Exit Programming

Duration: 3 Days

Audience:

This course is designed for competent z/OS Systems Programmers with no prior knowledge of the Assembler Language.

Pre-requisites:

Delegates should be existing z/OS Systems Programmers.

A working knowledge of TSO/ISPF is assumed.

Course Objectives

With IBM's latest mainframe announcements, perhaps it is time to consider the fact that z/OS does have a future and some of the skills that have been lost will need replicating. One classic example of this is Assembler. It used to be that every new z/OS Systems Programmer was taught to write Assembler as a matter of course. This has not been the case for quite some time. This means that upgrades to the z/OS operating system are heavily reliant on finding one of the "Old Guys or Gals" to review your exit code and establish if any changes are required for the new z/OS release.

This course takes a unique approach in that the ability to write Assembler is not a prerequisite. Using a combination of NLP/Brain Friendly techniques we take any competent Systems Programmer and teach them to read their own environment's system exits in three days.

Course Content

Module 1: Assembler for non-Assembler Programmers.

Assembler documentation sources.
Statement layout
Compilation differences
Starting and ending a module
General Purpose Registers and conventional use
Working Storage and some field definitions
The lack of data structures
How to redefine memory and fields
Decision making
Input / Output processing

Exit Programming

Moving data around
Mathematics and Editing data
Invoking sub-routines and passing parameters
Exploiting z/OS Services

Module 2: Instruction Types

Traditional instructions

- RR – Format
- RS – Format
- RX – Format
- SI – Format
- SS - Format

More modern interpretations

- 2-byte Format
- 4-byte Format
- 6-byte Format

General instructions versus Privileged Instructions

Module 3: Reference Material

General reference material
Types of exits, and their reference material

- BCP Exits
- CICS Exits, GLUES and TRUES
- IMS Exits
- IPCS Exits
- JES2 Exits
- JES3 Exits
- RACF Exits
- RMF Exits
- SMF Exits
- SMS Exits
- TSO Exits
- VTAM Exits
- z/OS Exits include MPFLST

User Supervisor Call (SVC)

Exit Programming

Module 4: Coding Requirements

What is re-entrant code?
Why have Re-Entrant Code?
What are the benefits of Re-Entrant code?
Re-Entrant code requirements
Avoiding in-line parameter lists
Acquiring memory – STORAGE Macro
Releasing memory – STORAGE Macro
AMODE / RMODE switching
RSECT versus CSECT
Exercise in converting non-reentrant to re-entrant code
Where to store exits
Testing

Module 5: New Release Changes

Is the exit still required?
A scenario
MPFLIST versus AUTORxx PARMLIB member
The introduction and Release Guide
Manuals and sample Exit Source Code

Module 6: Reading what is there

Is the source code available?
Here are two samples that can be reviewed

- A least one message handling exit triggered via the MPFLST
- IKJEFF10 the TSO Submit Exit

What if the source code is not available.

- AMBLIST Service Aid to produce list of CSECT in a multi-CSECT module
- AMASPZAP Service Aid to produce a CSECT listing
- Link-Editor to isolate a CSECT
- ASMDASM – Disassembler object code to source

Module 7: Review of customer exits

The content for this module will depend upon what the client is prepared to disclose.
If there is no client content module 6 will have achieved this goal albeit in an abstract fashion.

Exit Programming

Module 8 – Relative Addressing

Relative Addressing versus Multiple Base Registers
SYSSTATE and IEABRCX macros
STRL and LARL instructions
Qualified USING statements
How to handled embedded data issues
EX versus EXRL instruction
Continuing requirement for local addressability