Here is a quick reference guide for Ruby developers:

**What is Ruby?**

Ruby is a pure object-oriented programming language. It was created in 1993 by Yukihiro Matsumoto of Japan. Ruby is a general-purpose, interpreted programming language like PERL and Python.

**What is IRb?**

Interactive Ruby *IRb* provides a shell for experimentation. Within the IRb shell, you can immediately view expression results, line by line.

This tool comes along with Ruby installation so you have nothing to do extra to have IRb working. Just type `irb` at your command prompt and an Interactive Ruby Session will start.

**Ruby Syntax:**

- Whitespace characters such as spaces and tabs are generally ignored in Ruby code, except when they appear in strings.
- Ruby interprets semicolons and newline characters as the ending of a statement. However, if Ruby encounters operators, such as `+`, `-`, or backslash at the end of a line, they indicate the continuation of a statement.
- Identifiers are names of variables, constants, and methods. Ruby identifiers are case sensitive. It means Ram and RAM are two different identifiers in Ruby.
- Ruby comments start with a pound/sharp `#` character and go to EOL.

**Reserved words:**

The following list shows the reserved words in Ruby. These reserved words should not be used as constant or variable names in your program, however, be used as method names.

```plaintext
BEGIN    do     next    then
END      else   nil     true
alias    elsif  not     undef
and      end     or      unless
begin    ensure  redo    until
break    false   rescue  when
case     for     retry   while
class    if      return  while
def      in      self    __FILE__
defined? module  super   __LINE__
```

**Here Docs in Ruby:**

Here are different examples:
#!/usr/bin/ruby -w
print <<EOF
This is the first way of creating her document ie. multiple line string.
EOF
print "EOF";
# same as above
print `EOC`
# execute commands
echo hi there
echo lo there
EOC
print <<"foo", <<"bar"  # you can stack them
I said foo.
foo
I said bar.
bar

Ruby Data Types:

Basic types are numbers, strings, ranges, arrays, and hashes.

Integer Numbers in Ruby:

<table>
<thead>
<tr>
<th>Number</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td># Fixnum decimal</td>
</tr>
<tr>
<td>1_6089</td>
<td># Fixnum decimal with underline</td>
</tr>
<tr>
<td>-5000</td>
<td># Negative Fixnum</td>
</tr>
<tr>
<td>0377</td>
<td># octal</td>
</tr>
<tr>
<td>0xee</td>
<td># hexadecimal</td>
</tr>
<tr>
<td>0b1011011</td>
<td># binary</td>
</tr>
<tr>
<td>?b</td>
<td># character code for 'b'</td>
</tr>
<tr>
<td>?\n</td>
<td># code for a newline (0x0a)</td>
</tr>
<tr>
<td>12345678901234567890</td>
<td># Bignum</td>
</tr>
</tbody>
</table>

Float Numbers in Ruby:

<table>
<thead>
<tr>
<th>Number</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023.4</td>
<td># floating point value</td>
</tr>
<tr>
<td>1.0e6</td>
<td># scientific notation</td>
</tr>
<tr>
<td>4E20</td>
<td># dot not required</td>
</tr>
<tr>
<td>4e+20</td>
<td># sign before exponential</td>
</tr>
</tbody>
</table>

String Literals:

Ruby strings are simply sequences of 8-bit bytes and they are objects of class String.

- 'VariableName': No interpolation will be done
- "#{VariableName} and Backslashes \n:" Interpolation will be done
- %qVariableName: No interpolation will be done
- %QVariableNameandBackslashes\n: Interpolation will be done
- %VariableNameandBackslashes\n: Interpolation will be done
- `echo command interpretation with interpolation and backslashes
- %xechocommandinterpretationwithinterpolationandbackslashes

Backslash Notations:

Following is the list of Backslash notations supported by Ruby:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Character represented</th>
</tr>
</thead>
</table>

Ruby Arrays:

Literals of Ruby Array are created by placing a comma-separated series of object references between square brackets. A trailing comma is ignored.

Example:

```ruby
#!/usr/bin/ruby
ary = ["Ali", 10, 3.14, "This is a string", "last element", ]
ary.each do |i|
  puts i
end
```

This will produce the following result:

```
Ali
10
3.14
This is a string
last element
```

Ruby Hashes:

A literal Ruby Hash is created by placing a list of key/value pairs between braces, with either a comma or the sequence => between the key and the value. A trailing comma is ignored.

Example:

```ruby
#!/usr/bin/ruby
hsh = colors = { "red" => 0xf00, "green" => 0x0f0 }
hsh.each do |key, value|
  print key, " is ", value, "\n"
end
```

This will produce the following result:

```
green is 240
red is 3840
```
Ruby Ranges:

A Range represents an interval, a set of values with a start and an end. Ranges may be constructed using the s..e and s...e literals, or with Range.new.

Ranges constructed using .. run from the start to the end inclusively. Those created using ... exclude the end value. When used as an iterator, ranges return each value in the sequence.

A range 1..5 means it includes 1, 2, 3, 4, 5 values and a range 1...5 means it includes 2, 3, 4 values.

Example:

```ruby
#!/usr/bin/ruby
(10..15).each do |n|
  print n, ' ' end
```

This will produce the following result:

```
10 11 12 13 14 15
```

Variable Types:

- `$global_variable`
- `@@class_variable`
- `@instance_variable`
- `[OtherClass::]CONSTANT`
- `local_variable`

Ruby Pseudo-Variables:

They are special variables that have the appearance of local variables but behave like constants. You can not assign any value to these variables.

- `self`: The receiver object of the current method.
- `true`: Value representing true.
- `false`: Value representing false.
- `nil`: Value representing undefined.
- `__FILE__`: The name of the current source file.
- `__LINE__`: The current line number in the source file.

Ruby Predefined Variables:

Following table lists all the Ruby's predefined variables.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$!</code></td>
<td>The last exception object raised. The exception object can also be accessed using =&gt; in rescue clause.</td>
</tr>
<tr>
<td><code>$@</code></td>
<td>The stack backtrace for the last exception raised. The stack backtrace information can retrieved by Exception#backtrace method of the last exception.</td>
</tr>
<tr>
<td><code>$/</code></td>
<td>The input record separator newlinebydefault. gets, readline, etc., take their</td>
</tr>
</tbody>
</table>
input record separator as optional argument.

\$
  The output record separator *nilbydefault*.

,
  The output separator between the arguments to print and Array#join
  *nilbydefault*. You can specify separator explicitly to Array#join.

;
  The default separator for split *nilbydefault*. You can specify separator
  explicitly for String#split.

.
  The number of the last line read from the current input file. Equivalent
  to ARGF.lineno.

<
  Synonym for ARGF.

>
  Synonym for $defout.

0
  The name of the current Ruby program being executed.

?
  The process pid of the current Ruby program being executed.

:
  The exit status of the last process terminated.

DEBUG
  True if the -d or --debug command-line option is specified.

defout
  The destination output for *print* and *printf* ($stdout by default).

F
  The variable that receives the output from *split* when -a is specified.
  This variable is set if the -a command-line option is specified along with
  the -p or -n option.

FILENAME
  The name of the file currently being read from ARGF. Equivalent to
  ARGF.filename.

LOAD_PATH
  An array holding the directories to be searched when loading files with
  the load and require methods.

SAFE
  The security level

  0 --> No checks are performed on externally supplied *tainted* data.
  default

  1 --> Potentially dangerous operations using tainted data are
      forbidden.

  2 --> Potentially dangerous operations on processes and files are
      forbidden.

  3 --> All newly created objects are considered tainted.

  4 --> Modification of global data is forbidden.

stdin
  Standard input *STDINbydefault*.

stdout
  Standard output *STDOUTbydefault*.

stderr
  Standard error *STDERRbydefault*.

VERBOSE
  True if the -v, -w, or --verbose command-line option is specified.

-x
  The value of interpreter option -x $x = 0, a, d, F, i, K, l, p, v. These options
  are listed below

-0
  The value of interpreter option -x and alias of $/.

-a
  The value of interpreter option -x and true if option -a is set. Read-only.
$-d  
The value of interpreter option -x and alias of $DEBUG

$-F  
The value of interpreter option -x and alias of $;

$-i  
The value of interpreter option -x and in in-place-edit mode, holds the extension, otherwise nil. Can enable or disable in-place-edit mode.

$-l  
The value of interpreter option -x and alias of $.

$-l  
The value of interpreter option -x and true if option -lis set. Read-only.

$-p  
The value of interpreter option -x and true if option -pis set. Read-only.

$_  
The local variable, last string read by gets or readline in the current scope.

$~  
The local variable, MatchData relating to the last match. Regex#match method returns the last match information.

n(1, 2, 3...)  
The string matched in the nth group of the last pattern match. Equivalent to m[n], where m is a MatchData object.

$&  
The string matched in the last pattern match. Equivalent to m[0], where m is a MatchData object.

$`  
The string preceding the match in the last pattern match. Equivalent to m.pre_match, where m is a MatchData object.

$'  
The string following the match in the last pattern match. Equivalent to m.post_match, where m is a MatchData object.

$+  
The string corresponding to the last successfully matched group in the last pattern match.

$+  
The string corresponding to the last successfully matched group in the last pattern match.

Ruby Predefined Constants:

The following table lists all the Ruby's Predefined Constants.

**NOTE:** TRUE, FALSE, and NIL are backward-compatible. It's preferable to use true, false, and nil.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Synonym for true.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Synonym for false.</td>
</tr>
<tr>
<td>NIL</td>
<td>Synonym for nil.</td>
</tr>
<tr>
<td>ARGF</td>
<td>An object providing access to virtual concatenation of files passed as command-line arguments or standard input if there are no command-line arguments. A synonym for $&lt;.</td>
</tr>
<tr>
<td>ARGV</td>
<td>An array containing the command-line arguments passed to the program. A synonym for $*.</td>
</tr>
<tr>
<td>DATA</td>
<td>An input stream for reading the lines of code following the <strong>END</strong> directive. Not defined if <strong>END</strong> isn't present in code.</td>
</tr>
<tr>
<td>ENV</td>
<td>A hash-like object containing the program's environment variables. ENV can be handled as a hash.</td>
</tr>
<tr>
<td>RUBY_PLATFORM</td>
<td>A string indicating the platform of the Ruby interpreter.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>RUBY_RELEASE_DATE</td>
<td>A string indicating the release date of the Ruby interpreter.</td>
</tr>
<tr>
<td>RUBY_VERSION</td>
<td>A string indicating the version of the Ruby interpreter.</td>
</tr>
<tr>
<td>STDERR</td>
<td>Standard error output stream. Default value of $stderr.</td>
</tr>
<tr>
<td>STDIN</td>
<td>Standard input stream. Default value of $stdin.</td>
</tr>
<tr>
<td>STDOUT</td>
<td>Standard output stream. Default value of $stdout.</td>
</tr>
<tr>
<td>TOLEVEL_BINDING</td>
<td>A Binding object at Ruby's top level.</td>
</tr>
</tbody>
</table>

**Regular Expressions:**

**Syntax:**

```
/pattern/
/pattern/im    # option can be specified
%r!/usr/local! # general delimited regular expression
```

**Modifiers:**

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Ignore case when matching text.</td>
</tr>
<tr>
<td>o</td>
<td>Perform #{()} interpolations only once, the first time the regexp literal is evaluated.</td>
</tr>
<tr>
<td>x</td>
<td>Ignores whitespace and allows comments in regular expressions</td>
</tr>
<tr>
<td>m</td>
<td>Matches multiple lines, recognizing newlines as normal characters</td>
</tr>
<tr>
<td>u,e,s,n</td>
<td>Interpret the regexp as Unicode UTF-8, EUC, SJIS, or ASCII. If none of these modifiers is specified, the regular expression is assumed to use the source encoding.</td>
</tr>
</tbody>
</table>

**Various patterns:**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>.</td>
<td>Matches any single character except newline. Using m option allows it to match newline as well.</td>
</tr>
<tr>
<td>[...]</td>
<td>Matches any single character in brackets.</td>
</tr>
<tr>
<td>[^...]</td>
<td>Matches any single character not in brackets</td>
</tr>
<tr>
<td>re*</td>
<td>Matches 0 or more occurrences of preceding expression.</td>
</tr>
<tr>
<td>re+</td>
<td>Matches 0 or 1 occurrence of preceding expression.</td>
</tr>
<tr>
<td>re{ n}</td>
<td>Matches exactly n number of occurrences of preceding expression.</td>
</tr>
<tr>
<td>re{ n,}</td>
<td>Matches n or more occurrences of preceding expression.</td>
</tr>
<tr>
<td>re{ n, m}</td>
<td>Matches at least n and at most m occurrences of preceding expression.</td>
</tr>
</tbody>
</table>
a| b  Matches either a or b.

re   Groups regular expressions and remembers matched text.

?imx Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.

? − imx Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.

?:re Groups regular expressions without remembering matched text.

?imx:re Temporarily toggles on i, m, or x options within parentheses.

? − imx:re Temporarily toggles off i, m, or x options within parentheses.

?#... Comment.

?=re Specifies position using a pattern. Doesn't have a range.

?!re Specifies position using pattern negation. Doesn't have a range.

?>re Matches independent pattern without backtracking.

\w Matches word characters.

\W Matches nonword characters.

\s Matches whitespace. Equivalent to [\t\n\r\f].

\S Matches nonwhitespace.

\d Matches digits. Equivalent to [0-9].

\D Matches nondigits.

\A Matches beginning of string.

\Z Matches end of string. If a newline exists, it matches just before newline.

\z Matches end of string.

\G Matches point where last match finished.

\b Matches word boundaries when outside brackets. Matches backspace \008 when inside brackets.

\B Matches nonword boundaries.

\n, \t, etc. Matches newlines, carriage returns, tabs, etc.

\1...\9 Matches nth grouped subexpression.

\10 Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

**File I/O:**

Common methods include:

- `File.join(path1, path2, ..., pathN) => "p1/p2/.../pN" platform independent paths`
- `File.new(path, mode= "r") => file`
- `File.new(path, mode= "r", permission= [0, 0, 0]) => file`
Here is a list of the different modes of opening a file:

<table>
<thead>
<tr>
<th>Modes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Read-only mode. The file pointer is placed at the beginning of the file. This is the default mode.</td>
</tr>
<tr>
<td>r+</td>
<td>Read-write mode. The file pointer will be at the beginning of the file.</td>
</tr>
<tr>
<td>w</td>
<td>Write-only mode. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.</td>
</tr>
<tr>
<td>w+</td>
<td>Read-write mode. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.</td>
</tr>
<tr>
<td>a</td>
<td>Write-only mode. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.</td>
</tr>
<tr>
<td>a+</td>
<td>Read and write mode. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.</td>
</tr>
</tbody>
</table>

Operators and Precedence:

Top to bottom:

```
:: .
[] **
-(unary) +(unary) ! ~
* / %
+ -
<< >>
& ~
| ^>
>= < <=
<> == === != =~ !~
&&
||
.. ...
=(+=, -=,=...)
not
and or
```

All of the above are just methods except these:

```
=, ::, ., .., ..., !, not, &&, and, ||, or, !=, !~
```

In addition, assignment operators `+=` etc. are not user-definable.

Control Expressions:

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Control Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. if bool-expr [then]
   body
   elsif bool-expr [then]
   body
   else
   body
end

2. unless bool-expr [then]
   body
   else
   body
end

3. expr if bool-expr

4. expr unless bool-expr

5. case target-expr
   when comparison [, comparison]... [then]
   body
   when comparison [, comparison]... [then]
   body
   ...
   [else
   body]
end

6. loop do
   body
end

7. while bool-expr [do]
   body
end

8. until bool-expr [do]
   body
end

9. begin
   body
end while bool-expr

10. begin
    body
end until bool-expr

11. for name[, name]... in expr [do]
    body
end

12. expr.each do | name[, name]... |
- **break** terminates loop immediately.
- **redo** immediately repeats w/o rerunning the condition.
- **next** starts the next iteration through the loop.
- **retry** restarts the loop, rerunning the condition.

**Defining a Class:**

Class names begin w/ capital character.

```ruby
class Identifier [< superclass ]
  expr..
end
```

Singleton classes, add methods to a single instance

```ruby
class << obj
  expr..
end
```

**Defining a Module:**

Following is the general syntax to define a module in ruby

```ruby
module Identifier
  expr..
end
```

**Defining a Method:**

Following is the general syntax to define a method in ruby

```ruby
def method_name(arg_list, *list_expr, &block_expr)
  expr..
end
# singleton method
def expr.identifier(arg_list, *list_expr, &block_expr)
  expr..
end
```

- All items of the arg list, including parens, are optional.
- Arguments may have default values `name=expr`.
- **Method_name** may be operators see above.
- The method definitions can not be nested.
- Methods may override following operators:
  - `.., |, ^, &<=>, ==, ===, =~,`
Access Restriction:

- **public** - totally accessible.
- **protected** - accessible only by instances of class and direct descendants. Even through hasA relationships, see below.
- **private** - accessible only by instances of class must be called nekkid no "self." or anything else.

Example:

```ruby
class A
  protected
  def protected_method
    # nothing
  end
end
class B < A
  public
  def test_protected
    myA = A.new
    myA.protected_method
  end
end
b = B.new.test_protected
```

Raising and Rescuing Exceptions:

Following is the syntax:

```
raise ExceptionClass[, "message"]
begin
  expr...
  [rescue [error_type [=> var],...]
    expr...]
  [else
    expr...]
  [ensure
    expr...]
end
```

Catch and Throw Exceptions:

- `catch :label do ... end`
- `throw :label` jumps back to matching catch and terminates the block.
- `+` can be external to catch, but has to be reached via calling scope.
- `+` Hardly ever needed.

Exceptions Classes:

Following is the class hierarchy of `Exception` class:

- `Exception`
  - `NoMemoryError`
  - `ScriptError`
    - `LoadError`
- NotimplementedError
- SyntaxError
- SignalException
- Interrupt
- StandardError default for rescue
  - ArgumentError
  - IOError
    - EOFError
  - IndexError
  - LocalJumpError
  - NameError
    - NoMethodError
  - RangeError
    - FloatDomainError
  - RegexpError
  - RuntimeError default for raise
  - SecurityError
  - SystemCallError
    - Errno::*
  - SystemStackError
  - ThreadError
  - TypeError
  - ZeroDivisionError
- SystemExit
- fatal

Ruby Command Line Options:

$ ruby [ options ] [ . ] [ programfile ] [ arguments ... ]

The interpreter can be invoked with any of the following options to control the environment and behavior of the interpreter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Used with -n or -p to split each line. Check -n and -p options.</td>
</tr>
<tr>
<td>-c</td>
<td>Checks syntax only, without executing program.</td>
</tr>
<tr>
<td>-C dir</td>
<td>Changes directory before executing equivalent to -X.</td>
</tr>
<tr>
<td>-d</td>
<td>Enables debug mode equivalent to -debug.</td>
</tr>
<tr>
<td>-F pat</td>
<td>Specifies pat as the default separator pattern $; used by split.</td>
</tr>
<tr>
<td>-e prog</td>
<td>Specifies prog as the program from the command line. Specify multiple -e options for multiline programs.</td>
</tr>
<tr>
<td>-h</td>
<td>Displays an overview of command-line options.</td>
</tr>
</tbody>
</table>
-i [ ext] Overwrites the file contents with program output. The original file is saved with the extension ext. If ext isn't specified, the original file is deleted.

-1 dir Adds dir as the directory for loading libraries.

-K [kcode] Specifies the multibyte character set code e or E for EUC (extended Unix code; s or S for SJIS Shift-JIS; u or U for UTF-8; and a, A, n, or N for ASCII).

-l Enables automatic line-end processing. Chops a newline from input lines and appends a newline to output lines.

-n Places code within an input loop as in while gets; ... end.

-0[ octal] Sets default record separator $/ as an octal. Defaults to \0 if octal not specified.

-p Places code within an input loop. Writes $ for each iteration.

-r lib Uses require to load lib as a library before executing.

-s Interprets any arguments between the program name and filename arguments fitting the pattern -xxx as a switch and defines the corresponding variable.

-T [level] Sets the level for tainting checks 1 if level not specified.

-v Displays version and enables verbose mode

-w Enables verbose mode. If programfile not specified, reads from STDIN.

-x [dir] Strips text before #!ruby line. Changes directory to dir before executing if dir is specified.

-X dir Changes directory before executing equivalent to -C.

-y Enables parser debug mode.

--copyright Displays copyright notice.

--debug Enables debug mode equivalent to -d.

--help Displays an overview of command-line options equivalent to -h.

--version Displays version.

--verbose Enables verbose mode equivalent to -v. Sets $VERBOSE to true

--yydebug Enables parser debug mode equivalent to -y.

Ruby Environment Variables:
Ruby interpreter uses the following environment variables to control its behavior. The ENV object contains a list of all the current environment variables set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLN_LIBRARY_PATH</td>
<td>Search path for dynamically loaded modules.</td>
</tr>
<tr>
<td>HOME</td>
<td>Directory moved to when no argument is passed to Dir::chdir. Also used by File::expand_path to expand &quot;~&quot;.</td>
</tr>
<tr>
<td>LOGDIR</td>
<td>Directory moved to when no arguments are passed to Dir::chdir and environment variable HOME isn't set.</td>
</tr>
<tr>
<td>PATH</td>
<td>Search path for executing subprocesses and searching for Ruby</td>
</tr>
</tbody>
</table>
programs with the -S option. Separate each path with a colon semicolon in DOS and Windows.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUBYLIB</td>
<td>Search path for libraries. Separate each path with a colon semicolon in DOS and Windows.</td>
</tr>
<tr>
<td>RUBYLIB_PREFIX</td>
<td>Used to modify the RUBYLIB search path by replacing prefix of library path1 with path2 using the format path1;path2 or path1path2.</td>
</tr>
<tr>
<td>RUBYOPT</td>
<td>Command-line options passed to Ruby interpreter. Ignored in taint mode where $SAFE is greater than 0.</td>
</tr>
<tr>
<td>RUBYPATH</td>
<td>With -S option, search path for Ruby programs. Takes precedence over PATH. Ignored in taint mode where $SAFE is greater than 0.</td>
</tr>
<tr>
<td>RUBYSHELL</td>
<td>Specifies shell for spawned processes. If not set, SHELL or COMSPEC are checked.</td>
</tr>
</tbody>
</table>