Enhancing CLIST, REXX Exec, and ISPF Panel Development

By Lionel B. Dyck

How often, when you are developing or updating CLISTs, Execs, or ISPF Panels, have you wanted an easy way to verify that your updates work? How many times, while working on an assembler program, have you wanted to assemble the program while still in ISPF Edit? I have, and from the response to a new tool that I recently posted on my website, it would appear that I am not alone. This article will present to you a tool that I call TRYIT.

TRYIT can be used to test your JCL, CLISTs, your REXX Execs, your ISPF Panels, and to assemble your assembler programs, without the need to split screens, invoke ISPF Dialog Test, or open ISPF option 6. It will dynamically determine the type of member being edited and perform the appropriate actions to test your code. Part of the appropriate actions is the dynamic allocation using ALTLIB or LIBDEF of the data set in which the member being edited resides so that any CLISTs, Execs, or Panels called by the current member can be found. This means that you no longer have to place the files being updated into your SYSPROC, SYSEXEC, or ISPPLIB allocations. Furthermore, with ISPF Panels you can now perform a test of the panel, make changes, and test again all without resorting to ISPF Dialog Test. For JCL, you need to have a JCL syntax verification product installed that can be invoked as an ISPF Edit command (all of them that I am familiar with can do that).

Does this sound good so far?

TRYIT is written in REXX as an ISPF Edit Macro. Therefore, you have the complete source available to learn from and to modify or enhance if you wish (and since it is being released under the GPL License this is an open source tool).

How does TRYIT work? The brief answer is it works very well, thank you. The longer answer is as follows:

When TRYIT is entered on the ISPF Edit command line, the first test is to determine if the member being edited has been changed. If so, TRYIT will optionally issue an ISPF message telling the user to save their changes and to try again. TRYIT does this so that the user is fully aware of what is happening. This notification setting is defined within the TRYIT code as a site installation option.

FIGURE 1: TRYIT COMMAND

```
EDIT       hlq.test.exec(sample) - 01.01
Command ===> tryit opt1 opt2
```

Next, TRYIT will attempt to determine the type of member being edited. This involves looking at the text of the first line of data. The word PROC followed by a numeric value is determined to be a CLIST. The word REXX on the first line will set the type as a REXX Exec. The test for an ISPF Panel is more involved, including the ability to detect a pre-processed Panel (which you can not edit but can still test). JCL is determined by a // in the first two columns of the first record.

Should that fail, the next step involves testing the suffix of the active data set name and if that fails, then testing the first line of data within the member. Data sets with a suffix of CLIST, SYSPROC, or CMDPROC are considered CLIST members. A suffix of REXX, EXEC, or SYSEXEC will set the type as a REXX Exec. The type of ISPF Panel is set if the suffix is PANEL, PANELS, or ISPPLIB. JCL suffixes are JCL and CNTL with Assembler suffixes of ASM and ASSEM.

Once the type has been determined, the proper routine can be called to test the member.

For JCL, assuming an appropriate JCL syntax verification tool has been defined, the command will be executed as an ISPF Edit command on the active member.

For a CLIST or REXX Exec the active data set is allocated using ALTLIB and the member is then executed, passing any parameters that were provided on the TRYIT command (see FIGURE 1).

Upon completion of the execution of the member, the return code is saved and the ALTLIB released. The user is then informed using ISPF short and long messages about the results of the test.

For ISPF Panels the active member is copied into a temporary PDS that is then allocated using LIBDEF, along with the active library. Any TRYIT parameters are then evaluated to determine how the PANEL is

to be tested. By default, the member is displayed. The supported TRYIT parameters supported are:

- POP to cause the panel to be displayed within an ISPF Popup window.
- SEL to select the panel instead of just displaying it.
- APPL followed by a 1 to 4 character ISPF applications ID will be used if SEL is also requested.
- TUT to display the panel as an ISPF Tutorial.

Upon return from testing, the return code is saved and the LIBDEF released. If the panel processing returns cleanly, then an ISPF short message indicates a return code of zero. If there are problems, then the ISPF short and long messages are displayed using the values set by the ISPF service that was used.

For Assembler programs, the active member is saved and then an ISPF panel is displayed prompting for assembly time options and optionally linkage editor options. Note that the ISPF panel is included inline within the TRYIT REXX code and is dynamically loaded into a temporary data set that is allocated using LIBDEF. Thus, TRYIT is fully contained within a single REXX program.

The result is that you will be able to develop, update and test your CLISTs, REXX Execs, ISPF Panels, and Assembler programs faster and easier and thus be more productive.

The approach taken by TRYIT can be expanded to test other member types. For example, it would be possible to detect that the member is a C program and to invoke the C compiler. On the other hand, TRYIT can detect that the member contains Script/VS (DCF) text and invoke the Script/VS processor to create a document. The variations are numerous. I will leave these and others up to you.

Should you enhance TRYIT, I would be very interested in hearing from you and would encourage you to share your updates with others and me. This can be easily done using the phpBB-based forums available on my website at http://www.lbdsoftware.com/forum.

The TRYIT package can be found at http://www.lbdsoftware.com on the ISPF Tools page. Here you will also find many other tools to enhance your and your user's productivity.

NaSPA member Lionel B. Dyck is a z/OS systems programmer for a large health maintenance organization in California. He has been in systems programming since 1972 and is an active participant at SHARE. lionel.b.dyck@kp.org is his e-mail address.