ISPF vs Zowe: Old School Meets New School

ISPF/TSO and Zowe are both tools that are uniquely yet incredibly important for both new and long-time mainframe developers. This lesson will discuss various topics, including these tools, such as, but not limited to: What is ISPF/TSO? What is Zowe? What can they do the same? What can they do differently? And the question everyone will probably be wondering: Which tool is “better”?

After clearly defining what ISPF/TSO and Zowe are, as well as comparing and contrasting the two, this lesson will transition into information about Z/OS data sets, since ISPF/TSO and Zowe are both tools used to manipulate Z/OS data sets. This will provide a student who either 1) has no Z/OS experience or 2) has some experience with Z/OS, but is confused aboudt anything relating to the ISPF/TSO terminal or data sets, a good foundation in understanding how to navigate around a Z/OS terminal, what the purpose of using a green screen is, even if it may not seem efficient, as well as how different data sets work, so if a student were to start coding in a Z/OS environment after participating in this lesson, they would have a much larger grasp of what’s going on, vs just typing commands because an instructor told you to so you can get around the terminal.

What is ISPF/TSO?

ISPF stands for “Interactive System Productivity Facility”, and TSO stands for “Time Sharing Option”, but what do either of those mean, and how do they relate to mainframes at all?

Firstly, TSO is an essential tool when interacting with mainframes, usually accessed through a terminal emulator, which in the case of CSCI360 and CSCI465, we use the terminal called “Vista TN3270”. To summarize, the use of TSO allows users to interact directly with the z/OS operating system through the use of a command-line interface. This is a very important tool because it supports multiple users working concurrently to share the z/OS resources. This is where the “Time Sharing” in the name comes from. Through this command line, some common commands that may be done include: submitting jobs, monitoring those job logs, and managing any associated data sets. And while these are some really useful commands that can be done, there are also various other important commands that a user can use at this command line.

Next, ISPF is the user interface that the user is presented with after logging on to TSO. ISPF runs on top of TSO, which is meant to serve as well well-structured and user-friendly menu to help users execute TSO commands, but does so in a way that breaks up each command to perform and organizes them into panels, so that users can just move between panels to find a command they are looking for, rather than memorizing every command and performing them at the TSO command line. Another way of looking at it is like this: every command or task executed within the ISPF menu could theoretically also be done by entering the command directly into the TSO command line. However, once you consider the fact that most mainframe developers, analysts, or testers likely do not want to memorize every command they will need to use, it can be easy to see the usefulness of the ISPF user interface.

What is Zowe?

Zowe is an open-sourced framework developed by the Open Mainframe Project that, similar to TSO/ISPF, allows users to connect and interact directly with z/OS systems. It was designed in mind of being a more user-friendly interface, specifically to try and attract younger programmers to mainframe technology, especially considering the reputation of the “Green Screen”. In other words, this was made to be a tool to bridge generational gaps between various generations of developers in mainframe development. Also, it is commonly used as an extension through Visual Studio Code, which is a highly popular application used for development, which is another reason Zowe is becoming highly popular and is seen as a more modern mainframe development tool. Within Zowe, you can do many of the basic commands that you would do through TSO/ISPF, including allocating data sets and submitting jobs, and being a part of Visual Studio Code, it also provides a really user-friendly terminal for developing within data sets.

What can TSO/ISPF and Zowe Both do Well?

While TSO/ISPF and Zowe may handle processes in different ways, there are several purposes that both of these provide to make them great resources for handling. Those include:

- Managing datasets: This includes creating new datasets, editing/deleting datasets, copying datasets, and viewing dataset content
- Managing JCL: JCL is the tool that allows you to properly submit jobs to a mainframe and have it handle a program appropriately. This includes submitting jobs, viewing job status, and viewing the SYSOUT (job output).

- Executing TSO commands: Both platforms allow for running TSO commands through a command line. However, they are slightly different in this way, as Zowe can't do quite as much as ISPF can.
- Secure access to z/OS: Both platforms require you to log on to an account to access them. This ensures that each user will follow the correct security permissions assigned to them, regardless of what platform they are using.

These are just a couple of the most significant similarities between the two, and it is easy to see how they both provide easy access to core z/OS functions. And while they may have some core similarities, there are also some ways they are useful in different ways from each other.

How are TSO/ISPF and Zowe Different?

As mentioned before, both platforms are useful for connecting to and interacting with mainframes and their resources. However, they are also different in many ways. Those include:

- User Interface: TSO/ISPF uses what is known as the "Green Screen", also known as the 3270 terminal, whereas Zowe uses more modern interfaces, such as Visual Studio Code
- How z/OS is accessed: TSO/ISPF requires logging into a TSO session, whereas Zowe uses various APIs to communicate with the operating system, but not through a TSO session.
- Modern tools: TSO/ISPF is not designed in mind for modern tools, whereas Zowe is in many ways. One of these tools that is almost essential to modern

development among a team would be a tool like Git and Github, which can be easily accessed through Zowe, but not TSO/ISPF

- Learning Curve: Learning to use a TSO/ISPF terminal can be painfully difficult (speaking from experience here), as while it becomes easier to use it, the more comfortable you get with it, the learning curve is quite steep. Especially when in comparison to Zowe, which has a much less steep learning curve, and is often much easier for a new mainframer to understand.
- What are they most efficient for? Due to the modern tools mentioned earlier, Zowe is better for modern development. Also, I would argue the terminal is better to develop within, but that's a matter of preference. However, since Zowe can't quite do everything TSO/ISPF does, specifically, more in-depth tasks, as well as handling legacy systems, TSO/ISPF does get the edge in these areas.

Overall, these two platforms are both similar and different in various ways, but they both help developers work towards a common goal, which is developing good software. And while Zowe may be the go-to choice for younger, more modern developers, TSO/ISPF is also the go-to choice for many older, more experienced developers. It is easy to understand why both groups may be attracted to one over the other, as they both have their perks.

So, which platform is better?

While it would be easy to say that one is better than the other, the reality is that this question is deeper than a simple yes or no answer. This is because of the upsides of each, as well as the accessibility that each provides. Are you on a development team meant to work together more remotely? Well then, almost certainly Zowe would be a

better choice for you, because its upsides fit better for what you are doing. Are you working on legacy systems in legacy programs? Well then, TSO/ISPF would likely be better for you, once again, because of the upsides of the TSO/ISPF platform, which would make that the easy choice for you.

However, while certain scenarios would make one platform appear advantageous over the other, ultimately, a lot of the answer to the question “Which platform is better?” comes down to personal preference among developers. If you ask most experienced developers, who have been using legacy systems using the green screen for the last 30-40 years, they will tell you how amazing the green screen is, and would much prefer that, but if you ask most younger developers, they will tell you the exact opposite, about how amazing Zowe is and how amazing it is to use compared to the green screen. So overall, an important part of becoming a mainframer is learning how to navigate and use a green screen, as well as becoming comfortable with Zowe, as in the modern world of mainframe, both platforms serve their purposes, and understanding how both work and being able to use both gives you extra flexibility as a developer.

Getting Started with z/OS Data Sets

Now that you have a base understanding of TSO/ISPF and Zowe, the next important thing to discuss is z/OS data sets, as being able to create/edit/delete/view these data sets is one of the most crucial aspects of using either platform. This section will discuss the various types of data sets within the z/OS setting and what they are used for. I will start with the most common types of data sets and go more in-depth with those, but will also mention others that exist as well.

Most Commonly Used Types of Data Sets

The two types of data sets that we will mostly focus on are the two that are commonly used by new mainframe developers, those being sequential and partitioned data sets.

Sequential Data Sets

Sequential data sets are useful in the case of saving data as individual records, and the data is stored one record after another, similar to a flat file. Whenever data is intended to be used for a report or line-by-line processing, storing that data in a sequential data set would be the most effective, and for that reason, these data sets will frequently be used as input/output files for batch COBOL and HLASM programs. Also, another key point with sequential data sets is that the records can only be accessed from the beginning to the end, meaning that they are not useful in cases where you need to access only a certain subset of the records, unless you have logic in your program that filters only the data records that you need.

Partitioned Data Sets

There are 2 types of partitioned data sets that you will see. These are partitioned data sets (PDS) and partitioned data sets extended (PDSE). Both of these refer to a similar concept, however, which is that these data sets represent something closer to a folder with files of data within it, rather than a single file, as is seen with sequential data sets. Within a PDS or a PDSE, each data set contains what are known as “members”, which will typically contain COBOL code, HLASM code, and/or JCL (at least in the context of some mainframe development). The key difference between PDSs and PDSEs has to do with how each types manage storage. With a PDS, members are stored in a fixed location (static storage), and with a PDSE, members are stored using dynamic space

reuse, meaning they are not stored in a fixed location (dynamic storage). This means that with a PDS, if a member is deleted, the space is not reclaimed until a utility is run to clear the space up, whereas with a PDSE, space will not require manual cleanup. Also, PDSEs include a nice feature, called member versioning, which allows storage for multiple versions of a member, which can be useful if you want to update a member, but keep the original version as well to reference later, and for these reasons, PDSEs are considered to be more modern and more effective than PDSs. PDSEs are most effectively used as libraries for source code, such as for COBOL source code, JCL source code, and/or HLASM source code.

Other Types of Data Sets

Another type of commonly used data set type is temporary data sets. These are never meant to be allocated, but rather used as a placeholder, specifically within JCL code. It is created with the intent of being short-term, which will likely mean during the duration of a single job that it's used within. They are automatically deleted after they are done being used, and can simply be used within JCL by putting in place of a data set name. They will always start with two ampersands, followed by the name of the temporary data set (example: &&TEMP1). These are mostly used between job steps of a job, for reasons like sorting data or filtering data before processing it.

The next type of data set to discuss is virtual storage access method (VSAM) data sets. There are several types of VSAM data sets, and they are more complicated than the previously discussed data sets, so I will briefly summarize them all.

- Key-Sequenced Data Set (KSDS): Records are stored in order by a key, and you can access any record directly by its key. The records are automatically indexed, which helps in keeping the data sorted. Essentially serves the purpose of a small database.
- Entry-Sequenced Data Set (ESDS): Records are stored in the order that they are added to the data set, and they can only be accessed sequentially, or by relative byte address. Records are not indexed, meaning they are not easily updated or deleted either. These are good for situations where data is meant to be it once, and read many times after.
- Relative Record Data Set (RRDS): Records are stored in fixed positions and accessed by a relative record number, meaning that any record can be referred to pretty easily based on its relative record number. These are good for situations when every record is the same size, and the order of the records matters.
- Linear Data Set (LDS): A stream of bytes with no record structure, unlike the other types of VSAM data sets. Meant to be used for a specific purpose, as random access can be done using relative byte addressing, which is to be determined by the specific program that will be accessing this data. These are meant to be used as the building blocks for databases.

Useful Related Links

- Zowe documentation: https://docs.zowe.org/
- IBM TSO/ISPF documentation: https://www.ibm.com/docs/en/SSLTBW_2.4.0/pdf/f54ug00_v2r4.pdf

- IBM z/OS data set documentation:

  https://www.ibm.com/docs/en/zos-basic-skills?topic=more-what-is-data-set