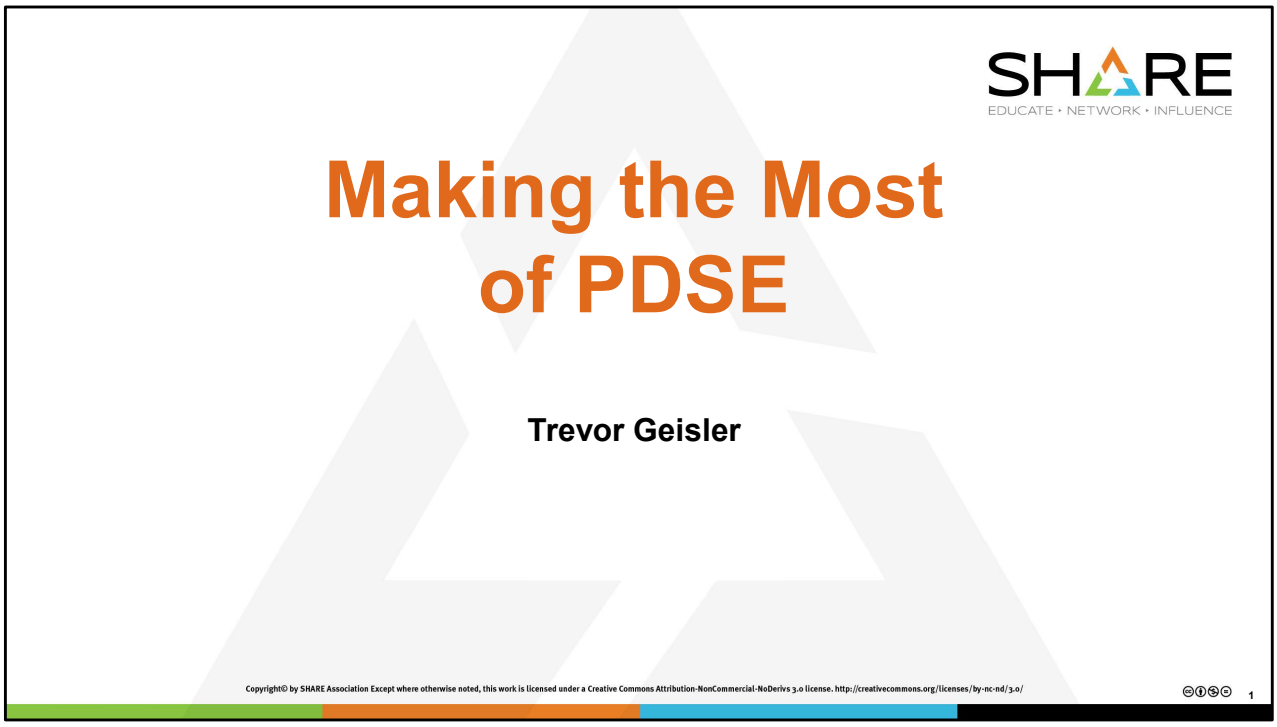


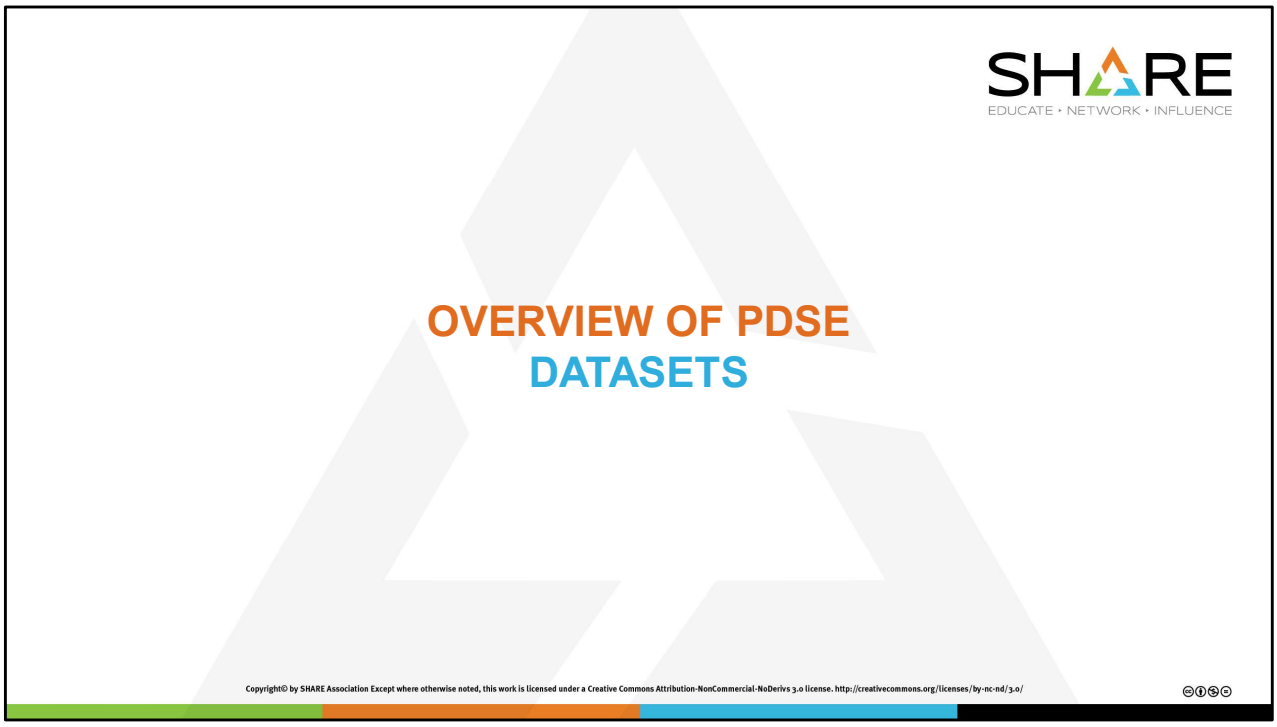
Hello everyone, and welcome to my session on Making the Most out of PDSE. My name is Trevor Geisler, and I've been working on PDSE for over 11 years in a support perspective. This presentation is designed to showcase some of the optional features and parameters of PDSE, setting some recommended defaults, and guiding you on how to make the most of your PDSEs.

Agenda

- Overview of PDSE
  - Dataset Version
- Optimal Setup
  - Sharing Datasets and Concurrent Access
    - Resource Monitor
    - Sharing Across the Sysplex
  - Restart-able Address Space
  - Hiperspace and Other Caching
    - Statistics and Cache Hits
  - Encryption and Compression
- Best Practices
  - LINKLIST Modifications
  - Problem Determination and Recovery
    - Dump Analysis for Dataset Name
    - Recover from Resource Contention

2

We are going to start by going over what PDSE is (both the datasets and the Address Spaces that drive them) and then jump into the optional; optimal; settings for PDSE, ranging from dataset sharing and serialization to fine-tuning advanced caching options. We will then talk about some of the steps you may need to take to recover from some of the more common types of problems we see, such as dataset corruption and orphaned latches and locks.



With the general concept map for this presentation out of the way, lets go ahead and dive into the first topic, which is explaining what PDSE really is.

What is a PDSE?

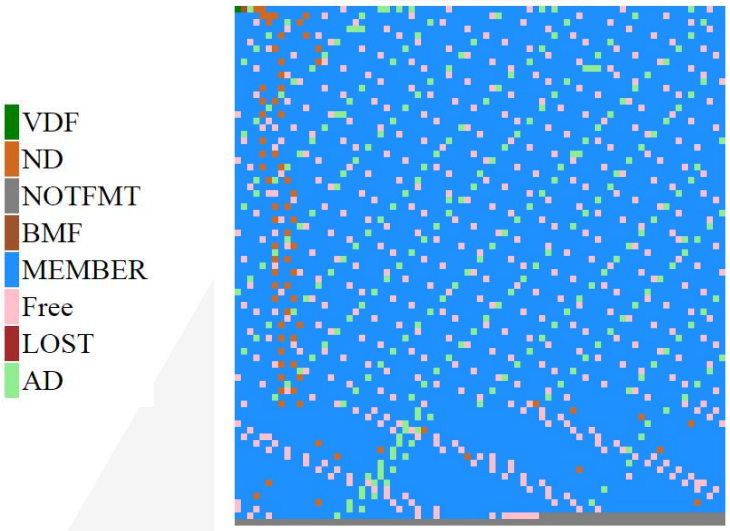
SHARE
EDUCATE • NETWORK • INFLUENCE

- PDSE: Partitioned Data Set Extended
- A PDSE is a homogenous collection of directory and data pages
- PDSE server consists of one or two address spaces (SMSPDSE and SMSPDSE1)
- The SMSPDSE(1) address spaces serve client access requests for PDSE data sets
- Under the hood SMSPDSE(1) also manages PDSE serialization and buffering

4

PDSE, or Partitioned Data Set Extended refers to both the datasets themselves, as well as the Address Spaces that handle requests for the datasets and members. In addition to dataset access, these Address Spaces, referred to as SMSPDSE and SMSPDSE1 (More on SMSPDSE1 later) provide many additional benefits such as serialization, buffering and caching, and much more for the datasets. The datasets, the core of PDSE, are different from PDS, in that everything in a PDSE dataset is comprised of pages 4k in size. These pages can contain the directory structures, the member data, as well as various meta-data related to the dataset and are all intermingled within the confines of the PDSE dataset.

What does a PDSE Look Like



5

This image here shows a visual representation of what a PDSE dataset looks like on DASD, with each little square representing one of those 4k pages. You see the AD and ND pages which make up the directory interspersed throughout the member and free pages that comprise the data, amongst various other according to the key on the left. Due to the nature of being comprised of 4k pages, and the ability for directory and members to intermingle in the dataset space, there is no fragmentation, like you might otherwise see in PDSs preventing a lot of the maintenance work that might otherwise be involved in using datasets of that type.

PDSE Version 2 Datasets

- Version 2 datasets increase the chance that partial release will be able to release space
- Version 2 brings better performance and efficiency
  - Removes unneeded structures
  - Reworked variable record member indexing
- Reduces CPU and Storage utilization
- Basis for additional PDSE features
  - PDSE Member Generations
  - Encrypted PDSEs
  - Compressible PDSEs

6

While not new anymore, PDSE datasets have an upgraded version, aptly named Version 2. These version 2 PDSE datasets bring with them a host of new enhancements on the back end, increasing efficiency and performance by re-working or removing some of the support structures. This allows for reduced CPU consumption and storage utilization as opposed to the legacy PDSE version 1. There are also numerous features that rely on a PDSE being version 2, such as member generations, PDSE encryption and PDSE compression. For this reason, the first recommendation for you all is to make the change to Version 2 PDSE datasets.

Allocating a PDSE Dataset – V2

- IGDSMSxx
  - DSNTYPE(LIBRARY|PDS|ZFS) – Library to have default datasets be PDSE
  - PDSE_VERSION({1|2}) – Specify 2 to default to Version 2 PDSE
- ISPF Panels

```
Data set name type    Library        (LIBRARY, PDS, LARGE, BASIC, *
                                      EXTREQ, EXTPREF or blank)
Data set version  . : 2
```

- JCL/DD Keywords
  - DSNTYPE=(LIBRARY,2)

7

There are a few ways that you can specify version 2 when allocating a new dataset, whether on an individual allocation such as through the ISPF panels or JCL/DD statement by specifying Library,2; however the recommendation is to go ahead and make this the default version for PDSE by making that change in the IGDSMSxx parmlib member. By setting the PDSE_VERSION to 2 (from its default of 1) you can rest assured that your PDSE allocations will default Version 2, making use of the enhancements that provides.

Allocating a PDSE Dataset – Space

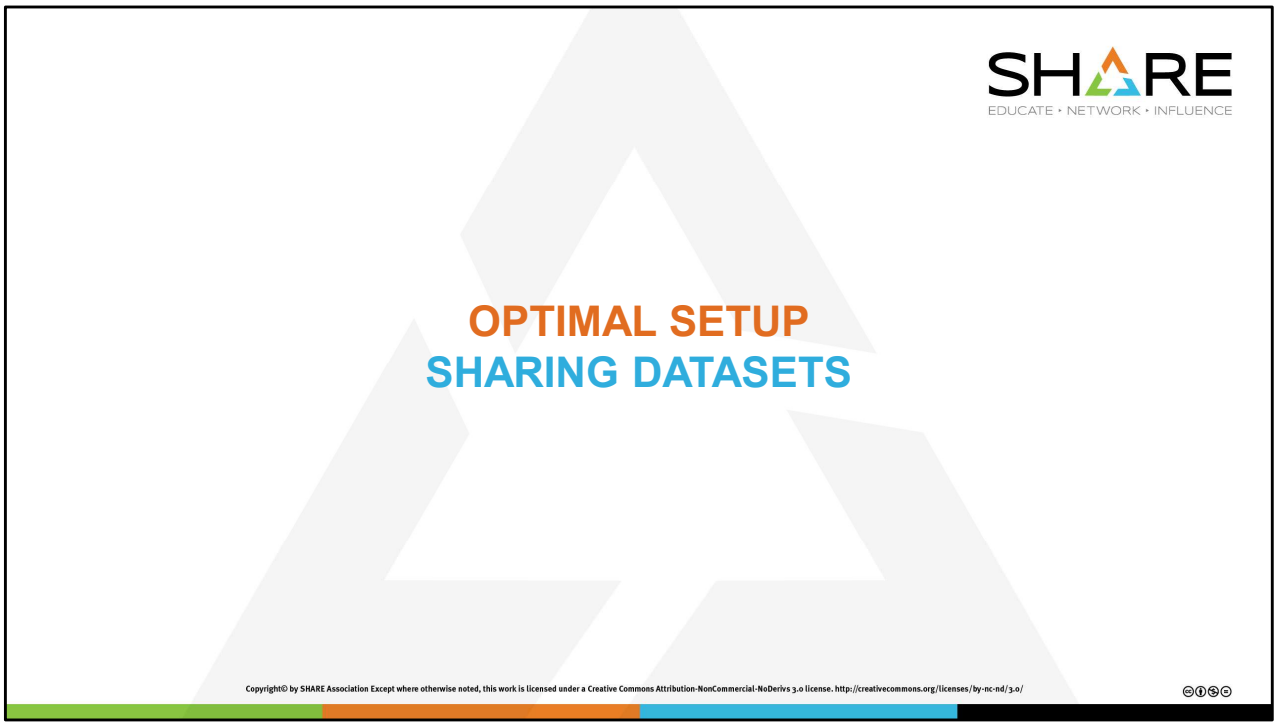
SHARE
EDUCATE • NETWORK • INFLUENCE

- Secondary Allocation
  - Providing a value other than 0 will allow the dataset grow
    - A PDSE can have 123 extents
  - PDSE Extents do not count toward the LINKLIST Extent Limit
    - A PDSE counts as a single extent
    - Having secondary allocations allows LINKLIST datasets to grow (for PDSE)
    - PDS Extents DO count
      - Historical recommendation to avoid Secondary Allocation (for PDS)

8

On the topic of dataset allocation, our next recommendation is in relation to Secondary Space, and allowing the PDSE dataset to grow as needed when updates are performed against the members therein. PDSE datasets, when you specify a secondary allocation other than 0 will be able to grow into a new extent to hold the new data automatically. This extent expansion can happen 122 times (for a total of 123 extents), as opposed to the old limit of 16 for PDS datasets.

Speaking of PDS datasets, if any of you have been around the mainframe world for a while, you probably are familiar with the historic recommendation to not have PDS datasets allocated with Secondary Extents if they are in the LINKLIST due to those extents eating into the limit of Extents available in the LINKLIST. This recommendation does not apply to PDSE datasets, as a single PDSE regardless of the number of secondary allocations still only counts as a single extent from a LINKLIST perspective. Because of this, the recommendation is that you DO specify secondary allocations for the PDSEs so that you can apply maintenance to your LINKLISTED datasets without the need to reallocate an entire new dataset due to space constraints and simply let the dataset grow to meet the need for extra space.



Next, we will talk about the different types of sharing modes and accessing PDSE datasets across multiple systems and then take a look into the serialization of PDSE resources and how to monitor for potential issues.

Sharing Datasets – Sharing Mode

SHARE
EDUCATE • NETWORK • INFLUENCE

Important Terminology:

- Two sharing modes, NORMAL and EXTENDED
  - NORMAL is the default and fallback mode
  - EXTENDED is preferred in the SYSPLEX environment
- GRSPLEX Scope: A set of systems connected by only GRS
- SYSPLEX Scope: A set of systems connected by both XCF and GRS

10

First off are some terms we will be using for this section, such as Normal Or Extended sharing, as well as the definitions of GRSPLEX and SYSPLEX. Normal sharing uses GRS to communicate resources between systems, and all sharing systems must be within the same GRS configuration, known as a GRSPLEX. Extended sharing uses a combination of GRS and XCF for sharing and serializing resources between systems, so all sharing systems must be within the same XCF and GRS configuration known as a SYSPLEX.

Sharing Datasets – Sharing Mode

- Normal Sharing
  - Shareable only at a dataset level between systems
  - Member-level sharing only on the system with access
  - Cannot be used with the Restart-able Address Space
  - PDSESHARING(NORMAL)
- Extended Sharing
  - Member-level sharing between all systems in SYSPLEX
  - Can be used with one or both PDSE Address Spaces
  - PDSESHARING(EXTENDED)

11

In regards to the sharing mode themselves and how they interact with PDSE datasets, there are 2 modes as mentioned. The legacy sharing mode, NORMAL Sharing, does so at a dataset level. This means that only one system can access the dataset at a time in a multi-system environment. For the system that has access to that dataset, multiple jobs/tasks can access members of the dataset. Also, if you are using the Restart-able Address Space (SMSPDSE1) discussed in detail later, you cannot use NORMAL Sharing. This is enabled by default or by specifying PDSESHARING(NORMAL) in the IGDSMSxx parmlib member.

Next is the EXTENDED Sharing mode, which allows for multiple systems to access a dataset and its members at the same time. You can use Extended Sharing even when the restartable address space is not in use, but with both address spaces active, you must use Extended Sharing. This is enabled by specifying PDSESHARING(EXTENDED) in the IGDSMSxx parmlib member.

Sharing Datasets – Cross-Sysplex

- Source / Copy Relationship
  - Maintain a dataset where all maintenance/updates are done within 1 SYSPLEX
  - Using XMIT or FTP, copy this dataset out to the other SYSPLEXs that need to share that dataset
  - This ensures that all data is equal between SYSPLEXs, but does not violate the Extended Sharing serialization leading to improper sharing

12

Taking a bit of an aside, before we get into the nitty-gritty of dealing with serialization of PDSE resources and multiple concurrent access and how that can lead to resource contention, I want to talk about something that we get asked a lot about in PDSE Support, and that is Cross-sysplex sharing of PDSE datasets. In short, any access of a PDSE dataset outside of the serialization communication by GRS or XCF is referred to as improper sharing and can cause issues due to changes not being seen by systems within the plex. Because this can lead to dataset corruption (discussed later) there is no supported way to share a single dataset across sysplexes. As such, we recommend that if you need the same dataset across multiple plexes, to set up a Source/Copy relationship between the various plexes, such that all maintenance and changes are done within a single plex, and then a copy of that is XMIT'd or FTP'd to the other plexes. This ensures that all data is equal between SYSPLEXs, but does not violate the Extended Sharing serialization requirements leading to improper sharing

Sharing Datasets – Resource Monitoring

SHARE
EDUCATE • NETWORK • INFLUENCE

- Multiple concurrent access
  - Various Jobs/Tasks running may require access to the same PDSE (dataset or members) at a time
  - This can lead to a situation where one job/task must wait for the serialization

13

Moving back to the main topic of concurrent access and serialization of PDSE resources, you may frequently find that you have multiple jobs or tasks running at the same time that require access to the same PDSE or member. While PDSE is good about only holding the resources for as long as needed, there still may be times where one resource is being held in use, while another job or task is stuck waiting for that resource. We refer to this as resource contention.

Resource Contention

- IGW038A
  - Indicates that there is potential contention issues
  - Governed by the PDSE MONITOR
    - SMSPDSE(1)_MONITOR_DURATION
    - SMSPDSE(1)_MONITOR_INTERVAL
  - A single message is OK
    - Long Running Jobs or Increased CPU Utilization
- IGW031I
  - VARY SMS,PDSE(1),ANALYSIS
  - Shows the latches/locks that are currently held, while other tasks are waiting for the resource
  - Compare resources, time held, and waiters between instances to determine scope of issue
    - Changes in resources are often false positive, resultant from increased CPU or long jobs
    - No change in resources or a long time held generally indicates a problem

14

Resource contention itself does not always refer to a real problem, simply that a resource is held while there are waiters for that resource. This condition is monitored for, and when detected, triggers an IGW038A message, which simply states 'potential contention issue' and then recommends issuing the V SMS,PDSE,ANALYSIS Command. Following that recommendation, which incidentally we recommend to automate, you will get an IGW031I message, which shows the latches and locks that are being held, the jobs/tasks holding them, the list of waiters for that resource, as well as how long that resource has been held.

Using that information, you can quickly determine if the resource contention is indicating a real issue such as a hang or an orphaned resource, or if you are simply running into a false positive. If there are no changes in the resources in the messages across multiple instances, and the time held keeps increasing, this may be a real issue and require some steps for further analysis and recovery (discussed later). If the resources are changing, or the messages stop appearing then likely you encountered a false positive, where a long-running job or increased CPU utilization is causing PDSE processing to slow down while other tasks are working. In such cases, the contention will clear on its own, and there is no cause for concern.

Resource Contention

- Tuning for False Positives
  - SMSPDSE(1)_MONITOR_DURATION
    - Defaults to 15 seconds
    - Signifies how long a resource must be held (with active waiters) before notifying user
  - SMSPDSE(1)_MONITOR_INTERVAL
    - Defaults to 60 seconds
    - Signifies how frequently the MONITOR makes these checks
  - If seeing a lot of transient connection or false positives due to long running jobs or high CPU utilization, consider increasing thresholds

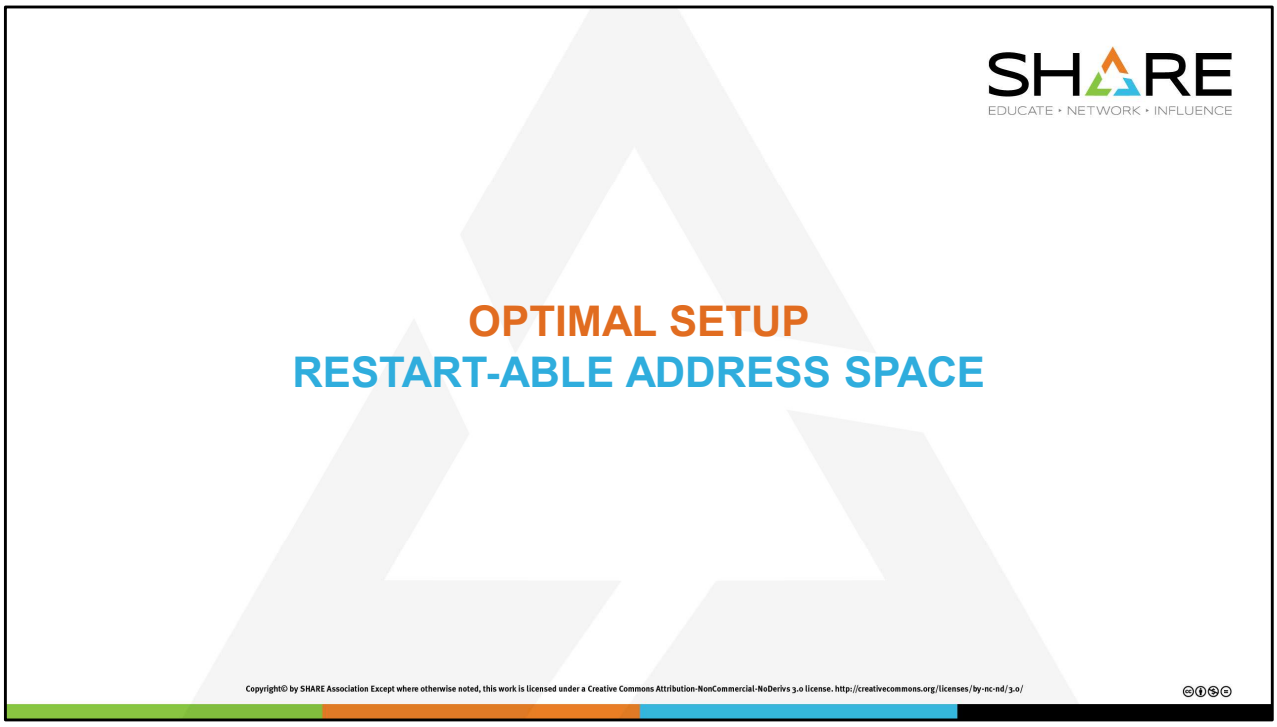
15

Taking a step back to the IGW038A messages, however, you can fine-tune the control parameters for the resource Monitor to stop the system from triggering these messages if you frequently are getting false positives, so that it only surfaces when there is a higher chance of it being a real contention issue. The monitor is governed by the pair of parameters SMSPDSE_MONITOR_DURATION, which says how long in seconds a resource must be held with waiters on the resource before it is considered to be a real contention issue. This default is 15 seconds. Next is the SMSDSE_MONITOR_INTERVAL which indicates how long in seconds between check of the resources. The default here is 60 seconds, meaning that every 60 seconds the system will check to see if resources are being held with waiters for longer than 15 seconds. In heavy CPU environments, this may trigger quite frequently, even though there is not a real issue, so you may consider increasing those values, so that a resource must be held for 30 seconds, for example., before being considered a real issue.

Resource Contention

- Tuning for False Positives
  - PDSE_SYSEVENT_DONTSWAP(NO|YES)
    - If enabled, prevents jobs or tasks holding PDSE resources from being swapped out
    - May delay the system from being swapped out but prevents PDSE processing from being delayed by a swapped address space
    - In addition, it causes the SYSEVENT ENQHOLD to be issued against a latch holder to ensure that the holder completes its processing and releases the latch
    - Very useful in high CPU processing times, to prevent further PDSE contention

The final piece of the puzzle for tuning out these false positives is the optional parameter PDSE_SYSEVENT_DONTSWAP. What this does is, if enabled, prevents a job or a task from being swapped out while it is holding PDSE resources. If a job holding resources is swapped out to allow for other tasks to process and it is still holding those PDSE resources, that can lock up other jobs and tasks as well since that resource will not be freed while the job is swapped out. Enabling this parameter can eliminate that particular issue and can be very useful in high CPU processing times, to prevent further PDSE contention.



The next bit of recommended setup for getting the most out of your PDSEs is the enablement of the Restart-able Address Space, SMSPDSE1.

Restart-able Address Space

- Requirements for Enabling
  - PDSE_RESTARTABLE_AS(YES|NO)
  - Requires Extended Sharing
- PDSE processing once enabled
  - The non-restartable SMSPDSE address space is used for PDSEs that are contained in the LNKLIST.
  - The restartable SMSPDSE1 address space is used for all other PDSEs in the system.

Enabling the SMSPDSE1 address space is very simple, assuming you meet the requirements as discussed previously. This does require Extended Sharing to enable, but if able to enable extended sharing as recommended, it is also recommended to enable the restartable PDSE address space. This can be done via the IGDSMSxx parameter PDSE_RESTARTABLE_AS and setting it to yes. When both Address spaces are active, the non-restartable address space SMSPDSE, will be used for global connections and those datasets within the linklist and the restartable address space will be used for all other PDSEs in the system.

Restart-able Address Space

SHARE
EDUCATE • NETWORK • INFLUENCE

- Why perform a SMSPDSE1 restart?
  - To recover from a situation that would otherwise require an IPL
  - Recover from a PDSE latch hang situation
  - Recover from in-core corruption of a PDSE
  - Recover from excessive PDSE storage usage
- Best Practices and Performance Considerations
  - Very Safe!
  - One at a time – Do not route restart command across plex
  - What are the side effects?
    - A small amount of CSA is lost in the restart
    - Queisce may take minutes

Now, you may be wondering why you would want the restartable address space, and the primary answer to that is to avoid IPLs in the event of an emergency. An SMSPDSE1 restart can be used to recover from resource contention, dataset corruption if in-core, and even situations that might otherwise require an IPL.

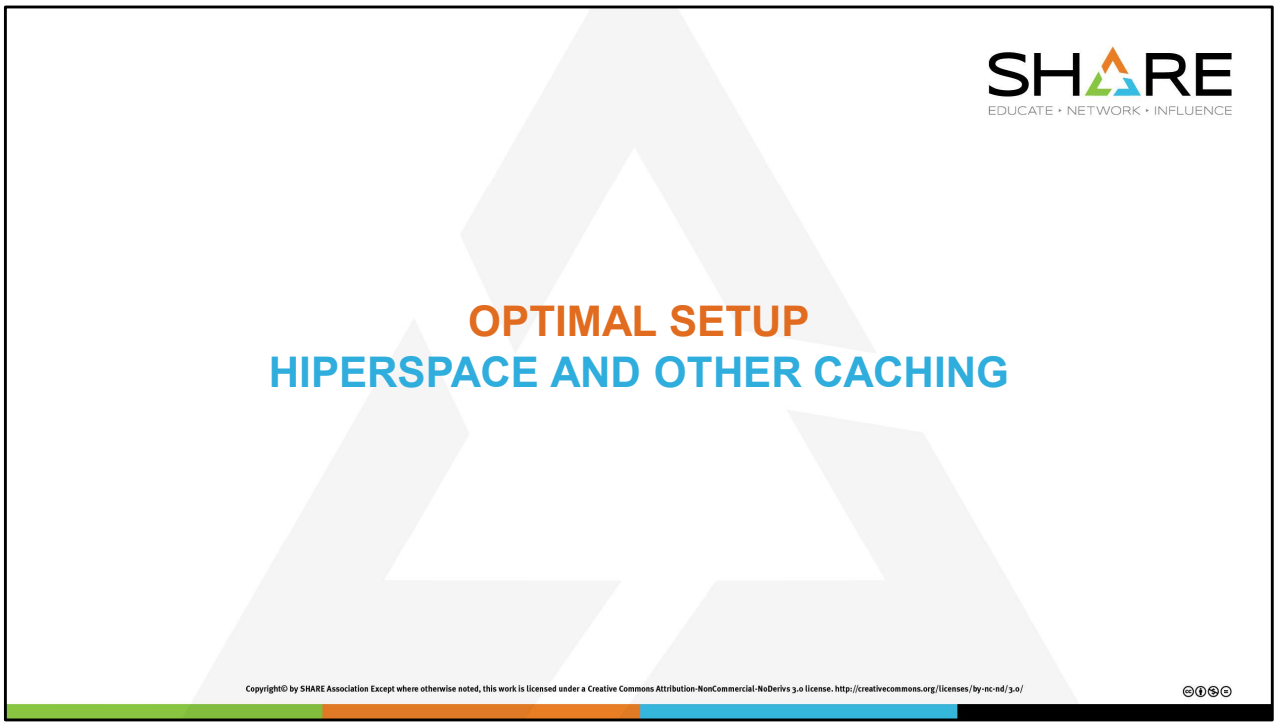
The restart itself is very safe, however, a restart of SMSPDSE1 must be done one system at a time and must fully come back up before a restart on another system is attempted.

Restart-able Address Space

SHARE
EDUCATE • NETWORK • INFLUENCE

- Issue the restart command
  - V SMS,PDSE1,RESTART [,QUIESCE(duration | 15)[,COMMONPOOLS(NEW|REUSE)]
  - QUIESCE option determines how long in-flight operations have to quiesce
  - COMMONPOOLS option determines whether ECSA cell pools are reused
  - Only select NEW if there was a cell pool problem

Along with being safe, the SMSPDSE1 restart is also very easy, just one command, V SMS,PDSE1,RESTART, with a few optional parameters affecting quiesce time and cellpools, although in most cases the defaults are sufficient.



With the discussion of the restart-able address space completed, I will re-iterate the sort of CORE recommendation for getting the most out of your PDSEs, which is to make use of extended sharing and the restart-able SMSDSE1 address space to manage your Version 2 PDSEs. The next items we will discuss can further enhance the performance and security of your PDSE datasets. As you can see, we will first talk about caching, both for the directory and the member pages of your PDSE datasets.

Other Caching – Directory Caching

- PDSE caches the directory to optimize reads and writes
  - Cache space is shared amongst all OPEN PDSEs
    - Will grow to meet need
  - OPENs for the same PDSE utilize same pages
  - Periodically reaped to cache limit based on usage
- Cache invalidation
  - Data set closure (default behavior)
  - EXTENDED mode only: another sysplex member *writes* to the PDSE
    - More contention = more cache invalidation
  - IEBPDSE without specifying NOFLUSH

Directory caching for PDSE datasets is automatic. As datasets are opened, the directory pages will be read and copied into the cache, This allows for other jobs/tasks on the same system to make use of these cached pages to optimize the reading and writing of the directory and avoid IO.

The directory cache does not have a maximum size, and will instead grow to meet demand, though it will try and meet a soft-limit, by periodically reaping cached pages that have not been referenced recently. In addition to pages that have not been referenced, the cache will also remove pages when a dataset is closed (with an exception we will discuss shortly), or if they have been updated by another system in the sysplex, as any changes from those systems would not be reflected in the cached pages, requiring that we re-read the changed directory from the disk. Additionally, you can manually purge the cached pages for a PDSE dataset using IEBPDSE or the V,SMS,PDSE,REFRESH,DSN() command.

Other Caching – Directory Caching

SHARE
EDUCATE • NETWORK • INFLUENCE

- IGDSMSxx Parameters
  - Tunable for both SMSPDSE and SMSPDSE1
  - PDSE{1}_BUFFER_BEYOND_CLOSE(**NO**|YES)
    - Setting to YES enables caching beyond the last closure. Only honored with EXTENDED mode sharing.
  - PDSE{1}_DIRECTORY_STORAGE(**2G**|nnn)
    - Soft limit for the amount of memory used by the directory page cache in the SMSPDSE{1} address space.
  - PDSE{1}_LRUTIME(**60**|n)
    - Time, in seconds, between cache adjustment cycles.
  - PDSE{1}_LRUCYCLES(**15**|n)
    - LRU Cycles that a cached page must not be referenced to be invalidated

There are a few options for tuning the caching behavior of PDSE datasets, found within the IGDSMSxx parmlib. The first, PDSE_DIRECTORY_STORAGE, provides the soft limit for the maximum size of the directory cache, which the cache invalidation algorithm will try to honor by removing pages that haven't been referenced recently. This leads into the next parameters, LRUTIME and LRUCYCLES which together determine how long a cached page must be unreferenced before being invalidated and removed from the cache. The default is 15 60-second cycles. Lastly, the PDSE_BUFFER_BEYOND_CLOSE parameter will allow for cached pages of a dataset to remain in the cache after the dataset has closed. This is especially useful for datasets that frequently open and close, so that we do not have to do IO for every open, increasing efficiency and performance.

Other Caching – Directory Caching

- Statistics in SMF Type 42 Subtype 1
  - SMF42TDT – Number of directory page reads
  - SMF42TDH – Number of directory page cache hits
  - SMF42BUF – Number of active directory pages in cache
  - SMF42BMX – Largest number of directory pages

Using the following SMF records, you can review the caching statistics of the Directory cache, such as how many directory reads were handled by the cache, how many cached pages did you have at the peak, and more, in order to fine-tune the LRU cycles and soft-limit to better fit your needs.

Other Caching – Hiperspace Cache

- Retain read member pages in memory
  - Supported for both program object and data PDSEs
- Disabled by default
- SMS-managed PDSEs only
  - Requires a Millisecond Response Time of 1
- Hard limit on cache storage
- Cache invalidation
  - Pages age out
  - Data set is closed

Unlike the Directory pages, the member pages are not cached by default, but with a small amount of setup, you can enable member pages to be cached as well, in what is called the Hiperspace cache. First, you need to ensure that the datasets are cache eligible, and that requires that the datasets are SMS managed, with a data class that specifies a Millisecond response time of 1.

Other Caching – Hiperspace Cache

- PDSE{1}_HSP_SIZE(**0**|nnn)
  - Size of the member cache, in MB.
  - Limit of 2GB
- PDSE{1}_BUFFER_BEYOND_CLOSE(**NO**|YES)
  - Useful for member caching too!
  - Retains member pages beyond OPEN lifetime
- PDSE{1}_LRUTIME(**60**|n) LRUCYCLES(**15**|n)
  - Member cache is adjusted on the same interval as directory pages

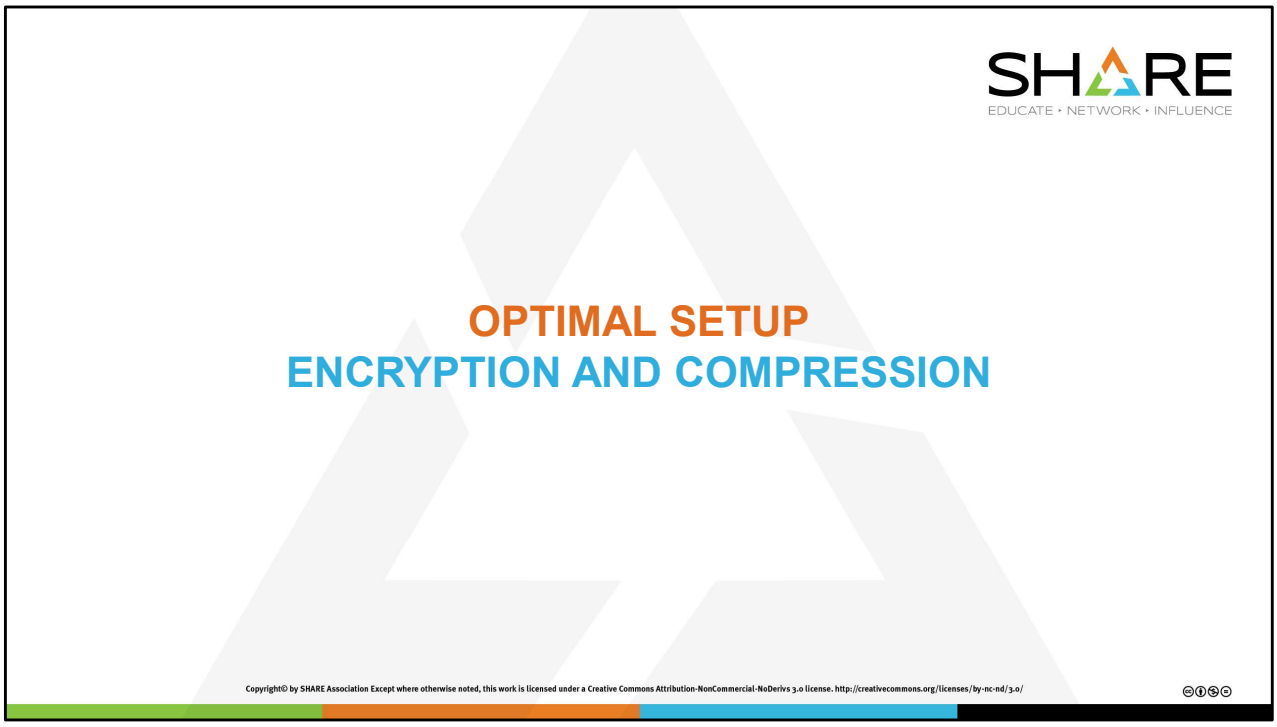
To enable the member cache, you need to specify the PDSE_HSP_SIZE parameter with the size of the member cache you want, up to 2 Gbs, which is a hard limit, unlike the directory cache. With the cache enabled and the datasets eligible, the caching functions the same way as the directory cache, with pages following the same logic for invalidation as the directory cache.

Other Caching – Hiperspace Cache

- Statistics in SMF Type 42 Subtype 1
  - SMF42TRT– Number of member page reads
  - SMF42TRH – Number of member page cache hits

Likewise, there are a set of SMF records that showcase the amount of member page reads handled by the cache, which can help you tune the Hiperspace cache to your needs.

Of note, the statistics here is for all PDSEs, therefore the number of page cache hits may be significantly smaller than the total page reads, as all datasets are counted here, not just those datasets that you made cache-eligible.



The next set of features to discuss when making the most of your PDSEs is that of encryption and compression.

Encryption

- **Encrypted PDSE's are V2 PDSE's**
  - Structure is identical
  - Allocated as V2 PDSE data sets
  - Become Extended Format
  - Only allocation change is that a keylabel is provided
- **Encryption at rest**
  - All user data and user metadata is stored as encrypted on disk
    - That includes the PDSE directory and member generations
    - In practice this means every page in the PDSE is encrypted
  - Transparent at Application Level

Encryption of PDSE datasets is encryption at rest. This means that every one of those 4k pages that make up a PDSE dataset can be encrypted on disk and then decrypted only when needed. From an end-user or application perspective, there is no difference between an encrypted and non-encrypted dataset. Even when considering allocation, the only difference is that a keylabel would have been specified for the encrypted PDSE dataset.

Compression

SHARE
EDUCATE • NETWORK • INFLUENCE

How to Enable:

- Discrete FACILITY class STGADMIN.SMS.ALLOW.PDSE.COMPACT
- Data Class: Compaction = ZP or ZR

Characteristics:

- Member-Level compression, determined at a per-member basis
  - Not compressed if:
    - Member size < 64k bytes
    - Compression of member would cause growth
    - Program Object
- Directory entries are not compressed
- Members cannot be Open for UPDATE
- Must be SMS managed

Compression of the PDSE datasets is only slightly more involved than encryption and has a few more restrictions on what can be compressed, but when applicable can provide some great space optimization benefits for PDSE datasets.

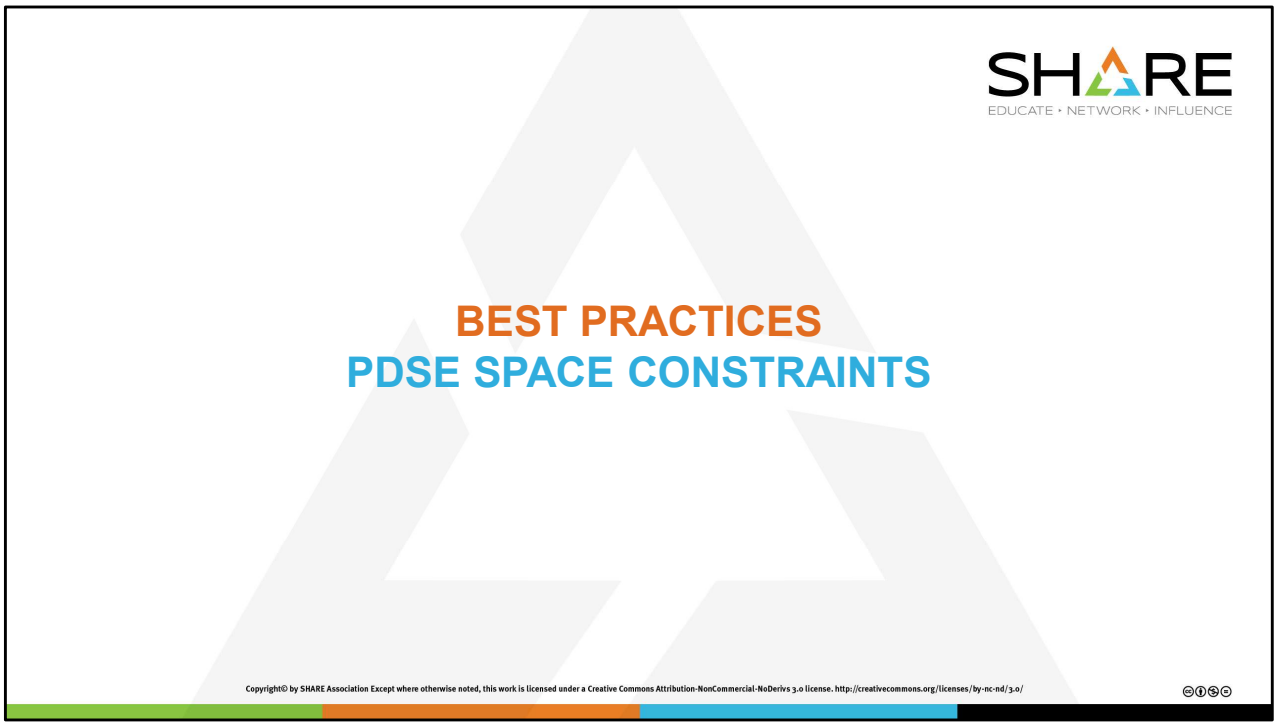
To start, PDSE compression must be enabled by specifying a discrete facility class, STGADMIN.SMS.ALLOW.PDSE.COMPACT, as well as having the SMS managed datasets be associated with a data class indicating ZP or ZR indicating zEDC Preferred or zEDC Required for the compaction parameter.

As for the restrictions I mentioned, PDSE directory pages are not compressible, nor are program objects. Additionally, data members need to be above 64k bytes in size to be compression eligible.

Encryption and Compression

- **Performance Considerations**
  - **Directory Pages**
    - Cached while dataset is open
    - Cuts down on decryption/decompression processing
    - Further enhanced by PDSE(1)_BUFFER_BEYOND_CLOSE
  - **Member Pages**
    - Not cached by default
    - Cache Eligible with Hiperspace Caching
    - Further enhanced by PDSE(1)_BUFFER_BEYOND_CLOSE
  - Increased performance while maintaining security

However, when considering both encryption or compression there are options for reducing the performance impact of repeated decryption or decompression, and that is the caching we discussed previosuly. You can make use of the many different caching options; LRU tuning and buffering beyond close, as well as enabling the Hiperspace cache, to retain these PDSE pages in their decrypted and decompressed state while in use. This allows for you to increase performance while maintaining the security and other optimizations that these new features provide.



So now that we have talked about the full feature suite of what PDSE has to offer, and how enabling and tuning these enhancements can help PDSE be utilized to its fullest, I want to switch gears a bit and discuss some common topics that we see in support, and pull from them some best practices that you can use to keep your PDSE environment processing the best it can.

First, we will talk about some space related topics you might encounter, then discuss datasets located in the linklist, and finally take a quick look into recovery from the 2 most common types of cases we see in support, dataset corruption and actual resource contention.

Space Management and Pending Deletes

SHARE
EDUCATE • NETWORK • INFLUENCE

What Are They:

- When a member is deleted with active connections, it is not deleted, but marked as a pending delete
  - No new connections to 'Pending Delete' member
  - Once all connections are dropped, simply takes up space

How To Verify:

- The IEBPDSE job will display how many members are in a 'Pending Delete' state

3 Methods to Clear:

- Performed automatically on first OPEN for OUTPUT of PDSE dataset
- IEBPDSE with PARM='PERFORMPENDINGDELETE,NOANALYSIS'
- IGDSMSxx parm PDSE_PENDING_DELETE_INTERVAL(nnnn)
  - Specified in minutes, must be greater than LRU Time and Cycles Interval
  - A 0 indicates no automatic Pending Delete Processing
  - Ideal for datasets that remain Open for OUTPUT for a long time

33

First off, I want to talk about Pending Deletes. A Pending Delete is a member that was deleted while it had active connections. We essentially mark it for deletion later but keep it around while that connection is still active. No new jobs or tasks can connect to the member in pending delete status, so once that existing connection is dropped, it essentially becomes a waste of space. To verify if a dataset is taking up extra space due to members in a pending delete state, you can run an IEBPDSE against the dataset, which will show how many members are in a pending delete state.

If a dataset has pending deletes that you'd like to clear, there are a few methods of doing so. The first of which is automatic and will happen when the dataset is first open for output. It will trigger a purge of all the pending delete members. You can also trigger this manually to remove any pending delete members with no remaining connections using IEBPDSE with the 'Perform Pending Delete' parameter. Lastly for PDSE datasets that remain OPEN for OUTPUT for a long time, those that would not benefit from the Automatic cleaning of a First Open, you can specify a PDSE_Pending_Delete_Interval parameter in the IGDSMSxx parmlib. This will allow the pending delete process to purge pending delete members periodically.

Out Of Space - D37-04 on Member Deletion

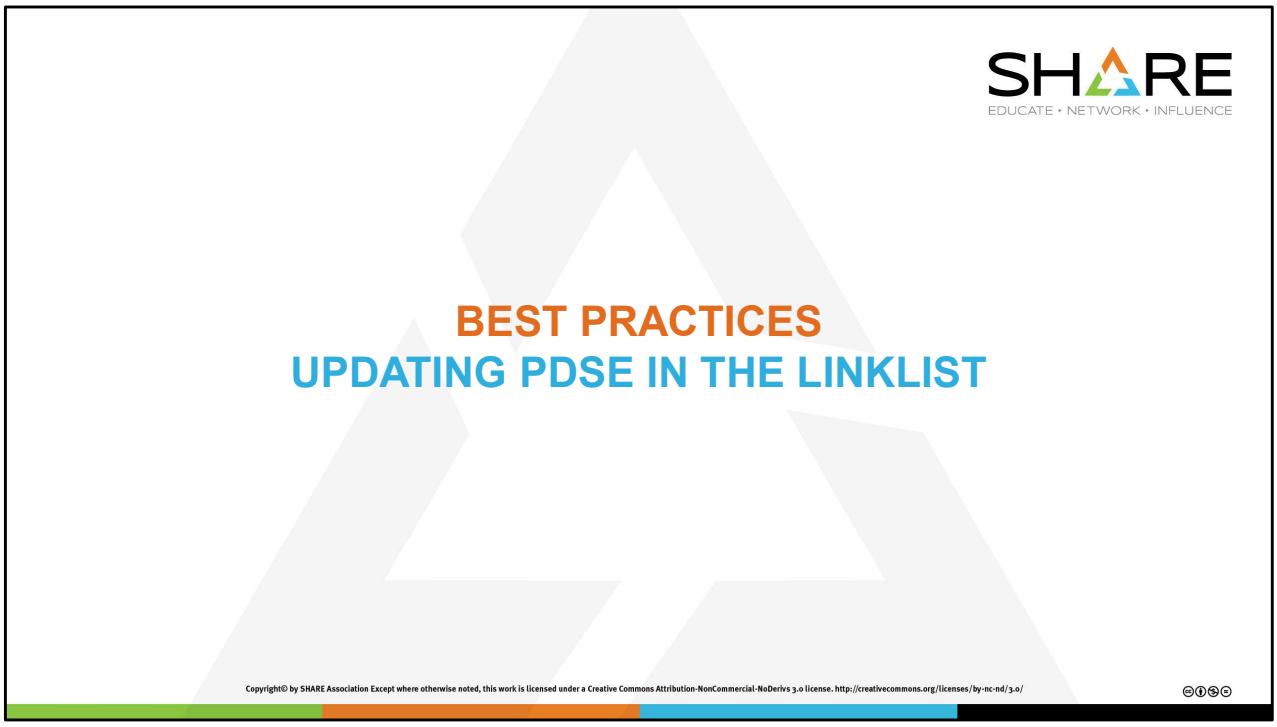
SHARE
EDUCATE • NETWORK • INFLUENCE

- Guaranteed Space for Member Deletion
  - PDSE Deletion Process
    - builds its new internal structures in free space,
    - replaces them to ensure that any change is made atomically
  - Led to a situation where a sufficiently full PDSE would fail a delete operation with an “out-of-space” error
  - PDSE was updated to reserve space for member deletion
  - New in 3.2
    - No Pre-Requisites, No Coexistence
    - Existing datasets may need to re-size 1 last time

34

Next is a bit of a unique one. Historically, if a PDSE dataset was fully utilized, it could actually be too full to delete any members to free up space.

This was due to the method we use for deleting members from the dataset, which builds the new internal structures for the dataset post delete within the free space of the dataset, and then replaces them once the delete is complete. This led to the situation described where a dataset could be too full to create those new structures thus preventing the deletion of members. In 3.2, we have updated PDSE to reserve space for member deletion meaning that you will no longer run into this situation.



Next, we will look into dataset modification for linklisted datasets. There are a few items we have discussed so far in the presentation that led into this topic, from the extents and secondary allocations, which PDSE addresses the historic issue that plagued PDS datasets, to the discussion of pending deletes. Datasets in the linklist have persistent connections from LLA, and as such any modifications to those datasets would leave those pre-updated members taking up space in the dataset as a pending delete. This could cause the datasets to be full unexpectedly, and when allocated without secondary extents, could prevent maintenance application without manual intervention.

Problem Prevention – Modifying LINKLIST PDSEs

- Dynamically Updating a LINKLST PDSE
  - PDSE's cannot be updated while in an active LNKLST set
  - Remove Data Set from LINKLST
    - SETPROG LNKLST,DEFINE,NAME=LNKLST2,COPYFROM=CURRENT
    - SETPROG LNKLST,DELETE,NAME=LNKLST2,DSNAME=MY.LNKLST.DATASET
    - SETPROG LNKLST,ACTIVATE,NAME=LNKLST2
    - SETPROG LNKLST,UPDATE,JOB=*
    - MODIFY LLA,REFRESH
  - Update/Modify dataset, or Replace with new one
  - Add Data Set back to LINKLST
    - SETPROG LNKLST,DEFINE,NAME=LNKLST3,COPYFROM=CURRENT
    - SETPROG LNKLST,ADD,NAME=LNKLST3,DSNAME=MY.LNKLST.DATASET
    - SETPROG LNKLST,ACTIVATE,NAME=LNKLST3
    - SETPROG LNKLST,UPDATE,JOB=*
    - MODIFY LLA,REFRESH

36

Using the guide here, put together by the LLA team in conjunction with PDSE, is the IBM recommended method for modifying PDSE datasets in the linklist. Essentially, you temporarily remove the dataset from the active linklist, and update the in-progress jobs and refresh LLA to remove those active connections. Once freed, you can then modify the dataset, before reversing the process to add the dataset back into the linklist and re-updating the in-progress jobs and refreshing LLA. This will prevent pending deletes from taking up space, and we have also seen it mitigate issues that might arise by directly modifying these datasets while active.

Finally, our last topic is problem determination and recovery from the 2 most common types of issues that we see in support; dataset corruption and real resource contention.

Problem Prevention – Improper Sharing

- Improper sharing can allow for unserialized access to PDSE data sets
  - There is no warning that a data set has been accessed in an unserialized manner
  - The results are unpredictable but may include:
    - Invalid index data in-core
    - Corrupt index data on DASD
    - Corrupt member data
    - Mismatched extent information
    - Nothing at all

As we discussed earlier, sharing PDSE datasets outside of the sysplex is unserialized, and changes being made without ensuring all systems know about it can lead to cases where the caches becomes out of sync, or even where data becomes corrupted on disk. Unfortunately, due to the very nature of this access being unknown to the sharing systems, there is no warning or message that a dataset has been accessed in an unserialized manner, only evidenced by the resultant corruption.

Problem Prevention – Improper Sharing

- There is no 100% safe way to circumvent EXTENDED mode's serialization requirements
- PDSE data sets cannot be serialized by third party products
  - Specifies RNL=NO
  - MIM does not serialize PDSEs
- Asking users not to update PDSEs from outside the SYSPLEX
  - Inevitably someone forgets
  - New users may not know the rules
- Reserves can cause serialization deadlocks

The key takeaway from this slide is that there is no 100% safe way to bypass the serialization requirements. PDSE cannot be serialized with 3rd party products, simply asking users not to update datasets from systems outside the plex is prone to human error and still doesn't prevent possible corruption on those systems. Even the use of reserves can cause serialization deadlocks. The main point is that in order maintain dataset integrity, you need to ensure that all sharing systems are limited to those within the sysplex.

Problem Prevention – Improper Sharing (Recovery)

- Determination
  - PDSE corruption is often surfaced through S0F4 abends on dataset access
  - IEC143I 213-50 OPEN Macro error
  - CSV031I LIBRARY SEARCH FAILED
  - Others
- Location
  - In-core/On-disk
  - EXEC PGM=IEBPDSE
- Recovery
  - In-Core: V SMS,PDSE(1),REFRESH,DSN(data.set.name)
  - On-Disk: IEBCOPY or Restore from valid backup

So next, lets talk about how to recover a dataset that is exhibiting signs of corruption. This often surfaces as a S0F4 abend on dataset access, a 213-50 OPEN error, or a few others. This corruption, like I mentioned previously, can be either in just the cached copy of the directory pages, where the cached pages become inconsistent with the valid on-disk data, or the pages can be corrupted on the disk itself.

First, you want to determine where the corruption is, and to do that, you can run the IEBPDSE job specifying the dataset in question. IEBPDSE validates the directory on disk, so if it runs clean then the corruption was only in the cache, otherwise we know that the dataset is corrupted on disk. If the corruption is in the cache, you can resolve the issue buy flushing the cached pages, with the Refresh command seen here. If the corruption was on disk, then the dataset will need to be recreated from scratch or restored from a valid backup.

Proper Sharing - IEC413I 213-70

- Issued to indicate prevention of serialization state that could lead to corrupted datasets
  - Warning message instead of actual error
  - Attempted access is stopped

Job 2 on System 2 Attempts OPEN For:	Job 1 on System 1 has PDSE Currently OPEN For:			
	Input	**Output**	**Update – Not Positioned to a Member**	**Update – Positioned to a Member**
Input	Successful	Successful - Job 2 reads old data, new data after last close.	Successful	OPEN ABEND – IEC143I 213-74
Output	Successful – Job 1 reads old data, new data after last close.	Successful – Contains data from last close.	Successful	OPEN ABEND – IEC143I 213-74
Update	OPEN ABEND – IEC143I 213-70	OPEN ABEND – IEC143I 213-70	OPEN ABEND – IEC143I 213-70	OPEN ABEND – IEC143I 213-70

Before we move away from dataset sharing, I do want to briefly mention a scenario that you may see when you are sharing your datasets correctly, and that is the IEC143I 213-70 or 74 message. Essentially, due to the serialization requirements to ensure dataset integrity, there are some concurrent access requests that cannot be honored, primarily surrounding the Update-in-Place processing for members of a PDSE dataset. This message is simply a warning that says the access attempted concurrently with a task/job doing an update-in-place is not granted. This is not an error, just a warning as we prevented a serialization state and disallowed the access that could have otherwise led to corruption. You can refer to this table for more information.

Problem Prevention – Resource Contention (Recovery)

- **Step 1: Get a Dump**
  - This easiest way to ensure that the dump captures the required information is via the V SMS,PDSE(1),DUMPNEXT command
- **Step 2: Finding the Scope**
  - Changing resources? → Check CPU utilization, not necessarily a hang
  - Single Holder/Resource Contention? → Investigate further
- **Step 3: AS Report**
  - IP VERBX SMSXDATA 'f(AS) jobname(SMSPDSE[1]) comp(CLM)'
  - Note the TCB and ASID of holder, and Latch Address + Offset
- **Step 4: Clear the Holder**
  - Cancel or Force if necessary → recovery may clear contention
- **Step 5: FREELATCH**
  - Ensure the holder is cleared out of the system before

And finally, we will discuss the recovery from actual resource contention, a job hang or an orphaned resource. This high-level overview will provide you with the basic steps needed to quickly resolve and restore operations that have been impacted by resource contention. First, you will want to take a dump, which can be automatically captured by specifying the DUMPNEXT command. You will also want to review the IGW031I messages to see if the resources are changing, in order to rule out those false positives I talked about earlier. Assuming that the resources are not changing and the contention time keeps increasing, you will then want to find the job/resource that is the source of the contention. This information can be found from the AS report in the dump, making note of the TCB, ASID, and Latch information. With that found, you will want to go clear the holder job from the system, either via cancel or a force of the job, if necessary, which might clear the contention during recovery processing. If the contention remains after the job is out of the system, you can then issue a FREELATCH command using the information pulled from the AS report, to manually release the resource and free up the system to continue.

There is a deeper dive into the recovery process that can found from a technote, which I have linked in the appendix of the presentation, as well as other relevant slides with sample jobs, cheat sheets, and other links all located within the appendix for your reference as well, so it may be worth downloading this presentation for your future reference.

The DFSMS team is proud to announce the brand-new z/OS DFSMS Guild!

Inspired by the success of the z/OSMF Guild, we wanted to provide a way for new to mid tenure storage administrators to learn directly from our subject matter experts on a wide range of DFSMS concepts; from the basics to advanced functions to new enhancements!

Our first session will be DFSMS Back to Basics on March 25th, 2026 at 10am EST with Frank McCune and Trevor Geisler. There is a QR code there that brings you to the community page with more information and the link to register. We hope to see you there!

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation

And with that, the presentation is complete. Please let me know if there are any questions you have.
Thank you.

SHARE
EDUCATE • NETWORK • INFLUENCE

APPENDIX
CHEAT SHEETS, PARAMETERS, COMMANDS AND JCL

Resource Contention

- **IGW038A POSSIBLE PDSE PROBLEM(S). (SMSPDSE|SMSPDSE1)**
  RECOMMEND ISSUING V SMS,PDSE(1),ANALYSIS
- **IGW031I PDSE ANALYSIS Start of Report**
  ++ no exceptional data set conditions detected ← Not an Issue
  End of Report
- **IGW031I PDSE ANALYZE Start of Report**
  ---------data set name---------- ---vsgt----------
  SYSPLEX.TEMP.PDSE 01-XP0201-000104
  ++ Unable to latch HL1b:hhhhhhhh
  Latch:llllllll Holder(aaaa,tttttttt)
  Holding Job:jjjjjjjj tttt.tt
  PDSE ANALYSIS END OF REPORT (SMSPDSE|1)

Appendix – NORMAL and EXTENDED Sharing Modes

SHARE
EDUCATE • NETWORK • INFLUENCE

- Normal Sharing is the Legacy PDSE Sharing mode
  - It provides the ability to share at the *dataset* level between systems.
    - It can share at a *member* level only within a single system.
  - It can only be implemented with the Non-Restartable Address Space (SMSPDSE).
    - If you wish to use the restartable Address Space you **must** use Extended Sharing.
  - Getting started with Normal Sharing
    - PDSESHARING(NORMAL) must be specified in the IGDSMSxx member.
    - In order to change from a Extended Sharing Mode to Normal Sharing Mode, you **must** IPL.
  - Normal Sharing Mode is not limited to systems in the same SYSPLEX.
    - However, all participating systems must belong to the same GRSPLEX

- Extended Sharing is the preferred method of sharing
  - It provides the ability to share at the *member* level between systems.
  - It works with either and/or both of the SMSPDSE Address Spaces active.
  - Getting started with Extended Sharing
    - PDSESHARING(EXTENDED) must be specified in the IGDSMSxx member.
    - The SYSPLEX sharing type is determined by the first PDSE Address Space to start.
    - IPL is recommended to start Extended Sharing.
      - ACTIVATE Command can be used, but may cause PDSE problems.
  - Extended Sharing is strictly limited to systems within the same SYSPLEX.
    - All participating systems must belong to the same GRSPLEX **AND** XCFPLEX

47

Appendix – NORMAL and EXTENDED Sharing Modes

SHARE
EDUCATE • NETWORK • INFLUENCE

- Sharing a PDSE outside of the XCFPLEX
  - By sharing a PDSE outside of the XCFPLEX in Extended Sharing, you are introducing unpredictable problems.
  - Corruption can cause 0F4 ABENDs
    - Varied symptoms make improper sharing harder to diagnose
  - Improper sharing can result in unserialized access to datasets.
    - There is **no** warning that a dataset has been accessed in this manner.
    - Potential issues:
      - Invalid index data in-core, Corrupt dataset on DASD, corrupt member data, mismatched extent data, or even nothing at all.
  - There is no sure fire way to circumvent Extended Sharing Mode's serialization requirements.
    - PDSE datasets cannot be serialized by 3rd party products.
    - Asking users not to update PDSEs from outside SYSPLEX
      - Too hard to enforce, inevitably someone forgets, new users may no know all the rules
    - Reserves can cause serialization deadlocks.

Appendix – PDSE Parameters, Commands, and JCL

- PDSE Console Dump Parameters

```
COMM=(PDSE PROBLEM)
JOBNAME=(*MASTER*,SMSPDSE*),
SDATA=(PSA,CSA,SQA,GRSQ,LPA,LSQA,RGN,SUM,SWA,TRT,COUPLE,XESDATA),END
```

- IGDSMSxx Parameters:
  - SMSPDSE1 restartable address space:

```
PDSE_RESTARTABLE_AS(NO | YES)
```

  - PDSE Sharing Modes:

```
PDSESHARING(EXTENDED|NORMAL)
```

  - PDSE Member Generations Installation Limit

```
MAXGENS_LIMIT=n
```

Appendix – PDSE Parameters, Commands, and JCL

- PDSE Console Commands
  - SMSPDSE1 Restart Command

```
V SMS,PDSE1,RESTART
  [,QUIESCE(duration | 15 )[,COMMONPOOLS(NEW|REUSE) ]
```

  - SMSPDSE1 Activate Command

```
V SMS,PDSE1,ACTIVATE
```

  - PDSE Analysis Command

```
V SMS,PDSE(1),ANALYSIS
```

  - PDSE Freelatch Command

```
V SMS,PDSE|PDSE1,FREELATCH(<latch address>,asid,tcb)]
```

Appendix – PDSE Parameters, Commands, and JCL

- IEBPDSE JCL (1.13 and above only)

```
//VALIDATE EXEC PGM=IEBPDSE
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSLIB DD DISP=SHR,DSN=INPUT.PDSE.BAD
```

Appendix – PDSE Parameters, Commands, and JCL

- DSS PHYSICAL dump JCL

```
//DUMP     EXEC PGM=ADRDSSU
//SYSPRINT DD  SYSOUT=*
//OUT      DD  UNIT=3390,
//             VOL=SER=xxxxxx,
//             DISP=(NEW,KEEP),
//             SPACE=(CYL,(100,100)),
//             DSN=hilev.DSSDUMP,
//             DCB=BLKSIZE=32760                    //SYSIN    DD  *
                                    DUMP  PIDY(vvvvvv) -
      OUTDD(OUT)  -
     DATASET(INCLUDE(pdse.dataset.name)) -
     ALLDATA(  *  )                                 /*
```

Appendix – Reference Links

- S0F4 PDSE Corruption
  - https://www.ibm.com/support/pages/pdse-abend0f4-reason-codes-may-indicate-pdse-corruption
- What to do when you get an IGW038A
  - https://www.ibm.com/support/pages/what-do-when-you-get-igw038a-message
- Dynamically Updating PDSE datasets in the LINKLST
  - https://www.ibm.com/support/pages/node/879235