PROGxx and LLA Enhancements z/OS 1.12

Session 9703

Peter Relson
IBM Poughkeepsie
relson@us.ibm.com

9 August 2011

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

AIX*	FlashCopy*	Parallel Sysplex*	System Storage	z10
CICS*	HiperSockets	ProductPac*	System z	z10 BC
DB2*	IBM*	RACF*	System z9	z10 EC
DFSMSdss	IBM eServer	Redbooks*	System z10	z/OS*
DFSMShsm	IBM logo*	REXX	System z10 Business Class	zSeries*
DFSMSrmm	IMS	RMF	Tivoli*	
DS6000	Infiniband*	ServerPac*	WebSphere*	
DS8000*	Language Environment*	SystemPac*	z9*	
FICON*				

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

InfiniBand is a registered trademark of the InfiniBand Trade Association (IBTA).
Intel is a trademark of the Intel Corporation in the United States and other countries.
Linux is a trademark of Linux Torvalds in the United States, other countries, or both.
Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
UNIX is a registered trademark of The Open Group in the United States and other countries.
All other products may be trademarks or registered trademarks of their respective companies.
The Open Group is a registered trademark of The Open Group in the US and other countries.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.

Abstract

The presentation will cover the new functions made available within Contents Supervisor component in z/OS 1.12. PROGxx enhancements are in the areas of dynamic exits, dynamic LNKLST, dynamic LPA, and defaults. There are also LLA enhancements.

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults
- **Summary**

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

LLA Dynamic Exits CSVLLIX1, CSVLLIX2

- CSVLLIX1 (LLA fetch exit) and CSVLLIX2 (LLA staging exit) are added to the dynamic exits facility, so they can be managed as are other dynamic exits through PROGxx parmlib members
- The default for each: when the EXIT1 or EXIT2 statement in CSVLLAxx does not indicate OFF, if no exit routines are associated with the exit then the system will add exit routine CSVLLIX1 to the first and/or CSVLLIX2 to the second.
- This preserves current behavior

LLA Dynamic Exits (cont)

- If exit routine(s) have been associated via PROGxx parmlib member and/or SETPROG command, then the default processing is not done. In that case, all manipulations are to be done via the dynamic exits facility
- Once the exit routines are managed by your use of PROGxx for those exits, use of the EXIT statements within CSVLLAxx is no-op'd. Thus you do not disable the EXIT by changing EXIT1 or EXIT2 statement in CSVLLAxx to indicate OFF but use the dynamic exits facility (such as the EXIT statement in PROGxx with the DISABLE option)

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

LLA Busy

- LLA is manipulated by a modify command
- Prior to z/OS 1.12, if one modify is in-process when another is received, the second get a “busy” response
- With z/OS 1.12, up to 255 modify's can be started without getting “busy” (a 256th would get “busy”). Requests received while an early one is in-process get queued and processed in turn; the requestor does not get a “busy” response.

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

LLA Automatic Restart

- Suppose SUB=MSTR is incorrectly omitted when LLA starts, such as START LLA,nn=XY
  which asks to use parmlib member CSVLLAXY
- LLA detects the omission of the recommended SUB=MSTR and terminates this START and begins a new one adding SUB=MSTR
- Prior to z/OS 1.12: it did not propagate nn=XY so the restart was with the default parmlib member. This likely was not what you wanted.
- With z/OS 1.12: it does what you surely want: propagate nn=XY so the restart uses parmlib member CSVLLAXY

*

LLA Restart z/OS 1.13

- Suppose you started LLA (e.g., S LLA,LLA=XY,SUB=MSTR) then terminated LLA (P LLA) and then restarted LLA but omitted the “LLA=” specification (S LLA,SUB=MSTR)
- Prior to z/OS 1.13, this would default to “no parmlib member” and LLA would use its default of “only the LNKLST”.
- As of z/OS 1.13, this now defaults to “the parmlib member that you successfully used the last time this IPL that you started LLA” (CSVLLAXY in the example)
- If you truly want “only the LNKLST” when you restart, you may specify LLA=NONE when you start LLA (S LLA,LLA=NONE,SUB=MSTR).

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

Dynamic Exits

- REPLACE
- PARAM
- ExitType
- FoundButError

Dynamic Exits Replace

- Have you ever wanted to be able to change an exit routine?
- You can do it today using DELETE then ADD, or MODIFY to INACTIVE followed by MODIFY back to ACTIVE. But both of these have the disadvantage that there is some time before the re-ADD when the exit routine does not exist at all.
- This can be a show-stopper for a security-related exit routine where it could be vital that the exit routine always be active
- The REPLACE function of the EXIT ADD statement in PROGxx or SETPROG EXIT,REPLACE command, or the CSVDYNEX REQUEST=REPLACE macro addresses this situation.
- When REPLACEing an active exit routine, there is always one (and only one) copy of an exit routine that gets control at the exit point. Not 0. Not 2.

Dynamic Exits Param

- An exit routine might be able to take advantage of having a constant parameter passed to it
- This can be defined on the EXIT ADD statement of PROGxx or the SETPROG EXIT,ADD command, or by the CSVDYNEX REQUEST=ADD macro
- The 8-byte parameter is placed into access registers 0/1 on entry to the exit routine (this is a bit strange, but is the only place available for all situations)
- DISPLAY PROG,EXIT,EXITNAME=xx,DIAG displays the data (it assumes the data is printable, so you will only see the data if it is printable; for this reason, we recommend to exploiters that they use printable data)

Dynamic Exits ExitType

- There are two main types of exits
  - Installation Exits (type is “installation”)
  - Exits intended for use by programs (type is “program”)
- z/OS 1.12 provides functionality by which
  - The owner of an exit can identify which of these types their exit is (many exits are not yet identified)
  - The DISPLAY PROG,EXIT command can ask to display
    - All exits marked as “installation”
    - All exits either marked as “installation” or not marked at all. This is the “NotProgram” option as it indicates that the exit is not marked as a program exit

You might be interested only in NotProgram exits

*

Dynamic Exits FoundButError

- On Exit ADD / REPLACE, you can request that the system issue a message (or not) when there is an error.
- A new option is added, Message=FoundButError,
- This indicates to write a message on any of the cases covered by Message=Error except for the case of “Exit routine not found”
- This was used in the implementation of the LLA dynamic exits, but could be useful to other dynamic exits exploiters

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

Dynamic LNKLST

- Update Delay
- LNKLST UPDATE (via PROGxx parmlib member or SETPROG command) is unpredictably dangerous in uncounted numbers of ways, yet we know that you like to try it.
  - Part (not all) of the danger comes from in-flight processing that runs aground when the “old LNKLST” is closed and freed
- With the DELAY operand value (numbers of seconds), you can ask that the completion of this command wait the specified time before closing (and hence freeing). This gives in-flight requests additional time to complete.
- The command itself will not complete until this delay has been taken into account

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

Dynamic LPA

- AddAlias
- SVC Number
- LPA Add by fully qualified pathname
- Deferred LPA wait
- Query Only

Dynamic LPA AddAlias

- When you are adding a module that has aliases, it is up to you whether you want dynamic LPA to add those aliases or not (or perhaps just do the main name). You provide all the names that you want processed.
- Prior to z/OS 1.12, If you do want “all the aliases”, you have to know what they are and specify them.
- With z/OS 1.12, if you want all the aliases, you can either specify them all or you can simply request “AddAlias” via
  - (parmlib) PROGxx: LPA ADD MOD(...) ADDALIAS
  - (command) SETPROG LPA ADD MOD(...) ADDALIAS
  - (macro) CSVDYLPA REQUEST=ADD,ADDALIAS=YES.

Dynamic LPA SVC Number

- When replacing an SVC routine on behalf of some authorized application, it can be a nuisance even if you can get the module into storage. The SVCUPDTE programming interface has existed for a long time to hook together the module with the SVC table processing.
- With z/OS 1.12, that replacement is simplified in some cases, as you can specify on a dynamic LPA operation both to fetch the module and also to update the SVC table using the SVC number information that you provide.
- Note that this does not help much with new SVCs because this function updates only the module address. But SVCs usually have other attributes (such as the “type”). Those other attributes are not supported by this mechanism.

Dynamic LPA by Fully Qualified Pathname

- Prior to z/OS 1.12, dynamic LPA modules could come from a PDS or a PDSE. But they could not come from a file system.
- The CSVDYLPA macro is enhanced to allow you to indicate the file from which the module is to be fetched by specifying the fully qualified file system pathname
- CSVDYLPA REQUEST=ADD,...,
  PATHNAME=pathname,PATHNAMELEN=pnl
- This functionality is not provided for use in PROGxx or by the SETPROG command. It is available only by the programming interface.

Dynamic LPA Deferred LPA Wait

- LPA built during IPL (PLPA, MLPA, FLPA) cannot use PDSEs. Dynamic LPA can use PDSEs. Some applications might need to use PDSE program objects in LPA and need to know when it is safe to do so
- Prior to z/OS 1.12, you could add PDSE program objects to LPA at the tail end of the IPL (COMMNDxx parmlib member) but an application could not know that things were ready for it. This left it up to you to sequence these events
- With z/OS 1.12, you can have LPA ADD statements in your IPL-time PROGxx specification (prior to z/OS 1.12 these specifications were not processed). Now they will be, at a deferred point late in the IPL.

Deferred LPA Wait (cont)

- New function is provided for an application to ask that it wait until all of the deferred ADD processing is complete
- Thus you can set up for this application by placing data into the IPL-time PROGxx, and the application can wait, knowing that when it wakes up the ADD processing is done and the application can use what you provided.
- If the ADD processing is already done when the WAIT request is received, the WAIT is a no-op

*

Deferred LPA Wait (cont)

- If the application can be changed to do things itself:

  -
    ```
    CSVDYLPA REQUEST=QUERYDEFLPA,
          DEFLPASTATE=the_state
    CLI   the_state,CsvdylpaDefLpaComplete
    JE    deferred_LPA_is_complete
    ```
  - CSVDYLPA REQUEST=DEFLPAWAIT (requires authorization) or
  - LINK EP=CSVDLPAW (does not require authorization)
  - (The QUERY is optional. You could just do the “WAIT” and it works fine if deferred LPA is already complete)
- If the application is not changed
  - EXEC PGM=CSVDLPAW (a pre-step in a job if a subsequent step would have required waiting)

Dynamic LPA Query Only

- When adding an entire library of parts to dynamic LPA (or a subset using a mask other than “*”), have you wondered how much storage ((E)CSA, (E)SQA) this was going to take, in case it was “too much”?
- The CSVDYLPA macro for REQUEST=ADD with MODINFOTYPE=MEMBERMASK supports a QueryOnly=YES option.
- It does much of the processing that normal REQUEST=ADD does, except it stops short of doing the actual “ADD”, instead just keeping track of how much storage would have been used if the “ADD” were done
- Output is mapped by new DSECT LPMEAQ within macro CSVLPRET
- The idea is to use this prior to an IPL so that you can adjust your (E)CSA and (E)SQA amounts accordingly

Agenda

- **Library Lookaside (LLA)**
  - Dynamic Exits
  - LLA Busy
  - LLA restart improvement
- **PROGxx**
  - Dynamic exits
  - Dynamic LNKLST
  - Dynamic LPA
  - Defaults

PROGxx Defaults

- Sometimes there are options that you know you always want, but they are not required and you might forget.
- New syntax is provided to let you define some things to be defaulted
- For example, when creating a new LNKLST set SETPROG LNKLST DEFINE NAME(xxx)
- You almost always want to copy from what you currently have, thus adding COPYFROM(CURRENT)
- If you forget, you get a practically useless LNKLST set that has only the SYS1.xxxLIB data sets (more or less)
- This is one of the defaults that you can now define

PROGxx Defaults (cont)

- When I create a LNKLST set, always do COPYFROM(CURRENT)
  - DEFAULTS LNKLST COPYFROMCUR
- When I create a LNKLST set, make me always specify COPYFROM (I will identify the source to copy from)
  - DEFAULTS LNKLST REQCOPYFROM
- When I do dynamic LPA, always process aliases for the modules I specify whether I provide the aliases or not
  - DEFAULTS LPA ADDALIAS
- Default DISPLAY PROG,EXIT to display only exits of a particular type
  - DEFAULTS EXIT INSTALLATION | NOTPROGRAM | ALL

PROGxx Defaults (cont)

- The defaults apply to
  - Statements within PROGxx
  - The SETPROG command
- The defaults do not apply to the programming interfaces (CSVDYNEX, CSVDYNL, CSVDYLPA)

Coexistence Considerations

- There is a coexistence consideration: The new PROGxx-specified functions will fit into one of two cases:
  - Rejected by an earlier release (subsequent parmlib statements in the same member will still be processed)
  - Accepted by the earlier release (the function was shipped but not documented and possibly not fully tested at that earlier release)
- Primarily due to the second bullet above, even though it is somewhat of a pain if you need to fall back, we recommend that you use a unique parmlib member for specifying the functions new in R12.

Summary

- There are many CSV-related enhancements in the areas of
  - LLA
  - Dynamic Exits
  - Dynamic LNKLST
  - Dynamic LPA
  - PROGxx in general

Publications

- z/OS V1R12.0 MVS System Commands – SA22-7627-13
- z/OS V1R12.0 MVS Initialization and Tuning Reference – SA22-7592-21
- z/OS V1R12.0 MVS Installation Exits SA22-7593-16
- z/OS V1R12.0 MVS Programming Authorized Assembler Services Reference (ALESERV – DYNALLOC) - SA22-7609-11