GMI Parser Application Notes

GMI Command Utility Parse Function
General Management Interface Foundation
http://www.gmi-foundation.org

Introduction

This document provides a reference manual and application notes for the "parser" functions of the GMI command library, and "GMI-Cmd.exe" executable program.

The parser function is a simple built-in parsing utility that is useful for locating and parsing values from GMI agent listings, incorporated into the "GMI-Cmd.exe" program. Also, the GMI parser function is also available via the "GMI-Cmd32.lib" link library via the "GMIC_get()" and "GMIC_parse()" functions of that library.

A complete understanding of parser functions is not necessary to use the GMI Agent, utilities, and libraries. The parser function described herein is easily substituted by other external programs, such as "awk", and by link libraries such as "regex()". However, the parser interface is very simple to learn and use, and can offer good benefits for intermediate and advanced users of the GMI software looking to accomplish straightforward parsing of GMI listings and values.

Background

The GMI Agent can return multiple values from a single query of the GMI-Cmd.exe utility. For example, the agent "/system/SysInfo" folder can contain a single line, or can consist of an entire structured or unstructured file. Likewise, the "/install/apps" folder contains certification information that can have multiple tagged entries as part of the list.

Because a listing, fetched from the GMI agent, is often more than just a single one-line value, some mechanism may be necessary to parse data from the multi-line listing. Specifically, the user can list data to a file using standard ">" and ">>" redirect symbols, and can pipe data to another program using the standard "|" delimiter. The captured data can then be parsed using any number of tools or programs.

A built-in alternative to using an external parsing program is to pipe data to the GMI "parse" function, which then parses the data using an "awk" like program syntax. This parser takes advantage of the regular nature of most GMI folder lists to permit easy access to individual data elements commonly found throughout the agent.

To use the parser from the GMI-Cmd.exe program, simply pipe the output of a command to the "parse" command (as if it was an external command.) The parse command then filters and parses the data, and writes the results to standard output, with redirection to a file also possible.

Additionally, the "GMI-Cmd32.lib" link library provides access to the parse functions of the GMI-Cmd.exe program, and permits users to easily parse a single value from a listing to acquire a metric such as "bytes received", "disk full", "root folder", or any other value produced by the agent.

Parser Syntax Rules

The "parse" command accepts either one or two arguments. The first argument is a match pattern or keyword, and the second argument is a field specification. If the first argument is omitted, the parser matches all lines. If the second argument is omitted, the matched lines are simply written out without modification.

The general syntax of the "parse" command is therefore as follows:

```
parse "matchpatt" fieldspec
```

At the GMI-cmd.exe utility command prompt, the user can parse any standard output of the program. For example, to parse the output of a folder or Application subdirectory, the user can issue a command such as the following:

```
GMI> ls (agentpath) | parse "matchpatt" fieldspec
```

Specifically, to parse the contents of a file from the GMI-cmd.exe command line (such as for test purposes) the user can issue the following command (where the "type" function is the built-in GMI command to display the contents of a file to standard output.)

```
GMI> type (filename) | parse "matchpatt" fieldspec
```

As illustrated above, the GMI-Cmd.exe "parse" command accepts standard input, and is only useful when executed on the right side of a command pipe.

For each line of standard input, when a line matches the optional "matchpatt" specification, the line passed to the "fieldspec" specification, which selects the particular words to write to standard output.

Parser "Match Pattern" Argument

The first argument to the parser, if supplied, is the required keyword or phrase that must match all lines in standard input. The "matchpatt" argument can be specified several ways, using the following conventions.

1. The "matchpatt" argument can be a single keyword or a phrase containing blank characters. Leading and trailing blanks are ignored.

2. The "matchpatt" argument is case insensitive;

3. The "matchpatt" argument can be a regular expression contain the following special wildcards as follows: (a) "*" asterisk matches zero or more characters; (b) "?" question mark matches one character; (c) "^" circumflex matches start of string; (d) "$" dollar character matches end of string; (e) "\" backslash escapes the special meaning of a character.

4. The "matchpatt" argument can be enclosed in double or single balanced quotes, (to delimit white space);

5. To match a single or double quote, or other special character as part of the match pattern, the character can be preceded by a backslash '\' character to escape the special meaning of the character. (Special characters include slashes, quotes, braces, and the dollar sign character.)

6. The "matchpatt" argument can be specified in "awk" like syntax, such as "/mypatt/", to accommodate users familiar with the "awk" program;

7. The "matchpatt" argument can be omitted to match all lines of standard input.

Parser "Field Specification" Argument

The second argument to the parser, if supplied, is a specification of which particular words to write to standard output in the matched input. The "fieldspec" argument can be specified by itself, or following the "matchpatt" argument above. The "fieldspec" argument has the following conventions:

1. The "fieldspec" argument consists of word references where $1 specifies the first word, $2 specifies the second word, $3 specifies the third word, and so on. This syntax will be familiar to "awk" program users, where the $dollar-number convention simply specifies the particular ordinal position of any word in the line.

2. If a field does not exist for a specified "fieldspec" reemergence (for example $99 is used and there are only five words in the standard input) then no error occurs and the specified field is simply ignored. Any other valid fields are output.

3. The "fieldspec" argument can be enclosed in double quotes, or "awk" like brace "{" and "}" characters, to accommodate users familiar with "awk" syntax.

4. The "fieldspec" argument can be omitted (if a "matchspec" argument is specified) so that the entire matched line is written to standard output.

The operator must supply either a "matchpatt", or a "fieldspec", or both. If the operator includes just a "fieldspec", then all lines are matched. If the operator includes just the "matchpatt" argument, then all lines that match the pattern are returned.

Parser Examples

Given the above rules of syntax, the following examples illustrate the various ways to use the parser function.

1. Return all lines in standard input that match the "system" keyword. The command below uses the "parse" function to display any lines, implementing a function similar to the MS Windows "find" utility program, or UNIX "grep" utility program.

   ```
   parse "system"
   ```

2. Same as above, but the match pattern is delimited with forward slashes, in a convention that will be recognized by "awk" program users. Otherwise, the command below is identical to the above.

   ```
   parse /system/
   ```

3. Same as above, but use a wildcard in the match pattern:

   ```
   parse /^s*tem/
   ```

4. Match all strings containing a forward slash "/" character. Note that the slash character has to be escaped with a backslash in order to use the character in any match pattern (irrespective of whether the match pattern is enclosed in slashes or quotes.)

   ```
   parse "/\//"
   ```

5. Match all strings containing a brace "{" character. As above, the brace character is special, hence must be escaped before it can be used in a match pattern.

   ```
   parse "\{"
   ```

6. For all lines in standard input, return just the first word of the line. Any blank lines (that have no first word) are omitted from the listing. Otherwise, the first word of all lines is written to standard output:

   ```
   parse $1
   ```

7. For all lines in standard input, return the fifth word, the third word, and the seventh word. If a line has less than seventh words (but more than five words) then just the fifth and third word is returned.

```
parse $5 $3 $7
```

8. Return the second and fourth words for all lines in standard input that match the "test" keyword. If the line matches "test", but has less than two words, no value is returned for that line

```
parse test $2 $4
```

9. Same as above, but the match pattern is delimited with forward slashes, in a convention that will be recognized by "awk" program users. Otherwise, the command below is identical to the above.

```
parse /test/ $2 $4
```

10. Same as above, but the field specification (second argument) is delimited by braces. The command below will look very familiar to "awk" program users. Otherwise, the command operates the same as the above two commands:

```
parse /test/ {$2 $4}
```

Conclusion

The "parse" function is intended to provide simple access to the type of data often found when working with GMI Agent applications. This type of data is distinguished by highly structured output, usually consisting of row data with fixed numbers of columns, where each column has at least one unique label.

Using the "parse" function (either at the GMI-Cmd.exe program prompt or within the GMI-Cmd32.lib link library) an operator will have no problem selecting certain values from any typical GMI Agent listing.

Users of the "awk" program will find the syntax of the program quite familiar and accommodating. However, note that the parser is not intended to be a complete substitute for "awk" or other highly proficient parsing systems; the "parse" command is strictly intended as a standard and simple method of extracting data from a GMI Agent listing, such as the numeric or textual value associated with some labeled data item.

In particular, the "awk" program (which is readily available for all types of platforms from the web) works perfectly as a substitute for all the functions described herein, and should be considered for any more elaborate parsing requirements.

For more information on the parse function, see the "GMI-Cmd.exe Reference Manual", and the "GMI Command API Reference Manual, available from the GMI Foundation website.

About The GMI Foundation

The GMI Software consists of agents, utilities, documentation and API's intended to promote secure and flexible management of end-points. GMI fully supports Open Source Initiatives, and views its role as one of not-for-profit promotion of software to correct the endemic problems associated with system and software management. More information on the GMI Foundation is available at the following location:

GMI Foundation
http://www.gmi-foundation.org
mailto: info@gmi-foundation.org

About Our Technical Partners

Additional information on the GMI Software, including a selection of useful GMI Apps, as well as professional and support services, is available from various members of the GMI Foundation.

A central clearinghouse for GMI applications is provided by Vallum Software, LLC, which provide support, certification, and development solutions, as well as the "Halo Management System", based on the GMI agent. Visit the link below.

Vallum Software, LLC
http://www.vallumsoftware.com
mailto: info@vallumsoftware.com