Choosing a Command Language—
An Application-Centric Approach

Hal German
GTE

Choosing a Command Language -- An Application-Centric Approach

Hallett German

GTE Laboratories, Incorporated.

An Introduction to this paper

For over four years, the author has discussed a means for beginning and intermediate command language users to quickly choose the essential elements of their application without using a single piece of code. This paper is the first time the approach has been presented to the movers and shakers of the REXX world. It supplements the presentation by covering the following:

1. Why use such an approach?
2. Concepts behind the approach.
3. The approach itself
4. Conclusions
5. References

The presentation at the REXX Symposium will provide an overview of the approach, as well as an example of how to use, and include other factors to consider. Handouts can be obtained by contacting the author.

Why use such an approach?

In "the old days" it was easy. You used a mainframe host that had one command language and one editor (that usually had ties to the command language). Then PCs and UNIX systems snuck in from somewhere and the issues became more complex. There were more than one command language and editor to choose from. Programs could run on more than one operating system (and simultaneously if needed). Unfortunately, the theories and software practices for command languages were not enhanced to match the new realities. The approach listed below is a modest attempt to provide command language developers a strategy to deal with the new realities so they don't have to say "What do I do next?"

Concepts behind the approach

1. **What is a command language?**

Unfortunately, we only can briefly look at this area. My definition of a command language is the following:

A programming language consisting of a series of high-level English-like commands entered interactively (e.g., a keyboard, mouse, or other input device) or non-interactively (that is created with an editors, saved in a file, and executed in foreground or background). An interpreter or compiler for the command language then determines which user-specified operating system tasks to perform and processes them using corresponding task values.

Whew! A real mouthful. So what does it mean?

* Command languages are almost always interpreted languages. (REXX is one of the exceptions to this.)

* Command languages are usually executed in foreground. (Again, REXX is one of the exceptions.)

* Command languages are comprised of English-like verbs describing the task to perform. REXX is typical with instruction keywords like SAY and PULL.

* Command language provides a mean to directly or indirectly access the operating system. REXX shines in this area with the ADDRESS instruction and the environment model.

* Command languages offer user and third-party extensions. For REXX this includes functions, sub procedures, and interfaces to external environments.

2. Identifying the types of command language applications

In their CLIST manual, IBM talked about three types of command language applications. My eight years of working with various command languages have verified that this typology is a good match for the type of applications found in the real world.

These types are the following:

Front-end -- Also called "housekeeping" applications. In this case, the command language sets up the proper environment for an application to execute. This could be allocating files, creating files, or creating environment variables. They also can receive output from or send input to the application. I view the startup or login programs as a special example of a front-end application.

System and Utility -- This is like front-end command language application. However, the emphasis is on doing system tasks (such as backing up files) and utility operations (Such as being a function/sub procedure that performs a date operation.)

Self-contained -- The other two types are "blue-collar" applications. The "white-collar" application type is the self-contained application. It provides a dialog with the user (usually full-screen) while maintaining strict control over the process.

3. The Command Language Component

The last and most important piece of the puzzle is the command language component. All command languages that have examined to date have the following components:

* Input/Output (File operations, Stack operations, Output to the screen, Input from the keyboard)

* Flow Control (Conditional, Loop, Exception handling, Exit and return codes, Array operations)

* General Features (Debugging, Symbolic Substitution, Labels, Global Options, Numeric format, Interpreter Version)

* Interfaces (Internal functions, interface to operating system and external environments)

* Built-in (Functions, and System variables)

What the approach does is combine all three of the above elements. First determine your type of application, once you know that, you know the command language components that are usually used by that application type.Finally look up the commands corresponding to that command language component. **And not a single piece of code has been yet been written.**

The approach is application-centric because it encourages you to know your application requirements and data as much as possible before starting to code.

The approach

The following are the steps of the approach:

1. What type of application do I have?

   The three types were discussed above.

2. Which command language should I use?

   This is discussed in the presentation. This includes a look at the following:

   * Type of data
   * File type
   * benchmarks
   * Ease of use vs. power
   * features

3. Which command language components should I use?

   The components were listed above.

4. Which command language match these components?

   This is the crucial step. Table 1 lists a summary of the components.

5. Which command language components match these commands?

   Space does not permit listing this step. However, tables with this information can be found in the references section.

6. Where can I find more about these commands?

   There are many places you can learn about a command language command. These include: user guides, books, on-line references, summary references, electronic information servers, electronic mailing lists, user groups, and colleagues.

7. Do I need third-party extensions?

   Third-party extensions should be used in the following situations:

   * When portability is not a concern.

   * When the third-party extension performs an operation not found in the command language such as network and database operations.

   * When you can afford the run-time license costs for distributing the extension.

   * When the extension greatly enhances the look and feel of the application. Such as any of the "Visual REXXes."

Conclusions

I hope that this will be of use to you the next time that you are considering developing a command language application. I encourage others to look into this area.

Getting in touch with me

Hallett German
GTE Laboratories Inc
40 Sylvan Road
Waltham, Ma 02254
617-466-2290
hhg1@.gte.com

References

German, Hallett **Command Language Cookbook**, VNR 1992
[The approach is covered in detail. Looks at many different type of REXX implementations.]

German, Hallett **OS/2 2.1 REXX Handbook**, VNR 1994
[A Rexx tutorial and the approach with some enhancements.]

Table 1 Command Language Components by Application Type

Front-end
- Operating System Commands
- External Interfaces
- Input/Output: File operations & Command Line operations.
- Built-In Functions
- Flow Control: Conditional

System/Utility
- Operating System Commands
- External Commands
- Internal Commands
- Input/Output: File/Screen Operations
- Command line input
- Built-in Functions: String Operations
- Flow Control: Loops
- General: Batch Operations, Arrays

Self-contained
- External Commands
- Functions/Sub-procedures
- Input/Output: Command Line operations, user validation
- Flow control: Multiple conditions
- Built-in Functions: Text Case & string.
- General (Interactive operations, arrays)