Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Diplomant:	Michal Vašák
Vedoucí diplomové práce:	Ing. Rudolf Pecinovský, CSc.
Recenzent	Ing. Miloš Forman

Preparation of a hands-on tutorial teaching basics of Mainframe ISPF programming

školní rok 2010/2011

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval(a) samostatně a že jsem uvedl(a) všechny použité prameny a literaturu, ze kterých jsem čerpal(a).

V Praze dne

…….

Podpis

Abstract

English

This paper is primarily focused on developers wanting to learn the essentials of developing mainframe applications on the IBM`s Interactive System Productivity Facility (ISPF). It describes the components of the ISPF environment, focusing mainly on the two components most used in mainframe application development: The ISPF Dialog Manager and the ISPF Program Development Facility (PDF).

Second part of the paper is a hands-on tutorial, where the reader is guided through the creation of a set of simple interactive ISPF applications, with focus on covering the most used features of the platform. Within the paper it is assumed that any reader interested in this topic has basic mainframe skills and knowledge, including user knowledge of ISPF PDF, as it is usually the primary tool for dataset manipulation and editing in the z/OS mainframe environment. A wider skill set is assumed for readers who want to go through the tutorial, for example REXX and HLASM assembler basics. However none of these skills are necessary, nor expected, from those who are mainly interested in finding out what ISPF is and what it can offer to z/OS applications and their developers.

Keywords: Mainframe, z/OS, ISPF, Application, Development

Czech

Tato práce je primárně zaměřena na vývojáře, kteří se chtějí naučit základy vývoje mainframe aplikací pro Interactive System Productivity Facility (ISPF) od IBM. Práce popisuje jednotlivé komponenty prostředí ISPF, se zaměřením především na dvě nejvíce používané při vývoji mainframe aplikací: ISPF Dialog Manager a ISPF Program Development Facility (PDF).

Druhá část práce je ve formě tutoriálu, návodu, ve kterém je čtenář veden krok za krokem při vytváření sady jednoduchých interaktivních ISPF aplikací, se zaměřením na pokrytí většiny běžně používaných funkcí této platformy. V této práci se předpokládá, že čtenář se zájmem o toto téma ovládá alespoň základy práce s mainframem, včetně uživatelské znalosti ISPF PDF, které je obvykle primárním nástrojem pro manipulaci se soubory v z/OS a a pro jejich úpravy. U čtenářů, kteří chtějí projít tutoriál, se předpokládá větší šíře znalostí, například základy REXXu a HLASM assembleru. Nicméně tyto nejsou nutné, ani očekávané, od čtenářů, kteří chtějí především zjistit, co to je ISPF a co může nabídnout z/OS aplikacím a jejich vývojářů.

Klíčová slova: Mainframe, z/OS, ISPF, Vývoj, Aplikace

Table of Contents

1 Introduction

1.1 Aim and scope of this paper

The main goal this paper sets to accomplish is to serve as a source of information and a learning material for mainframe developers, who wish to learn about developing mainframe applications that make use of ISPF panel-driven UI (dialogs) and the wide array of ISPF application services. This paper aims to provide a hands-on tutorial, preceded by a concise overview of the ISPF platform and explain what individual parts of ISPF do and what are the services they can offer to the developed application as well as to the developer during the development process. It is not my goal to replace the ISPF product documentation provided by IBM, but rather to give a quick overview with enough background information to enable the reader to understand the things shown in the tutorial, which makes up the main part of this paper. In this tutorial, concrete usage of the ISPF environment will be demonstrated on a series of 3 increasingly complex sample applications. Starting with the obligatory "Hello, World!" application, and ending with a KSDS VSAM display tool, these sample applications should cover a wide range of the services ISPF can offer to user applications, and show the reader the capabilities of ISPF and how these can be leveraged from within applications written in REXX (REstructured eXtended eXecutor) or mainframe Assembler. As the range of services offered by ISPF is fairly large, it will not be possible to cover them all in this tutorial. Nor would that be desirable, as there are detailed reference documents available from IBM and duplicating them would not bring any added value. Therefore this tutorial will often point to additional documentation for details about syntax and operation of individual commands, instead of focusing on making the user aware of the services available and demonstrating their use, while leaving it up to the reader to read the corresponding documentation when implementing these services in their applications.

As ISPF is, from application point of view, essentially a set of services called from the programming language of the application, I needed to choose a programming language to be used in the sample applications. Two languages were selected, REXX and z/OS assembler. The first two samples are written with REXX. It was chosen because it is a scripting language that is easy to code and easy to read, with support for imbedding commands from various command interfaces (ISPF, TSO) making it very useful for handling the logic behind a panel-based user interface. REXX code should be fairly easy to understand even for programmers who have not used it before, as it has been designed to be easily readable and to sound like spoken English. However, as REXX is not able to perform some system-dependent actions, and system tools are applications commonly created using ISPF, the third sample application will show the

reader how ISPF services can be utilized from mainframe Assembler, and how ISPF dialogs and ISPF services can tie together the different load modules and REXX scripts to form a single application.

1.2 Intended audience

The ISPF overview part of this paper can be useful for anyone interested in gaining a general understanding of what ISPF is and what it is for. No special skills or knowledge are expected or required, and although the overview is primarily aimed at developers and does contain technical information, the first part of this paper can be enjoyed even by people lacking the interest or knowledge needed for the tutorial part. However, not having at least some general experience with mainframes and the z/OS operating system can make it harder to understand some of the basic concepts mentioned, especially for Windows users used to the graphical environment of modern PCs. In such a case, visit the Mainframe and z/OS primer chapter for the main differences between Windows/Unix PCs and a z/OS mainframe.

The tutorial part presents some of the essential techniques for building interactive mainframe applications using the combination of ISPF and REXX for presentation and basic functions and mainframe Assembler for the more specialized tasks. It is created with participants in the CA mainframe training in mind, but can be enjoyed by anyone who knows the basics of Assembler and REXX and wants to learn how to combine these with ISPF to create applications using the interactive panel interface.

The target of this tutorial is to guide the readers trough creation of an ISPF mainframe application, teaching them how to build ISPF applications. Skills that the reader should gain include:

- How to call ISPF services from REXX and from Assembler
- How to pass control between ISPF, REXX and Assembler load modules
- How to create ISPF panels
- How to use ISPF variables and tables
- How to use ISPF messages
- …as well as how to tie these all together.

This tutorial presumes that the reader already has the following knowledge:

- User-level knowledge of mainframes and z/OS, like dataset listing, allocation and editing, submitting jobs and reading job output, and understanding basic concepts.
- At least basic knowledge of the REXX programming language. REXX will be used in this tutorial mostly for scripting simple operations and calling ISPF services and specialized Assembler

modules, therefore very little experience is needed. However, the basics of REXX are not explained in this tutorial and are presumed to be known by the reader.

- At least basic knowledge and experience with mainframe Assembler. The reader should know how to write, compile and link simple assembler programs following the linking conventions, to understand assembler addressing and concepts like DSECTs, dynamic memory allocation or dataset IO using DCBs.

Knowledge of ISPF services is not necessary. It is expected that the reader knows how an ISPF interface looks like, and that they have used the basic ISPF panels like 3.4, but building panels and using ISPF services from program code will be explained within the course of this tutorial and no previous knowledge should be necessary.

Related work

Other guides and tutorials to ISPF do exist on the internet, but these are in vast majority focused on using the Program Development Facility and the ISPF editor. As the PDF user interface is used by most mainframers for their daily work, the audience for guides focusing on the user perspective is relatively large and it is easy to find the information required. However, when looking for a good learning material for future ISPF application developers, I have found that the only freely available good source of information are the IBM guides – unfortunately to get the full picture as a developer interested in developing for the ISPF platform, you need to combine information from at least 7 guides ([4][5][6] [7][9][10][11]).

As for a hands-on tutorial for ISPF developers, IBM created one under the name Interactive System Productivity Facility Getting Started. It is actually not easily located, and the last revision I could find is from 1995, but it does contain a walkthrough trough building a sample ISPF application. Apart from this tutorial being 16 years old, the reasons why I believe a new tutorial will be beneficial to ISPF developers is that it will focus a lot more on using the ISPF as a framework, that can glue together parts of your application, and on combining it with common mainframe programming languages, whereas the older tutorial is mostly focused on building a UI.

1.3 Information sources used in this paper

The information sources for this paper were gathered as follows: two main sources of theoretical information have been the official IBM documentation and publications regarding ISPF dialog development and related subjects, and a web search in the IBM mainframe developer community for other information sources used by ISPF developers. A third source, so to say, has been my own experience with

developing ISPF applications. Since I am in no way an expert in this field, I have used my own experience only to help me understand the technical documentation and guides and, unless explicitly stated, I have not used my own findings as a source in the theoretical part of this work. The tutorial part, on the other hand, is based on a program I have written myself, with the intention of demonstrating some of the functionality of ISPF that I have considered to be the most useful, both based on the documentation I have read and on my own information needs when I was learning to develop applications for ISPF.

1.3.1 IBM documentation search

The reason why I chose to use IBM`s publications as the main information source was that the ISPF, as well as the whole z/OS environment it operates in, has been developed by IBM and the official documentation provides the deepest and most reliable source of technical information available. It is also freely available, making it easy to obtain and allowing referencing documentation sections where the reader can find more detailed information on topics that do not fit within the scope of this paper. The main problem with the documentation provided by IBM is that while it surely contains a lot of accurate technical information, it is very extensive and it can be rather hard to piece together the information you need.

1.3.2 Mainframe developer community search

My reason for choosing this as the second direction in which to search for information sources was a simple assumption – many programmers before me had needed to find these sources, and had spent time trying to find them. Therefore it will be more efficient to search for the results of their searches than to just repeat what they did. To do this, I have searched in the community of mainframe developers – on the forums and blog sites dedicated to mainframe application development and mainframe technology in general, like www.felgall.com, www.tsotimes.com, IBM`s www.share.org, ibmmainframes.com and other similar sites I have met during my work or found using Google.

Concerning the reliability, while such sources are generally considered highly unreliable, in case of my research, I was not looking for actual information about ISPF, but rather for references to sources of such information. The plan was to sift through the findings, and filter out the more reliable sources, such as articles from authors with good credentials, or papers accepted to academic or professional conferences and journals.

1.3.3 Search results

While the IBM-published documentation search yielded the expected results, searching the community proved to be nearly useless. What I have found was that whenever sources were cited in the web content found, they would almost always lead either directly or through another source back to the IBM

documentation. Indeed, a common suggestion found on mainframe forums in an answer to various technical questions was a link to the relevant IBM documentation. (By the way, this is not as unhelpful as it sounds, as finding the correct place to look for something in the official publications can be difficult) Even though I have not gained new sources from the community search, it was not completely useless, as it helped me notice some IBM documents that I could have missed otherwise (possibly [1] and almost surely [12]) and to confirm their relevance. I have also acquired some lesser-known printed sources from my colleagues at work.

2 Mainframe and z/OS introduction

For those who are not familiar with the mainframe environment, the following chapter should help them to understand how the mainframe differs from PCs and what the context in which ISPF applications operate is. Those who already have experience using and/or developing for the mainframes can feel free to skip this chapter and continue with the next one.

2.1 What is a mainframe

A mainframe is a computing system that businesses use to host the commercial databases, transaction servers, and applications that require a greater degree of security and availability than is found on smaller-scale machines. It is accessed indirectly through terminals, dedicated devices used for user input/output. (Today, terminal emulator programs on PCs or other platforms are used.) The power of a mainframe provides computing speed and capacity, enabling it to perform very high volumes of processing. It provides unparalleled levels of reliability and data throughput and can handle very large numbers of transactions and concurrent users. It provides built-in virtualization and workload management, and can also achieve the highest single-thread processing speed among commercially used computers, making it very efficient at all tasks that require exclusive access to a resource. While usually not visible to the public, present day mainframes are found at the center of information systems of large government agencies, banks, insurance companies, airports and other enterprises that need to process large amounts of data and/or serve large numbers of concurrent users with the highest possible reliability and availability.

The history of mainframe computing dates as far back as the 1950s. The first commercially used computers of that time had many characteristics of a mainframe. These computers were very expensive, so they have been owned only by large companies or dedicated data centers. These have then sold time on their computers to other companies. At first these computers were not standardized, and were built for specific function, as either scientific or commercial computers. First general purpose computer was the IBM System/360™ (or S/360™) introduced to the public in 1964. Since then, mainframes have been continuously improving in functionality, reliability and processing power, while keeping a high degree of stability and backwards compatibility. In fact, some of the code written for S/360 is still in use on today's mainframes. Over the time, other types of computing have arrived. The minicomputers were a smaller, more affordable computing platform for smaller businesses, sharing many of the mainframe characteristics. After these came the even smaller microcomputers, among them the PC (this is what the micro- in Microsoft stands for). The mainframe platform has not been replaced by these, but continued to evolve, creating a place alongside the newer forms of computing. Today, mostly for marketing reasons,

the only manufacturer of mainframes, IBM, now refers to its latest mainframes as zEnterprise servers [1][17].

Figure 1 IBM employees James Geuke (top) and Larry Terpak (standing) install covers on the new IBM zEnterprise System mainframe. (Photo: Feature Photo Service for IBM)

3 Mainframe operating systems

The operating systems available for the current IBM mainframes are: z/OS – the primary operating system of interest in this paper, z/VM, z/VSE, z/TPF and the mainframe Linux.

3.1 z/OS operating system

z/OS is the most widespread operating system on IBM mainframes. It has evolved from the first mainframe operating system, and maintains a high level of compatibility with its predecessor mainframe operating systems like MVS. The first versions of the system were batch-oriented, and submitted batch jobs (programs that are started by the user, and continue their execution in the background without user interaction) are still prominent in z/OS. The interaction with a mainframe user happens through a text terminal or a terminal emulator program on a PC, and is limited to text only. The most basic form of operation, accessing z/OS using the TSO Ready prompt (discussed in chapter 5.1.4 TSO) is similar to using UNIX OS from the command line. However, there are important differences between the two in how the systems operate.

3.1.1 Z/OS Datasets

Data in z/OS is stored in datasets instead of files, with main differences being that datasets are usually composed of records, while UNIX files are a binary stream. A record is a segment of the file, containing data. Most often, individual records represent some data entities, much like rows in a database. Parts of these records that have a distinct meaning (like street name in a record representing an address) are called fields. Of course a binary file can contain a similar structure, the difference here is that the record structure of z/OS datasets is visible to the operating system, and multiple access methods exist in z/OS, optimized for accessing different dataset formats. The records also serve as the smallest unit of processing, so dataset read, write or update operations deal at minimum with single records. Records are usually organized in Blocks – larger chunks of records that are individually addressed for faster access. Selecting the right record and block size depending on the type and usage of the data has significant impact on application performance. All this leads to one immediately noticeable difference from Windows or UNIX - you need to specify the record types (fixed or variable-length) and size whenever allocating a new dataset. Creating datasets without the record structure is also possible, but the majority of datasets in z/OS use records.

Allocating datasets in itself is also a distinct feature of z/OS. On Windows or UNIX systems, when creating a new file you do not have to know anything about its future usage. New files are created as empty binary files, and all you need to define them is their new name and folder. Windows systems use

file extensions to distinguish file formats, but on an empty file this is for information only – you can change the extension at any time and the operating system itself lets you save any data into any extension.

In z/OS creating (called allocating) a new dataset works differently. When you are allocating a new dataset, you need to specify 2 sizes –primary and secondary. Primary space is immediately given (allocated) to the dataset, so even while empty, the dataset occupies at least this much space on the volume and no other dataset can use this space. The secondary size is used for extends – when the allocated space is used up, z/OS will extend the dataset by a continuous block of space with the secondary size. There are different types of dataset types available in z/OS, some of which may not use the record and/or block structure at all. The most important are:

- Sequential datasets
- Partitioned datasets
- VSAM datasets

Sequential datasets are the oldest of the three. They are easy to read and edit by humans, thanks to being editable by the ISPF editor. When storage and retrieval of larger data volumes is needed, they are outperformed by the VSAMs. That makes them useful for configuration files, program source code, human readable logs and similar.

Partitioned datasets (PDS) are special datasets, that contain member datasets and a directory listing that allows direct access to the individual members. They are similar to a folder on UNIX or Windows, with the difference that they cannot be nested to create a hierarchy. They are used for storing load modules of programs (this is required by the OS) and as libraries for program source code and jobcards. (see JCL in chapter 5) An extended format of PDS exists – the PDSE – that provides some additional capabilities.

VSAM datasets are a group of dataset types optimized for speed of access. Unlike sequential datasets, they require specialized tools to edit, and are typically written and read by applications rather than humans. Different types of VSAM datasets exist, depending on the function they are meant to perform. As an example, KSDS datasets have a data part and an index part and associate their records with indexed keys, allowing efficient random access to a record anywhere within the dataset based on the key. VSAM datasets are used for holding and processing business data, implementing databases, holding the hierarchical file systems of the USS (see UNIX System Services in 3.2.4) and generally in situations where larger amounts of data are processed.

3.1.2 z/OS File Hierarchy

z/OS does not use a hierarchy of folders like Windows or UNIX systems. Dataset organization in z/OS is actually quite a lot different. The dataset names are composed of a number of 8-letter *qualifiers* separated

by commas. The first qualifier is called the *high level qualifier* (HLQ) and is usually determined by the security system, and the last one is called the *lowest-level qualifier* (LLQ). A dataset name can consist of up to 44 characters, including the commas. By custom, qualifiers are used to group similar datasets together, for example by giving datasets belonging to the same application the same beginning qualifiers. Many applications allow datasets to be filtered and manipulated based on their qualifiers (for example listing only datasets starting with given qualifiers, or setting access rules for the qualifiers) so the qualifiers are somewhat mimicking the folders of other operating systems. However z/OS does have its own variant of folders, the PDS libraries mentioned earlier. These are special datasets that contain member datasets. Their members share the attributes defined to the PDS (record and block sizes etc.) and can serve special purposes, for example binaries of executable programs have to be stored in specially formatted libraries to be useable by z/OS. The libraries cannot be nested (cannot contain other libraries) so unlike folders in Windows or UNIX they form only a 2-level hierarchy – first level being sequential and VSAM datasets (and other types not discussed here) and second level being members in libraries.

To make things a bit more complicated, datasets can either be catalogued or uncatalogued. When z/OS looks for a dataset by name, it refers to its system of catalog data to find which volume the dataset resides on. However, it is also possible to have uncatalogued datasets that can only be found by giving z/OS both their name and the volume they reside on. While the name of catalogued datasets has to be unique within a system, for uncatalogued datasets it only needs to be unique on the volume it resides on. Usually, most datasets are catalogued and their DSN (Data Set Name) is all that is needed to locate them.

3.1.3 TSO

TSO (Time-Sharing Option) is a subsystem of z/OS that provides a single-user logon capability to the z/OS by managing interactive user sessions. It also provides a limited set of commands to the users to allow them to interact with the z/OS. The full, correct name of the subsystem is TSO/E (TSO/Extensions) but it is generally referred to as simply TSO [1].

Upon connecting to it, TSO displays a Logon panel, where the user needs to provide his TSO login name and password. If correct, TSO starts an interactive session for the user in the system, and displays the so-called Ready prompt – an empty screen with the word Ready indicating the system is waiting for user input. Here the user can type TSO commands to interact with the z/OS in much the same way as a Windows or a UNIX user working from the command prompt.

Using z/OS from the Ready prompt is called using TSO in native mode. The native TSO is rather cumbersome and offers only limited functionality, and it is seldom used. This is thanks to the ISPF, Interactive System Productivity Facility, on which this paper is focusing. ISPF adds a CUI, a character-based user interface to the z/OS, and a number of applications like the ISPF editor and ISPF dataset

listing, that let the user perform actions a lot easier and without having to memorize and type as many commands as would be required in a command-prompt style of work. (ISPF does contain a command input line and typed commands still play a major role though) You will learn much more about ISPF in the following chapter About ISPF.

3.1.4 3270 terminal

One of the things that make ISPF panel-driven interaction possible is the 3270 protocol. When a user wants to access the z/OS mainframe, they need to logon to the TSO system by connecting to it from a "3270 display device" – that means from a terminal device supporting the 3270 protocol or, much more commonly, from a terminal 3270 emulation program. The 3270 is the latest version (only 40 years old!) of the protocol used for communication between the terminal (or today the terminal emulator) and the mainframe. It is based on text panels, sending panels of text with marked fields where the user can type. When the user types some input and presses Enter, one of 24 Function keys or other keys recognized by the protocol, the text of the changed fields is sent back to the mainframe for processing. Since the first terminals used to have screens with green background, the 3270 terminals and emulators are commonly referred to as "the green screen".

3.2 Other mainframe operating systems

3.2.1 z/VM

The z/VM is the latest, 64-bit version of the VM operating system. VM is an operating system primarily focused on providing a virtualization layer for running multiple instances of other operating systems. VM stands for Virtual Machine, a virtual environment that simulates a real hardware machine, complete with CPU, memory and I/O devices. For programs running inside a virtual machine, it is impossible to tell that they are inside a VM. z/VM is usually used for simulating large numbers of Linux servers or combining Linux and z/OS, z/VSE or z/TPF workloads on the same machine and enables seamless transition of virtualized servers between physical machines without stopping them. [19]

3.2.2 z/TPF

z/TPF is a specialized mainframe operating system focused on high-volume transaction processing. It can manage extreme transaction volumes - z/TPF can process tens of thousands of transactions per second from hundreds of thousands of end users. To achieve this, it lacks the batch processing typical of z/OS and provides direct support for transaction processing on the OS level. (while z/OS has to rely on transaction processing systems like CICS or IMS) z/TPF is used in airline reservation systems, internet banking and other large transaction-based information systems.[17]

3.2.3 z/VSE

Another mainframe operating system from IBM, developed alongside the more popular z/OS, is VSE (Virtual Storage Extended). It originated from DOS, a temporary operating system used on the System/360 mainframes while OS/360 (from which z/OS is an indirect descendant) was in development. Some customers preferred the simpler and smaller DOS though, and IBM decided to continue developing it in parallel to its other operating systems. Although it is less widespread that z/OS, it is still fully supported and further developed by IBM and has a significant customer base. z/VSE is the latest version of VSE, running on the 64-bit z/Architecture mainframes.[17]

3.2.4 UNIX System Services

The USS (Unix System Services) is a version of the Linux operating system for the mainframe. Created to satisfy customer demand to process Linux-based workloads, it implements a shell conforming to the POSIX standard, with standard UNIX commands and file structure. The USS shell is based on the UNIX System V shell, with some features from the UNIX Korn shell. It has its own standard hierarchical UNIX file system, complete with user home directories and the usual system directory structure. The mainframe stores this whole file system in special HFS datasets, or their newer variant, the zFS (both being a type of VSAM). Each HFS contains a part of the UNIX file system, and can be mounted into directories in a root HFS. For the UNIX shell user this is transparent and they see a standard hierarchical UNIX file system containing binary files and supporting standard UNIX commands and utilities. However, this file system is not completely isolated from the rest of the mainframe, as there are commands available to interact with z/OS or z/VSE, and you can copy files between the operating systems, start jobs in the job subsystem, and communicate across the OS boundary. One important difference from other Linux operating systems is that security can be handled by outside security managers, like RACF or TopSecret running in z/OS, so you only have to deal with one set of user accounts and rights across both operating systems.

The USS is very useful for running a Java Virtual Machine and therefore Java applications, as well as many standard web and application servers like IBM`s Websphere or Apache`s Tomcat and other programs that need the hierarchical file system to operate. However in this paper the UNIX side of z/OS is of little interest to us, as ISPF does not rely on this subsystem in any way. If you want to learn more about z/OS UNIX, see the *UNIX System Services User's Guide [18]* for your z/OS version.

4 About ISPF

4.1 Basic overview

ISPF stands for Interactive System Productivity Facility. According to the ISPF User Guide ISPF is *"a multifaceted development tool set for the z/OS operating system"* [10, page 1]. Probably all mainframe users (aka. mainframers) working on the z/OS green screen today know the ISPF Program Development Facility, often simply called ISPF, as the interface they use for their day-to-day mainframe work. Most of them actually do their daily work almost exclusively in ISPF PDF and other ISPF applications. In reality, the programs that are provided with ISPF are just the visible tip of the ISPF iceberg. ISPF is also a complex and powerful platform on which these applications run. From the developer`s point of view, ISPF provides an environment and tools for developing and running interactive MVS and z/OS applications called Dialogs, making use of the ISPF CUI (Character-based User Interface) and ISPF services, that simplify application access to various z/OS resources and provide a lot of useful functions that programmers can use without having to code these themselves [10].

```
VASMI03 ISREDDE4 3.TEST.TUT.EXEC(TUTVSAM) - 01.64                Columns 00001 00080
Command ===> _____________________________________________ Scroll ===> CSR
****** ******************************** Top of Data ********************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 /******************************** REXX ***************************/
000002 address ISPEXEC
000003 "CONTROL ERRORS RETURN"
000004 "TBCREATE VSAMTAB KEYS(VKEY) NAMES(VREC) NOWRITE REPLACE"
000005 "DISPLAY PANEL(VSAM1)"
000006 RESTART:
000007 IF RC>0 THEN EXIT
000008 address TSO "ALLOC F(INVSAM) SHR REUSE DA('"INDSN"')"
000009 if RC>0 then signal IOError
000010 /*READ VSAM AND FILL IN THE VSAMTAB AND OTHER VARS USING ASM PGM*/
000011 VOFFSET = 0
000012 "VPUT VOFFSET PROFILE"
000013 "SELECT PGM(VSAMASM)"
000014 if RC>0 then signal IOError
000015 /**/
000016 SCR = "PAGE"
000017 "TBSORT VSAMTAB FIELDS(VKEY)"
000018 "TBVCLEAR VSAMTAB"
000019 "TBTOP VSAMTAB"         /*POINTER BACK TO START OF TABLE*/
000020 "VGET VLRECL"
000021 VCOLINFO = 'Column 'VOFFSET' to 'VOFFSET+80' of 'VLRECL
000022 "TBDISPL VSAMTAB PANEL(VSAM2)"
000023 AFTERDISPLAY:
000024 if ZCMD = 'PREV' then do
000025   VOFFSET=VOFFSET-80
000026   if VOFFSET < 0 then VOFFSET = 0
000027   signal REREAD
000028 end
000029 if ZCMD = 'NEXT' then do
000030   VOFFSET=VOFFSET+80
000031   if VOFFSET > VLRECL - 80 then VOFFSET = VLRECL - 80
000032   SIGNAL REREAD
000033 end
000034 exit
000035 REREAD:
000036 VCOLINFO = 'Column 'VOFFSET' to 'VOFFSET+80' of 'VLRECL
000037 "VPUT VOFFSET PROFILE"
000038 "SELECT PGM(VSAMASM)"
000039 if RC>0 then signal IOError
000040 "TBTOP VSAMTAB"         /*POINTER BACK TO START OF TABLE*/
000041 "TBDISPL VSAMTAB"
000042 signal AFTERDISPLAY
000043 IOERROR:
000044 "DISPLAY PANEL(VSAM1) MSG(TUT004)"
000045 signal restart
****** ******************************** Bottom of Data ********************************
```

Figure 2 Example of the ISPF CUI – the ISPF Editor (courtesy of CA Technologies)

Main components of ISPF

4.1.1 ISPF Dialog Manager (DM)

Dialog Manager is the "invisible" part of ISPF, providing callable services to applications. The name refers to ISPF dialogs, a term describing interactions with the user as described in chapter 4.2, and has nothing to do with dialogs as graphical popup windows known from Windows OS. Among the services offered by the dialog manager are ISPF-managed variable pools, creation and display of ISPF panels for interaction with the user, all sorts of operations with dataset, including I/O, management and file-tailoring and access to many MVS system facilities. It is the goal of this tutorial to show you some of these services and how to make use of them to create an interactive ISPF application for the mainframe [10][12].

4.1.2 Program Development Facility

The PDF is the visible part of ISPF, a set of ISPF applications forming a single menu structure. The PDF provides a user interface in which the developer or user can create, edit, and manage datasets, execute scripts, JCLs and programs and provides other useful tools like dataset comparison, searching utilities etc. For most mainframe users this is where they do most (or all) of their work. The user interface of PDF consists of ISPF panels just like any ISPF application, and the panels may be modified to include any other ISPF-based applications outside the PDF. It is expected in the tutorial part of this paper that the reader is already familiar with using the PDF and capable of performing basic actions like dataset creation and editing [10][1].

4.1.3 Software Configuration and Library Manager (SCLM)

The SCLM provides ISPFs version of source control system, allowing versioning of libraries, promotion of changes into development stages, managing configurations and build routines etc. This part of ISPF is not within the scope of this work [10].

4.1.4 Client/Server component:

This component allows ISPF to connect to workstations running other operating systems and display the ISPF panels in their native GUI environment. This part of ISPF is also out of the scope of this paper. Also, applications that work under Dialog Manager should be able to use this component without any modifications, so there is actually no need to know this component in order to write applications for it [10].

4.2 ISPF Dialogs

According to IBM`s ISPF Dialog Developer`s Guide and Reference, *A dialog is the interaction between a person and a computer* [4]. That means a dialog can be considered as a transaction or a series of transactions between the application and its user. This interaction is facilitated by ISPF through its Dialog Manager module. Each dialog is composed of a command procedure or a program, controlling the flow of the dialog, and so-called *dialog elements* [4], facilitating the communication. The dialog elements in ISPF, according to the ISPF User Guide [10], are:

- Functions
- Variables
- Command tables
- Panel definitions
- Message definitions
- File-tailoring skeletons
- Tables

The following descriptions are a compilation from IBM`s documentation, mostly ISPF User Guide [10][11] and ISPF Dialog Developer`s Guide and Reference [4].

4.2.1 Functions

In ISPF terminology, a function is a program or a command procedure doing the work the user has requested. We will discuss these in greater detail in a separate chapter: ISPF Functions.

4.2.2 Variables

ISPF uses variables to pass data between dialog elements, like functions, panels, tables and skeletons. Variables are manipulated by a set of ISPF commands, and maintained in 2-level variable pool, enabling variable sharing in the context of a single split-screen, or in the context of a user profile across multiple splits. A special set of variables, called Z-variables because they all begin with letter ‘Z’, is reserved for special uses.

Z Variables

Z variables are ISPF special purpose variables, all beginning with letter “Z”, that are defined by the system. These are used to return values from ISPF services, to provide additional information for panel processing, and for other input or output. Z variables can be read and modified like any other variable; the

only difference between them and other variables is that at least one ISPF service uses these as input or output. That means you should avoid using these variables for different purpose, as they may get overwritten or used as input by ISPF. You can find out if an ISPF service uses any Z variables in ISPF Service Guide [9].

4.2.3 Command Tables

Every time a user types anything into the ISPF command line, ISPF uses the command tables to determine how to process the input. If the first word matches one of the commands in the table, ISPF takes the action indicated in the table, using the rest of the command line as parameters, when parameters are expected. Three levels of command tables can be defined. The system command table is always defined and cannot be changed. It contains the default commands accepted by ISPF and is distributed as a part of the ISPF installation. The site command tables are custom command tables defined by system programmers or administrators for a given installation. Finally the application command tables can be defined by individual applications, and associated with their applid (application identifier). These command tables are used only while the associated application is running. The tables are consulted in following order: First the application command table, if it exists. Next, the active site command table(s), and last the system table. The first table that contains an entry matching the input being processed determines what action will be taken. If none of the tables has a matching entry, the input is passed to the application that displayed the panel trough z-variable `ZCMD`.

4.2.4 Panel Definitions

Panel definitions are another essential part of a dialog. They define the look and behavior of panels, that ISPF uses to present data and gather input from the user. They will be covered in greater detail in chapter ISPF Panel Definitions

4.2.5 Messages

Messages are used for displaying a short text to the user, like an error or warning, without leaving the currently active panel. The advantages of using ISPF messages over simply having a field on each panel dedicated to displaying such texts are:

- No need to modify your panels to be able to display them.
- Saving Panel space, as messages display over part of panel normally used for something else.
- They can be displayed on a panel, directed into a log, or both.

- Easy to modify – as they are referred to by ID, changing the message text is easy even when used in multiple panels and functions.
- ISPF messages have a short form, displayed as overlay, and a longer explanation, displayed as popup when the user presses F1 on a panel that is currently displaying a message.

4.2.6 File-Tailoring Skeletons

Often referred to as simply skeletons, they are sequential dataset templates that can be used to quickly generate a sequential output. They contain pieces of text that will be written "as is", dynamic fields that will be filled in by ISPF variables, control statements allowing some sections to be repeated or included only if conditions are met, and data record sections, that are used to format and include records from an ISPF table. Skeletons are typically used for dynamic generation and submitting of JCLs and for generating reports.

4.2.7 Tables

ISPF Tables are two-dimensional arrays used for storing data in a way similar to a table in a relational database. They have a header defining one or more keys and zero or more other columns, followed by a set of records. The key columns uniquely identify a record (row) in a table, and adding a new record with the same key column(s) overwrites the previous record. ISPF Tables are deeply integrated in the ISPF Dialog Manager. They can be displayed in an ISPF panel through the)MODEL section (as will be described in ISPF Panel Definitions) or through a file-tailoring skeleton and there are services available for searching, ordering and filtering the tables. ISPF Tables reside in an applications address space, and they can be either temporary (using the NOWRITE attribute) and exist only while the application is running or they can be permanent, meaning they are saved into and loaded from datasets. Tables can also be used exclusively by one application, or shared between multiple applications or instances of an application.

4.3 ISPF Functions

In ISPF terminology, a function is a program or a command procedure meant to do the work the user requested. It can be written in any programming language that lets you access the ISPF command interface. It is called by ISPF, and returns control to it once it finishes. Thanks to this, individual functions can be written in different programming languages, and can even contain a mix of command procedures (CLIST), interpreted programming language code (like REXX) and binary load modules (created by Assembler or compiled from higher languages like COBOL or PL/I). The Dialog Manager component of ISPF handles the calls between different functions, and takes care of transferring variables between them, where possible. One important thing to keep in mind though is that ISPF will only consider a program or

command procedure to be a new function if it is called using ISPF. This usually means using the SELECT command, either directly or indirectly. However, if for example a load module links to a different load module using the CALL macro directly in code, it is considered to be a part of a single function. This is because ISPF does not understand the code inside the function; it only prepares the environment and passes control to the function, and after the function finishes, ISPF returns control to the calling program.

ISPF offers its function programs a wide array of services, that can be called trough ISPF`s command processor (for example by setting it as default command processor by using the Address command in REXX, and passing a command to it), or through a direct assembler CALL to one of its 2 command processing sub-routines (ISPEXEC of ISPLINK). Both of these techniques will be covered in the tutorial. The services provided by ISPF allow for example various file manipulations, access to some of the operating system functions, sharing of variable pools or access to and usage of other dialog elements (tables, messages).

4.4 ISPF Panel Definitions

Panel definitions tell ISPF how to display a character-based user interface panel on the terminal emulator. The panel definition consists of sections that describe the panel itself, the input and output fields on it, pre- and post-processing of variables used in the panel, actions to take when an option is selected and more. Most sections do not have to be defined for every panel, however the order of the sections is fixed and needs to be adhered to even if some are missing.

4.4.1.1 The sections of a panel definition (in the order they must be defined)

)CCSID	Specifies the Coded Character Set Identifier (the character set used by the panel). Only code this section if you want to change the default. Rarely used.
)PANEL	You can use this section to specify a key list for this panel – a custom mapping of PF keys to commands, overriding the defaults. This will only work if the user allows PF keys overriding in ISPF options, so you should not rely on it.

)ATTR	This section defines special characters (or short strings) that will be used in the Body section to denote fields of special type and/or format. There are 3 default characters predefined (unless changed by using the DEFAULT keyword).
	`% = text (protected) field, high intensity`
	`+ = text (protected) field, low intensity`
	`_ = input (unprotected) field, high intensity`

)ACB	Defines a single choice in the Action Bar (menu bar on the topmost line of the panel), the choices in its associated dropdown, and actions taken for each of these choices. This section and the following two make up a block, that defines a single menu in the Action Bar; This block can be repeated multiple times to define multiple menus. If left out, the Action Bar is not used (saving one line of on-screen space).
)ACBINIT	Defines actions ISPF will perform before displaying the Action Bar.
)ACBPROC	Defines actions ISPF will perform after displaying the Action Bar.
)BODY	This is the "main" section, describing the contents of the panel. It is a template of the panel as it will be seen by the user, with characters from the Attribute section used to mark fields used for input and output (and variables linked to them) and fields with special formatting. This is the only section that has to be present on every panel.
)MODEL	This section is used on panels that display the contents of an ISPF table. It is a template for 1 table row, using a dedicated attribute character to denote fields filled with table column values. However, a model can also contain static text, a normal ISPF variable or an input field.
)AREA	Defines a scrollable area, with its contents described by the same syntax as is used in Body section; A single panel can contain multiple scrollable areas, and scrolling will be done in the one where the cursor is currently located.
)INIT	In this section, you can define processing ISPF will do before displaying the panel, like setting up variables.

)REINIT	This section is called when the same panel is displayed 2 or more times in a row instead of the Initialization section before the panel is re-displayed.
)PROC	This section contains actions ISPF will execute after the panel is done displaying. This can include checking user input, taking an action based on user choice, converting user input to another format etc.
)HELP	Defines help panels for individual parts of the panel; to define a general help panel for the whole panel, assign it to .HELP special variable in the Initialization section.
)PNTS	Defines actions for fields in the Body (or Area) section that are marked as point-and-shoot fields. These actions will execute when the user presses enter on the point-and-shoot field or double-clicks it. Rarely used.
)LIST	Specifies a list to build on the panel. It is used in conjunction with the attribute keywords DDLIST(name), LISTBOX(name), and COMBO(name) to build a list of list box and combo choices.
)END	This is not really a section, but just a tag marking the end of a panel definition. Compulsory.

4.4.2 Field types in a panel

Each field in the Body or Area section displays and behaves according to its properties: Its Type, governing the function of the field, and its other attributes, describing how the field will look. All of these are determined by the definition of the control character preceding it, defined in the Attribute section. A field will always start with a control character or line start (in which case Text without highlight is assumed) and end at the next control character or line end:

Field 1 (control1) Field 2 (control2) Field3

The length of a field is given by the distance between its start and end, and its position in the resulting panel is given by its position (row and column) in the panel definition. This brings some problems, as for example an input field definition has to contain an input variable name between the control characters, but you may want the actual field in the panel to be shorter than the variable name. This is solved by a workaround solution using the .ZVARS, which will be discussed in Special variables (Z variables).

4.4.2.1 The basic attribute types (defined by default)

NT – Normal Text	The text inside this field is displayed in the panel, letter-by-letter. This applies even to spaces, so NT fields without actual text are a common way to end a previous field (for example an input or output field).
INPUT - Input	Instead of text, this field has to contain a name of an ISPF variable (plus arbitrary number of spaces) and when the panel is displayed, this field will become a writeable input field initialized with the current content of the variable. After the panel closes, the variable will be changed to what was written into the field by the user.
OUTPUT - output	Same as input field, but does not allow modification by the user.

4.4.2.2 Some other useful field types:

AB – Action Bar unselected choice	Defines an action bar menu. Contains the displayed menu name. The menu contents are defined by an)ABC section and connected to the field by order – first)ABC section in a panel describing the first AB field, etc.
ABSL – Action Bar Separator Line	Action Bar separator line – divides the AB and the rest of the panel.
PS - Point-and-shoot	Defines a point-and-shoot field; Connected with a point-and-shoot field definition in the)PNTS section by order (first with first, etc.).
PT - Panel Title	Panel title – behaves like normal text, but its default color is based on ISPF`s Panel title color (which is by default different than normal text, and consistent across the panels in PDF and other ISPF applications that honor this color setting.

SAC – Selection Available Choice	Used in selection panels to highlight one of the choices the user can select. It has the advantage, that if SAC fields are defined, the user input is automatically verified by ISPF to be one of the SAC choices.

For a full list, see "Defining the attribute section" in chapter 7 in ISPF Dialog Developer`s Guide and Reference [4].

4.4.3 Special variables used in panel definitions

Several special variables are used by ISPF when interpreting panel definitions. Apart from Z-Variables (mentioned in Variables) used for special kinds of input/output fields, for example for Command Line (variable `ZCMD`), Scroll amount (`ZSCR`) or point-and-shoot fields (`ZPCnnnnn`) a second type of special variables exists in panel definitions: these variables begin with a dot (".") and are meant to provide additional information to the Dialog Manager when it is interpreting the panel definition. The "dot" variables are not changed by user interaction with the panel, they are defined before the panel is displayed, either by the calling function (program) or in the `)INIT` section of the panel definition. A simple example of this type of variable is the `.HELP` variable. By setting this variable to the name of another panel definition, you tell ISPF which panel to display by default when the user presses F1 or types the `help` command. (This setting can be overridden for specific parts of the panel in the)HELP section, as well as by any message being displayed.) The name of the associated help panel is something provided by the application, and not expected to be displayed to or changed by the user.

4.4.4 Dialog Tag Language

The Dialog Tag Language (DTL) provides a second way to define a panel. It will be discussed in a separate chapter – chapter 4.6 Dialog Tag Language.

4.5 ISPF Dialog hierarchy

When a selection panel or a dialog function invokes a new dialog function by using the SELECT service, the previous dialog element is not disposed of by ISPF. Instead, the new panel is displayed or the new function is given control, and after it finishes, ISPF redisplays the calling panel or returns control to the calling function. This creates a structure similar to a call stack in programming. Unlike the stack of programs calling subprograms, the ISPF dialog hierarchy contains both programs (functions) and panels, which are not executable code, but only descriptions interpreted and shown by the Dialog Manager. A dialog organization may look like this:

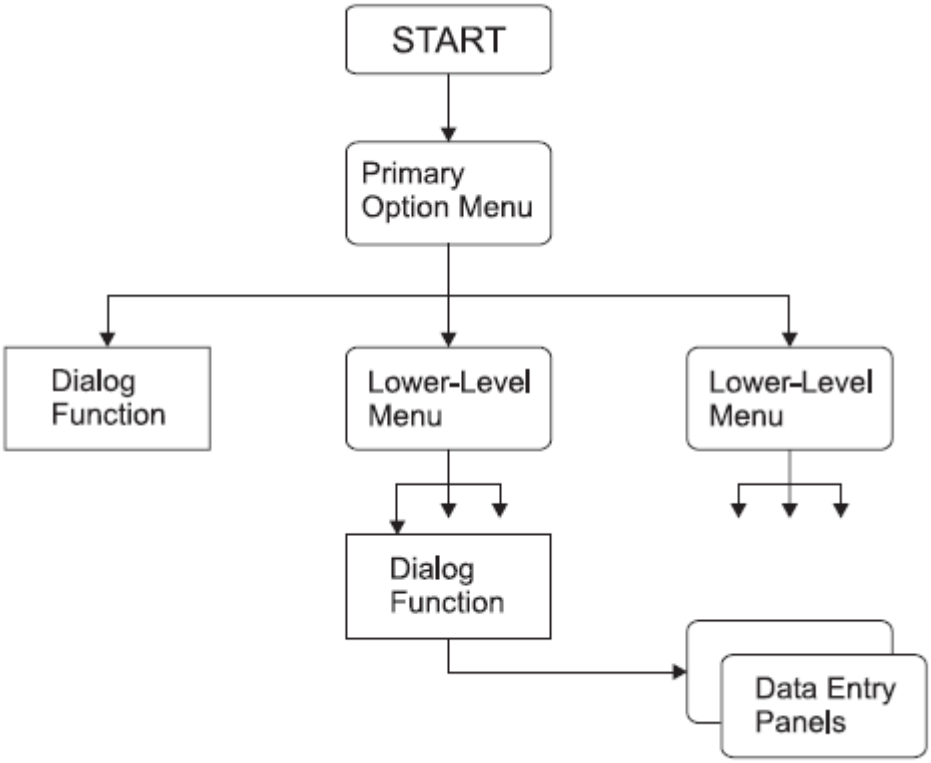


Figure 3 Typical dialog organization starting with a menu [4]

As long as the user stays in the panel part hierarchy, ISPF Dialog Manager will keep executing, moving down the tree when a user selects an option in a selection panel (= panels that contains at least one SAC field) and up the tree when F3 or cancel/exit commands are used. Control is passed out of ISPF only when a function is invoked by one of the dialogs. ISPF makes sure control is correctly passed between panels and functions, that resources used by a panel or function that are no longer part of the "stack" are freed, and provides error handling. For ISPF to be able to manage the hierarchy correctly, functions need to invoke other functions or panels using the SELECT service. If a function contains direct calls to other programs or services, they are all considered as part of the same function by ISPF.

4.5.1 Applid

When selecting a panel or function with the Select service, you have the option of selecting a new applid (Application Identifier). This lets ISPF know that the selected program/panel is the starting point of a stand-alone application. From now on, variables within application scope, dataset allocations to a DD name or other application context dependent operations will work with this new applid. You can specify whether allocations from the previous (higher) applid should carry over to the new (lower) one, or not. Defining a distinct applid lets you separate your ISPF application from the ISPF application that called it. In what I believe to be the most common case, the calling application would be a site's PDF or other

primary ISPF interface, where users do their daily work. As you as the developer have no control over what the calling application can do, using your own applid can help shield your application from other ISPF applications running, as well as protect other ISPF applications from being affected by yours.

4.6 Dialog Tag Language (DTL)

The Dialog Tag Language is a language developed by IBM that provides a second way to define some of the dialog elements – ISPF Panels, Messages, application Command tables, and keylist mappings. The DTL is a SGML-based markup language somewhat similar to HTML, sharing many common tags with IBM`s BookMaster markup language. The definitions written in DTL are not directly understood by Dialog Manager and need to be converted by ISPF conversion utility ISPDTLC (ISP Dialog Tag Language Converter) to standard panel, message or table definitions first. This means that anything that can be defined in for example a DTL panel definition can also be defined directly in an ISPF panel definition. The DTL is only a tool that can make writing the panel definition easier, faster, or more standardized [8].

4.6.1 Advantages of using DTL

As mentioned above, there is nothing that DTL can do that could not be done without it. However, there are reasons why you may consider using DTL instead of the standard code:

- Ease of use and the SGML-based markup: Especially if you are familiar with HTML or XML, the DTL will be easier for you to use. On the other hand, if you need to deal with code of existing ISPF applications, you will probably have to learn to write standard panel definitions too. In that case having to learn two different ways to define panels may bring more problems than advantages.
- Consistency of panel design when multiple developers work on a project.
- Generated panel definitions formatting conforms to IBM`s Common User Interface (CUI) guidelines, making it easier to implement CUA in your application.
- DTL enables National Language Support (NLS) and the conversion utility provides NLS translations for certain key words.

According to the ISPF Dialog Tag Language Guide and Reference [8], there are 2 more advantages:

- “DTL lends flexibility to application development. Panels can be quickly changed without you having to tediously line up text and fields. This gives you greater control over application development and updates. “

- “DTL techniques improve the way in which interactive programs, like ISPF applications, are developed. The language concentrates on the role of the various elements and their interrelationships, and ISPF takes care of their form and appearance at run time. “

As these are really relative, depending on what you are trying to achieve, I have set these apart from the rest. The fact that DTL takes care of the panel formatting can be an advantage if you are developing an independent application, but if you need to format your UI to match the look and feel of an existing application or a standard set by your organization, having less control over panel format can become more of a problem than an advantage.

4.6.2 DTL syntax

DTL syntax is based on the SGML (Standard Generalized Markup Language) – that means it is composed of *elements* forming a rooted acyclic directed graph, marked by enclosing them with *tags*. *Tags* are short text strings enclosed by sharp brackets (“<” and “>”) that exist in 2 forms: start tags (e.g. `<panel>`) and end tags (recognized by “/” as their second character, for example `</panel>`) Each element has to begin with a start tag, and most have to end with a corresponding end tag. For some DTL tags, end tag can be omitted, and the tags are ended by any following tag. Elements can be nested inside other elements, provided these do have end tags. Example:

```
<area>
  <info width=38>
    <p>You'll love the wide selection of merchandise
    in our Widgets department.
    <p>And, like all of our merchandise, Widgets come
    with our &guar;.
  </info>
</area>
```

Each DTL definition is preceded by a document declaration, which determines the type of definition (panel, message, command-table or keylist) and through that the set of tags that can be used in the document. A document declaration begins with “<!doctype” and ends with “>”. The parameters in the doctype definition are:

DM - Indicates that this is a DTL source file defining dialog elements.

SYSTEM - Indicates that the rules for the DOCTYPE are contained in an external file. This lets you include definitions from another file. If there is no value assigned to the system parameter, the entity name is used as the file name.

Entity definitions – SYSTEM parameter can be followed by entity definitions, a set of symbols that represent strings in the document text (used to make changing a text that occurs on multiple places easier)

Větší font A document type definition for a DTL panel definition with a single entity would look like this:

```
<!doctype dm system [
  <!entity guar "full, unconditional, money-back guarantee">
 ]>
```

Describing the individual tags and their functions is beyond the scope of this paper, as we will only use the standard panel definitions in the tutorial. However, all interactions between panels and other parts of a dialog you will learn in the tutorial remain valid also if you decide to use DTL.

5 Programming languages used in the tutorial

5.1 JCL – Job Control Language

A special language used for creating jobcards – instructions for running a batch job on the mainframe. Two variants exist, one for z/VSE and one for z/OS. In this tutorial we will be using JCL as a convenient way to define datasets needed during the tutorial, and no actual knowledge of JCL is necessary. When you are told to use a JCL from the appendix, just copy the JCLs to the mainframe and submit using the "submit" command from the ISPF editor or "sub" line command from PDF utility 3.4 (Dataset listing).

5.2 REXX programming language

REXX stands for REstructured eXtended eXecutor. It is a general purpose programming language with emphasis on ease of use, most suitable for uses like command scripts, application frontends, user macros or application prototyping [3]. REXX is typically used as an interpreted language, but it is also possible to compile REXX programs to create binary executable modules. REXX language uses variables without declarations, much like PHP, with no data-types defined at compile-time. All variables contain the string equal to their name as the default value. One advantage of REXX when developing ISPF applications is that REXX can share and directly use variables defined by ISPF – as long as the variable has the same name in both the REXX code and ISPF-specific code (ISPF command, ISPF panel, file-tailoring skeleton) and the variable type is supported by both of them (meaning no compound variables for example), it`s value is shared. In other programming languages, it is necessary to use the VDEFINE, VGET and VPUT ISPF commands to manipulate ISPF variables from code.

5.2.1 General syntax

REXX program consists of clauses, individual statements ended with a semicolon (;). However, the semicolons are implicitly filled in by the compiler or interpreter: at end of line, after some statements, and after a colon (:) following a single symbol. Exceptions to these rules are line breaks inside comments and quoted strings, and lines ending with the continuation character – comma (,). Therefore it is not necessary to actually use any semicolons in your code, unless you need to enter two clauses on a single line. This helps make the REXX language format easier to read while retaining the option of using semicolon to chain multiple clauses on a line. If you need to block multiple commands together (for example to insert them into an `IF-THEN-ELSE`) use the DO and END keywords to mark the beginning and end of the block. The same block can also be used for iterating, as shown in the next chapter. Offsetting the start of the lines inside a block is not required, but it is a good coding practice to improve code readability [3].

```
if ZCMD = 'PREV' then do
  VOFFSET=VOFFSET-80
  if VOFFSET < 0 then VOFFSET = 0
    signal REREAD /* Read data again using the new offset */
end
```

The color-coding used here is provided by the ISPF editor to help you read your REXX code. To enable it, use editor command “hilite rexx”.

- REXX keywords are shown in red.
- Variables, numbers and operators are shown in green.
- Strings are shown in gray.
- Comments are blue.

5.2.2 Control flow statements

REXX uses control flow structures common to other languages. It uses the `IF-THEN-ELSE` and `SELECT-WHEN` (same as `SWITCH-CASE` in most languages) constructs to branch the code depending on a condition. It uses the `DO-END` pair to mark blocks of statements, and the same block can be also used for iterating by adding iteration information to it: `DO 7` will repeat the code block 7 times. `DO FOREVER` will repeat the block forever (relying on a command inside the block to break the cycle). `DO variable = x TO y BY z` is equivalent to most languages `FOR variable = x TO y` and `DO WHILE condition` and `DO UNTIL condition` is the equivalent of the common `WHILE` and `UNTIL` loops. You can also combine these conditions in any way that makes sense, creating complex loops like this:

```
DO FOR i = 1 TO 15 WHILE x > 10 UNTIL y = 0
    .. code ..
END
```

Unlike most higher languages, REXX also contains the `SIGNAL` command for direct jump to any place within the program, either when the `SIGNAL` statement is encountered, or when a certain type of program interruption happens - like a runtime error, or an invalid statement.

REXX programming language contains procedures, although not as fully developed as in true procedural or object-oriented languages. Using the Signal or Call instructions, or a symbol immediately followed by a parenthesis, makes the program jump to the indicated label, where `PARSE ARGS` command can be used to obtain parameters passed to the procedure, and `RETURN` command can be used to return control to the instruction immediately following the calling instruction, with or without a return value. However, the

procedures are not encapsulated, and their parameters and results are not explicitly defined or type-checked. It is entirely up to the calling code to deliver the correct parameters and deal with the result appropriately [3][1].

5.2.3 Command environments and the Address command

One of the advantages of REXX and main reasons that it is going to be used for the samples in the tutorial is the ability to easily send commands into multiple system environments. Any clause that is not recognized as an instruction, assignment, label or a null clause is passed to a currently set environment for execution. By default this is TSO, however REXX is capable of forwarding commands to multiple environments. Switching between these is done by the `address` command. It lets you select an environment to receive either one command typed directly in the `address` clause, or all commands following it until any other address is encountered.

5.3 High Level Assembler (HLASM) for z/OS

The HLASM is provided freely with the z/OS operating system. It consists of instructions that are translated in a 1:1 relation into the processor instructions supported by z/OS, instructions that are processed at assembly time to generate or modify the processor instructions, and macros that translate into a series of both types of instructions, based on parameters they receive. Assembler is a low-level programming language, but macros give it some higher-level programming language capabilities, and frameworks exist for implementing common higher language commands like IF-THEN-ELSE or DO-WHILE as assembler macros. Before any assembler program can run, it must first be assembled and linked. Before assembly its macros are expanded, changing them into blocks of instructions. During assembly the instructions are processed, generating object modules – blocks of machine language instructions with a set of metadata. This is then linked by a link-editor (or its newer variant, the Binder), which replaces references to other modules with their actual addresses (links them) and prepares the code for execution. The result is an executable binary load module.

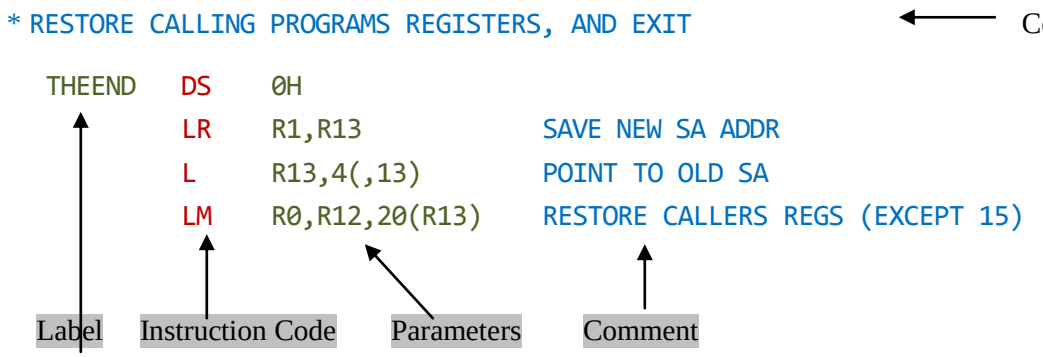
5.3.1 Uses of HLASM

HLASM, often referred to as simply assembler, is most useful when low-level access to system resources is required, as it can access any resources within z/OS. While many system functions are accessible through higher programming languages, assembler operates on a lower level and allows you to access resources not available to other languages. In fact, since assembler lets you operate with the instruction set of the CPU itself, there are no hard limits on what functionality or resources you can access (except that you may need your program to run in privileged mode to perform some operations) since you can work directly with the memory blocks of the operating system, I/O devices etc. Apart from this, many

mainframe applications were written in assembler, meaning that understanding them requires knowledge of assembler and extending them or interfacing with them is usually easier trough assembler programs or at least assembler modules. These reasons make assembler important to mainframe developers even today.

5.3.2 General syntax

HLASM statements consist of 4 parts – a label, operation code, parameters and comment. These are separated by 1 or more spaces, with an end of line ending the statement. The whole statement has to fit into columns 1 to 71, if this is not enough for the statement, placing any character into column 72 signals to Assembler compiler that the statement continues on the next line. Example:



Label – labels this instruction. It can be used to reference the address of this instruction within a load module. Referencing a DC or DS instruction (which reserves a piece of memory for data) is the closest thing to a variable you can get in assembler. A label always has to start in column 1. When a label is omitted, there has to be a space in column, with the instruction code starting at column 2 or higher (to let the compiler know this is not a label).

Instruction Code – contains either the instruction mnemonic code, or a macro instruction name. An instruction mnemonic is an alias for the machine instruction operation code, meant to be easier to remember, for example ‘L’ for Load instruction instead of hex code ’58’. If the instruction code does not match any instruction mnemonic, the compiler will look through all defined macro libraries for a macro matching this code. If found, the corresponding macro is expanded, that means it is translated into a series of machine instructions depending on input parameters given to the macro.

Parameters – parameters for the instruction or macro. Must match the parameters expected by the instruction (found in z/Architecture Principles of Operation [16]) or macro. For instructions, the parameters become part of the machine instruction created during compilation.

Comment – a comment describing what the instruction does. Due to the low level nature of Assembler, commenting the code is very important for making the code readable.

6 Hands on: Building ISPF utilities application

Our first step will be to create a basic program that will show a menu with one selectable item and display a message when it's selected. We will later extend this to serve as the main menu of our application. To create it we need to create 2 things:

- REXX code to display the panel
- Definition of the panel
- REXX code to execute the selected action

6.1 Setting things up

Before starting with our first application, let's make some preparations. Our application will consist of different sources, like REXX code, Assembler code, ISPF panels, ISPF tables and other. So, to stay organized, we will first create some libraries to hold all these different source types.

Choose a high level qualifier (HLQ) for which you have full access rights, and use `TUTALLOC` JCL from Appendix A - JCLs used in this tutorial to create the following libraries:

`HLQ.TUT.ASSEMBLE.SOURCE`	Assembler source code
`HLQ.TUT.ASSEMBLE.OBJMOD`	Assembler compiled object modules
`HLQ.TUT.ASSEMBLE.LOADLIB`	Assembler linked load modules
`HLQ.TUT.ASSEMBLE.MACLIB`	Assembler macros
`HLQ.TUT.EXEC`	REXX scripts
`HLQ.TUT.ISPF.PANEL`	ISPF panel definitions
`HLQ.TUT.ISPF.MESSAGE`	ISPF messages
`HLQ.TUT.ISPF.TABLE`	ISPF tables

You may have to modify the allocation JCL by adding volume or management/storage class to the datasets, if your mainframe environment requires it.

6.1.1 Startup REXX

We will start with the REXX startup code. We will create it as member "START" in our EXEC library. First we want to display our applications menu panel. We will invoke it using the `SELECT` command. You can find its full syntax in the ISPF Services Guide [10]. The `SELECT` command can do many things, but to select a panel called MAIN all we need is this:

```
SELECT PANEL(MAIN)
```

This will make ISPF look through its panel libraries for a member called MAIN, and try to interpret it as a panel definition. However, since we have our own library that we want to use for our applications panels, we will need to add another statement in front of the select:

```
LIBDEF ISPPLIB DATASET ID('HLQ.TUT.ISPF.PANEL')
```

The `LIBDEF` tells ISPF to define our panel library as its ISPPLIB – the DD where ISPF will look for panels before looking into its standard libraries.

Now, after adding `ADDRESS ISPEXEC` to tell REXX we are using ISPF commands, we get a code looking like this:

```
/********************************** REXX ***********************************/
address ISPEXEC
"LIBDEF ISPPLIB DATASET ID('VASMI03.TEST.TUT.ISPF.PANEL')"
"SELECT PANEL(MAIN)"
```

This is enough to show our panel, after we have defined it in our panel library. But since we want to use it to run additional REXX scripts and to display messages, we need to make some more definitions. This may seem a bit complicated the first time, but it will help us to get all the definitions out of the way so we don't have to keep modifying the start script when we start adding new stuff to our application. This is what our start script will look like:

```
/*************************** REXX **********************************/
/* ALLOCATE LIBRARIES                                              */
/* LAUNCH PANEL MAIN                                               */
/*******************************************************************/
address ISPEXEC
"LIBDEF ISPPLIB DATASET ID('VASMI03.TEST.TUT.ISPF.PANEL')"
"LIBDEF ISPMLIB DATASET ID('VASMI03.TEST.TUT.ISPF.MESSAGE')"
"LIBDEF ISPTLIB DATASET ID('VASMI03.TEST.TUT.ISPF.TABLE')"
"LIBDEF ISPLLIB DATASET ID('VASMI03.TEST.TUT.ASSEMBLE.LOADLIB')"
```

```
address TSO
"ALLOC F(REXXLIB1) SHR REUSE DA('VASMI03.TEST.TUT.EXEC')"
"ALTLIB ACT APPLICATION(EXEC) LIBRARY(REXXLIB1)"
address ISPEXEC
"CONTROL ERRORS RETURN"
"SELECT PANEL(MAIN) NEWAPPL(TUT) PASSLIB"
address TSO "ALTLIB DEACT APPLICATION(EXEC)"
```

Here, we need to use both ISPF and TSO commands, so we have to keep using the ADDRESS statement to specify. The reason for this is the use of TSO ALLOC and ALTLIB commands. While TSO is not part of this tutorial, the commands used here are useful for ISPF applications. They add our REXX library to the system concatenation of EXEC libraries. Unlike ISPF LIBDEF, this command needs a deallocation at the end of the program, to avoid concatenating our library multiple times when restarting the program.

The ISPMLIB used in the first new LIBDEF statement is the DD used for an applications Message library. We will be using these in REXX scripts called from the main panel, but defining all the libraries at the entry point will help us to keep things organized. Any application invoked using the SELECT command can use the definitions made by the caller.

The ISPTLIB is our table library. Here ISPF will create and look for our applications tables, as well as our application command table. (more about that later)

The ISPLLIB is our load library. Here ISPF will look for our load modules, when we tell it to invoke a binary program

The CONTROL ERRORS RETURN statement tells ISPF that if the application encounters an error, it will not crash, but return control to our START script with RC > 0. We could use this to deal with the error. Or, as in this case, to make sure the deallocation statements following the SELECT will be executed even if there is some kind of error in the rest of the application.

6.1.2 Definition of the panel

Each ISPF panel definition is composed of sections, starting with ")" and ending with start of next section or with the END section. The sections are always defined in a fixed order. For the complete list, see chapter 4.4 ISPF Panels, or page **106** in z/OS V1R9.0 ISPF Dialog Developer's Guide and Reference [4]. Now let's look at the code for our MAIN panel, section after section. Create a member MAIN in your TUT.ISPF.PANEL library. Now add the following code:

```
)ATTR DEFAULT(@+_)
      /*  @ TYPE(TEXT) INTENS(HIGH)  default for highlighted text        */
```

```
        /*  + TYPE(TEXT) INTENS(LOW)   default for normal text         */
        /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT) default entry field*/
```

This is the)ATTR section is used to define characters, that will be used in the following)BODY section to define fields and their formats. In this case, we are telling ISPF to use markers @+_ for the three default field types:

@ as highlighted text, + as normal text, _for input/output field

The next 3 commented lines are definitions that would be used to denote these characters if they were not already default, to remind us of what each character stands for. We will cover these more later.

```
)BODY
%-----------------  ISPF TUTORIAL APPLICATION -------------------------------
%SELECTION ===>_ZCMD                                                        +
+
+   Select the action you would like to perform:

   @1+ Say Hello World
+
+
```

The)BODY section is the main section of the panel definition, here you create the panel itself using the special characters defined in)ATTR. Any character after + and before another special character (or end of line) will be displayed as simple text in the panel, any character after % is displayed as highlighted text and any character after _ is displayed as a field. Following the _ is name of the ISPF variable being displayed. Length of the name must be at most 8 characters. Length of the displayed field is given by the distance of _ to the next special character or end of line.

The ZCMD is a special variable, which is used for command input. Its inclusion makes this a selection panel. Like all fields, its contents will be available to the calling application in a variable of the same name. However, ZCMD will also try to interpret the input as ISPF user command.

```
)INIT
  .HELP = HELPMAIN              /* insert name of help panel  */
```

)INIT section can contain any initialization actions, that should be done before the panel is shown. The .HELP line defines a help panel – a panel called if the user presses F1 on this panel. It will be internally invoked by a SELECT command, using your defined panel library – by setting a system variable .HELP

```
)PROC
   &ZSEL=TRANS(TRUNC(&ZCMD,'.')
```

```
            1,'CMD(TUTHELLO)'
             )
)END
```

The)PROC section contains processing instructions for the panel – after user inputs some data in its fields, and presses enter, code in this section can be executed. The code used here a common way of handling selection panels. It takes the users input from system variable ZCMD, and converts it into system variable ZSEL (the & marks variables outside the body section) This variable tells ISPF to execute a SELECT command, with parameters defined by the ZSEL. So in this case, if user input is "1", ISPF should execute

```
SELECT CMD(TUTHELLO)
```

which should start command script TUTHELLO. Since we have concatenated our EXEC library to the systems EXEC, it will look in there, and execute member TUTHELLO. The TRANS command also has the side effect of verifying the input – that means not closing the panel until user input corresponds to one of its defined translations. The section is ended with the)END tag, because it is the last section in this panel.

6.1.3 REXX code for TUTHELLO

Member TUTHELLO in our TUT.EXEC library should contain the code needed to perform the desired action – saying Hello, World. The easiest way to do this would be:

```
/*REXX*/
say "Hello, World!"
```

This will work, and you can try coding this and testing your application by executing START script.

If you do this, you will notice the message created by say appears in a rather ugly way somewhere in the middle of the panel. This is not what we want our output to look like. To inform the user about something without switching to a new panel, 2 things can be used – popup, or message. For such a short text, message is more fitting. We display a message by putting this code into TUTHELLO

```
/************************** REXX ***************************/
address ISPEXEC 'DISPLAY PANEL(MAIN) MSG(TUT000)'
```

This causes a redisplay of our MAIN panel, but with ISPF message TUT000. ISPF messages are identified by ID of at most 8 characters, with at least 3 of them digits. It will look for this message in our message library (defined by DD ISPMLIB) under member name TUT00. The member name is generated by truncating any digits after the 2nd. Each member can therefore contain multiple ISPF messages.

6.1.4 ISPF message TUT000

Now all that is left, is defining the message. Create member TUT00 in TUT.ISPF.MESSAGE:

```
TUT000 'HELLO, WORLD!' .HELP=* .ALARM=YES
'THE APPLICATION IS GREETING THE WORLD AND ALL ITS PEOPLE'
```

This is the standard message format. First line is message id and message text.

.HELP is the associated help panel. If * is used, the help panel defined by the panel displaying this message will be used.

.ALARM=YES tells the terminal to beep to alert the user. If set to no, the alarm can still sound if it is set on the panel displaying this message.

The second line contains Long message text, explaining the message in greater detail. It is displayed if user presses F1 while the message is up. It can be up to 512 bytes on multiple lines concatenated by "+"

That is all we need to get this application working. You can now try starting the application by executing the START script. When you are content with how the application works, proceed to the next chapter.

6.2 Improving the basic application

6.2.1 Point and shoot fields

These are not widely used, but can be useful to make your panel somewhat more interactive. They are defined in)PNTS section of a panel, placed after)PROC section. They mark fields in your panel that can be clicked by mouse (or by moving a cursor to them and pressing enter) to execute an action. They are used on selection panels as an alternative way to select an option from the available option list, or on panels that let you check and uncheck some settings.

To add Point and Shoot fields to our panel MAIN, we need to do 3 things:

1) Modify the)ATTR section by adding a point and shoot field definition character:

```
! TYPE(PS)                          /* Point and Shoot text          */
```

2) Modify the)BODY section by putting the new point and shoot field definition character "!" in front of the option text, and "+" (normal text definition character) behind it to end the field

```
@1+!Say Hello World +
```

3) Add the)PNTS section after)PROC:

```
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL(1)
```

The ZPC00001 is a z variable, denoting the first point and Shoot field on the panel (from top to bottom and left to right) Other fields will have similar variables depending on their position on the panel (ZPC00002, ZPC00003, etc..) This statement tells ISPF that if field ZPS is clicked, it is equal to the user typing "1" into field ZCMD (our command line) and pressing enter. So the ZCMD value handling in)PROC section is activated just like if we typed the option number.

6.2.2 Adding options

We will add some additional options, so our user actually has something to choose from. In the next part of this tutorial, we will be creating a Dataset Listing utility, so we will add this as option 2. Last part of the tutorial will be examining a KSDS VSAM dataset, so we will add Examine KSDS as option 3. Also, it is customary to have an "exit" option in the primary menu, so we will add that as option "X". You can try it yourself to practice a bit, or if you are not yet sure, you can follow these steps:

First, we must add the lines into)BODY to show the new options to the user:

```
@1+!Say Hello World+
@2+!List Datasets+
@3+!Examine KSDS+
@X+!Exit+
```

Notice the point and shoot fields are used. Next we add corresponding SELECT actions into)PROC:

```
)PROC
&ZSEL=TRANS(TRUNC(&ZCMD,'.')
            1,'CMD(TUTHELLO)'
            2,'CMD(TUTDSLST)'
            3,'CMD(TUTVSAM)'
            X,'EXIT'
             )
```

The 4th option inserts the standard 'EXIT' command into the command line (the ZCMD variable) which is exactly the same action as is usually assigned to the PF3 key. And as a last step, we define the new point and shoot fields in)PNTS:

```
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL(1)
  FIELD(ZPS00002) VAR(ZCMD) VAL(2)
  FIELD(ZPS00003) VAR(ZCMD) VAL(3)
  FIELD(ZPS00004) VAR(ZCMD) VAL(X)
```

6.2.3 Help Panel

All panels in ISPF should have a pointer to a help panel that provides information related to the displayed panel; this does not mean there has to be a 1 on 1 relation between application panels and help panels though. Usually, there is a tree hierarchy of panels that creates the help. Its root is a selection panel, providing options to access main help topics. Leafs of the tree are made of normal (non-selection) panels

providing details. The tree can correspond to the hierarchy of application menus, or follow a different logic (based for example on topics)

For our application, we will create only a simple help, with one main panel, and one help panel for each available option except exit. The help panels are created just like ordinary panels and they can even have a .help defined to create help panels for help panels. (It can make sense to have 1 panel telling the users how to navigate your help) That means you should now know all you need to try creating the main help panel and the help panel for option 1 yourself. Go ahead and create them. If you run into trouble, or want to verify your solution, you can find examples in Appendix B: HELPMAIN and HELP1. When you are content everything works as it should, proceed to the next chapter.

6.3 Second utility: Dataset listing

Now that we have tried out all the basic components, let's create a second utility that looks a bit more like a real application. We will try to create a listing of datasets under a HLQ displaying dataset type, size in chosen units, a count of library members, and the ability to sort the list by any column, with an action bar menu and its own line command. If it sounds overwhelming to you now, do not worry, we will do this step by step.

6.3.1 DSLIST Panel

Let us start with the easy part first: The input panel for the new utility. Before listing any datasets, we need to ask the user to tell us the High Level Qualifier he wants to list and what units to use for displaying size.

So, we will have to create a panel with 2 input fields, one for the HLQ text, and the other offering a choice of one out of 4 size units (tracks, cylinders, bytes or megabytes)

You can find the whole code for this panel in Appendix B: DSLIST, here we will go only trough things that can be new to you. It is best if you first copy the code to your mainframe dataset, so you can easily look into the code as you read through the explanation. We will start with the first thing you have not seen so far:

```
)INIT
  .HELP = HELP2
  .ZVARS = sel
```

Here you can see a new "dot" variable. `.ZVARS` are set at initialization time to a variable or a list of variables.

Then in)BODY, variable Z can be used in their place (if multiple z variables are used, the order in the list must correspond to the order of occurrences of Z in body).

This is a bit of a workaround for one problem in ISPF panels – the name of the variable must fit into the field it creates in the panel. So to let us create a 1-character field for a variable with a longer name, we have to use a ZVAR. In this case, the "_z" in body is a one character long field for variable "sel".

```
)PROC
  VER (&dshlq,DSNAME)
  VER (&dshlq,LEN,GT,0)
  VER (&sel,LIST,1,2,3,4)
)END
```

The other thing we haven't used before are VER (verify) statements. With these, we can let ISPF do some basic input checking for us, so we do not have to worry about it in our program. When a variable does not pass the VER statement, user is prompted to correct it, and control is not passed away from the panel until they do. You can find the format and parameters of VER statement in ISPF Dialog Developers Guide[4], the general syntax is

```
VER (variable [NONBLANK] keyword [,MSG=value])
```

where keyword is one of a selection of evaluation conditions (possibly with their own parameters, like on the second VER here), NONBLANK specifies that the value is mandatory (variable must not be empty) and MSG specifies message to display if verification failed. The MSG parameter is optional, as ISPF has prepared default messages for each condition, so you need your own messages only if you have some additional information to deliver to the user.

6.3.2 Result display panel DSLIST2

The second panel for the new utility will be displaying a list of results. Fortunately, ISPF has a special type of panels just for this purpose. To display a list of results, complete with standard ISPF scrolling, put the results into an ISPF table and create a panel with a)MODEL section. This section is a model of how ISPF should display a single table row. To use it, we need the function code to insert the output data into the ISPF table and to call the panel by a different ISPF command (TBDISP). This will be done by the REXX program TUTDSLST we will create in the next step. For now, let's go through our new panel. As always, you can find its code in Appendix B.

```
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                    /* Variable Output Information   */
  ¢ TYPE(CH)                    /* Column Heading                */
  $ TYPE(FP)                    /* Field Prompt                  */
  ? TYPE(ET)                    /* Emphasized Text               */
```

```
* TYPE(LI) PADC(USER)           /* List Item                    */
! TYPE(PS)                     /* Point and Shoot text         */
# TYPE(LI) PADC(USER) JUST(RIGHT)/* List Item Right Justified    */
½ TYPE(AB)                      /* Action Bar                   */
¼ TYPE(ABSL)                    /* Action Bar Separator Line    */
```

The)ATTR section will be a bit longer than before. That is because we need new types of fields to mark action bar (AB) items and separator, the headers of our table columns, and the fields inside the model called List Items. List Item fields refer to a table column name, just like Entry Fields refer to a variable. The reason for having 2 definitions for the List Item here is that we want text to be left justified, and numbers to be right justified for readability.

```
)ABC DESC('ACTIONS') MNEM(1)
  PDC DESC('SORT BY NAME')
  PDC DESC('SORT BY TYPE')
  PDC DESC('SORT BY SIZE')
  PDC DESC('SORT BY MEMBER COUNT')
  PDC DESC('CHANGE HLQ')
  ACTION RUN('ISRROUTE') PARM('SELECT PANEL(HELPMAIN)')
  PDC DESC('EXIT')
  ACTION RUN('EXIT')
```

The)ABC section defines a single Action Bar item. DESC tells ISPF what to display as this items name, MNEM says which character will be highlighted and used as a hotkey. In our example you can see some basic usage of ABC. If our application had multiple items in its action bar, there would be multiple)ABC sections preceding the)BODY section. The simplest usage is to just fill the)ABC section with PDC definitions with descriptions, like the first four items in our ABC. ISPF will automatically add numbers to the options, and an input field allowing the user to choose from the items, including a check for valid values. The other 2 options use ABCs built-in function to immediately take action when an option is selected. Using the Action keyword followed by Run, you can execute a command from your applications command table. A special command is available: ISRROUTE can be used to process an ISPF command like a SELECT. This is executed immediately after the user selects this option.

```
)ABCINIT
   .ZVARS=HELPSEL
```

When one of the other options is selected, the whole panel disappears, returning control to the application that called it. The value of the users choice is put into a .ZVAR. To let our program read it, we have to

assign one of the regular variables to the ZVAR in)ABCINIT section. This section does for)ABC what)INIT does for the whole panel – put anything that needs to be done before displaying the menu popup, including, but not limited to, defining .zvars. Each)ABCINIT refers to the)ABC section immediately preceding it. Similarly,)ABCPROC can be used for post-processing an)ABC.

```
)BODY
@-----------------  ISPF TUTORIAL APPLICATION -------------------------------
$Option ===>_ZCMD                                          $Scroll ===> _SCR
+
+   Listing Datasets for: %dshlq
+
+   Type command?'SORT <column>'+or click column header to sort the list
+
¢  !Name¢                        !Dataset type¢  !Size%unit   ¢  !Member count¢
```

The)BODY section defines the static part of the panel. It shouldn't be too big, or there will not be enough space for the displayed list and the user will have to scroll more. The list will be appended right under the body, so table headers are usually located on the last line. Notice that the headers are defined as point-and-shoot fields, to allow easy sorting by column names.

```
)MODEL clear(sel)
_z+*llq                 +|#dsorg        +|#size        +|#count    +
```

This is the model of a single list entry. The List Entry fields correspond to columns, and will be filled in by values form each row of the table. When a normal display or entry field is used in the model, it will show the value of its corresponding ISPF variable, and will be the same on all rows in the list. The z is the special variable used with .ZVARS definition mentioned earlier. It creates an input field 1 character long, and the .ZVARS definition maps it to variable "sel". When any kind of input field is used inside the model definition (using .ZVARS or not) you will need additional commands in the function code to determine the line on which the user input was made. But we will get to that in the REXX code.

```
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL('SORT NAME')
  FIELD(ZPS00002) VAR(ZCMD) VAL('SORT DSORG')
  FIELD(ZPS00003) VAR(ZCMD) VAL('SORT SIZE')
  FIELD(ZPS00004) VAR(ZCMD) VAL('SORT COUNT')
)END
```

Last section used is the point and shoot section. You can see another technique of using PNTS fields here. Instead of selection numbers, we use the fields to create text commands on the command prompt (ZCMD variable).

So hitting a field has exactly the same effect as typing its corresponding command into the command line.

In our case, the point and shoot fields are table headers, and work as shortcuts for commands our application will recognize as requests to sort the table by a given column.

6.3.3 Detail display panel DSLIST3

This is a simple panel, that we will display if the user types the 'S' line command (the customary line command for selecting a row) into our dataset list. There should be nothing new for you in this panel, it is a simple display panel showing a set of variables as fields.

6.3.4 Main program code - TUTDSLST

This is the REXX script where the interesting stuff happens. This script uses more ISPF services than just select or display, making use of ISPF tables to store data, and other ISPF services for locating and listing datasets. You can find details about them and their syntax in ISPF Services Guide. Now, take a deep breath, copy the full code from Appendix C: TUTDSLST to your mainframe EXEC dataset as member TUTDSLST, and let's take a look at the commands used in this program:

```
address ISPEXEC
'TBERASE TUTTAB'
'TBCREATE TUTTAB NOWRITE
            KEYS(LLQ) NAMES(SIZE COUNT DSORG)'
```

This `TBCREATE` ISPF command creates a new ISPF table, with key column called "`LLQ`" and 3 other columns called `SIZE`, `COUNT` and `DSORG`. The preceding statement makes sure the table from previous run of the program was not kept, otherwise the create statement would not work and we would be reusing the table. That can often be desirable, but in our case, we want a clean table.

```
dshlq = ''
sel = '1'
'DISPLAY PANEL(DSLIST)'
if RC>0 then EXIT
SELECTION = SEL
```

This `DISPLAY` command displays panel DSLIST within this dialog function, without creating a new dialog step like `SELECT` would have done. After the dialog is closed (by enter, exit or cancel) the program gets control back, with all variables updated according to user input. The special variable `RC` will contain the return code, with `RC` = 0 indicating the user entered the input and pressed enter and `RC`=8 indicating the user canceled or exited the panel. Our panel here contains 2 input fields, one for the HLQ listed, and one for a choice of size units. Since the panel already checks the input for us, we can assume the values in these variables (`dshlq` and `sel`) are valid as long as `RC`=0. If `RC`=8, the user exited the panel, so we exit

the REXX script as well to return control to a higher member of the select hierarchy (in our case the primary menu).

```
dslevel = dshlq".*"
"LMDINIT LISTID(IDV) LEVEL(&DSLEVEL)"
if rc>0 then leave
```

The LMDINIT creates a dataset list ID, and is used later for processing the list. Note that DSLEVEL is a wildcard HLQ.* The id IDV is thus assigned to the list of all datasets starting with HLQ. The RC tells us if allocating the ID was successful. If not, we should handle this by notifying the user and requesting a new input. For simplicity of this example, however, we will just exit the script.

```
do forever
    "LMDLIST LISTID("IDV") OPTION(LIST) DATASET(DSNAME) STATS(YES)"
    if rc = 0 then do
      "LMINIT DATAID(IDL) DATASET('"DSNAME"') ENQ(SHR) ORG(DSORG)"
```

Now, we start the cycle going through all the dataset members. LMDLIST with option LIST gives us next dataset from the list we have defined by LMDINIT, or RC=8 if there are no more datasets in the list. The name of this dataset is saved in variable DSNAME and statistics about the dataset are created and saved in special variables starting with prefix ZDL. For full list see the ISPF services guide, some of the basic ones will be used later in this example. If RC = 0, meaning we got a new DS from the list, we will continue with its processing by assigning an ID to the dataset (similar to the list ID assigned earlier)

Don't get confused, we are assigning the ID here, so DATAID is the variable where we want the new ID stored, and DATASET is the input, the DSN of the dataset.

```
      if dsorg="PO" then do
        count = 0
        memname = '        '
        "LMOPEN DATAID("IDL")"
        do forever
          "LMMLIST DATAID("IDL") OPTION(LIST) MEMBER(MEMNAME)"
          if rc = 0 then do
            count = count + 1
          end
          else leave
        end
        "LMCLOSE DATAID("IDL")"
      end
```

This part of the code uses the dsorg obtained from LMINIT to determine if the current DS fetched from the list is a library. If so, it counts its members by opening it, and running a LMMLIST loop. LMMLIST is similar in function to LMDLIST, the only difference is it iterates trough members of a library instead of a list of datasets. After counting the members, we close the library to avoid blocking other users.

Rest of the cycle deals with getting the part of a datasets name that follows the specified HLQ, and with calculating the size of the datasets in units given. It does so by using the ZLD- variables:

ZDLSIZE containing the datasets size in tracks, and ZLDDEV to find out what device they are stored on. Using the device type, size in units other than tracks is calculated based on characteristics of the common device types. At the end of each cycle is this ISPF statement:

```
TBMOD TUTTAB
```

This is a critical step in the cycle. TBMOD adds a line to our table TUTTAB, using data from variables, whose names are identical to names of table columns. So, ISPF looks into variable llq (defined as key column) – and finds a line in the table with key = llq. If it is not found it is created, otherwise it is overwritten. Each of the columns is filled in by values found in variables with corresponding names (size, count and dsorg).

```
lastsort = 'SORT NAME'
sortorder = 'a'
sorttype = 'c'
sortfield = 'llq'
redisplay:
"TBSORT TUTTAB FIELDS("sortfield","sorttype","sortorder")"
'TBVCLEAR TUTTAB'
'TBTOP TUTTAB'          /*POINTER BACK TO START OF TABLE*/
'TBDISPL TUTTAB PANEL(DSLIST2)'
```

After closing the cycle, use this bit of code to display panel DSLIST2 with the results. Notice we use TBDISPL instead of DISPLAY or SELECT. This function displays an ISPF table (TUTTAB) using an ISPF panel (DSLIST2) The panel has to contain the)MODEL section telling ISPF how to format the table rows. ISPF then takes care of building the table, and making it scrollable if all the rows do not fit on the terminal screen. The preceding command TBSORT sorts the table by our defined column, using sort type of ‘c’ for texts and ‘n’ for numbers, in ascending or descending order.

TBVCLEAR clears table variables (llq, size, count, dsorg) that could be used for filtering the table, if we wanted to show only partial result set. Finally TBTOP tells ISPF to position the cursor to the top of the table – since ISPF tables have a cursor that can be used for retrieving rows one at a time, and this cursor

now rests on the last inserted line. Leaving it there would display an empty table, as TBDISPL uses the iteration mentioned, and thus displays only rows between cursor position and the end of the table.

```
if zcmd == "SORT NAME"  then do
  sortfield = 'llq'
  type = 'c'
end
```

(if statements for other sort commands follow, left out for readability)

```
if zcmd == lastsort & sortorder == 'a' then
  sortorder = 'd'
else
  sortorder = 'a'
if zcmd == lastsort then do
  if sortorder == 'a' then sortorder = 'd'
  else sortorder = 'a'
end
```

These lines of the script are used to redisplay the panel DSLIST2 if user enters a "SORT" command or clicks one of the table header point-and-shoot fields. It sets the sort-related variables according to the column that should be sorted, reversing the sort order if the same command is received twice in a row. Afterwards it cycles back to the table sort and display. If no sort command was entered, the `signal redisplay` statement is not invoked and the script ends, returning control to the calling dialog member (in our case, the MAIN selection panel). Note that in order for ISPF to sort numeric fields correctly, type must be set to 'n' as numeric (c is default) and the value in the field has to be formatted by FORMAT command to include leading spaces.

```
if sel == 'S' then do
  dsn = dshlq'.'llq
  "DISPLAY PANEL(DSLIST3)"
  signal redisplay
end
```

Finally this little piece of code handles the line command "s". When any input is made into a field inside the)MODEL section, after the panel returns, the table cursor will automatically be positioned on the first line with any input in it and the input variable will have the value. So when we detect the S, we know that all of the table variables (`llq, size, count, dsorg`) are set to the values on the first selected line, so we can use the values of `llq, size, count, dsorg` in the DSLIST3 panel to display detailed info about the dataset that was selected.

6.3.5 Messages TUT001-TUT003

We are using 3 new messages in our Rexx code, TUT001, TUT002 and TUT003. To define these to ISPF, we have to edit the TUT00 member in our message library, and add their definitions:

```
TUT001 'UNKNOWN COMMAND' .HELP=* .ALARM=YES
'THE COMMAND YOU HAVE TYPED IS NOT VALID IN THIS APPLICATION'
TUT002 'INVALID OPTION' .HELP=* .ALARM=YES
'ONLY THE SELECT (S) INLINE OPTION IS SUPPORTED'
TUT003 'NO DATASETS FOUND' .HELP=* .ALARM=YES
'NO DATASETS WERE FOUND MATCHING THE HLQ YOU SPECIFIED'
```

6.4 Last utility: KSDS VSAM reader

The dataset listing utility taught us some techniques useful in ISPF user interface development, but functionally it was very simple. To create a utility that can do something more complicated within the operating system, we will now try to add a bit of assembler into the mix. We will focus on combining an assembler function accessing system data with a REXX script driving the user interaction using ISPF to create a utility that can display the contents of a KSDS VSAM dataset – something that cannot be done by the standard set of utilities provided in ISPF Program Development Facility. And we will also be learning some more ISPF tricks like horizontal scrolling and using command tables.

6.4.1 Command table TUTCMDS

Before we begin coding the core program working behind the scenes, we need to set up a command table. Whenever a user types a command (or uses a PF key to do so) the command tables are checked first. Command tables can exist in 3 levels, system, user or application. What we want to do is define an application command table, associated with our applications ID. The easiest way to do this is by using ISPF option 3.9 to define the table. Choose 'TUT' as the Application ID and press enter. On the following screen, define 2 commands LEFT and RIGHT, and assign PASSTHRU as their action:

```
   File  Menu  Utilities  Help
------------------------------------------------------------------------------
CA31 VASMI03                    Update TUTCMDS                Row 1 to 2 of 2
 Command ===> ____________________________________________________ Scroll ===> PAGE

 Insert (I), delete (D), repeat (R) and edit (E) command entries.
 Enter END command to save changes or CANCEL to end without saving.
```

```
 Verb              T  Action
____ LEFT      0  PASSTHRU
___ RIGHT     0  PASSTHRU
****************************** Bottom of data ********************************
```

The PASSTHRU action means the command is passed to our application for processing. The reason we needed to define this is to override system command table that uses these commands for scrolling scrollable areas left and right. Using PASSTHRU we can force ISPF to ignore this and pass the commands to our application, allowing us to handle side scrolling (PF10, PF11) by ourselves.

Similar overrides can be used for other default system commands, like Exit (PF3), Up and Down (PF7, PF8) etc. However, to keep things simple, we will let ISPF handle these and only override the side scrolling commands, as the default implementation of these does not work for ISPF tables.

When you are done editing the table, press F3 to save it. ISPF will put it into the first library in ISPTABL concatenation, which is usually your ISPF profile library. Copy it from there to your applications TABLE library to allow other users to use the command table when they run your application. The name of the member will be TUTCMDS. If you have trouble locating the library, use TSO command ISRDDN and look for ISPTABL – the first dataset assigned to this DD name will contain the saved table.

6.4.2 Input panel VSAM1

The first panel we need to create is a simple one, with some kind of introduction, and a single input field for the DSN of the KSDS VSAM dataset we want to view. That should be easy for you now, all you need to know is:

1) The program code will expect the input field to be called INDSN
2) We want to verify the input is really a DSN – meaning this VER statement should be in)PROC:

```
VER (&indsn,NONBLANK,DSNAME)
```

Try coding the panel yourself now. As always, you can find an example of the correct panel definition in Appendix B: VSAM1.

6.4.3 Main panel VSAM2

This is the panel that displays the contents of the KSDS. It consists of 2 parts, a fixed part displaying overall information about the VSAM, and a list of records from the table. Unlike previous example, we will want the list to scroll horizontally as well as vertically. This is not directly supported by ISPF like the vertical scrolling, so it will take some extra work. This is how to do the trick: The ISPF table being displayed will only contain first 80 columns of each record. Earlier we have set up the command table to

pass through the RIGHT and LEFT commands (F10,F11) to our program code. Every time our program receives these commands, it will repopulate the ISPF table with new 80 columns for each line (jumping forward or back by 80 or as many as the record length allows) and redisplay the panel. From the user`s point of view, it looks like they never exited the panel, but only scrolled its contents right or left. Please note that this is only an example - for a large KSDS it will be inefficient, as all the records need to be read and updated during the horizontal scroll. To avoid this, vertical scrolling would have to be handled by the program code too, so that only the table rows currently displayed would be read at any scroll request. This is a bit easier due to support from ISPF, if you are interested in learning how to implement vertical scrolling using ISPF-supplied variables see *Example: dynamic table expansion* in chapter 3 of *ISPF Dialog Developer's Guide and Reference* [4] For now though, mixing both could be confusing, so we will only implement the horizontal scrolling, and let ISPF deal with the vertical scroll on its own.

Now we are going to have a look at the source in Appendix B:VSAM2. Copy it to your PANEL library and follow the explanations in code as usual.

First, let us get the static part out of the way:

```
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                       /* Variable Output Information   */
  ! TYPE(PS)                       /* Point and Shoot text          */
  @ TYPE(PT)                       /* Panel Title                   */
  ¼ TYPE(VOI) PADC(-) JUST(RIGHT)  /* Normal Field           */
  $ TYPE(FP)               /* CUA Attribute for Field Prompt          */
  * TYPE(LI) PADC(USER)             /* List Item                     */
```

The attributes are same as usual, except we will need 2 types of output fields – one left-justified (the default) for general use, and one right-justified to show horizontal scroll position in upper right corner, under the vertical scroll position shown automatically by ISPF.

```
)BODY
@------------------------------ KSDS VIEWER ----------------------------------
+                                  ¼VCOLINFO                                  +
$Command ===>_ZCMD                                          $Scroll ===>_SCR
+
+ VIEWING KSDS %INDSN                                                        +
+
+ KEY LENGTH:    %VKEYLEN +        RECORD COUNT:  %VNLOGR  +
+ KEY OFFSET:    %VRKP    +        RECORD LENGTH: %VLRECL  +
+                                  ENDING RBA:    %VENDRBA +
```

```
@-------------------------------------------------------------------------
```

The body consists mostly of output fields displaying various properties of the VSAM. However, notice the right-justified VCOLINFO field.

```
)MODEL
*VREC
```

The whole line in the model will be created by the program code, so we make it as one big output field (terminated by end-of-line).

```
)INIT
  &ZCMD = ''
)REINIT
  &ZCMD = ''
  REFRESH(ZCMD,VCOLINFO)
```

Because we will be redisplaying the table on every horizontal scroll, we need the `)REINIT` section. It will be called every time the table is re-displayed, making sure that the `LEFT` or `RIGHT` commands are not left on the command line, and refreshing the value of the horizontal position info. The `REFRESH` is important for variables changed between displays, except for lines in the `)MODEL` section, which are always generated anew after redisplay.

```
)PROC
)PNTS
)END
```

6.4.4 Intermediate layer Rexx script TUTVSAM

While we are going to use assembler for the actual VSAM reading, using a Rexx script middle layer will make the assembler code a lot less complex. The purpose of the script will be to handle the displaying and redisplaying of panels, checking and processing of user input and commands, and generally doing anything that does not involve the actual VSAM processing. The communication between the Rexx script and the assembler module will be handled entirely through ISPF – using the ISPF Select command to call the module and ISPF variables to pass information between the module, the Rexx code and the ISPF panels. This has the advantage of having a single set of variables shared across the whole application, as well as the added benefit of ISPF providing the values to the assembler module in defined format (for example packed decimal) without need for parsing of a parameter list and converting to the desired format inside the program.

Now, copy the REXX code for TUTVSAM from appendix C:TUTVSAM to your EXEC library and lets go over the interesting parts:

```
"TBCREATE VSAMTAB KEYS(VKEY) NAMES(VREC) NOWRITE REPLACE"
```

This is the table for our output - VKEY is the key in KSDS records, or a number in RRDS, and VREC will contain the currently viewed 80 columns of each record. The table will be filled with actual values by our assembler module. But defining it here will make it available to both the rexx and assembler code, as both run under the same applid.

```
address TSO "ALLOC F(INVSAM) SHR REUSE DA('"INDSN"')"
if RC>0 then signal IOError
```

Allocate the VSAM dataset the user requested to a DD INVSAM – this is the DD statement where our assembler code expects the VSAM dataset; If the allocation fails, we go to the error routine, that will display the first panel again, together with a message informing the user there is something wrong with the DSN they provided.

```
/*READ VSAM AND FILL IN THE VSAMTAB AND OTHER VARS USING ASM PGM*/
VOFFSET = 0
"VPUT VOFFSET PROFILE"
"SELECT PGM(VSAMASM)"
if RC>0 then signal IOError
```

This is the first call to our assembler module. The VOFFSET variable is used by the assembler code to determine which columns from the VSAM to put into our ISPF table – to make its value available to the module, we need to use VPUT to insert it into our PROFILE variable pool, from where it can be retrieved by using the VGET command. After the input variable is set, we use the SELECT service to call the assembler module. ISPF will take care of loading the module into memory and pass control to it, returning control to our Rexx script after it finishes, and setting the RC variable to the return code from register 15.

```
SCR = "PAGE"
```

We set the scroll amount to PAGE for the sake or consistency – as our horizontal scroll only scrolls by a whole page (80 columns) For a more serious application we should take the scroll amount defined by the user and honor it in our horizontal scrolling.

```
"TBSORT VSAMTAB FIELDS(VKEY)"
"TBVCLEAR VSAMTAB"
"TBTOP VSAMTAB"        /*POINTER BACK TO START OF TABLE*/
"VGET VLRECL"
VCOLINFO = 'Column 'VOFFSET' to 'VOFFSET+80' of 'VLRECL
"TBDISPL VSAMTAB PANEL(VSAM2)"
```

Now that the assembler module has filled VSAMTAB with the records, we display it using the same technique as in the previous sample. There is no need to do any VGETs for the variables used in the panel, as these are retrieved from the shared pool automatically. The only VGET we need is for VLRECL – a

variable containing the maximum length of records in this VSAM – because we need to use this variable later on in the REXX code.

```
if ZCMD = 'PREV' then do
   VOFFSET=VOFFSET-80
   if VOFFSET < 0 then VOFFSET = 0
   signal REREAD
 end
 if ZCMD = 'NEXT' then do
   VOFFSET=VOFFSET+80
   if VOFFSET > VLRECL - 80 then VOFFSET = VLRECL - 80
   SIGNAL REREAD
 end
```

Here we handle the horizontal scroll commands received thanks to the PASSTHRU option in our command table. If we receive one of them, we modify the VOFFSET accordingly and loop back to update the VSAMTAB table and redisplay the panel.

6.4.5 Message TUT004

You know what to do, add a message TUT004 stating something like ‘CANNOT READ VSAM’

6.4.6 VSAMASM assembler module

Thanks to the REXX middle layer, this module has a rather straightforward job – to read records from VSAM defined by DD statement INVSAM and put columns starting at ISPF variable VOFFSET to ISPF table VSAMTAB, and to fill in general properties of the VSAM file into corresponding ISPF variables.

To accomplish this, there are 3 main tasks than need to be done:

- Acquiring access to the ISPF variables used for input and output
- Collecting the VSAM file properties
- Reading the records and putting them into the table

You can find an example of the whole module in Appendix D: VSAMASM module. Copy it into your ASSEMBLE.SOURCE library now. The program execution starts and ends with code taking care of the standard linking conventions, with the main part of the program consisting of 4 internal routines. Before we have a look into them, let us examine how ISPF commands are called from Assembler code. There are 2 routines provided for this purpose by ISPF, ISPEXEC and ISPLINK. To call ISPF, you need to load one of these routines and pass control to it by using standard routine call conventions (easiest to do with the CALL macro) The difference between the two routines is in the parameters you need to pass to them:

ISPEXEC expects 2 parameters – length of the ISPF command, and full text of the ISPF command you wish to execute.

ISPLINK expects a variable list of arguments – first argument is a 8-character service name of the ISPF command, for example `select`, `vput`, `display`, etc. You can find the service name for each command in ISPF Services Guide[9] The following arguments are the arguments of the command. For example, to get variable var1 from profile pool, we would need to call ISPLINK like this:

```
LOAD  EP=ISPLINK
LR    15,0
CALL  (15),(VGET,VARNAME,POOL),V
```

And in the data definitions:

```
VGET      DC     CL8'VGET'
VARNAME   DC     CL8'VAR1'
POOL      DC     CL8'PROFILE'
```

This is the preferable way if we need a dynamic command – for example, if we needed to change the value of VARNAME between each call to retrieve different variables from the pool. You could of course use ISPEXEC and change the variable name in the command string passed to it, but using ISPLINK will make the code easier to read. Now, let us have a look at the main program sections:

PREPISPF – This section of code is preparing ISPF variables for use in the rest of the module. Since Assembler does not share variables with ISPF (and in fact does not really have variables, but only labeled memory blocks) to obtain or modify an ISPF variable, we need to first tell ISPF where it should put the contents of the variable, and what data format (binary, packed decimal, character) it should use. For this we need to use the `VDEFINE` command.

```
LOAD  EP=ISPEXEC
ST    R0,ISPEXADR
LOAD  EP=ISPLINK
ST    R0,ISPLIADR
```

First we load the ISPF service routines and store their entry point addresses. In this sample, we will not really be using `ISPEXEC`, but usually you would want to have both routines, as both have their advantages depending on what you need to do.

```
L     R15,ISPLIADR
CALL  (15),(VDEFINE,N_OFFSET,V_OFFSET,FIXED,LEN4),VL
L     R15,ISPLIADR
CALL  (15),(VGET,N_OFFSET,PROFILE),VL
```

The rest of PREPISPF consists of VDEFINE calls for each variable used. For each of these calls, there are constants defined in the data area, where N_VAR is 8-character name of a variable, and V_VAR is the actual area in memory to which the variable should be mapped with length corresponding to the last parameter in the call. The last but one parameter determines the type, FIXED means fixed binary, CHAR means character string. For full list of parameters, refer to ISPF Service Guide chapter 2.28 [9].

VERIFY – This step refreshes the VSAMs RBA, an action that prevents us from getting incorrect results after repeated access which has nothing to do with ISPF. Feel free to copy the code.

GETSTATS – This section extracts some of the VSAM file properties available through the SHOWCB macro, and saves them in ISPF variables.

```
SHOWCB ACB=INACB,AREA=V_KEYLEN,OBJECT=DATA,                    +
       FIELDS=(KEYLEN,NLOGR,LRECL,ENDRBA,RKP),                 +
       LENGTH=20
```

The SHOWCB macro is set to put 5 4-byte fields into 20 bytes of memory starting at V_KEYLEN. In PREPISPF these 20 bytes have been mapped by VDEFINEs to 5 ISPF variables. (see the data section of the program) This is a DFSMS macro, and not connected with ISPF in any way. Access to macros like this one is one reason why we use assembler here.

```
L     R15,ISPLIADR
CALL  (15),(VPUT,N_KEYLEN,PROFILE)
L     R15,ISPLIADR
CALL  (15),(VPUT,N_NLOGR,PROFILE)
```

Calls like these are made for all 5 affected variables to store them in the PROFILE level shared pool. Notice that the name of the variable is enough, as ISPF knows which memory area it should read and what data format to expect on input thanks to the VDEFINEs.

READVSAM – This is the read loop, going through records in the VSAM using GET in keyed sequential locate mode to load the records into a system buffer (to avoid having to worry about length of the record) and moving 80 bytes starting at VOFFSET from each record into rows of the VSAMTAB table. We will not go into the details of VSAM processing here, as this is not connected to ISPF and learning that would be a topic for another tutorial. However what is interesting for us is the way the results are put into the table:

```
MOVEKEY  EQU   *                KEY VARIABLE ADDR
         LA    R2,V_KEY         KEY VARIABLE ADDR
         L     R4,INPUTADR      ADDR OF INPUT RECORD IN BUFFER
         A     R4,V_RKP         KEY OFFSET
         LHI   R3,80
         L     R5,V_KEYLEN
```

```
            ICM   R5,B'1000',=C' '       FILL CHARACTER FOR MVCL
            MVCL  R2,R4               MOVE KEY TO VKEY VARIABLE
...
   MOVEREC  EQU   *
            LA    R2,V_REC              RECORD VAR ADDR
            L     R4,INPUTADR           ADDR OF INPUT RECORD IN BUFFER
            A     R4,V_OFFSET           OFFSET OF CURRENT HORIZ SCROLL
            LHI   R3,80                 LENGTH OF VREC VARIABLE
            L     R5,RECLEN             LENGTH OF RECORD IN BUFFER
            S     R5,V_OFFSET            - OFFSET = REMAINING LENGTH
            ICM   R5,B'1000',=C' '       FILL CHARACTER FOR MVCL
            MVCL  R2,R4               MOVE PART OF RECORD TO VREC VAR
```

These code sections move the record key into V_KEY (or its 80 characters if it is longer, as V_KEY is defined as CL80) and the 80 bytes of the record starting at VOFFSET to V_REC.

```
   ADDTOTAB EQU   *
            L     R15,ISPLIADR
            CALL  (15),(TBMOD,VSAMTAB,,ORDER),VL
```

By calling TBMOD, value in variable VREC is inserted into VSAMTAB at position given by key variable VKEY. Since we have defined by VDEFINE that V_KEY and V_REC hold the actual values of these variables, the data we have moved there will be used. ISPF will convert this data from the format we have specified in VDEFINE to its internal format, allowing it to for example to display numbers given to it as binary in the panels (where they have to show as characters).

After finishing writing the VSAMASM module, compile and link it using JCL TUTCOMP and TUTLINK in our JCL library. This will create a load module VSAMASM in the hlq.TUT.ASSEMBLE.LOADLIB library. Now we have all the pieces together, try to run the VSAM utility and use it to examine a KSDS dataset in your system. If you do not have any dataset to test with, use JCL TUTVSAM to create a simple KSDS VSAM (80 columns, so horizontal scrolling will not work) or modify this JCL by changing DATA DD statement to refer to an external sequential dataset with longer record length, containing records in ascending order. Do not forget changing the record lengths in first steps SYSIN to the length of records in your external dataset.

7 Conclusion

If you followed the tutorial, you should now have a fair understanding of how to code simple ISPF dialogs. There is still a lot to learn, if you want to be developing applications using the ISPF dialog manager on a professional level. The techniques shown here are meant to give you an introduction to coding for ISPF, give you some idea about what ISPF can do for your application and help you decide whether you are interested in what it has to offer. Depending on the needs of your application it is likely you will need to use commands and techniques not shown here, but you should now have some base ISPF knowledge you can build upon. In the present mainframe world, chances are that if you are developing ISPF dialogs, you are more likely to be extending or modifying an existing application, than starting a brand new mainframe product. In that case, you should take time to notice and understand the coding patterns used in the existing code and, where possible, replicate these. There can often be different ways to do the same thing, and consistency can be very helpful for anyone reading the code later (including yourself).

For everyone reading this paper, no matter if you went through the tutorial or not, I hope it has helped you to gain an understanding of what ISPF is and what it can do, as well as learn about the context in which it operates. With the new generation of mainframes retaining an important niche in present-day information systems, mainframe technologies are here to stay. And while there is a trend toward off-platform user interfaces like web-based UIs, many system tools keep using ISPF to interact with their users and this is not likely to change anytime soon.

I believe this paper and the tutorial contained in it will prove useful to the new mainframers who are just beginning their careers as well as their more experienced colleagues who have been recently assigned a development task that requires ISPF programming knowledge. I am sure there are other areas in the mainframe world where such tutorial materials could be helpful, especially in the present era when the first generation of mainframers is reaching retirement age and companies are trying to set up training programs teaching mainframe skills to the next generation of mainframe operators and developers. This new generation has been raised on the PC with Windows and UNIX and without education provided by the hiring companies it is unlikely to possess the necessary mainframe skills. For that reason I believe educational materials geared towards tech-savvy people who come from other platforms and have little to no mainframe experience will be in greater demand than ever.

8 References

[1] Ebbers, M. O'Brien, W. Ogden, B. *Introduction to the New Mainframe: z/OS Basics,* Redbooks, 2011, 3rd edition, ISBN 0738433012 available at: http://www.redbooks.ibm.com/redbooks/pdfs/sg246366.pdf

[2] Fosdick, H. *Using IBM`S ISPF Dialog Manager,* Van Nostrand Reinhold Company Inc 1987, ISBN 0-442-22626-8877

[3] International Business Machines Corporation *DFSMS Macro Instructions for Data Sets*, IBM publications 2009, 9th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[4] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Dialog Developer's Guide and Reference z/OS Version 1 Release 9.0*, IBM publications 2007, 7th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[5] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Edit and Edit Macros z/OS Version 1 Release 10.0*, IBM publications 2008, 9th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[6] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Messages and Codes z/OS Version 1 Release 10.0,* IBM publications 2008, 9th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[7] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Reference Summary z/OS Version 1 Release 10.0,* IBM publications 2008, 9th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[8] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Services Dialog Tag Language Guide and Reference*, IBM publications 2006, *6th edition*, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[9] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) Services Guide z/OS Version 1 Release 10.0*, IBM publications 2008, 9th edition, recent version available at: http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[10]International Business Machines Corporation *Interactive System Productivity Facility (ISPF) User's Guide Volume I z/OS Version 1 Release 10.0*, IBM publications 2008, 9th edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[11] International Business Machines Corporation *Interactive System Productivity Facility (ISPF) User's Guide Volume II z/OS Version 1 Release 10.0,* IBM publications 2008, 9th edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[12] International Business Machines Corporation *Interactive System Productivity Facility Getting Started,* IBM publications 1995, 2nd edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[13] International Business Machines Corporation *MVS Programming: Assembler Services Reference, Volume 1 (ABEND-HSPSERV),* IBM publications 2008, 11th edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[14] International Business Machines Corporation *MVS Programming: Assembler Services Reference, Volume 2 (IARR2V-XCTLX),* IBM publications 2008, 11th edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[15] International Business Machines Corporation *TSO/E REXX Reference*, IBM publications 2006, 8th edition, recent version available at:
http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.f54dg00%2Fispzdg90.htm

[16] International Business Machines Corporation *z/Architecture Principles of Operation*, IBM publications 2005, 5th edition, available at: http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/dz9zr002/CCONTENTS

[17] International Business Machines Corporation *z/OS Basic Skills Information Center Mainframe concepts,* IBM publications 2005, 2008, available at:
http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zmainframe/zmainframe_book.pdf

[18] International Business Machines Corporation *z/OS V1R12.0 UNIX System Services User's Guide,* IBM publications 2010, available at:http://publibz.boulder.ibm.com/epubs/pdf/bpxza4b0.pdf

[19] International Business Machines Corporation *z/VM General Information,*

IBM publications 2010, version 6 release 1, available at: http://www.vm.ibm.com/pubs/hcsf8b31.pdf

9 Terminology dictionary

Term	Shortcut	Explanation [source]
3270 protocol		Protocol used for communication between a mainframe and a terminal
Dataset Name	DSN	Full name of a z/OS dataset[1]
Data Definition Name	DDN	A name of a data definition to which an input or output dataset can be assigned (for example by JCL DD statement)
Disk Operating System	DOS	Early operating system on System/360 mainframes, predecessor of VSE; Not related to PC operating systems like MS-DOS[17]
Hierarchical File System dataset	HFS	Older type of dataset used for storing USS file systems[18]
High Level Assembler	HLASM	Mainframe assembler – a low level programming language (in spite of the name) [13]
High Level Qualifier	HLQ	First qualifier in a dataset name in z/OS[1]
Integrated System Productivity Facility	ISPF	A multifaceted development tool set for the z/OS operating system [4]
Job Control Language	JCL	Language used for writing jobcards [15]
Jobcard		JCL code that tells the operating system how to execute a batch job[1]
Lowest Level Qualifier	LLQ	Last qualifier in a dataset name in z/OS[1]
Mainframe		A type of computer built for high performance, reliability, security and availability [1]
Multiple Virtual Storage	MVS	Older operating system on IBM mainframes [1]
Panel		The formatted textual content of a terminal 3270 screen, with marked fields for user input [4]
Partitioned dataset	PDS	Type of dataset that acts as a library, containing members[1]

Term	Shortcut	Explanation [source]
Portable Operating System Interface	POSIX	Standard interface of UNIX-like operating systems [http://www.pasc.org/plato/]
Restructured Extended Executor	REXX	Interpreted programming language on the mainframe, with ability to call different command environments and optional compilation [15]
Sequential dataset		Type of dataset with simple sequential record structure[1]
Terminal		A display device used to connect to the mainframe and perform user input and output [1]
Time Sharing Option	TSO	A component of z/OS handling user sessions[1]
Transaction Processing Facility	TPF	One of the operating system on IBM mainframes, focused on transaction processing [1]
UNIX System services	USS	One of the operating system on IBM mainframes, an implementation of Linux for the mainframe [1]
Virtual Machine	VM	One of the operating system on IBM mainframes, focused of virtualization [1]
Virtual Storage Access Method	VSAM	Type of dataset access method and dataset organization supporting this method [1]
Virtual Storage Extended	VSE	One of the operating system on IBM mainframes [1]
zArchitecture		64-bit hardware architecture of the most recent mainframes[1]
zEnterprise		Most recent model of the zArchitecture mainframes[1]
zSeries File System	zFS	Newer type of dataset used for storing USS file systems[18]
z/TPF	z/TPF	Latest version of the TPF operating system on IBM mainframes, focused on transaction processing [1]
z/VM	z/VM	Latest version of the VM operating system on IBM mainframes, focused of virtualization [1]
z/VSE	z/VSE	Latest version of the VSE operating system on IBM mainframes [1]

10 Appendix

10.1 Appendix A - JCLs used in this tutorial

10.1.1 TUTALLOC

Allocate libraries used throughout the course. Copy into a dataset member on the mainframe, set fields marked in red. Submit.

```
//TUTALLOC JOB accountno,name,REGION=0M,
//         CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*********************************************************************
//*        SET YOUR HIGH LEVEL QUALIFIER                              *
//         SET  HLQ=your-hlq                                          *
//*********************************************************************
//ALLOC    EXEC PGM=IEFBR14
//ASMSRC   DD   DSN=&HLQ..TUT.ASSEMBLE.SOURCE,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//ASMOBJ   DD   DSN=&HLQ..TUT.ASSEMBLE.OBJMOD,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//ASMLOAD  DD   DSN=&HLQ..TUT.ASSEMBLE.LOADLIB,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=U,LRECL=0,BLKSIZE=32000)
//ASMMAC   DD   DSN=&HLQ..TUT.ASSEMBLE.MACLIB,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//EXEC     DD   DSN=&HLQ..TUT.EXEC,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//ISPPLIB  DD   DSN=&HLQ..TUT.ISPF.PANEL,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//ISPMLIB  DD   DSN=&HLQ..TUT.ISPF.MESSAGE,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//ISPTAB   DD   DSN=&HLQ..TUT.ISPF.TABLE,DISP=(MOD,CATLG),
//   SPACE=(CYL,(1,2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
```

10.1.2 TUTCOMP

Compile assembler source. Copy into a dataset member on the mainframe and set fields marked in red. Use by putting name of the assembler source module into the blue field and submitting

```
//TUTCOMP  JOB accountno,name,REGION=0M,
//         CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
```

```
//         SET M=VSAMASM
//         SET HLQ=your-hlq
//A       EXEC PGM=ASMA90
//SYSIN    DD  DSN=&HLQ..TUT.ASSEMBLE.SOURCE(&M),DISP=SHR
//SYSLIB   DD  DSN=&HLQ..TUT.ASSEMBLE.MACLIB,DISP=SHR
//         DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=&HLQ..TUT.ASSEMBLE.OBJMOD(&M),DISP=SHR
//SYSUDUMP DD  SYSOUT=*
```

10.1.3 TUTLINK

Link assembler object module(s) into a load module. Copy into a dataset member on the mainframe and set fields marked in red. Use by modifying the blue fields and submitting

```
//TUTLINK    JOB accountno,name,REGION=0M,
//         CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//         SET   HLQ=your-hlq
//LINK     EXEC  PGM=IEWL,PARM='XREF,LIST'
//SYSLIB   DD DISP=SHR,DSN=&HLQ..TUT.ASSEMBLE.LOADLIB
//         DD DISP=SHR,DSN=&HLQ..TUT.ASSEMBLE.OBJMOD
//SYSUT1   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*,OUTLIM=20000
//INSTOBJ  DD DISP=SHR,DSN=&HLQ..TUT.ASSEMBLE.OBJMOD
//SYSLMOD  DD DISP=SHR,DSN=&HLQ..TUT.ASSEMBLE.LOADLIB
//SYSLIN   DD *
    INCLUDE  INSTOBJ(VSAMASM)
    INCLUDE  SYSLIB(VSAMASM)
    ENTRY    VSAMASM
    NAME     VSAMASM (R)
/*
```

10.1.4 TUTVSAM

```
//TUTVSAM  JOB accountno,name,REGION=0M,
//         CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//DEFINE   EXEC PGM=IDCAMS,COND=(0,NE)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 DELETE   hlq.TUT.TESTKSDS                          -
            CLUSTER                                 -
            PURGE
```

```
 SET MAXCC = 0
 DEFINE   CLUSTER (                                  -
             NAME (hlq.TUT.TESTKSDS)                 -
                    CYL     (1 1)                    -
                    FSPC    (0 10)                   -
                    KEYS    (11 0)                   -
                    RECSZ   (80 80)                  -
                    SHR     (3 3)                    -
                    SPEED                            -
                    )                                -
             DATA    (                               -
                    CISZ    (800)                    -
                    )                                -
             INDEX   (                               -
                    CISZ    (400)                    -
                    )
/*
//FILL     EXEC PGM=IDCAMS,COND=(0,NE)
//SYSPRINT DD  SYSOUT=*
//VSAM     DD  DSN=hlq.TUT.TESTKSDS,DISP=SHR
//DATA     DD *
DEUTER 31:6O BE STRONG AND COURAGEOUS. DO NOT BE AFRAID OR TERRIFIED BECAUSE OF
JOSHUA 1:9   THE LORD GAVE THIS COMMAND TO JOSHUA SON OF NUN: "BE STRONG AND
ROMANS 1:17  AND JESUS SAID UNTO THEM ... , "IF YE HAVE FAITH AS A GRAIN OF MUST
1 JOHN 4:18  THERE IS NO FEAR IN LOVE; BUT PERFECT LOVE CASTETH OUT FEAR ...
/*
//SYSIN    DD *
 REPRO -
    INFILE(DATA) -
    OUTFILE(VSAM)
/*
```

10.2 Appendix B – ISPF Panel definitions

Reduced to less than 80 characters width, for improved readability

10.2.1 MAIN

```
)PANEL
)ATTR DEFAULT(@+_)
      /*  @ TYPE(TEXT) INTENS(HIGH)  default for hilited text        */
      /*  + TYPE(TEXT) INTENS(LOW)   default for normal text         */
```

```
      /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT) default entry field*/
      ! TYPE(PS)                       /* Point and Shoot text          */
)BODY
@------------------  ISPF TUTORIAL APPLICATION -------------------------------
@SELECTION ===>_ZCMD                                                        +
+
+   Select the action you would like to perform:

   @1+!Say Hello World+
   @2+!List Datasets+
   @3+!Examine KSDS+
   @X+!Exit+
+
+
)INIT
  .HELP = HELPMAIN              /* insert name of help panel  */
)PROC
&ZSEL=TRANS(TRUNC(&ZCMD,'.')
             1,'CMD(TUTHELLO)'
             2,'CMD(TUTDSLST)'
             3,'CMD(TUTVSAM)'
             X,'EXIT'
              )
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL(1)
  FIELD(ZPS00002) VAR(ZCMD) VAL(2)
  FIELD(ZPS00003) VAR(ZCMD) VAL(3)
  FIELD(ZPS00004) VAR(ZCMD) VAL(4)
)END
```

10.2.2 HELPMAIN

```
)PANEL
)ATTR DEFAULT(@+_)
      /*  @ TYPE(TEXT) INTENS(HIGH)  default for hilited text        */
      /*  + TYPE(TEXT) INTENS(LOW)   default for normal text         */
      /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT) default entry field*/
      ! TYPE(PS)                       /* Point and Shoot text          */
)BODY
@------------------  ISPF TUTORIAL APPLICATION -------------------------------
```

```
$Selection ===>_ZCMD
+
+ The ISPF  Tutorial Application consists of 3 utilities, each having
+ a wholly different use. See the topics listed for help on each utility.
+
+ You can look at the following topics in sequence, or select each one
+ by number:
+
    @1+!Say Hello World Utility+
    @2+!List Datasets Utility+
    @3+!Examine KSDS Utility+
+
)PROC
     &ZSEL = TRANS( &ZCMD
                  1,HELP1
                  2,HELP2
                  3,HELP3
                  )
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL(1)
  FIELD(ZPS00002) VAR(ZCMD) VAL(2)
  FIELD(ZPS00003) VAR(ZCMD) VAL(3)
)END
```

10.2.3 HELP1

```
)PANEL
)ATTR DEFAULT(@+_)
      /*  @ TYPE(TEXT) INTENS(HIGH)  default for highlighted text        */
      /*  + TYPE(TEXT) INTENS(LOW)   default for normal text         */
      /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT) default entry field*/
      ! TYPE(PS)                       /* Point and Shoot text          */
)BODY
@------------------  ISPF TUTORIAL APPLICATION -------------------------------
$Selection ===>_ZCMD
+
+ The Hello World application shows the user a message, saying:
+
+          @Hello, World!+
+
+ While not usefull in any way, such a function is often used in tutorials
```

```
+ to show the basics of a programming language or environment.
)END
```

10.2.4 DSLIST

```
)PANEL
)ATTR DEFAULT(@+_)
)BODY
@------------------  ISPF TUTORIAL APPLICATION -------------------------------
$Option ===>_ZCMD                                                          +
+
+
+ Select HLQ for the dataset listing utility:
+
+  _dshlq                                        +
+
+ size units: _z+   @1+ Tracks
+                  @2+ Cylinders
+                  @3+ Bytes
+                  @4+ Megabytes
+
+
)INIT
  .HELP = HELP2
  .ZVARS = sel
)PROC
  VER (&dshlq,DSNAME)
  VER (&dshlq,LEN,GT,0)
  VER (&sel,LIST,1,2,3,4)
)END
```

10.2.5 DSLIST2

```
)PANEL
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                       /* Variable Output Information   */
  ¢ TYPE(CH)                       /* Column Heading                */
  $ TYPE(FP)                       /* Field Prompt                  */
  ? TYPE(ET)                       /* Emphasized Text               */
  * TYPE(LI) PADC(USER)            /* List Item                     */
  ! TYPE(PS)                       /* Point and Shoot text          */
  # TYPE(LI) PADC(USER) JUST(RIGHT)/* List Item Right Justified     */
  ½ TYPE(AB)                        /* Action Bar                    */
```

```
  ¼ TYPE(ABSL)                        /* Action Bar Separator Line    */
)ABC DESC('ACTIONS') MNEM(1)
  PDC DESC('SORT BY NAME')
  PDC DESC('CHANGE HLQ')
  ACTION RUN('ISRROUTE') PARM('SELECT PANEL(HELPMAIN)')
  PDC DESC('EXIT')
  ACTION RUN('EXIT')
)ABCINIT
  .ZVARS=HELPSEL
)BODY
½ ACTIONS+
½------------------  ISPF TUTORIAL APPLICATION --------------------------------
@Command ===>_ZCMD                                          +Scroll ===> _SCR
+
+   Listing Datasets for: %dshlq
+
+   Type command?'SORT <column>'+or click column header to sort the list
+
!S  !Name¢               !Dataset type¢  !Size%unit   ¢ !Member count¢
)MODEL clear(sel)
_z+*llq                 +|#dsorg        +|#size        +|#count    +
)INIT
  .ZVARS = sel
  .HELP = HELP2
)PROC
  &COMMAND =&ZCMD
)PNTS
  FIELD(ZPS00001) VAR(ZCMD) VAL('SORT NAME')
  FIELD(ZPS00002) VAR(ZCMD) VAL('SORT DSORG')
  FIELD(ZPS00003) VAR(ZCMD) VAL('SORT SIZE')
  FIELD(ZPS00004) VAR(ZCMD) VAL('SORT COUNT')
)END
```

10.2.6 DSLIST3

```
)PANEL
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                        /* Variable Output Information   */
)BODY
@Command ===>_ZCMD
+
```

```
+   Details for Dataset %dsn
+
+                Size:    %size        +
+                Dsorg:   %dsorg       +
+                Members: %count       +
)INIT
  .HELP = HELP2
)END
```

10.2.7 VSAM1

```
)PANEL
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                        /* Variable Output Information   */
  ! TYPE(PS)                        /* Point and Shoot text          */
  @ TYPE(PT)                        /* Panel Title                   */
  $ TYPE(FP)                        /* CUA Attribute for Field Prompt */
)BODY
@------------------------------  KSDS VIEWER ---------------------------------
$Command ===>_ZCMD                                                        +
+
+
+ WHICH VSAM DO YOU WISH TO VIEW?
+
+ DSN: _INDSN                                                              +
+
)INIT
)PROC
VER (&indsn,NONBLANK,DSNAME)
)PNTS
)END
```

10.2.8 VSAM2

```
)PANEL
)ATTR DEFAULT(@+_)
  % TYPE(VOI)                        /* Variable Output Information   */
  ! TYPE(PS)                        /* Point and Shoot text          */
  @ TYPE(PT)                        /* Panel Title                   */
  ¼ TYPE(VOI) PADC(-) JUST(RIGHT)  /* Normal Field            */
  $ TYPE(FP)               /* CUA Attribute for Field Prompt          */
  * TYPE(LI) PADC(USER)             /* List Item                     */
)BODY
```

```
@------------------------------  KSDS VIEWER ----------------------------------
+                                  %VCOLINFO                                   +
$Command ===>_ZCMD                                          $Scroll ===>_SCR
+
+ VIEWING VSAM %INDSN                                                         +
+
+ KEY LENGTH:    %VKEYLEN +        RECORD COUNT:  %VNLOGR  +
+ KEY OFFSET:    %VRKP    +        RECORD LENGTH: %VLRECL  +
+                                  ENDING RBA:    %VENDRBA +
@-------------------------------------------------------------------------------
)MODEL
*VREC
)INIT
  &ZCMD = ''
)REINIT
  &ZCMD = ''
  REFRESH(ZCMD,VCOLINFO)
)PROC
)PNTS
)END
```

10.3 Appendix C: REXX Execs

10.3.1 Start

```
/********************************** REXX ************************************/
/* ALLOCATE LIBRARIES                                                       */
/* LAUNCH PANEL MAIN                                                        */
/****************************************************************************/
ADDRESS ISPEXEC
"LIBDEF ISPPLIB DATASET ID('VASMI03.TEST.TUT.ISPF.PANEL')"
"LIBDEF ISPMLIB DATASET ID('VASMI03.TEST.TUT.ISPF.MESSAGE')"
"LIBDEF ISPTLIB DATASET ID('VASMI03.TEST.TUT.ISPF.TABLE')"
"LIBDEF ISPLLIB DATASET ID('VASMI03.TEST.TUT.ASSEMBLE.LOADLIB')"
ADDRESS TSO
"ALLOC F(REXXLIB1) SHR REUSE DA('VASMI03.TEST.TUT.EXEC')"
"ALTLIB ACT APPLICATION(EXEC) LIBRARY(REXXLIB1)"
ADDRESS ISPEXEC
"CONTROL ERRORS RETURN" /* COMMENT OUT IF YOU NEED TO FIND AN ERROR IN CODE */
"SELECT PANEL(MAIN) NEWAPPL(TUT) PASSLIB"
ADDRESS TSO "ALTLIB DEACT APPLICATION(EXEC)"
```

10.3.2 TUTHELLO

```
/********************************** REXX ************************************/
ADDRESS ISPEXEC 'DISPLAY PANEL(MAIN) MSG(TUT000)'
```

10.3.3 TUTDSLST

```
/****************************** REXX ************************************/
ADDRESS ISPEXEC
restart:
zcmd=''
/*"CONTROL ERRORS RETURN"*/
'TBERASE TUTTAB'
'TBCREATE TUTTAB NOWRITE
            KEYS(LLQ) NAMES(SIZE COUNT DSORG)'
dshlq = ''
sel = '1'
'DISPLAY PANEL(DSLIST)' /*user input: sel and libhlq*/
if rc>0 then exit
SELECTION = sel
DSLEVEL =DSHLQ".*"
```

```
"LMDINIT LISTID(IDV) LEVEL(&DSLEVEL)"
if rc>0 then exit
LIBRARIES=0
do forever
   "LMDLIST LISTID("IDV") OPTION(LIST) DATASET(DSNAME) STATS(YES)"
   if rc = 0 then do
     "LMINIT DATAID(IDL) DATASET('"DSNAME"') ENQ(SHR) ORG(DSORG)"
     if RC > 4 then dsorg = '-'
     if dsorg="PO" then do
       count = 0
       memname = '        '
       "LMOPEN DATAID("IDL")"
       do forever
         "LMMLIST DATAID("IDL") OPTION(LIST) MEMBER(MEMNAME)"
         if rc = 0 then do
           count = count + 1
         end
         else leave
       end
       "LMCLOSE DATAID("IDL")"
     end
     else count='-'
     llq = GETLLQ(dsname)
     size = ZDLSIZE /*size in tracks;PART OF STATS GENERATED BY LMDLIST */
     if size = '        ' then size = '-'
     else do
       size = size + 0 /*conv to number for better display*/
       unit = "(Tracks)"
       if (selection = 2) then do
         devtype = ZDLDEV /*device type;PART OF STATS GENERATED BY LMDLIST */
         if devtype = 3375 then size = size / 12
         else if devtype = 3380 then size = size / 15
         else if devtype = 3390 then size = size / 15
         else if devtype = 9345 then size = size / 15
         else size = '-'
         unit = "(Cyls)"
       end
       if (selection > 2) then do
         devtype = ZDLDEV /*device type;PART OF STATS GENERATED BY LMDLIST */
         if devtype = 3375 then size = size * 35616
```

```
          else if devtype = 3380 then size = size * 47476
          else if devtype = 3390 then size = size * 56664
          else if devtype = 9345 then size = size * 46456
          else size = '-'
          unit = "(Bytes)"
        end
        if (selection = 4) then do
          size = size / (1024 * 1024)
          unit = "(MB)"
        end
      end
      if size <> '-' then size = FORMAT(size,10,2)
      if count <> '-' then count = FORMAT(count,10,0)
      "TBMOD TUTTAB"
    end
    else leave
end
lastsort = 'SORT NAME'
sortorder = 'a'
sorttype = 'c'
sortfield = 'llq'
redisplay:
"TBSORT TUTTAB FIELDS("sortfield","sorttype","sortorder")"
"TBVCLEAR TUTTAB"
"TBTOP TUTTAB"          /*POINTER BACK TO START OF TABLE*/
"TBDISPL TUTTAB PANEL(DSLIST2)"
if helpsel == 1 then zcmd = 'SORT NAME'
if helpsel == 2 then zcmd = 'SORT DSORG'
if helpsel == 3 then zcmd = 'SORT SIZE'
if helpsel == 4 then zcmd = 'SORT COUNT'
if helpsel == 5 then zcmd = 'CHANGE'
helpsel = ''
message = ''
if RC=0 then do
  if zcmd == 'SORT NAME' then call resort('llq' 'c')
  else if zcmd == 'SORT DSORG' then call resort('dsorg' 'c')
  else if zcmd == 'SORT SIZE' then call resort('size' 'n')
  else if zcmd == 'SORT COUNT' then call resort('count' 'n')
  else if zcmd <> '' then do
    message='MSG(TUT001)'
```

```
  end
  else if sel == 'S' then do
    dsn = dshlq'.'llq
    "DISPLAY PANEL(DSLIST3)"
  end
  else if sel <> '' then do
     message='MSG(TUT002)'
  end
  signal redisplay
end
"LMDLIST LISTID("IDV") OPTION(FREE)"
"TBEND TUTTAB"
zcmd=''
exit
RESORT:
parse arg sfld stp
sortfield = sfld
sorttype = stp
if zcmd == lastsort & sortorder == 'a' then
   sortorder = 'd'
else
   sortorder = 'a'
lastsort = zcmd
signal redisplay
GETLLQ:
parse arg fulldsname
hlq = ''
targethlq = '.'dshlq
do until hlq == targethlq
  parse var fulldsname q'.'fulldsname
  hlq = hlq'.'q
end
return fulldsname
```

10.3.4 TUTVSAM

```
 /******************************** REXX ***************************/
address ISPEXEC
"CONTROL ERRORS RETURN"
"TBCREATE VSAMTAB KEYS(VKEY) NAMES(VREC) NOWRITE REPLACE"
"DISPLAY PANEL(VSAM1)"
```

```
RESTART:
IF RC>0 THEN EXIT
address TSO "ALLOC F(INVSAM) SHR REUSE DA('"INDSN"')"
if RC>0 then signal IOError
VOFFSET = 0
/*READ VSAM AND FILL IN THE VSAMTAB AND OTHER VARS USING ASM PGM*/
"VPUT VOFFSET PROFILE"
"SELECT PGM(VSAMASM)"
if RC>0 then signal IOError
/**/
SCR = "PAGE"
"TBSORT VSAMTAB FIELDS(VKEY)"
"TBVCLEAR VSAMTAB"
"TBTOP VSAMTAB"        /*POINTER BACK TO START OF TABLE*/
"VGET VLRECL"
VCOLINFO = 'Column 'VOFFSET' to 'VOFFSET+80' of 'VLRECL
"TBDISPL VSAMTAB PANEL(VSAM2)"
AFTERDISPLAY:
if ZCMD = 'PREV' then do
  VOFFSET=VOFFSET-80
  if VOFFSET < 0 then VOFFSET = 0
  signal REREAD
end
if ZCMD = 'NEXT' then do
  VOFFSET=VOFFSET+80
  if VOFFSET > VLRECL - 80 then VOFFSET = VLRECL - 80
  signal REREAD
end
exit
REREAD:
VCOLINFO = 'Column 'VOFFSET' to 'VOFFSET+80' of 'VLRECL
"VPUT VOFFSET PROFILE"
"SELECT PGM(VSAMASM)"
if RC>0 then signal IOError
"TBTOP VSAMTAB"        /*POINTER BACK TO START OF TABLE*/
"TBDISPL VSAMTAB"
signal AFTERDISPLAY
IOERROR:
"DISPLAY PANEL(VSAM1) MSG(TUT004)"
signal restart
```

10.4 Appendix D – Assembler source codes

10.4.1 VSAMASM module

```
         TITLE 'VSAMASM'
         EJECT
*************************************************************************
*        PROGRAM NAME: VSAMASM                                          *
*        DESCRIPTION:  LOAD A KSDS VSAM, AND WRITE ITS BASIC            *
*                      PARAMETERS AND ITS CONTENTS TO A PANEL           *
*                      EXPECTS TO BE CALLED BY THE TUTORIAL APP         *
*                                                                       *
*        AUTOR:        MICHAL VASAK                                     *
*                                                                       *
*        LAST MOD ON:                                                   *
*                                                                       *
*        REGISTER  R0                      R8                           *
*        USAGE     R1                      R9                           *
*                  R2                      R10 ROUTINE RETURN ADDR      *
*                  R3                      R11                          *
*                  R4                      R12 BASE                     *
*                  R5                      R13 SAVE AREA                *
*                  R6                      R14                          *
*                  R7                      R15                          *
*************************************************************************
*-----------------------------------------------------------------------*
*        PROGRAM INITIALIZATION                                         *
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
VSAMASM  CSECT
* SAVE REGISTERS
         STM   14,12,12(13)                    SAVE REGISTES

* ESTABLISH ADDRESSABILITY FOR PROGRAM
         LR    12,15             ESTABLISH FIRST BASE REGIST
         USING VSAMASM,12

* ESTABLISH NEW SAVE AREA AND POINT 13 TO IT
         GETMAIN R,LV=72
         ST    1,8(,13)          CHAIN NEW AND OLD
```

```
         ST    13,4(,1)           SAVE AREAS TOGETHER
         LR    13,1               POINT R13 TO NEW SA
         LM    0,1,20(13)         RESTORE REGISTERS 0 AND 1
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*-------------------------------------------------------------------*
*        PROGRAM CODE MAIN PART                                     *
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
         OPEN  (INACB)
         TRT   R15,R15
         BNE   ERROROPENINGVSAM
         BAS   R10,PREPISPF
*        BAS   R10,VERIFY
         BAS   R10,GETSTATS
         BAS   R10,READVSAM
         CLOSE (INACB)
         XR    R15,R15
*-------------------------------------------------------------------*
*        EPILOGUE                                                   *
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
* FREE MEMORY, RESTORE CALLING PROGRAMS REGISTERS, AND EXIT
THEEND   DS    0H
         LR    R1,R13             SAVE NEW SA ADDR
         L     R13,4(,13)         POINT TO OLD SA
         FREEMAIN R,LV=72,A=(1)   FREE DYNAMICALLY ACQUIRED SAVE AREA
         LM    R0,R12,20(R13)     RESTORE CALLERS REGS (EXCEPT 15)
```

```
         L     R14,12(R13)
         BR    R14
ERROROPENINGVSAM EQU   *
         DC    D'0'
         LHI   R15,8
         B     THEEND
ERRORRC8 EQU   *
         EX    R0,*
         CLOSE (INACB)
         LHI   R15,8
         B     THEEND
*-------------------------------------------------------------------*
*        PROGRAM CODE SECTIONS                                      *
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
PREPISPF EQU   *
         LOAD  EP=ISPEXEC
         ST    R0,ISPEXADR
         LOAD  EP=ISPLINK
         ST    R0,ISPLIADR
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_OFFSET,V_OFFSET,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VGET,N_OFFSET,PROFILE),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_KEYLEN,V_KEYLEN,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_NLOGR,V_NLOGR,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_LRECL,V_LRECL,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_ENDRBA,V_ENDRBA,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_RKP,V_RKP,FIXED,LEN4),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_KEY,V_KEY,CHAR,LEN80),VL
         L     R15,ISPLIADR
         CALL  (15),(VDEFINE,N_REC,V_REC,CHAR,LEN80),VL
         BR    R10
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
VERIFY   EQU   *
```

```
         VERIFY RPL=INRPL,ACTION=REFRESH
         BR    R10
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
GETSTATS EQU   *
         SHOWCB ACB=INACB,AREA=PROP_VARS,OBJECT=DATA,                  +
               FIELDS=(KEYLEN,NLOGR,LRECL,ENDRBA,RKP),                 +
               LENGTH=20
         L     R15,ISPLIADR
         CALL  (15),(VPUT,N_KEYLEN,PROFILE)
         L     R15,ISPLIADR
         CALL  (15),(VPUT,N_NLOGR,PROFILE)
         L     R15,ISPLIADR
         CALL  (15),(VPUT,N_LRECL,PROFILE)
         L     R15,ISPLIADR
         CALL  (15),(VPUT,N_ENDRBA,PROFILE)
         L     R15,ISPLIADR
         CALL  (15),(VPUT,N_RKP,PROFILE)
         BR    R10
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
READVSAM EQU   *
         CLC   V_NLOGR,X'00000000'
         BE    VSAMEMPT
         LHI   R6,1000               1000 RECORDS MAX - FOR TUTORIAL
INLOOP   GET   RPL=INRPL
         MVC   V_KEY(1),=C' '         CLEAR KEY AND REC FIELDS
         MVC   V_KEY+1(79),V_KEY
         MVC   V_REC(1),=C' '
         MVC   V_REC+1(79),V_REC
         SHOWCB RPL=INRPL,AREA=RECLEN,LENGTH=LEN4,FIELDS=(RECLEN)
         L     R0,V_KEYLEN
         LTR   R0,R0
         BNE   MOVEKEY
MAKEKEY  CVD   R6,V_KEY              NO KEY, SET RRN AS KEY
         B     MOVEREC
MOVEKEY  EQU   *                     KEY VARIABLE ADDR
         LA    R2,V_KEY              KEY VARIABLE ADDR
         L     R4,INPUTADR           ADDR OF INPUT RECORD IN BUFFER
         A     R4,V_RKP              KEY OFFSET
         LHI   R3,80
         L     R5,V_KEYLEN
```

```
         ICM   R5,B'1000',=C' '       FILL CHARACTER FOR MVCL
         MVCL  R2,R4              MOVE KEY TO VKEY VARIABLE
         CLI   RECLEN,X'0'
         BH    RECLENVALID        CHECK IF RECLEN IS FILLED IN
         MVC   RECLEN,V_LRECL     IF NOT, USE LRECL INSTEAD
RECLENVALID EQU *
         L     R5,RECLEN
         S     R5,V_OFFSET
         AHI   R5,-80
         LTR   R5,R5              IF RECLEN < OFFSET + 80
         BL    BUFFEROUT          DO NOT MOVE, YOU GET OUTSIDE BUFFER
MOVEREC  EQU   *
         LA    R2,V_REC               RECORD VAR ADDR
         L     R4,INPUTADR            ADDR OF INPUT RECORD IN BUFFER
         A     R4,V_OFFSET            OFFSET OF CURRENT HORIZ SCROLL
         LHI   R3,80                  LENGTH OF VREC VARIABLE
         L     R5,RECLEN              LENGTH OF RECORD IN BUFFER
         S     R5,V_OFFSET             - OFFSET = REMAINING LENGTH
         ICM   R5,B'1000',=C' '       FILL CHARACTER FOR MVCL
         MVCL  R2,R4              MOVE PART OF RECORD TO VREC VAR
         B     ADDTOTAB
BUFFEROUT EQU   *
         LA    R2,V_REC
         MVC   0(1,R2),=C' '
         MVC   1(79,R2),0(R2)
ADDTOTAB EQU   *
         L     R15,ISPLIADR
         CALL  (15),(TBMOD,VSAMTAB,,ORDER),VL
         BCT   R6,INLOOP
VSAMEOD  EQU   *
         BR    R10
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
VSAMEMPT EQU   *
         BR    R10
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
MVCINSTR MVC   0(0,R2),0(R3)
*-------------------------------------------------------------------*
*        DATA DEFINITIONS                                           *
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*
*--------ISPF
```

```
ISPEXADR DS    F
ISPLIADR DS    F

VDEFINE  DC    CL8'VDEFINE '
VGET     DC    CL8'VGET    '
VPUT     DC    CL8'VPUT    '
TBMOD    DC    CL8'TBMOD   '
ORDER    DC    CL8'ORDER   '
VSAMTAB  DC    CL8'VSAMTAB '
PROFILE  DC    CL8'PROFILE '
LEN4     DC    F'4'
LEN8     DC    F'8'
LEN80    DC    F'80'
PACK     DC    CL8'PACK    '
CHAR     DC    CL8'CHAR    '
FIXED    DC    CL8'FIXED   '

V_OFFSET DS    F
V_KEY    DS    CL80'K'
V_REC    DC    CL80' '

PROP_VARS DS   5F
         ORG   PROP_VARS
V_KEYLEN DS    F
V_NLOGR  DS    F
V_LRECL  DS    F
V_ENDRBA DS    F
V_RKP    DS    F

N_OFFSET DC    C'(VOFFSET)'
N_KEY    DC    C'(VKEY)'
N_REC    DC    C'(VREC)'
N_KEYLEN DC    C'(VKEYLEN)'
N_NLOGR  DC    C'(VNLOGR)'
N_LRECL  DC    C'(VLRECL)'
N_ENDRBA DC    C'(VENDRBA)'
N_RKP    DC    C'(VRKP)'
*--------OTHER
REGS     DS    16F
RECLEN   DS    F
```

```
RPLARG   DS    F
         DS    0D
INACB    ACB   AM=VSAM,MACRF=(KEY,SEQ,IN),DDNAME=INVSAM,              +
               EXLST=EXITS
INRPL    RPL   ACB=INACB,AM=VSAM,AREA=INPUTADR,AREALEN=LEN4,         +
               OPTCD=(SEQ,LOC),ARG=RPLARG
EXITS    EXLST AM=VSAM,EODAD=VSAMEOD
         LTORG
INPUTADR DS    F
         END   VSAMASM
```

10.5 Appendix E – ISPF Messages

10.5.1 TUT00 message dataset

Messages starting with TUT00 will be searched here by ISPF.

```
TUT000 'HELLO, WORLD!' .HELP=* .ALARM=YES
'THE APPLICATION IS GREETING THE WORLD AND ALL ITS PEOPLE'
TUT001 'UNKNOWN COMMAND' .HELP=* .ALARM=YES
'THE COMMAND YOU HAVE TYPED IS NOT VALID IN THIS APPLICATION'
TUT002 'INVALID OPTION' .HELP=* .ALARM=YES
'ONLY THE SELECT (S) INLINE OPTION IS SUPPORTED'
TUT003 'NO DATASETS FOUND' .HELP=* .ALARM=YES
'NO DATASETS WERE FOUND MATCHING THE HLQ YOU SPECIFIED'
TUT004 'CANNOT READ VSAM' .HELP=* .ALARM=YES
'DSN IS INVALID OR VSAM TYPE NOT SUPPORTED'
```