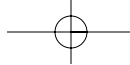




Answers to “Test Your Understanding” Questions

Chapter 1

1. Rexx is a *higher-level language* in that each line of code accomplishes more than does code written in traditional languages like C++, COBOL, or Pascal. Rexx derives its power from the fact it is a *glue language*—a language that ties together existing components such as other programs, routines, filters, objects, and the like. The industry-wide trend towards scripting languages is based on the higher productivity these languages yield.
2. Rexx is a *free-format language*. There are no requirements to code in particular columns or lines or in uppercase, lowercase, or mixed case.
3. Expert programmers sometimes mistakenly think that they don't need an easy-to-use language. Nothing could be farther from the truth. Expert programmers become wildly productive with easy-to-use languages. Their code lasts longer as well, because less skilled individuals can easily enhance and maintain it.
4. The two free object-oriented Rexx interpreters are roo! from Kilowatt Software and Open Object Rexx from the Rexx Language Association (formerly known as IBM's Object REXX). Both run standard or classic Rexx scripts without any alterations.
5. One outstanding feature of Rexx is that it runs on all sizes of computer, from handhelds to personal computers to midrange machines to mainframes. Rexx runs on cell or mobile phones, Palm Pilots, and mainframes.
6. One of the two current Rexx standards was established by the book *The Rexx Language*, second edition, by Michael Cowlshaw, published in 1990. The other was promulgated by the American National Standards Institute, or ANSI, in 1996. There is little difference between these two standards. Chapter 13 lists the exact differences between these two similar standards.



Appendix M

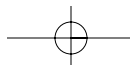
7. Rexx bridges the traditional gap between ease of use and power through: simple syntax; free formatting; consistent, reliable behavior; a small instruction set surrounded by a large set of functions; few language rules; support for modularity and structured programming; and standardization.

Chapter 2

1. Comments are encoded between the starting identifier `/*` and the ending identifier `*/`. They may span as many lines as you like. They may also appear as *trailing comments*, comments written on the same lines as Rexx code.
2. Rexx recognizes functions as keywords immediately followed by a left parenthesis: `function_name()` or `function_name(parameter)`. The `call` instruction can also be used to invoke functions. In this case, the function is encoded just like a call to a subroutine, and parentheses do not immediately follow the function name. Chapter 8 fully discusses how to invoke functions and subroutines.
3. Variables do not have to be predefined or declared in Rexx. They are automatically defined the first time they are used or referred to. If a variable is equal to its name in uppercase, it is uninitialized.
4. The basic instruction for screen output is `say`. The basic instruction for keyboard input is `pull`. Rexx also offers more sophisticated ways to perform input and output, described in subsequent chapters.
5. *Comparisons* determine if two values are equal (such as character strings or numbers). *Strict comparisons* only apply to character strings. They determine if two strings are identical (including any preceding and/or trailing blanks). The strings are not altered in any way prior to a strict comparison. For example, the shorter string is not blank-padded as in regular or “nonstrict” character string comparison.
6. Define a numeric variable in the same way you define any other Rexx variable. The only difference is that a numeric variable contains a value recognized as a number (such as a string of digits, optionally preceded by a plus or minus sign and optionally containing a decimal place, or in exponential notation).

Chapter 3

1. Structured programming is recommended because it leads to more understandable code, and therefore higher productivity. The first table in the chapter lists the structured programming constructs and the Rexx instructions that implement them. Subroutines and functions support modularity, the key structured programming concept of breaking code up into discrete, smaller routines.
2. Rexx matches an unmatched `else` with the nearest unmatched `if`.
3. Test for the end of input by testing for the user’s entry of a null string or by inspecting input for some special character string denoting the end of file in the input (such as `end` or `exit` or `x`). Chapter 5 introduces functions that can test for the end of an input file, such as `chars` and `lines`.

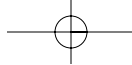


Answers to “Test Your Understanding” Questions

4. *Built-in functions* are provided as part of the Rexx language. The code of *internal routines* resides in the same file as that of the calling routine; *external routines* reside in separate files. A Rexx *function* always returns a single value; a *subroutine* may optionally return a value. Chapter 8 gives full details on how to pass information into and out of functions and subroutines.
5. TRUE tests to 1. FALSE tests to 0. Standard Rexx does not accept “any nonzero value” for TRUE. (However, some specific Rexx interpreters will accept any nonzero value as TRUE).
6. The danger of a `do forever` loop is that it will be an *endless loop* and never terminate. Avoid coding the `do forever` loop; use structured programming’s `do-while` loop instead. If you do code a `do forever` loop, be sure to code a manual exit of some sort. For example, the `leave`, `signal`, and `exit` instructions can end the otherwise endless loop.
7. The `signal` instruction either causes an unconditional branch of control, or aids in processing special errors or *conditions*. `signal` differs from the GOTO of other languages in that it terminates all active control structures in which it is encoded.
8. `do while` tests the condition at the top of the loop, while `do until` is a bottom-driven loop (it tests the condition at the bottom of the loop). Only the `do while` is structured. Its use is preferred. Any `do until` can be recoded as `do while`.

Chapter 4

1. Any number of subscripts can be applied to array elements. An array may have any desired dimensionality. The only limit is typically that imposed by memory. Array elements do not have to be referenced by numbers; they may be referenced by arbitrary character strings also. This is known as an *associative array*.
2. All elements in an array can be initialized to some value by a single assignment statement, but other operations cannot be applied to an entire array. For example, it is not possible to add some value to all numeric elements in an array in a single statement. Use a simple loop to accomplish this. A few Rexx interpreters do allow additional or extended array operations. The chapters on specific interpreters in Section II of this book cover this.
3. Rexx does not automatically keep track of the number of elements in an array. To process all elements in an array, keep track of the number of elements in the array. Then process all array elements by a loop using the number of array elements as the loop control variable. Alternatively, initialize the entire array to some unused value (such as the null string or 0), prior to filling it with data elements. Then process the array elements using a loop until you encounter the default value. These two array processing techniques assume you use a numeric array subscript, and that contiguous array positions are all used. To process all elements in an array subscripted by character strings, one technique is to store the index strings in a list, then process that list against the array, one item at a time.
4. Arrays form the basis of many data structures including lists, key-value pairs, and balanced and unbalanced trees. Create a list with a one-dimensional array (an array in which elements are referenced by a single subscript). Create key-value pairs by matching subscripts with their corresponding values. Create tree structures by implementing an element hierarchy through the array.



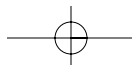
Appendix M

Chapter 5

1. The two types of input/output are *line-oriented* and *character-oriented*. Use the former to read and write lines of information, and the latter to read and write individual characters. Line-oriented I/O is typically more portable across operating systems. Character-oriented is useful in reading all bytes in the input stream, regardless of any special meaning they might have to the operating system.
2. The `stream` function is used either to return information about a character stream or file, or to perform some action upon it. The `stream` function definition allows Rexx interpreters to offer many implementation-dependent file commands, and most do. Look the function up in your product documentation to learn what it offers for I/O and file manipulation. The statuses it returns are `ERROR`, `NOTREADY`, `READY`, and `UNKNOWN`.
3. Encode the I/O functions immediately followed by parentheses—for example, `feedback = linein(filein)`—or through a `call` instruction (for example, `call linein filein`). Capture the return code as shown in the example for the first method, or through the `result` special variable for the `call` method. It is important to check the return code for I/O operations because this informs your program about the result of that I/O and whether or not it succeeded.
4. Rexx does not require explicitly closing a file after using it. Program end automatically closes any open files. This is convenient for short scripts, but for longer or more complex scripts, explicitly closing files is a good programming practice. This prevents running out of memory (because each open file uses memory) and may also be necessary if a program needs to “reprocess” a file. The typical way to close a file is either to encode a `lineout` or `charout` function that writes no data, or to encode the `stream` function with a command parameter that closes the file.
5. Rexx offers several options beyond standard I/O for sophisticated I/O needs. One option is to use a database package, such as one of those described in Chapter 15. Another option is to use Rexx interpreter I/O extensions (discussed in Chapters 20 through 30 on the specific Rexx interpreters).

Chapter 6

1. String processing is the ability to process text. It is critically important because so many programming problems require this capability. Examples include report writing and the building and issuing of operating system commands.
2. Concatenation can be performed *implicitly*, by encoding variables with a single space between them; by *abuttal*, which means coding variables together without an intervening blank; or *explicitly*, by using the string concatenation operator: `||`.
3. The three methods of template parsing are: by *words*, in which case each word is identified; by *pattern*, which scans for a specified character or pattern; and by *numeric pattern*, which processes by column position.
4. The functions to use are: `verify`, `datatype`, `pos`, `delstr`, `right` and `left`, and `strip`. Given the flexibility of the string functions, you might choose other string functions and combine them in various ways to achieve these same operations.
5. `wordindex` returns the character position of the *n*th word in a string, while `wordpos` returns the word position of the first word of a phrase within a string.



Answers to “Test Your Understanding” Questions

6. Hex characters each represent strings of four bits; character strings are composed of consecutive individual characters of variable length, where each character is internally made up of 8 bits; bit strings consist solely of 0s and 1s. Rexx includes a full set of *conversion functions*, including: `b2x`, `c2d`, `c2x`, `d2c`, `d2x`, `x2b`, `x2c`, and `x2d`.
7. Bit strings can be used for a wide variety of tasks. Examples mentioned in the chapter include bit map indexes, character folding, and key folding.

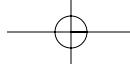
Chapter 7

1. `numeric digits` determines how many significant digits are in a number. This affects accuracy in computation and output display. `numeric fuzz` determines the number of significant digits used in comparisons. You might set `fuzz` in order to affect just a single comparison, while keeping the numeric precision of a number unchanged.
2. Scientific notation has one digit to the left of the decimal place, followed by fractional and exponential components. Engineering notation expresses the integer component by a number between 1 and 999. Rexx uses scientific notation by default. Change this by the `numeric form` instruction.
3. There are several ways to right-justify a number; one of them is the `format` function.
4. The `datatype` function allows you to check many data conditions, including whether a value is alphanumeric, a bit string, all lower- or uppercase, mixed case, a valid number or symbol, a whole number, or a hexadecimal number.

```
-22          valid
' -22      '  valid- the blanks are ignored
2.2.        invalid- has a trailing period
2.2.2      invalid- more than one decimal point
222b2      invalid- contains an internal character
2.34e+13   valid
123.E -2   invalid- no blanks allowed in exponential portion
123.2 E + 7 invalid- no blanks allowed in exponential portion
```

Chapter 8

1. Modularity is important because it underlies *structured programming*. Modularity reduces errors and enhances program maintenance. Rexx supports modularity through its full set of structured control constructs, plus internal and external subroutines and functions.
2. A *function* always returns a single string through the `return` instruction. A *subroutine* may or may not return a value. Functions return their single value such that it is placed right into the statement where the function is coded, effectively replacing the function call. Get the value returned by a subroutine through the `result` special variable. Functions can be coded as embedded within a statement or invoked through the `call` instruction. Subroutines can only be invoked via the `call` instruction.
3. Internal subroutines reside in the same file as the main routine or driver. External subroutines reside in separate files. `procedure` can be used to selectively protect or expose variables for an internal routine. External routines always have an implicit `procedure` so that all the caller's variables are hidden.



Appendix M

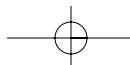
4. The *function search order* determines where Rexx searches for called functions. It is: internal function, built-in function, external function. If you code a function with the same name as a Rexx built-in function, Rexx uses your function. Override this behavior by coding the function name as uppercase within quotes.
5. Information can be passed from a caller to a routine by several methods, including: passing arguments as input parameters, `procedure expose`, and using global variables. Updated variables can be passed back to the calling routine by the `return` instruction, changing `expose'd` variables, and changing global variables.
6. A `procedure` instruction without an `expose` keyword hides all the caller's variables. `procedure expose` allows updating the variables, whereas those read in through `arg` are read-only. (Argh!)
7. In standard Rexx condition testing, expressions must resolve to either 1 or 0, otherwise an error occurs. Some Rexx interpreters are extended to accept any nonzero value as TRUE.

Chapter 9

1. The default setting for the trace facility is `trace n` (or Normal). `trace r` is recommended for general-purpose debugging. It traces clauses before they execute and the final results of expression evaluation. It also shows when values change by `pull`, `arg`, and `parse` instructions. `trace l` lists all labels program execution passes through and shows which internal routines are entered and run. `trace i` shows intermediate results.
2. The trace facility is the basic tool you would use to figure out any problems that occur while issuing operating system commands from within a script. The trace flags C, E, and F would be useful for tracing OS commands.
3. To start interactive tracing, code the `trace` instruction with a question mark (?) preceding its argument. The ? is a toggle switch. If tracing is off, it turns it on; if tracing is on, it turns it off. The first `trace` instruction or function you execute with ? encoded turns tracing on. The next one that executes with the question mark will turn it off.
4. When in *interactive mode*, the Rexx interpreter pauses after each statement or clause. Use it to single-step through code.

Chapter 10

1. The purpose of error or exception trapping is to manage certain kinds of commonly occurring errors in a systematic manner. The conditions are ERROR, FAILURE, HALT, NOVALUE, NOTREADY, SYNTAX and LOSTDIGITS. The ANSI-1996 standard added LOSTDIGITS.
2. To handle a control interrupt, enable an error trap for the HALT condition. Enable this exception condition, then Rexx automatically invokes your error routine when this condition occurs.
3. `signal` applies to all seven error conditions. `call` does *not* apply to SYNTAX, NOVALUE, and LOSTDIGITS errors. `signal` forces an *abnormal change* in the flow of control. It terminates any `do`, `if` or `select` instruction in force and unconditionally transfers control to a specified label.



Answers to “Test Your Understanding” Questions

`call` provides for normal invocation of an internal subroutine to handle an error condition. The `result` special variable is not set when returning from a called condition trap; any value coded on the `return` instruction is ignored.

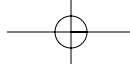
4. Use the `interpret` instruction to dynamically evaluate and execute an expression.
5. Enable a condition routine through a `signal` or `call` instruction. You can have multiple routines to handle a condition by coding `signal` or `call` multiple times, but only one condition routine is active for each kind of error trap at any one time.
6. If the error trap was initiated by the `signal` instruction, after executing the condition trap routine, you must reactivate the error condition by executing the `signal` instruction again.
7. Whether it is better to write one generic error routine to handle all kinds of errors, or to write a separate routine for each different kind of error, depends on what you’re trying to do and the nature of your program. Both approaches have advantages. Sometimes it is convenient to consolidate all error handling into a single routine, other times, it may be preferable to have detailed, separate routines for each condition.

Chapter 11

1. All Rexx implementations covered in this book have a stack; however, how the stack is implemented varies. Review the product documentation if you need to know how the stack is supported within your version of Rexx.
2. Stacks are last-in, first-out structures, while queues are first-in, first-out. Instructions like `push` and `queue` place data into the stack, and instructions like `pull` and `parse pull` extract data from it. The `queued` built-in function reports how many items are in the stack.
3. The limit on the number of items the stack can hold is usually a function of available memory.
4. The answer to this question depends on the Rexx interpreter(s) you use and the platforms to which you wish to port. Check the documentation for the platforms and interpreters for which you intend to port.
5. Some Rexx interpreters support more than one stack, and more than one memory area or buffer within each stack. Functions or commands like `newstack` and `delstack` manage stacks, while `makebuf`, `dropbuf`, and `desbuf` manage buffers. Check the documentation for your specific Rexx interpreter regarding these features, as they do vary by interpreter.

Chapter 12

1. Consistency in coding is a virtue because it renders code easier to understand, enhance, and maintain.
2. Some programmers deeply nest functions because this makes for more compact code. Some developers find it an intellectually interesting way to code, and others even use it to demonstrate their cleverness. If overdone, it makes code indecipherable and result in slower execution of the program.



Appendix M

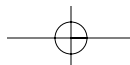
3. A good comment imparts information beyond what the code shows. It explains the code further, in clear English. Rexx comments may appear on the end of a line, in stand-alone lines, or in comment boxes.
4. Modularity and structured programming permit a limited number of entry and exit points from discrete blocks of code. This makes code easier to understand and follow, and restricts interactions between different parts of programs. The result is higher productivity, and more easily understood code that is easier to maintain.
5. `do until` and `signal` are unstructured control instructions. Any code using them can be rewritten as structured code by use of the `do while` and `if` statements.
6. A good variable name is descriptive. It is not short or cryptic but long and self-explanatory. It does not employ cryptic abbreviations but instead fully spells out words. Good variable naming makes a program much more readable.
7. Global variables can be highly convenient when coding. But best programming practice limits their use in larger and more complex programs. Structured programming involves carefully defined interfaces between routines, functions, and modules. Variables should be localized to routines and their use across routines should be carefully defined and limited. Global variables do not follow these principles. Different programmers and sites may have their own standards or opinions on this matter.

Chapter 13

1. No. Writing portable code typically takes more effort than writing nonportable code. In some cases, where the goal is quick coding, a nonportable solution may meet the goal more effectively.
2. Scripts can learn about their environment in several ways. Key instructions for this purpose include `parse version` and `parse source`. The chapter also lists many other instructions and functions that help scripts learn about their environment.
3. `arg` automatically translates input to uppercase, while `parse arg` does not.
4. The `sourceline` function either returns the number of lines in the source script, or a specific line if a line number is supplied as an argument.
5. The appendix of the TRL-2 book lists all its differences from TRL-1.

Chapter 14

1. Rexx sends a command string to the environment when it does not recognize it as valid Rexx code. The default environment, the environment to which external commands are directed by default, is typically the operating system's shell or command interface.
2. Enclose commands in quotation marks when their contents will otherwise be incorrectly interpreted or evaluated by the Rexx interpreter. For example: `dir > output.txt` will not work because the carrot symbol will be interpreted as a "greater than" symbol by Rexx, rather than as a valid part of the OS command. So, this OS command would fail unless enclosed within quotation marks. Some developers like to enclose *all* of the OS command string in quotes, except the parts they specifically want Rexx to evaluate. Other developers quote only those parts of the



Answers to “Test Your Understanding” Questions

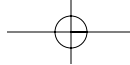
command that must not be evaluated. We generally follow the latter approach in this book. Either technique works fine; it is a matter of preference as to which you use.

To prepare a command in advance, assign the command string to a Rexx variable prior to issuing it to the operating system. You can easily inspect the variable’s contents merely by displaying it.

3. Basic ways to get error information from OS commands include inspecting their command return codes, capturing their textual error output, and intercepting raised condition traps within error routines. Look up return code information for OS commands in the operating system documentation.
4. Two ways to redirect command input/output from within a script are the `address` instruction or through the operating system’s redirection symbols. Command I/O redirection works on operating systems in the Windows, Linux, Unix, BSD, and DOS families, among others. Not all operating systems support I/O redirection.
5. Sources and targets can be specified as arrays or streams (files). They may be intermixed within the same `address` command.
6. To direct all subsequent external commands to the same interface, specify `address` with a system target only (without any external command encoded on the same instruction). Repeated coding of `address` without any environment operand effectively “toggles” the target for commands back and forth between two target environments.

Chapter 15

1. Rexx/SQL is free, open source, universal, and standardized. Database programming provides sophisticated I/O for multiuser environments. Advantages to database management systems include backup/recovery, database utilities, central data administration, transaction control, and many other features.
2. Scripts typically start by loading the Rexx/SQL function library for use through the `RxFuncAdd` and `SQLLoadFuncs` functions.
3. Connect to a database by `SQLConnect`, and disconnect through the `SQLDisconnect` function. Check connection status by `SQLGetInfo`. You can also check the return code for any Rexx/SQL function to verify its success or failure.
4. Consolidating error handling in a single routine is typical in database programming. It allows consistent error handling, while minimizing code. Here are the SQLCA variables set by Rexx/SQL:
 - `SQLCA.SQLCODE` — SQL return code
 - `SQLCA.SQLERRM` — SQL error message text
 - `SQLCA.SQLSTATE` — Detailed status string (N/A on some ports)
 - `SQLCA.SQLTEXT` — Text of the last SQL statement
 - `SQLCA.ROWCOUNT` — Number of rows affected by the SQL operation
 - `SQLCA.FUNCTION` — The last Rexx external function called
 - `SQLCA.INTCODE` — The Rexx/SQL interface error number
 - `SQLCA.INTERRM` — Text of the Rexx/SQL interface error



Appendix M

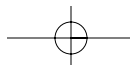
5. Assigning a SQL statement to a Rexx variable makes its coding clearer. It can also be verified simply by displaying the variable's value via a simple `say` instruction.
6. `SQLDisconnect` terminates a connection with a database and closes any open database cursor(s) for that connection. `SQLDispose` releases the work area (memory) resources originally allocated by a `SQLPrepare` function.
7. Rexx/SQL is a database-neutral product that is free, open source, very capable, and widely used. It offers both generic and native database interfaces to nearly any available database. Database-specific Rexx interfaces are also available from a few relational database companies. These products are proprietary and support only that company's database. In exchange, they often offer access to database-unique features, for example, the ability to write Rexx scripts for database administration or for controlling database utilities. An example of a proprietary interface is IBM Corporation's DB2 UDB interface described in this chapter.

Chapter 16

1. Both Rexx/Tk and Rexx/DW are free, open source interfaces that allow you to create portable graphical user interfaces for Rexx scripts. Rexx/Tk is based on the widely used Tk toolkit and provides Rexx programmers entrance into the Tcl/Tk universe. Rexx/DW is a lightweight protocol; it does not have the overhead that Rexx/Tk does. Both products offer very large libraries of GUI functions and features.
2. Rexx Dialog was designed specifically for scripting Windows GUIs. It works with both the Reginald and Regina Rexx interpreters.
3. A widget is a control or object placed on a window with which users interact. Widgets are added in Rexx/Tk by functions like `TkAdd` and `TkConfig` and others. Widgets are packed onto DW window layouts.
4. The basic logic of GUI scripts is the same, regardless of whether Rexx/Tk or Rexx/DW is used. Register and load the function library; create the controls or widgets for the topmost window; display the top-level window; wait for user interaction with the widgets, handle the interactions requested through event-handling routines; and terminate the window and the program when requested by the user.
5. Tcl/Tk GUI toolkit has achieved worldwide use because it renders inherently complex windows programming relatively simple. It is also portable and runs on almost any platform.
6. Rexx/gd creates graphical images, not GUIs. These images can be used as part of a GUI (for example, as components placed on a Web page). Rexx/gd creates its images in memory work areas. The images are typically stored on disk after developed, then the script releases the image memory area and terminates.

Chapter 17

1. Yes, you could write Web programs without using any of the packages described in the chapter. However, you would be doing a lot more work, and essentially duplicating code and routines that already exist and that you can freely use.



Answers to “Test Your Understanding” Questions

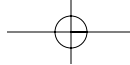
2. Functions `htmltop` and `htmlbot` write standard headers and footers, respectively. Scripts write the *Content Type header* by the `PrintHeader` function. The content type header must be the first statement written to the browser. It tells the browser the kind of data it will receive in subsequent statements. Read user input through the `ReadForm` function, among others.
3. Function `CgiInit` initializes and sets up the CGI header, while `CgiEnd` typically ends a script. `CgiHref` generates a hyperlink.
4. `Mod_Rexx` makes the Apache open source Web server completely programmable by Rexx scripts. Apache scales better than traditional CGI Web server programming because Apache handles incoming connections much more efficiently. `Mod_Rexx` gives the same capabilities to Rexx scripts as Perl programs get from `mod_perl` and PHP scripts from `mod_php`.
5. Rexx Server Pages are analogous to Java Server Pages or embedded PHP scripting, in that RSPs permit embedding Rexx code directly into the HTML of Web pages. This permits “dynamic pages” that are tailored or customized in real time.
6. Short- and long-form delimiters identify and surround Rexx code within HTML pages. They are used with Rexx Server Pages, or RSPs. There is no functional difference between short- and long-form delimiters.
7. To customize Apache’s log processing, use the `Mod_Rexx` package. It enables you to code Rexx scripts that control any of the 14 or so processing steps of the Apache Web server, including the one that manages log processing.

Chapter 18

1. XML is a self-describing data language. XML files contain both data and tags that describe the data. They are textual files. XML is useful for data interchange between applications or companies. XPath is a standard for identifying and extracting parts of XML files. XSLT applies definitional templates called stylesheets to XML files. It can be used to transform XML files. HTML is a language that defines Web pages.
2. Function `xmlParseXML` can be used to load and optionally validate a document. Function `xmlSaveDoc` saves a document, while function `xmlFreeDoc` frees resources.
3. Function `xmlParseHTML` can parse or scan a Web page written in HTML. `xmlFindNode` and `xmlNodesetCount` may also be useful, as per the example in the chapter.
4. Rexx does not include regular expressions. However, many packages are freely available that add this facility to the language, including *RexxRE* and *Regular Expressions*. Appendix H lists many of the free and open source packages, tools, and interfaces for Rexx programmers.
5. Apply an XSLT stylesheet to a document by the `xmlApplyStylesheet` function. `xmlParseXSLT` parses and compiles an XSLT stylesheet, while `xmlFreeStylesheet` frees a compiled stylesheet. `xmlOutputMethod` reports the output method of a stylesheet.

Chapter 19

1. BRexx runs fast and runs natively on Windows CE. Regina runs under virtually any imaginable platform. Rexx interpreters that are extended specifically for Windows are *r4*, *Reginald*, and *Regina*. Several interpreters offer extensions for Unix, Linux, and BSD, especially *Rexx/imc*,



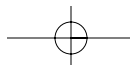
Appendix M

BRexx, and Regina. Kilowatt Software offers both their classic Rexx interpreter called `r4` and an upwardly compatible, fully object-oriented interpreter called `roo!`

2. The major Rexx standards are TRL-1, TRL-2, SAA, and ANSI-1996. Most Rexx interpreters adhere to TRL-2, while Regina is the primary offering that implements full ANSI-1996. SAA was IBM's attempt to rationalize its diverse operating systems in the early and mid- 1990s. SAA declared Rexx its common procedures language. The practical effect was that IBM ported Rexx to all its operating systems and established more rigorous standardization for the language across platforms.
3. `roo!` and Open Object Rexx are fully object-oriented Rexx interpreters. Both are supersets of standard or classic Rexx. This means that you can take a standard Rexx script and run it under either `roo!` or Open Object Rexx without any changes to that script. `roo!` is free under Windows, while Open Object Rexx is free under Linux, Windows, Solaris, and AIX.
4. NetRexx runs under the Java Virtual Machine (JVM). So, it runs anywhere Java runs. It presents an easy-to-use alternative to Java and may be freely intermixed with Java scripts. So, for example, NetRexx can make full use of the Java class libraries. You can write applets, applications, classes, Java Beans, and Servlets in NetRexx. NetRexx is a Rexx-like language; it does not meet the Rexx standards such as TRL-2 or ANSI-1996.
5. Regina is the open source Rexx that it includes many of the extended functions offered in other Rexx interpreters.
6. Rexx runs in native mode and emulation mode under the three major handheld operating systems. BRexx runs natively under Windows CE, Regina runs natively under Symbian/EPOC32, and Rexx for Palm OS runs natively under the Palm OS.
7. Emulation is slower and less efficient than running in native mode. However, emulation has the immediate benefit that it ports thousands of DOS applications to the handheld device without any code changes. Rexx interpreters are one example of the kinds of programs that can run under DOS emulation. Specifically, BRexx runs under DOS emulation (as well as in native mode under Windows CE).

Chapter 20

1. Regina is open source, widely popular, well supported, and meets all standards. It runs on virtually every platform, including any version of Windows, Linux, Unix, and BSD. It also runs natively under the handheld operating system Symbian/EPOC32, as well as a variety of less used operating systems such as BeOS, OS/2, AROS and others. Regina does not run under 16-bit DOS, but it does run from the Windows command prompt.
2. Use the stack to manage command I/O and pass information between routines. Regina also has a unique stack facility that permits communications between different processes on the same machine, and even between different processes on different machines. Regina also supports sending and receiving I/O to/from commands using the `address` instruction with standard keywords like `input`, `output`, and `error`.
3. Use `readch` and `writch` to read and write character strings, and `readln` and `writeln` to read and write lines. Use `open` to explicitly open a file, `close` to explicitly close a file, `eof` to test for end of file, and `seek` to move the file pointers.



Answers to “Test Your Understanding” Questions

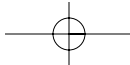
4. The SAA API is an interface definition that allows programs written in languages like C to use Regina as a set of services or a function library. Regina uses offers SAA-compatible functions for loading external function libraries. These include `rxfundadd` to register an external function and `rxfundrop` to remove external functions from use.
5. Regina supports a wide variety of parameters on the `options` instruction, including `CMS`, `UNIX`, `BUFFERS`, `AREXX_BIFS`, `REGINA`, `ANSI`, `SAA`, `TRL2`, and `TRL1`. See the Regina documentation for full details on these and other `options` instruction parameters.

Chapter 21

1. Rexx/imc runs under all forms of Linux, Unix, and BSD. Rexx/imc meets the TRL-2 standards. Its advantages include its strong Unix heritage and orientation, extra Unix functions, good documentation, and a strong track record of support.
2. Rexx/imc includes C-like I/O functions such as `open`, `close`, `stream`, and `ftell`. This I/O model provides more explicit control than Rexx's standard I/O. Use the standard I/O functions for portability and standardization, and use the the C-like I/O functions for more explicit file control.
3. `select` can key off the values in a variable in Rexx/imc, rather than requiring condition tests. This is useful in implementing a CASE construct based on the value of a variable.
4. Use `rxfuncadd` to load and register and external function for use, and `rxfuncdrop` to drop an external function. Use `rxfuncquery` to determine if a function is already loaded.
5. Rexx/imc accepts any nonzero value as TRUE, whereas standard Rexx only accepts 1 as TRUE. In this respect, any standard Rexx script will run under Rexx/imc, but a script written for Rexx/imc's approach to TRUE conditions might fail when run under standard Rexx.

Chapter 22

1. BRexx's advantages include its high performance, small footprint, wide array of built-in functions, and extra function libraries. It is effective for a wide variety of problems. It runs on a wide variety of platforms and in addition is uniquely positioned among Rexx interpreters to run on smaller, limited-resource environments (such as Windows CE), and older systems (like 16- and 32-bit DOS).
2. Position a file pointer through the `seek` function. Using `seek`, you can position to anywhere in the file including its beginning or end. The `seek` function can also return current file pointer positions. Use it to determine the size of a file by this statement: `filesize = seek(file_pointer, 0, "EOF")`.
3. The EBCDIC functions convert data between ASCII and EBCDIC (these are the two predominant coding schemes, with the former being used on PCs and midrange machines, and the latter being used for mainframes). The Date Functions external function library can handle date arithmetic.
4. The stack buffer functions include `makebuf` (create a new system stack), `desbuf` (destroy all system stacks), and `dropbuf` (destroy the top *n* stacks).

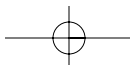


Appendix M

5. BRexx supports standard Rexx I/O, C-like I/O, and database I/O through MySQL. Standard Rexx I/O is for general use in standards-based, portable programs; C-like I/O is nonstandard but offers more explicit file control; and the MySQL functions provide the full power of a relational database.
6. Code operating system commands in the same manner as with any standard Rexx interpreter. You can also encode them as if they were functions, if they use standard I/O. Capture command output through the stack, or code a command as if it were a function and capture its return string through an assignment statement.

Chapter 23

1. Reginald's biggest advantage is that it is specifically tailored and customized for Windows. Furthermore, it offers many add-on tools, provides great documentation on how to use its extended features and functions, permits use of *any* Windows DLL, meets the Rexx TRL-2 standards, and supports the SAA API.
2. Reginald comes with complete documentation that includes examples of every new function and feature. You shouldn't require any information other than what comes with Reginald to use it.
3. Use Reginald's SAA-compliant functions like `rxfuncadd`, `rxfuncquery`, and `rxfuncdrop` to access external function libraries. Use `funcdef` to access DLLs that were not written to Rexx's specifications. You can autoload many function libraries through Reginald's Administration Tool and thereby avoid explicitly loading them in every script. This is very convenient and also reduces the amount of code you must write.
4. Reginald accesses virtually any external data source, including office products like Microsoft Excel and Access, and databases like SQL Server, MySQL, and PostgreSQL. The Open Database Connectivity or ODBC drivers are the means to accomplish this.
5. Reginald offers a freely downloadable tutorial on how to use Reginald with the Common Gateway Interface, or CGI. Other tutorials address subjects like mailslots, Internet access, sockets, and GUI programming.
6. `DriveInfo` and `MatchName` are two functions (among others) that supply information about disk drives. `MatchName` also provides attribute information. A file does not have to be open to retrieve information about it, but it must exist.
7. `ValueIn` reads binary values in as numeric values.
8. The `LoadText` function reads lines of a text file into a stem variable, or saves a stem variable's lines to a text file.
9. The `RxDlgIDE` add-on product helps generate GUI code.
10. `RxErr` establishes how GUI-related errors will be handled; `RxCreate` creates a new window with its controls; `RxMsg` controls user interaction with a window. The key values scripts check to determine how a user interacted with a window and its controls are `rxid` and `rxsubid`.
11. The Speech library allows the computer to synthesize speech. The MIDI function library controls the MIDI interface, which connects a computer to external instrumentation. Use MIDI to send information to a musical instrument, and the Speech library to read a document aloud.



Answers to “Test Your Understanding” Questions

Chapter 24

1. The three major families of handheld operating systems are Windows CE, Palm OS, and Symbian/EPOC32. The Rexx interpreters that run under them natively are BRexx, Rexx for Palm OS, and Regina (respectively).
2. Native scripts run faster, because they require no intermediate layer of software (or emulator) to run. Emulators are mainly useful for reasons of compatibility. They allow thousands of old DOS programs to run on the handheld, without any upgrading or changes being necessary. DOS emulation provides another way to run Rexx scripts on the handheld. The main advantage here is that Rexx can function as a glue language to tie together existing DOS programs or customize their use.
3. Tiny Linux is a version of Linux (or “kernel build”) that minimizes resource use by stripping out any unnecessary components. It is useful because it allows Linux to run on small embedded devices. Sometimes it is referred to as *embedded Linux*.
4. You need to supply a DOS with any emulator. PocketDOS ships with a free DOS, while XTM does not. Many free DOS operating systems are available on the Internet, including DR-DOS, Free DOS, and others.
5. The fact that Rexx for handhelds supports Rexx standards is very useful for reasons of portability and ease of learning. The Rexx standards mean that the Rexx skills you may have learned on other platforms apply to handhelds as well.
6. Rexx offers several advantages for programming handhelds and embedded programming versus C, C++, and Java. Among them are ease of use, ease of learning, and the fact that an easy-to-use language yields more reliable code.

Chapter 25

1. Rexxlets run concurrently with applications. This empowers them as a glue language for control and customization of applications. They can be started by any single action, such as a pen stroke or keypad entry.
2. URLs and URIs are essentially the same thing: a standard way to reference a resource such as a Web site address or file. Rexx for Palm OS uses them as a standard means of naming and accessing resources, such as databases, files, the display screen, and the clipboard.
3. Scripts identify and access handheld resources by their URI names. Scripts open their resources implicitly, in the same manner as Rexx scripts running on any other platform.
4. Databases contain structured information. Files are stream-oriented. Databases are the most popular form of storage on the Palm, but files are also useful because their size is unlimited and they match the default concept of Rexx I/O.
5. You could develop Rexxlets in environments other than the handheld, but testing them really requires running them on the target (the handheld). Resources references might need to be altered if development occurs on a machine other than the handheld.
6. A hack manager enables and manages operating system extensions. It is only required if you use an older version of the Palm OS (older than version 5). X-Master is a free hack manager for versions of the Palm OS prior to version 5.

Appendix M

7. I/O is the same in Rexx for Palm OS as it is in any other standard Rexx. What sometimes make Palm scripting different are the different kinds of devices programmed on the handheld (for example, the beamer or a serial port). Rexx for Palm OS is a TRL-2 standard Rexx interpreter.

Chapter 26

1. Among the advantages to r4/roo! are that they are specifically tailored for Windows, they fit together as a classic Rexx/object Rexx pair, and they share a large set of Windows-specific tools. Among the tools are AuroraWare! (to create GUIs), TopHat (for creating fill-in-the-blank forms), Poof! (with 135 command-line aids and tools), Revu (to view text), XMLGenie! (to convert XML to HTML), Chill (to hide Rexx source code), and the Exe Conversion Utility (to convert scripts to stand-alone executable files).
2. r4 scripts run under roo! without any alteration. roo! scripts will typically not run under r4, because roo! is a superset of classic Rexx and r4. Since r4 scripts are classic Rexx, they are widely portable across operating systems. roo! scripts are portable only across varieties of Windows, because roo! is a Windows-specific product.
3. Several of the r4/roo! utilities aid in building GUIs, AuroraWare! being the most directly pertinent. In the list of attributes, the r4/roo! GUI tools offer the best customization for Windows, are easiest to use, can be learned most quickly, and are easiest to maintain. Competing tools such as Rexx/Tk and Rexx/DW are the most portable and the most powerful. These statements are generalizations that may not apply across the board to all projects, so always assess the available tools versus the criteria and goals of your own project.
4. Installation of r4 and roo! is simple and similar to other Windows installs. The one variant is that there is a preinstall step that Web-installs on the user's system.
5. roo! supports *all* the principles of object-oriented programming, including classes and methods, inheritance, an hierarchical class structure, encapsulation, abstraction, and polymorphism. As in all object-oriented systems, messages invoke methods in the various classes.
6. roo! supports a number of new operators in order to bring full object-orientation to classic Rexx. These include the following:

Operator	Symbols	Use
^^	Double caret	Instance creation
^	Caret	Method invocation prefix operator
~	Tilde	Method invocation infix operator
[]	Brackets	Arraylike reference operator
{}	Braces	Vector class reference
!	Exclamation Point	Identifies a command

7. For line-oriented I/O, use classes such as `InLineFile` and `OutLineFile`. To manage the display screen, the `Console` class is useful. `InStream` and `OutStream` handle the default input and output streams. Use the `Socket` class to manage TCP/IP sockets.

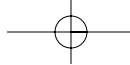
Answers to “Test Your Understanding” Questions

Chapter 27

1. Open Object Rexx is a superset of classic Rexx. Classic Rexx programs typically run under Open Object Rexx without any alteration. ooRexx features *inheritance*, the ability to create subclasses that inherit the methods and attributes of their superclasses. *Multiple inheritance* allows a class to inherit from more than one superclass. Open Object Rexx’s class hierarchy implements all this.
2. Encapsulation means that only an object can manipulate its own data. Send a message to the object to invoke its methods to manipulate that data. Polymorphism means that messages invoke actions appropriate to the object to which they are sent.
3. The `Stream` class has I/O methods. It goes beyond classic Rexx capabilities to include reading/writing entire arrays, binary, and text I/O, shared-mode files, direct I/O, and so on.
4. Open Object Rexx adds new special variables. Two are used for referencing objects: `Self` references the object of the current executing method, while `Super` references the superclass or parent of the current object.
5. The four directives are: `::CLASS` (to define a class), `::METHOD` (to define a method), `::ROUTINE` (to define a callable subroutine), and `::REQUIRES` (to specify access to another source script). Directives mark points within the script at which these definitions or actions apply.
7. Collection classes manipulate sets of objects and define data structures. A few of the collection classes are: `Array` (a sequenced collection), `Bag` (a nonunique collection of objects), `Directory` (a collection indexed by unique character strings), `List` (a sequenced collection which allows inserts at any position), and `Queue` (a sequenced collection that allows inserts at the start or ending positions).
8. The new `USER` condition trap allows user-defined error conditions. Explicitly raise an error condition (user-defined or built-in) by the new `raise` instruction.
9. As in classic Rexx, the `expose` instruction permits access to and updating of specified variables. In object programming, `expose` permits access to objects as well as string variables.

Chapter 28

1. Every classic Rexx program will run under Open Object Rexx, but this does not take advantage of any of the new object-oriented capabilities of Open Object Rexx. Compatibility both preserves legacy scripts and allows you to tip-toe into object-oriented scripting at a pace at which you are comfortable.
2. All instructions and functions of classic Rexx are part of Open Object Rexx. The latter adds a number of new and enhanced instructions and functions.
3. Collections are classes that implement data structures and allow manipulation of the objects in the class as a group.
4. Here are some solutions:
 - To return a string with its character positions reversed, code: `string~reverse`
 - To return a substring, code: `string~substr(1,5)`



Appendix M

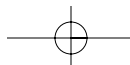
5. The `stream` class offers a superset of the I/O capabilities of classic Rexx. Extra features it includes are: reading/writing entire arrays, binary and text I/O, shared-mode files, direct I/O, and so on.
6. The four directives are: `::CLASS` (to define a class), `::METHOD` (to define a method), `::ROUTINE` (to define a callable subroutine), and `::REQUIRES` (to specify access to another source script). Classes and methods are placed after the main routine or driver (towards the end of the script).
7. These symbols can be used to express `at` and `put` in many collections:

Operator	Use
[]	Same as the <code>at</code> method
[] =	Same as the <code>put</code> method

8. These monitor objects are applied against the default streams. The default streams are `.stdin`, `.stdout`, and `.stderr`.

Chapter 29

1. Mainframe Rexx generally meets the TRL-2 and SAA standards. There are slight differences between Rexx interpreters on the different mainframe platforms as well as between mainframe Rexx and the standards. See the manual *SAA Common Programming Interface REXX Level 2 Reference, SC24-5549* for SAA definition and information.
2. Mainframe Rexx enhances the instructions `options` and `parse` and adds the extended mainframe instruction `upper`. Mainframe extended functions include `externals`, `find`, `index`, `justify`, `linesize`, and `userid`. In addition, both VM and OS Rexx add their own external functions, but what is included in this list varies between VM and OS.
3. Mainframe Rexx supports the Double-Byte Character Set, which allows encoding for ideographic languages, such as Asian languages like Chinese and Korean.
4. I/O on the mainframe is often performed using `EXECIO`.
5. Immediate commands can be entered from the command line and they affect the executing script in real time. Immediate commands can turn the trace on or off, suspend or resume script execution, and suspend or resume terminal (display) output.
6. VM Rexx knows about and automatically loads any or all of these three named packages if any function within them is invoked from within a script: `RXUSERFN`, `RXLOCFN`, and `RXSYSFN`. Users may add their own functions to these libraries.
7. Rexx compilers separate the compile (or script preparation step) from its execution. This usually produces an executable that runs faster, purchased at the price of a two-step process. If a Rexx script is stable and unlikely to change, and is run frequently, compiling it might increase its run-time performance. Note that the mainframe compiler does not guarantee a performance increase.
8. VM Rexx requires a comment line, while OS TSO/E Rexx requires the word `REXX`. To satisfy both requirements, encode this as the first line in a mainframe Rexx script: `/* REXX */`.



Answers to “Test Your Understanding” Questions

9. VM Rexx scripts reside in files of type EXEC. XEDIT editor macros reside in files of type XEDIT. CMS pipelines use files of type REXX.

Chapter 30

1. NetRexx offers some of the traditional advantages of Rexx: easy syntax, ease of use, and ease of maintenance. You can intermix NetRexx and Java classes however you like.
2. The NetRexx translator can be used as a compiler or an interpreter. As an interpreter, it allows NetRexx programs to run without needing a compiler or generating .class files. As a compiler, it can compile scripts into .class files. NetRexx programs can be both compiled and interpreted in just one command. This is easy to do and machine-efficient. NetRexx can also generate formatted Java code, including original commentary, if desired.
3. NetRexx scripts run on any machine that offers a Java Virtual Machine (JVM). This makes them portable across all Java environments. As an interpreter, the NetRexx translator allows NetRexx programs to run without needing a compiler or generating .class files.
4. Indexed strings are NetRexx strings by which subvalues are identified by an index string. Arrays are tables of fixed size that must be defined before use. Arrays may index elements of any type, and the elements are considered ordered. To define a dictionary collection class, refer to the Java documentation. NetRexx uses all the Java classes and methods.
5. NetRexx uses all the Java classes and methods, so refer to the Java documentation for information.
6. *.nrx files contain the source of a NetRexx script. *.java files contain Java source code, and can be produced automatically from NetRexx scripts by the NetRexx translator. *.class files contain the binary or compiled form of a program.
7. Special names perform an action wherever they appear in the source. For example, ask reads a line from the default input stream and returns it as a string of type Rexx while length returns an array's length (the number of elements). The special methods include super, the constructor of the superclass, and this, the constructor of the current class.
8. To migrate classic Rexx scripts to NetRexx, download the free Rexx2Nrx classic Rexx to NetRexx automated conversion tool.

