

```
setenv LD_LIBRARY_PATH
    <mcr_root>/runtime/glnx86:
    <mcr_root>/sys/os/glnx86:
    <mcr_root>/sys/java/jre/glnx86/jre1.4.2/lib/i386/client:
    <mcr_root>/sys/java/jre/glnx86/jre1.4.2/lib/i386:
    <mcr_root>/sys/opengl/lib/glnx86:${LD_LIBRARY_PATH}
```

On **Mac OS X**, to run the MCRInstaller:

1. In MATLAB, at the MATLAB prompt, execute the command `buildmcr`.

2. Copy `<matlabroot>/toolbox/compiler/deploy/MCRInstaller.zip` to the server machine.

3. On the server machine, unzip `MCRInstaller.zip` into a directory (`<mcr_root>`).

4. Update the library path for the user running the GenePattern server:

```
setenv DYLD_LIBRARY_PATH
    <mcr_root>/<ver>/runtime/mac:
    <mcr_root>/<ver>/sys/os/mac:
    <mcr_root>/<ver>/bin/mac:
    /System/Library/Frameworks/JavaVM.framework/JavaVM:
    /System/Library/Frameworks/JavaEmbedding.framework/JavaEmbedding:
    /System/Library/Frameworks/JavaVM.framework/Libraries
setenv XAPPLRESDIR <mcr_root>/<ver>/X11/app-defaults
```

Writing the Launcher Script

When the MATLAB Compiler generates a standalone executable, it also generates a Component Framework (.ctf) file. The .ctf file must be on the path when you run the standalone executable. The easiest way to address this requirement is to create a launcher script (.bat or .sh file) that adds the .ctf file to the PATH or LIBPATH and then runs the standalone executable.

On Windows, for example, to launch the MATLAB executable `analyzeThis.exe`, create a launcher script, `mllaunch.bat`, that contains the following lines:

```
set LIBDIR=%1
set PATH=%LIBDIR%;%PATH%
analyzeThis %2 %3
```

On Linux, for example, to launch the MATLAB executable `analyzeThis.exe`, create a launcher script, `mllaunch.sh`, that contains the following lines:

```
#!/bin/csh
export MCR_ROOT=<path where you installed the files from MCRInstaller.zip>
export LD_LIBRARY_PATH=$1:$MCR_ROOT/runtime/glnx86:$MCR_ROOT/sys/os/glnx86:\
  $MCR_ROOT/sys/java/jre/glnx86/jre1.4.2/lib/i386/client:\
  $MCR_ROOT/sys/java/jre/glnx86/jre1.4.2/lib/i386:\
  $MCR_ROOT/sys/opengl/lib/glnx86

export PATH=$1:$PATH
chmod a+x $1/analyzeThis
analyzeThis $2 $3
```

The chmod line sets the executable permission on the executable file; by default, the GenePattern server does not set this permission for uploaded files.

On Mac OS X, for example, to launch the MATLAB executable `analyzeThis.exe`, create a launcher script, `mllaunch.sh`, that contains the following lines:

```
#!/bin/sh
export MCR_ROOT=/Volumes/os9/gpserv
export LD_LIBRARY_PATH=$1:/Volumes/os9/matlab7.2/sys/os/mac:
    /Volumes/os9/matlab7.2/bin/mac/
export DYLD_LIBRARY_PATH=$LD_LIBRARY_PATH
export PATH=$1:$PATH
chmod a+x $1/writeToFile
writeToFile $2 "$3"
```

The chmod line sets the executable permission on the executable file; by default, the GenePattern server does not set this permission for uploaded files.

Writing the Module Command Line

On the GenePattern task definition form, write a command line calls the launcher script, passing the <libdir> parameter as the first argument (so that it can be added to the path).

On Windows, the following command line calls the launcher script, `mllaunch.bat`:

```
<libdir>mllaunch.bat <libdir> <param1> <param2>
```

On Linux or **Mac OS X**, the following command line calls the launcher script, `mllaunch.sh`:

```
sh <libdir>mllaunch.sh <libdir> <param1> <param2>
```

In both command lines, the first <libdir> sets the path to the `mllaunch` script. The second <libdir> is passed as the first argument to the script so that the script can add this directory to the appropriate environment variables. The <param1> and <param2> variables are parameters to the MATLAB application, which you define in the task definition form and specify in the command line as usual.

Adding Support Files

For the compiled approach, you must specify at least two support files for the MATLAB application: the executable file and .ctf file. If your application requires additional files for its execution, also add those files as support files.

Distribution Licensing

Should you choose to distribute your MATLAB based module to others, you must ensure you are in compliance with the MATLAB licensing agreement:

```
http://www.mathworks.com/company/aboutus/policies_statements/agreement.pdf
```

Following are a few key points for GenePattern developers:

- You may not distribute code that uses MATLAB and that competes with any of The MathWorks products.

- You may not modify or remove any license file included with the MCR Libraries.

- Users of your GenePattern modules must be made aware of the MATLAB license agreement in documentation and accept it before installing your modules.

- Your MATLAB application must have an about box or equivalent "visible" location that includes the legend "MATLAB copyright 1984-*yyyy* the MathWorks Inc.", where *yyyy* is the year you released your module.

Please refer to the MATLAB licensing agreement for exact details. You are responsible for reviewing and complying with the MATLAB software license. The above summary does not exempt you from this responsibility.

Example: Deploying a Compiled MATLAB Application

This section provides a step-by-step example of deploying a simple M-file application as a GenePattern module on a GenePattern server. Where the instructions are platform specific, the example shows instructions for Windows, Linux, and Mac OS X.

Writing the M-file

The first step is writing the MATLAB M-file that you want to share. For this example, write a simple application that takes a filename and a String and writes the String out to a file with the given name. This application consists of the following lines:

```
% write the variable whatToWrite to a file called filename in the current
directory
     fid = fopen(filename,'w');
     fprintf(fid,'#writing to a file\n\n');
     fprintf(fid,whatToWrite);
     fclose(fid);
```

Adapting the M-file

To call the M-file from the command line and pass it parameters, you must turn this script into a no-return function. To do this, add a function definition line at the start of the M-file and save the file using the name of the function (for example, writeToFile.m).

```
function writeToFile( filename, whatToWrite)
% write the parameter whatToWrite to a file called filename in the current
directory
     fid = fopen(filename,'w');
     fprintf(fid,'#writing to a file\n\n');
     fprintf(fid,whatToWrite);
     fclose(fid);
```

Compile the M-file

Within the MATLAB environment, call the MATLAB Compiler to convert this function into an application:

```
>> mcc -m writeToFile
```

Within the current working directory, this creates a number of files, including the following:

- writeToFile.exe (Windows) or
  writeToFile (Linux, Mac OS X)

- writeToFile.ctf

Note: To use the MATLAB compiler on Mac OS X, you must have Xcode 2.2 installed; minimally, the Developer Tools, gcc 4.0, gcc 3.3, Mac OS X SDK, and BSD SDK. These instructions were tested using Xcode 2.2.1.

Prepare the GenePattern Server

Install the MATLAB Component Runtime (MCR) on the GenePattern server, if you have not done so already. If the GenePattern server has MATLAB installed, it also has the MCR installed.

Windows

To install the MCR:

1. Copy `<matlabroot>\toolbox\compiler\deploy\win32\MRCInstaller.exe` to the GenePattern server machine.

2. At the DOS prompt, or from Windows Explorer, run the following:

   ```
   MCRInstaller.exe
   ```

Linux or Mac OS X

To install the MCR:

1. Within the MATLAB environment, create the MCRInstaller zip file:

   ```
   >> buildmcr mcrdir
   ```

   This creates a directory, `mcrdir`, beneath the current working directory and creates a file within that directory called `MCRInstaller.zip`.

2. Copy the zip file to your GenePattern server (if it is a different machine) and install it into a directory. For example, add a directory, `matlab`, under the GenePattern server directory and install the library files in `MCRInstaller.zip` into that directory:

```
cd GenePatternServer
mkdir matlab
cd matlab
cp <path to mcrinstaller.zip>MCRInstaller.zip .
unzip MCRInstaller.zip
```

Create the Launcher Script

Create the launcher script that sets the environment variables and then calls the MATLAB application.

Windows

Create the launcher script as a batch file that sets the PATH variable for the environment and then calls the MATLAB application. To do so, in a text editor, create the following `mllaunch.bat` file:

```
set LIBDIR=%1
set PATH=%LIBDIR%;%PATH%
writeToFile %2 %3
```

Linux

Create the launcher script as an .sh file that sets the PATH and LD_LIBRARY_PATH variables for the environment, ensures that the application is executable, and then calls the MATLAB application. To do so, in a text editor, create the following `mllaunch.sh` file:

```
#!/bin/csh
export MCRROOT=/home/username/GenePatternServer/matlab/v70
export
LD_LIBRARY_PATH=$1:$MCRROOT/runtime/glnx86:$MCRROOT/sys/os/glnx86:$MCRROOT/sys
/java/jre/glnx86/jre1.4.2/lib/i386/client:$MCRROOT/sys/java/jre/glnx86/jre1.4.
2/lib/i386:$MCRROOT/sys/opengl/lib/glnx86
export PATH=$1:$PATH
chmod a+x $1/testTwo
writeToFile $2 $3
```

Note that the MCR_ROOT variable is set to the `v70` directory, which you created by unzipping `MCRInstaller.zip`.

Mac OS X

Create the launcher script as an .sh file that sets the LD_LIBRARY_PATH and DYLD_LIBRARY_PATH variables for the environment, ensures that the application is executable, and then calls the MATLAB application. To do so, in a text editor, create the following `mllaunch.sh` file:

```
#!/bin/sh
export MCR_ROOT=/Volumes/os9/gpserv
export LD_LIBRARY_PATH=$1:/Volumes/os9/matlab7.2/sys/os/mac:
   /Volumes/os9/matlab7.2/bin/mac/
export DYLD_LIBRARY_PATH=$LD_LIBRARY_PATH
export PATH=$1:$PATH
chmod a+x $1/writeToFile
writeToFile $2 "$3"
```

Create the GenePattern Module

In the GenePattern Web Client, create a task that executes the launcher script.

Windows

- For the command line, enter the following:

  `sh <libdir>mllaunch.bat <libdir> <fname> <txt>`

- Define two parameters:

15

- ▪ <fname> for the output file name
- ▪ <txt> for the text to write to the file
- Include the following support files:
  - ▪ mllaunch.bat
  - ▪ whatToWrite.exe
  - ▪ whatToWrite.ctf

Linux or Mac OS X
- For the command line, enter the following:

  ```
  sh <libdir>mllaunch.sh <libdir> <fname> <txt>
  ```
- Define two parameters:
  - ▪ <fname> for the output file name
  - ▪ <txt> for the text to write to the file
- Include the following support files:
  - ▪ mllaunch.sh
  - ▪ whatToWrite
  - ▪ whatToWrite.ctf

Save the Module and Test It

Save the task and execute it. The task should create two files:

- A stdout file that contains execution information.
- A file with the name and text that you specified.

Debugging (Linux Only)

If the following error appears in the stdout file, you have not correctly set the path to the libraries that you installed from `MCRInstaller.zip`:

```
error while loading shared libraries: libmwmclmcrrt.so.7.0: cannot open shared
object file: No such file or directory
```

Double check the path. If it is correct, you may be using a different Unix shell than the one used in this example. Check that the `mllaunch.sh` file uses the correct command (`export` in this example) to set PATH and LD_LIBRARY_PATH.

Using GenePattern from Java

Using Java as a GenePattern client allows you to run GenePattern tasks and visualizers from within a Java application. This section describes how you can use the GenePattern Java library to run GenePattern analyses as easily as calling a routine. It contains the following topics:

- Getting Started in Java
- GenePattern Java Library
- Running a Program
- Using LSIDs from Java

Getting Started in Java

If you are not familiar with Java, see the http://java.sun.com website, which provides downloadable programs, samples, tutorials, and book suggestions.

GenePattern Java Library

The GenePattern Java library allows you to invoke a GenePattern task as if it were a local Java method running on your client and to get back from the task a list of result files. A zip file containing the Java library (and Javadoc that describes the API for accessing the server and running tasks) is available on your GenePattern server.

To download the GenePattern Java library to your computer:

1. Start the GenePattern Web Client.

2. Under Programming Libraries in the right column, click the *zip* link for the Java library.

3. After downloading the zip file, unzip it into the directory where you will be doing your Java development.

Running a Program

This section explores a simple Java application that preprocesses a dataset and displays it using the HeatMapViewer. The included code can be copied and pasted into your Java program so that you can try it out, modify it, and create your own solutions. The full source code of the sample application is available here.

The first statements in the application initialize various settings, which you must do once in every application that accesses GenePattern. You will need to customize the *italicized* GenePattern server URL and GenePattern user name (typically, your e-mail address) with values appropriate for your GenePattern server.

```java
import org.genepattern.data.expr.ExpressionData;

import org.genepattern.client.GPServer;

import org.genepattern.webservice.JobResult;

import org.genepattern.webservice.Parameter;

import org.genepattern.io.IOUtil;

import java.io.File;


public class MyProgram {

 public static void main(String[] args)

   throws Exception {

 GPServer gpServer=new GPServer("http://localhost:8080",

   "your email address");
```

After initializing the required settings, the application runs the PreprocessDataset task to preprocess a dataset. This example references the dataset using a publicly-accessible URL, but a filename would be equally valid. When you invoke the `runAnalysis` method, the GenePattern library invokes the appropriate task on the server, passing all of the input parameters and input files. Control returns to your application when the task completes. (To run a task asynchronously, invoke the `runAnalysis` method in a separate thread.)

```java
 String inputDataset=

 "ftp://ftp.broad.mit.edu/pub/genepattern/all_aml/all_aml_train.res";

 JobResult preprocess=gpServer.runAnalysis("PreprocessDataset",

   new Parameter[] {

 new Parameter("input.filename", inputDataset)

 });
```

When the task completes, you can query the `JobResult` object for an array of filenames that are the output from the task. You can download the result files or leave them on the server and refer to them by URL. Referring to result files by URL is especially useful for intermediate results. In this example, the `JobResult` object named `preprocess` contains a list of filenames (of length 1, in this case), which the application displays in a heat map:

```java
 // view results in a HeatMapViewer visualizer

 gpServer.runVisualizer("HeatMapViewer",
```

```
    new Parameter[] {
  new Parameter("filename", preprocess.getURL(0).toString())
  });
}
```

The last statements in the application download the preprocessed data and load it into a matrix for further analysis:

```
  String downloadDirName=String.valueOf(preprocess.getJobNumber());
  // download result files
  File[] outputFiles = preprocess.downloadFiles(downloadDirName);
  // load data into matrix for further manipulation
  ExpressionData expressionData=
  IOUtil.readExpressionData(outputFiles[0].getPath());
  }
}
```

You can combine GenePattern analyses with any capabilities that the Java environment has to offer. Use Java's 2-D and 3-D graphics libraries to create graphic output, or summarize and report on the data using your own code. The basic idea to remember is that GenePattern tasks create result files and those files are available to the Java application for processing.

For a list of the GenePattern modules, with links to their documentation, see the Modules page. To have GenePattern generate the Java code required to run a task, build a pipeline that contains the desired task and then, in the Pipelines pane of the GenePattern Web Client home page, use the *Download Pipeline Code* field to generate the Java code for that pipeline.

Using LSIDs from Java

As of version 1.3 of the GenePattern server, Life Science Identifiers (LSIDs) can be used instead of task names to identify tasks for GenePattern to run. An LSID may be submitted in place of the task name in the methods `runAnalysis` and `runVisualizer`. When an LSID is provided that does not include a version, the latest available version of the task identified by the LSID will be used. If a task name is supplied, the latest version of the task with the nearest authority is selected. The nearest authority is the first match in the sequence: local authority, Broad authority, other authority. For more information about LSIDs, see Understanding Version Numbers in the *GenePattern Java Client Guide*.

Using GenePattern from MATLAB

Using MATLAB as a GenePattern client allows you to run GenePattern tasks and to manipulate and visualize the results in a powerful, commercial technical computing application that works on most major platforms. Using GenePattern allows you to invoke methods written in many other languages without having to worry about how to launch them. This section describes how you can use the GenePattern MATLAB library to run GenePattern analyses:

- Getting Started in MATLAB
- GenePattern MATLAB Library
- Running a Program
- Using LSIDs from MATLAB

Getting Started in MATLAB

Resources and documentation for MATLAB are available at http://www.mathworks.com/.

GenePattern MATLAB Library

The GenePattern MATLAB library allows you to invoke a GenePattern task as if it were a local MATLAB function running on your client and to get back from the task a list of result files. A zip file containing the MATLAB library is available on your GenePattern server.

To download the GenePattern MATLAB library to your computer:

1. Start the GenePattern Web Client.

2. Under Programming Libraries in the right column, click the *zip* link for the MATLAB library.

3. Download the zip file and unzip it into your `MATLAB7/toolboxes` directory. If you do not have permission to put files in that directory, unzip into any other directory.

4. After downloading and unzipping the files, add the directories to your MATLAB path:

    1. At a MATLAB prompt, open the pathtool:

        ```
        >>pathtool
        ```

    2. Use the MATLAB pathtool to add the `GenePatternServer` and `GenePatternFileSupport` directories, with subfolders, to the MATLAB search path.

Running a Program

This section explores a simple MATLAB program that runs a task, displays the resulting output, and loads it into a MATLAB matrix for further analysis. The included code can be copied and pasted into your MATLAB client so that you can try it out, modify it, and create your own solutions.

The first statements in the application initialize various settings, which you must do once in every application that accesses GenePattern. You will need to customize the *italicized* GenePattern server URL and GenePattern user name (typically, your e-mail address) with values appropriate for your GenePattern server.

```
% Create a GenePattern server proxy instance
gp = GenePatternServer('http://localhost:8080','my.email@my.domain');
```

After initializing the required settings, the application runs the TransposeDataset task to transpose a dataset. This example references the dataset using a publicly-accessible URL, but a filename would be equally valid. As shown below, you can call the GenePattern methods directly or by calling the `runAnalysis` method. When you call a GenePattern method, such as `TransposeDataset`, the GenePattern library invokes the task on the server, passing all of the input parameters and input files. Control returns to your application when the task completes. (To run a task asynchronously, invoke the method in a separate thread.)

```
% input dataset for transpose operation
params.output_file_name = 'transposed.out'
params.input_filename='http://www.broad.mit.edu/mpr/publications/projects/Leuk
emia/
ALL_vs_AML_train_set_38_sorted.res'

% transpose the dataset
transposeResult = gp.TransposeDataset(params)
% alternate call to transpose the dataset
transposeResult = runAnalysis(gp, 'TransposeDataset', params)
```

When the task completes, it returns a MATLAB structure that contains a list of filenames that are the output from the task. In this example, `transposeResult` is a structure with a list of filenames (of length 1, in this case). The application displays the results in a file viewer window and also loads them into a matrix so that further manipulation can be performed:

```
% display the transposed results
edit 'transposed.out.odf'

% now read the output into a matrix
% so we can do further manipulation in MATLAB
myData = loadGenePatternExpressionFile('transposed.out.odf')
```

You can combine GenePattern analyses with all of the rich functionality of MATLAB. For example, you can use MATLAB's plotting methods to create graphic output, save modified matrices to files using save, or summarize and report on the data using your own code. The basic idea to remember is that GenePattern tasks create result files and those files are available to the MATLAB client for processing.

For a list of the GenePattern modules available on your server, run the `listMethods` function on your `GenePatternServer` object. To view the names of the input parameters for a module, use the `describeMethod` function on your `GenePatternServer` object, passing it the module name.

```
% display the available GenePattern modules
listMethods(gp)

% now look at the parameters for the TransposeDataset module
describeMethod(gp, 'TransposeDataset')
```

Alternatively, to get the parameters with their default values filled in, use the `getMethodParameters` function of the `GenePatternServer` object. This returns a MATLAB structure with named elements for each parameter, filled in with the default value if one exists. After filling in the missing parameters and overriding defaults if desired, this structure can then be passed on to the `runAnalysis` method.

```
% display the available GenePattern modules
params2 = getMethodParameters(gp, 'TransposeDataset')
params2.input_filename='http://www.broad.mit.edu/mpr/publications/projects/Leu
kemia/ALL_vs_AML_train_set_38_sorted.res'

% transpose the dataset
transposeResult = gp.TransposeDataset(params2)
```

The GenePattern MATLAB library also has convenience methods to read and write GenePattern files (such as res, gct, and odf files). Even if you choose not to look in the library, you can extend the techniques shown above to implement your own analyses.

Using LSIDs from MATLAB

As of GenePattern version 1.3, you can use Life Science Identifiers (LSIDs) to identify a task when executing GenePattern code in MATLAB. An LSID may be submitted in place of the task name to `getMethodParameters` or `runAnalysis`. When providing an LSID to a method in addition to a task name, the LSID alone is used to determine what task to run. When an LSID is provided that does not include a version, the latest available version of the task identified by the LSID will be used. For more information about LSIDs, see Understanding Version Numbers in the *GenePattern Java Client Guide*.

```
% Example using LSIDs from MATLAB
params = getMethodParameters(gp,
'urn:lsid::broad.mit.edu:cancer.software.genepattern.
module.analysis:00026:0'); params.output_file_name = 'transposed.out'
params.input_filename='http://www.broad.mit.edu/mpr/publications/projects/Leuk
emia/ALL_vs_AML_train_set_38_sorted.res'

% transpose the dataset
transposeResult = runAnalysis(gp,
'urn:lsid::broad.mit.edu:cancer.software.genepattern.module.analysis:00026:0',
params)
```

Using GenePattern from R

Using R as a GenePattern client allows you to run GenePattern tasks and to manipulate and visualize the results in a powerful, free statistical desktop package that works on most major platforms. Using GenePattern allows you to invoke methods written in many other languages without having to worry about how to launch them or whether you are passing incorrect parameters. This section describes how you can use the GenePattern R library to run GenePattern analyses:

● Getting Started in R

● Accessing GenePattern from R

● Running a Program

● Using LSIDs from R

Getting Started in R

If you are not familiar with R, see the following resources on the www.r-project.org website:

● An Introduction to R (PDF, approx. 100 pages, 650kB), based on the former "Notes on R", gives an introduction to the language and how to use R for doing statistical analysis and graphics.

● A draft of the R language definition (PDF, approx. 60 pages, 400kB) documents the language; that is, the objects that it works on, and the details of the expression evaluation process, which are useful to know when programming R functions.

● Writing R Extensions (PDF, approx. 85 pages, 500kB) covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.

- R Data Import/Export (PDF, approx. 35 pages, 270kB) describes the import and export facilities available either in R itself or via packages which are available from CRAN.

- R Installation and Administration (PDF, approx. 30 pages, 200kB).

- The R Reference Index (PDF, approx. 2200 pages, 12MB) contains all help files of the R standard and recommended packages in printable form.

Accessing GenePattern from R

The GenePattern R library allows you to invoke a GenePattern task as if it were a local R method running on your client and to get back from the task a list of result files. The R library is available on your GenePattern server in both Windows (zip) and Unix (tar.gz) formats.

To download the R library to your computer:

1. Start the GenePattern Web Client.

2. Under Programming Libraries in the right column, click the *zip* or *tar.gz* link for the R library.

3. After downloading the zip file, extract it into your `R/library` directory.

   If you cannot add files to your `R/library` directory (because it is a publicly-shared version and you do not have appropriate privileges), you can load the GenePattern library by setting the environment variable `R_LIBS=<GenePattern install directory>/R/library` in your `autoexec.bat`, `.cshrc`, `.bashrc` or other shell startup file. R will then load from its usual location, but will also search for and find the GenePattern library.

Running a Program

This section explores a simple R program that runs a task, displays the resulting output, and loads it into an R matrix for further analysis. The included code can be copied and pasted into your R client so that you can try it out, modify it, and create your own solutions.

The first statements in the application initialize various settings, which you must do once in every application that accesses GenePattern. You will need to customize the *italicized* GenePattern user name (typically, your e-mail address). The `gpTasksAsFunctions` method creates a function for each GenePattern task on the server, which invokes the latest version of that task.

```
# Load GenePattern library
library(GenePattern)
defaultServer <- "cp409-6ac.domain.com:8080"

# The server will want to know whose tasks to provide
gpLogin("my.email.address@mydomain.com");

# Ask the server to create R methods that proxy
# each of the installed tasks.
gpTasksAsFunctions();
```

Note: In GenePattern 2.0.2, the R programming environment does not handle HTTP redirects; therefore, you must set the value of the defaultServer variable to the full host name of your GenePattern server, as shown above. To find the host name of your GenePattern server, open the GenePattern Web Client and check the URL displayed in the web browser window.

After initializing the required settings, the application runs the TransposeDataset task to transpose a dataset. This example references the dataset using a publicly-accessible URL, but a filename would be equally valid. When you call an R method, such as `TransposeDataset`, the GenePattern library invokes the appropriate task on the server, passing all of the input parameters and input files. Control returns to your application when the task completes. (To run a task asynchronously, invoke the method in a separate thread.)

```
# input dataset for transpose operation
input.ds <-
"http://www.broad.mit.edu/mpr/publications/projects/Leukemia/ALL_vs_AML_train_
set_38_sorted.res";

# transpose the dataset
transpose.out <- TransposeDataset(input.filename=input.ds,
output.file.name="transposed");
```

When the task completes, it returns an R list of filenames that are the output from the task. In this example, `transpose.out` now has a list of filenames (of length 1, in this case). The application displays the results in a file viewer window and also loads them into a matrix so that further manipulation can be performed:

```
# display the transposed results
file.show(transpose.out$transposed.odf);

# now read the output into a matrix
# so we can do further manipulation in R
data <- read.delim(transpose.out$transposed.odf, as.is=T, header=F, sep="\t",
skip=9, comment="";)
data <- as.matrix(data);
cols <- length(data[1,]);
rows <- length(data[,1]);
```

You can combine GenePattern analyses with all of the rich statistical functionality of R. For example, you can use R's plot and legend methods to create graphic output, output JPEGs of your visualized data using savePlot, save modified matrices to files using save, or summarize and report on the data using your own code. The basic idea to remember is that GenePattern tasks create result files and those files are available to the R client for processing.

For a list of the GenePattern modules, with links to their documentation, see the Modules page. For generated R code that describes the function of and the input parameters for a module, point your browser to http://localhost:8080/gp/taskWrapperGenerator.jsp?name=TransposeDataset, replacing TransposeDataset with the name of any GenePattern task.

The GenePattern R library also has convenience methods to read and write GenePattern files (such as res, gct, and cls files), to enable running of multiple tasks in parallel, to run tasks with input from files that were output from previous tasks without moving them from the server, and other utilities. Even if you choose not to look in the library, you can extend the techniques shown above to implement your own analyses.

Using LSIDs from R

As of version 1.3 of the GenePattern server, Life Science Identifiers (LSIDs) can be used instead of task names to identify tasks for GenePattern to run. For R, this is primarily useful when you want to specify a particular version of a task for GenePattern to run. The easiest way to specify a particular version of a task is to specify the LSID as an argument to the R method for a GenePattern task, such as TransposeDataset. For example, the following statement invokes version 1 rather than the latest version of the TransposeDataset task:

```
transpose.out <- TransposeDataset(input.filename="all_aml_train.res",
output.file.name="<input.filename_basename>.transposed",
LSID="urn:lsid:broad.mit.edu:cancer.software.genepattern.module.analysis:00026
:1")
```

For more information about LSIDs, see Understanding Version Numbers in the *GenePattern Java Client Guide*.