



zEnterprise Data Compression

What is it and how do I use it?

Dale F. Riedy
IBM

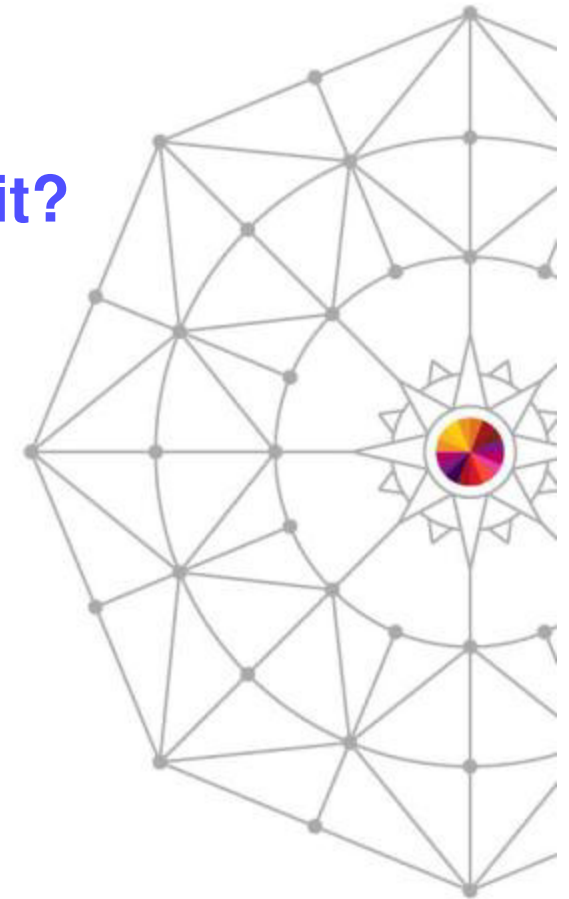
riedy@us.ibm.com

Anthony Sofia
IBM

asofia@us.ibm.com

March 12, 2014

Session 15081



Agenda

- Compression Overview
 - Why Compression
 - What is zEDC?
- zEDC Overview
 - What is DEFLATE?
 - Setup and configuration
- Using zEDC
 - zlib
 - Java
 - SMF
 - Sterling Connect:Direct

Other Sessions of Interest

- Session 14948 - Tools for Managing Big Data Analytics on z/OS
 - Presenters: Mike Stebner, Joe Sturonas
 - Wednesday, March 12, 2014 at 11:00
- Session 15207 - System z Batch Network Analyzer (zBNA) Tool
 - Presenter: John Burg
 - Wednesday, March 12, 2014 at 1:30
- Session 15080 - z/OS zEnterprise Data Compression Usage and Configuration
 - Presenter: Cecilia C. Lewis
 - Thursday, March 13, 2014 at 1:30

Explosive Growth In Data

Over 2000 Petabytes of Data Created Per Day



Data Compression will become pervasive

- I/O throughput is struggling to keep up with increasingly data driven applications
- Batch workloads are accessing more data from disk and network connections
- Business opportunities can be lost due to the cost prohibitive nature of keeping data online

Data needs to be shared across platforms

- Data is being exchanged among business partners
- Compression can substantially reduce the amount of data transferred
- Industry standard formats need to be used for transparent peer to peer communication

Compression solves problems in the enterprise

- Improves the effective throughput of data over storage and communication networks
- Allows more data to remain online for increases business value
- Less data to perform encryption operations against
- Makes high cost/byte storage technology such as flash memory more affordable

IBM z Enterprise Data Compression

New data compression offering



What is it?

- ✓ *zEDC Express is an IO adapter that does high performance industry standard compression*
- ✓ *Used by z/OS Operating System components, IBM Middleware and ISV products*
- ✓ *Applications can use zEDC via industry standard APIs (zlib and Java)*
- ✓ *Each zEDC Express sharable across 15 LPARs, up to 8 devices per CEC.*
- ✓ *Raw throughput up to **1 GB/s** per zEDC Express Hardware Adapter*

What Changes?

It is time to revisit your decisions about compression.

- **Disk Savings:** Many people are already getting value from CMPSC compression and software compression today
- **Performance:** High throughput alternative to existing System Z compression for large or active files.
- **Industry Standard:** Low cost compressed data exchange across all platforms
- **Pervasive:** Standard APIs allow quick adoption by middleware products running on System Z

What is the Value?

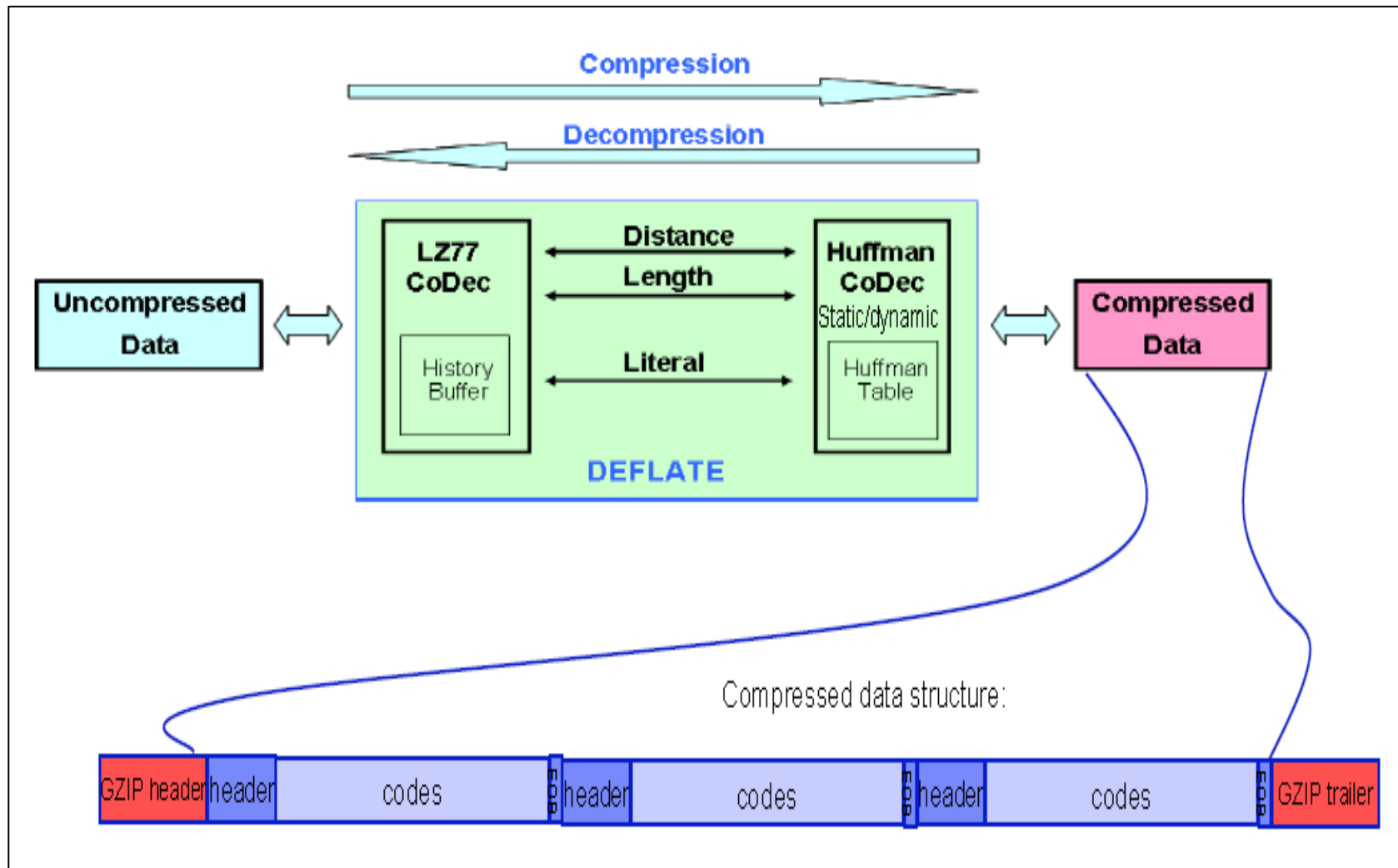
New sources of customer value

- **QSAM/BSAM** compression can save disk cost
- **Business Partner Data Exchange** can have higher throughput with lower CPU cost
- **Sterling Connect:Direct** saves additional link bandwidth, elapsed time.
- **ISV Products** delivery expanded customer value
- **Java** transparently accelerates java.util.zip
- **IBM Encryption Facility** for standard compliant data exchange
- **Improved availability** with SMF compression

What is DEFLATE?

- zEDC uses the DEFLATE file format – defined by the IETF RFC 1951 document
 - How deflate data is generated is up to the program
- The DEFLATE file is generated using the following two algorithms:
 - LZ77 (Lempel-Ziv 1977) – Provide pattern matching via a 32k rolling window in the data. Matches are replaced with a back reference.
 - Huffman encoding - Encodes the symbols into a set of bit patterns where the most frequently symbols get the smallest bit patterns
- Two types of Huffman encoding:
 - Static (Fixed) Huffman – predefined alphabet used to encode the symbols (RFC 1951)
 - Dynamic Huffman – alphabet is defined based on the symbols in the data. The Huffman tree alphabet is imbedded in the DEFLATE block
- The file format is bit aligned, meaning the symbols do not fall on byte boundaries

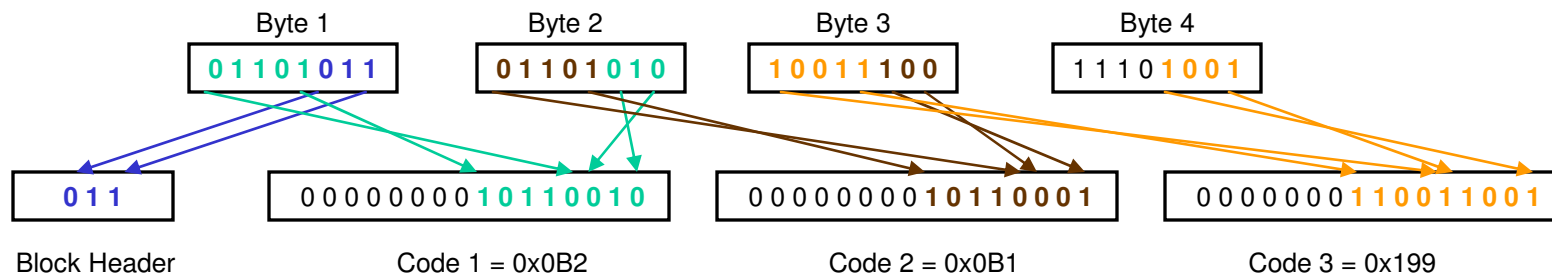
What is DEFLATE (cont)



What is DEFLATE (cont)

The alphabet of codes for the **Fixed Huffman** literals and length values are the following:

| Literal Value | Bit Length | Codes |
|-------------------------|------------|---------------|
| 0 – 143 (0x000-0x08F) | 8 | 0x030 – 0x0BF |
| 144 – 255 (0x090-0x0FF) | 9 | 0x190 – 0x1FF |
| 256 – 279 (0x100-0x117) | 7 | 0x000 – 0x017 |
| 280 – 287 (0x118-0x11F) | 8 | 0x0C0 – 0x0C7 |



The Block Header indicates that **BFINAL** is on and that this is a **BTYPE** of Static Huffman Codes (b'01'). The alignment of the header is such that the right-most bit represents **BFINAL** and the next two bits represent **BTYPE** with the most significant bit in the right-most position.

The DEFLATE RFC defines that each byte is formatted so the most significant bit is in the right-most position. Bits need to be collected until they form a valid code as code lengths vary from 7 to 9 bits.

Using the above example we can now determine the first three Literal Values for this stream.

$$0x0B2 - 0x030 + 0 \rightarrow 0x82 \text{ ('b')}$$

$$0x0B1 - 0x030 + 0 \rightarrow 0x81 \text{ ('a')}$$

$$0x199 - 0x190 + 144 \rightarrow 0x99 \text{ ('r')}$$

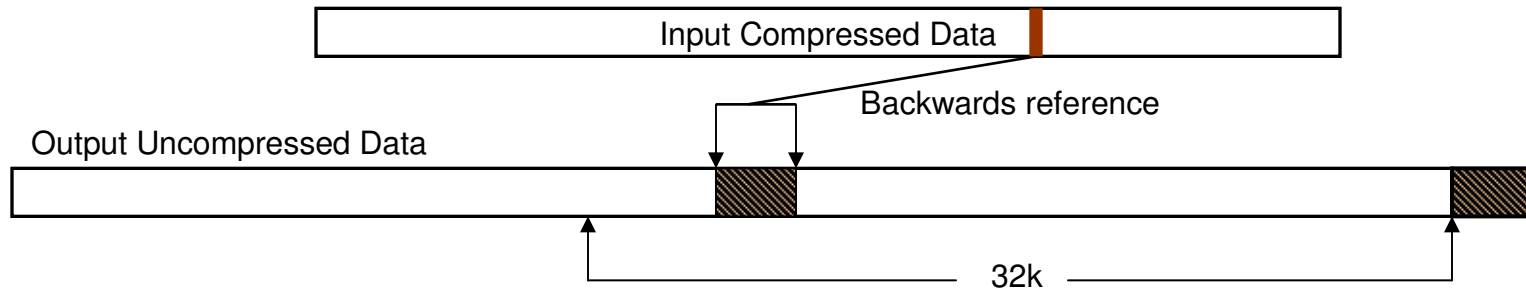
These calculations are done by taking the code, subtracting the start of the code range and adding the start of the literal range.

Example of Inflate (cont)

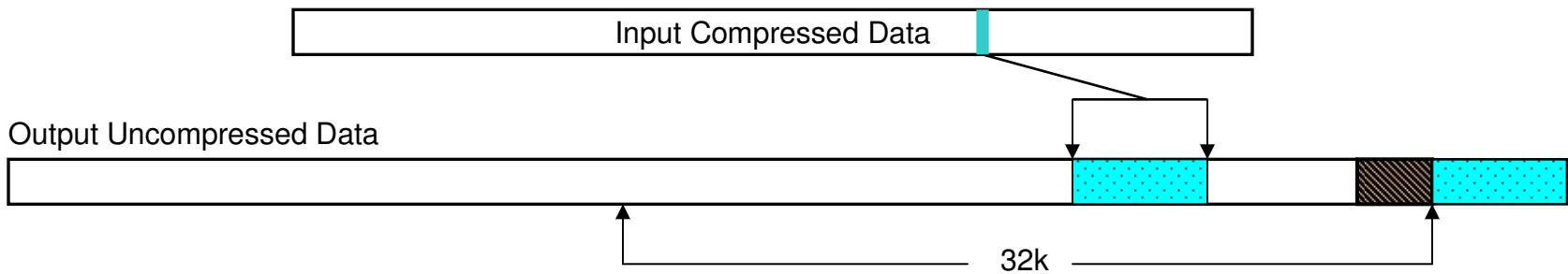
The DEFLATE compression format is considered dictionary-less. That means that the compressor of the file does not have to also save a dictionary of potential string matches.

This is accomplished with DEFLATE because the uncompressed data is the dictionary.

Take for instance a distance/length pair in the input compressed data stream.



After this symbol is decompressed, the rolling window looks like the following for the next symbol evaluation:



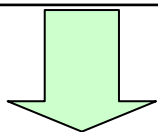
Example of Deflate

★ The complete example is in the appendix

Compressed Data
 bar bar bar bar
 boo boo boo boo
 blah blah blah blah
 foo boo blah
 blah foo blah
 hello

The 1st backwards reference length includes symbols that are part of the compressed symbols themselves as shown here:

| Byte# | String | Back Reference Len | Back Reference Character |
|-------|-----------------|--------------------|--------------------------|
| 4 | bar-b | n/a | n/a |
| 5 | bar-ba | 1 | 'a' at byte 1 |
| 6 | bar-bar | 2 | 'r' at byte 2 |
| 7 | bar-bar- | 3 | ' ' at byte 3 |
| 8 | bar-bar-b | 4 | 'b' at byte 4 |
| 9 | bar-bar-ba | 5 | 'a' at byte 5 |
| 10 | bar-bar-bar | 6 | 'r' at byte 6 |
| 11 | bar-bar-bar- | 7 | ' ' at byte 7 |
| 12 | bar-bar-bar-b | 8 | 'b' at byte 8 |
| 13 | bar-bar-bar-ba | 9 | 'a' at byte 10 |
| 14 | bar-bar-bar-bar | 10 | 'r' at byte 11 |



| Byte # | Codes and Values |
|----------|---|
| 00000000 | 00b2 = 0082 ('b') |
| 00000001 | 00b1 = 0081 ('a') |
| 00000002 | 0199 = 0099 ('r') |
| 00000003 | 0070 = 0040 (' ') |
| 00000004 | 00b2 = 0082 ('b') |
| 00000005 | <000a, 0004> = 0081 0099 0040 0082 0081 0099 0040 0082 0081 0099 |
| 00000015 | 0045 = 0015 ('\n') |
| 00000016 | 00b2 = 0082 ('b') |
| 00000017 | 0196 = 0096 ('o') |
| 00000018 | 0196 = 0096 ('o') |
| 00000019 | 0070 = 0040 (' ') |
| 00000020 | <000b, 0004> = 0082 0096 0096 0040 0082 0096 0096 0040 0082 0096 0096 |
| 00000031 | 0045 = 0015 ('\n') |
| ... | |

Start of overlap

Backward reference: length=10, distance=4 ('a' character at byte# 5)

zEDC Configuration Overview



▪ Operating system requirements

- Requires z/OS 2.1 and new zEDC Express for z/OS feature
 - PTF Support for BSAM/QSAM in 1Q
- z/OS V1.13 and V1.12 offer software decompression support only

▪ Server requirements

- Exclusive to zEC12 and zBC12
- New zEDC Express feature for PCIe I/O drawer (FC#0420)
 - One compression coprocessor per zEDC Express feature
 - Each feature can be shared across up to 15 LPARs
 - Up to 8 features available on zEC12 or zBC12
- Recommended high availability configuration per server is four features
 - No additional software cost is incurred by increasing the number of hardware features
 - Provides high availability during concurrent update (half devices unavailable during update)
 - Recommended minimum configuration per service is two features
- Hot pluggable
- For best performance, feature is ***needed on all systems accessing the compressed data***

▪ Capacity Planning

- The **z Batch Network Analyzer** now reports on potential zEDC usage for QSAM/BSAM data sets

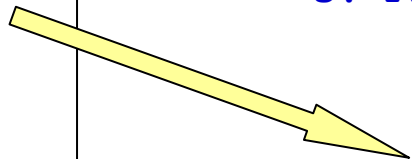
HCD - Define, Modify or View Configuration Data



Define, Modify, or View Configuration Data

Select type of objects to define, modify, or view data.

- 3_ 1. Operating system configurations
 - consoles
 - system-defined generics
 - EDTs
 - esoterics
 - user-modified generics
2. Switches
 - ports
 - switch configurations
 - port matrix
3. Processors
 - channel subsystems
 - partitions
 - channel paths
 - functions**
4. Control units
5. I/O devices
6. Discovered new and changed control units and I/O devices



HCD - Displaying the Processor List



```
Goto  Filter  Backup  Query  Help
-----
                                Processor List          Row 1 of 11 More:  >
Command ==> _____ Scroll ==> PAGE

Select one or more processors, then press Enter. To add, use F11.

/ Proc. ID Type +  Model +  Model+ Serial-# + Description
_ D76      2094   S28    LPAR   _____ ICSF z9
_ H87      2097   E40    LPAR   _____ H87 STP Secondary
_ H88      2097   E40    LPAR   _____ H88 STP Primary
_ H89      2097   E64    LPAR   _____ H89 IOS Test z10
_ M32      2818   M10    LPAR   _____ M32 IOS Test zMR
F PBUV4   2827   H43    LPAR   _____ PBUV4 zEC12
_ P87      2827   H66    LPAR   _____ P87 zEC12
_ P88      2827   H66    LPAR   _____ P88 zEC12
_ P89      2827   HA1    LPAR   _____ P89 zEC12
_ P92      2827   H89    LPAR   _____ P92 zEC12 JEMT
_ R89      2817   M80    LPAR   _____ R89 IOS Test z196
***** Bottom of data *****
```



HCD - Displaying PCIe Functions



```
Goto  Filter  Backup  Query  Help
-----
                                PCIe Function List      Row 1 of 7 More:      >
Command ==> _____ Scroll ==> PAGE

Select one or more PCIe functions, then press Enter. To add, use F11.

Processor ID . . . . : PBUV4          PBUV4 zEC12

/  FID    PCHID  VF+  Type+          Description
-  010    340    1    ZEDC EXPRESS  _____
-  011    340    2    ZEDC EXPRESS  _____
-  012    340    3    ZEDC EXPRESS  _____
-  013    340    4    ZEDC EXPRESS  _____
-  014    340    5    ZEDC EXPRESS  _____
-  020    360    1    ZEDC EXPRESS  _____
-  021    360    2    ZEDC EXPRESS  _____
***** Bottom of data *****
```



HCD - Displaying PCIe Functions – Part 2



```

Goto  Filter  Backup  Query  Help
-----
                                PCIe Function List          Row 1 of 7 More: <  >
Command ==> _____ Scroll ==> PAGE

Select one or more PCIe functions, then press Enter. To add, use F11.
Processor ID . . . . : PBUV4          PBUV4 zEC12
1=OS S01          2=OS S02          3=OS S03          4=OS TA4          5=OS S3A
6=OS TA0          7=  *          8=  *          9=  *          A=  *
B=  *          C=  *          D=  *          E=  *          F=  *

----- Partitions 0x -----
/ FID      PCHID  VF      1 2 3 4 5 6 7 8 9 A B C D E F
- 010      340    1      - - - - - a # # # # # # # # #
- 011      340    2      - - - - -   # # # # # # # # #
- 012      340    3      - - - - -   # # # # # # # # #
- 013      340    4      - - - - -   # # # # # # # # #
- 014      340    5      - - - a -   # # # # # # # # #
- 020      360    1      - - - - - a # # # # # # # # #
- 021      360    2      - - - - -   # # # # # # # # #
***** Bottom of data *****

```



HCD - Adding a PCIe Function



Add PCIe Function

Specify or revise the following values.

Processor ID : PBUV4 PBUV4 zEC12

Function ID 022

Type zedc express +

PCHID 360

Virtual Function ID . . 3_ +

Description _____

F1=Help F2=Split F3=Exit F4=Prompt F5=Reset F9=Swap
F12=Cancel



Enabling the zEDC Express for z/OS Feature

```

PRODUCT OWNER ('IBM CORP')
      NAME ('Z/OS')
      FEATURENAME ('ZEDC')
      ID (5650-ZOS)
      VERSION (*)
      RELEASE (*)
      MOD (*)
      STATE (ENABLED)
  
```

- Update the IFAPRDxx parmlib member with the new feature
- **Warning** – This must be set at IPL time!!!

```

d prod,state,featurename (ZEDC)
IFA111I 22.18.15 PROD DISPLAY 000
S OWNER          NAME          FEATURE          VERSION          ID
E IBM CORP      Z/OS          ZEDC            * .* .*         5650-Z
  
```

```

D IQP
IQP066I 22.21.59 DISPLAY IQP 003
zEDC Information
MAXSEGMENTS:           0 (0M)
Previous MAXSEGMENTS:  N/A
Allocated segments:    0 (0M)
...
Feature Enablement:    Enabled
  
```

- How to verify that the feature is enabled.

Configuring PCIe Devices to z/OS



CONFIG PFID(nn) command can be used to configure a PFID offline or online to z/OS

```
cf pfid(10),offline,force
IQP034I PCIE FUNCTION 0010 NOT AVAILABLE FOR USE.
  PCIE DEVICE TYPE NAME = (Hardware Accelerator    ).
IQP034I PCIE FUNCTION 0010 AVAILABLE FOR CONFIGURATION.
  PCIE DEVICE TYPE NAME = (Hardware Accelerator    ).
IEE505I PFID(10),OFFLINE
IEE712I CONFIG    PROCESSING COMPLETE
```

```
cf pfid(10),online
IQP034I PCIE FUNCTION 0010 ONLINE.
  PCIE DEVICE TYPE NAME = (Hardware Accelerator    ).
IEE504I PFID(10),ONLINE
IEE712I CONFIG    PROCESSING COMPLETE
```

Note: PFIDs can also be configured offline and online via the SE. z/OS will react to the PCIe availability events that are presented and react accordingly.

Displaying PCIe Devices

Device may appear twice in display after dynamic I/O or config. DP state means device waiting to be cleaned up.

ALLC state does not necessarily mean device is usable. Issue PCIE,PFID=nn command.

```

D PCIE
IQP022I 11.27.32 DISPLAY PCIE 313
PCIE      0010 ACTIVE
PFID  DEVICE TYPE NAME          STATUS  ASID  JOBNAME  PCHID  VFN
0000  Hardware Accelerator      DP      013A  FPGHWAM  0380  0001
0010  Hardware Accelerator      ALLC    013A  FPGHWAM  053C  0001
0000  Hardware Accelerator      CNFG                    0380  0001
0013  10GbE RoCE                  ALLC    002C  VTAM
0014  10GbE RoCE                  STNBY
0099  Hardware Accelerator      ALLC    013A  FPGHWAM  053C  0002
00AA  UNNAMED (1810 3D03)         CNFG
    
```

Devices are not necessarily displayed in PFID order.

Device type and vendor id are not known to z/OS

zEDC express devices show up as hardware accelerators. D PCIE,PFID=nn shows type of accelerator.

Address space that allocated device. FPGHWAM allocates all H/W accelerators.

Displaying PCIe Device Details



Application description specifies the type of H/W accelerator (if known)

Detailed device information is displayed only if the device is allocated by FPGHWAM address space.

```
D PCIE, PFID=10
IQP024I 16.33.20 DISPLAY PCIE 157
PCIE      0012 ACTIVE
PFID  DEVICE TYPE NAME          STATUS  ASID  JOBNAME  PCHID VFN
0010  Hardware Accelerator      ALLC    0013  FPGHWAM  053C  0001
CLIENT ASIDS: NONE
Application Description: zEDC Express
Device State: Ready
Adapter Info - Relid: 000000 Arch Level: 03
              Build Date: 06/12/2013 Build Count: 09
Application Info - Relid: 000000 Arch Level: 00
```

Device is usable only if in the Ready state. All other states, except intervention required are transient.

Adapter and application info are obtained from the device itself

zlib

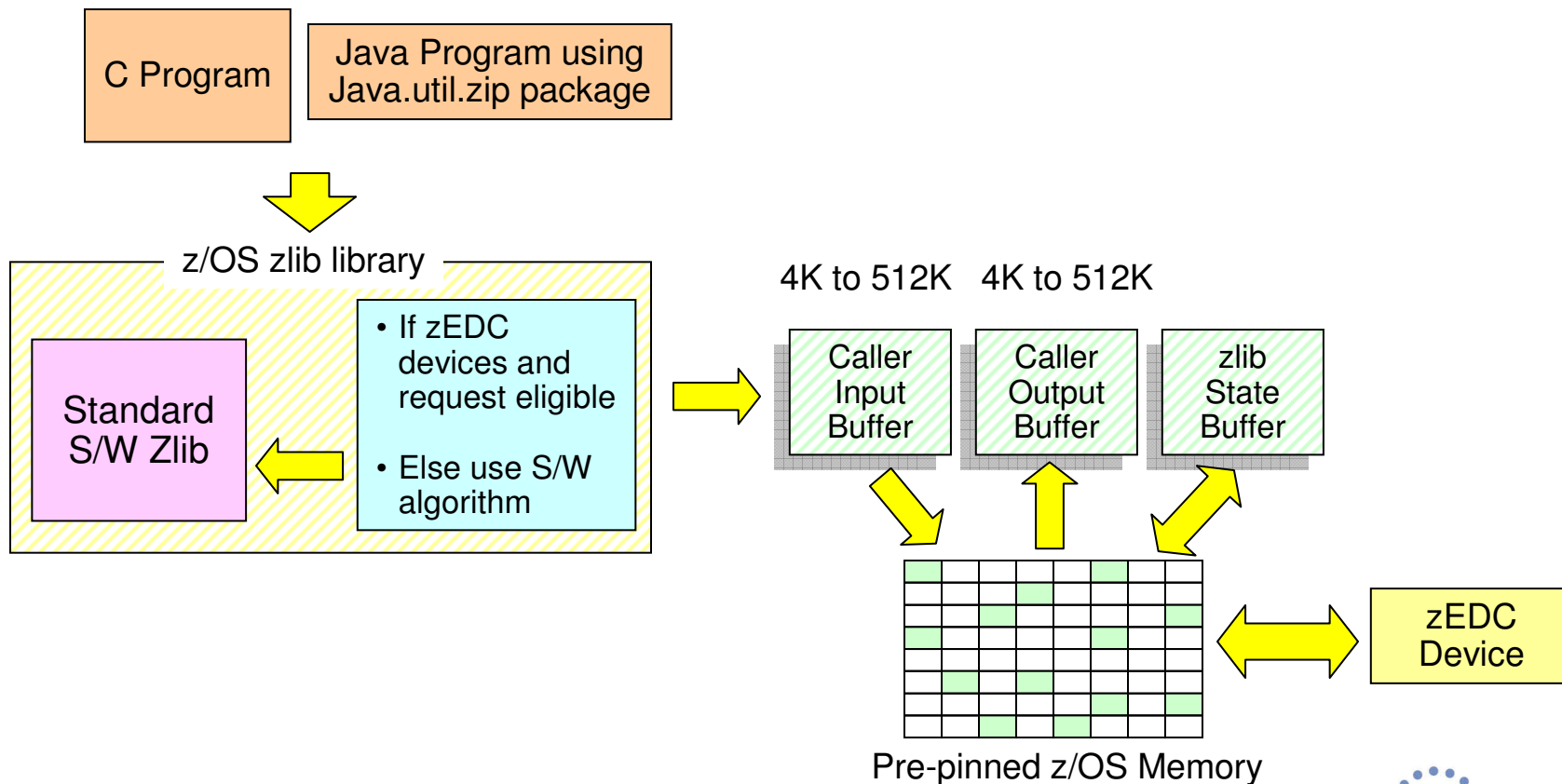


- zlib is a widely used open source C library that provides compression and decompression
 - Supports RFC 1950, 1951, and 1952
- Streaming model – data can be compressed or decompressed in chunks
- In order to use zEDC for compression/decompression
 - Relink applications with new IBM provided zLIB
 - Address spaces need READ access to the FPZ.ACCELERATORS.COMPRESSION SAF resource
 - First Inflate() or Deflate() must be at least as large as INFLATE_MIN_SIZE and DEFLATE_MIN_SIZE respectively in IQPPRMxx
 - The window size for deflate must be 32K
 - Note: The `_H_ZC_COMPRESSION_METHOD` environmental variable can be used to force S/W compression
- More information can be found in z/OS V2R1.0 MVS Callable Services for HLL

zlib Internal Structure

If the request can not be performed using zEDC then the software zlib code will be used for the request.

The zEDC requests are not done directly from user storage; the input data is copied into pre-allocated buffers and the output data is copied from these buffers back to user storage.



zlib Configuration Options



The IQPPRMxx member in SYS1.PARMLIB can be used to adjust internal settings for zlib behavior. For example

```
ZEDC, DEFMINREQSIZE=5, MAXSEGMENTS=7
```

This member will set the minimum input size of a deflate request via zlib to 5Kb and will set the maximum internal buffer size to 7 16Mb segments. The current settings and buffer usage can be displayed with the D IQP command:

```
D IQP
IQP066I 10.12.37 DISPLAY IQP 364
zEDC Information
MAXSEGMENTS:          7 (112M)
Previous MAXSEGMENTS: 4 (64M)
Allocated segments:   1 (16M)
Used segments:        0 (0M)
DEFMINREQSIZE:        5K
INFMINREQSIZE:        16K
Feature Enablement:   Enabled
```

Updated values from IQPPRMxx

Java and zEDC with java.util.zip



Java 7.1 provides zEDC access via the java.util.zip Inflater and Deflater classes. The same conditions that apply to zlib also apply to Java.

This example (try/catch blocks removed) shows the critical buffer sizes

```
byte buffer[] = new byte[64 * 1024];
byte outputFile[];

input = new FileInputStream(argv[0]);
output = new ByteArrayOutputStream();
gzStream = new GZIPOutputStream(output, 4096);

for(;;) {
    readBytes = input.read(buffer);

    if(readBytes < 0) {
        break;
    }
    else {
        gzStream.write(buffer, 0, readBytes);
    }
}
```

64Kb input buffer for deflate(). This must be >= DEFMINREQSIZE

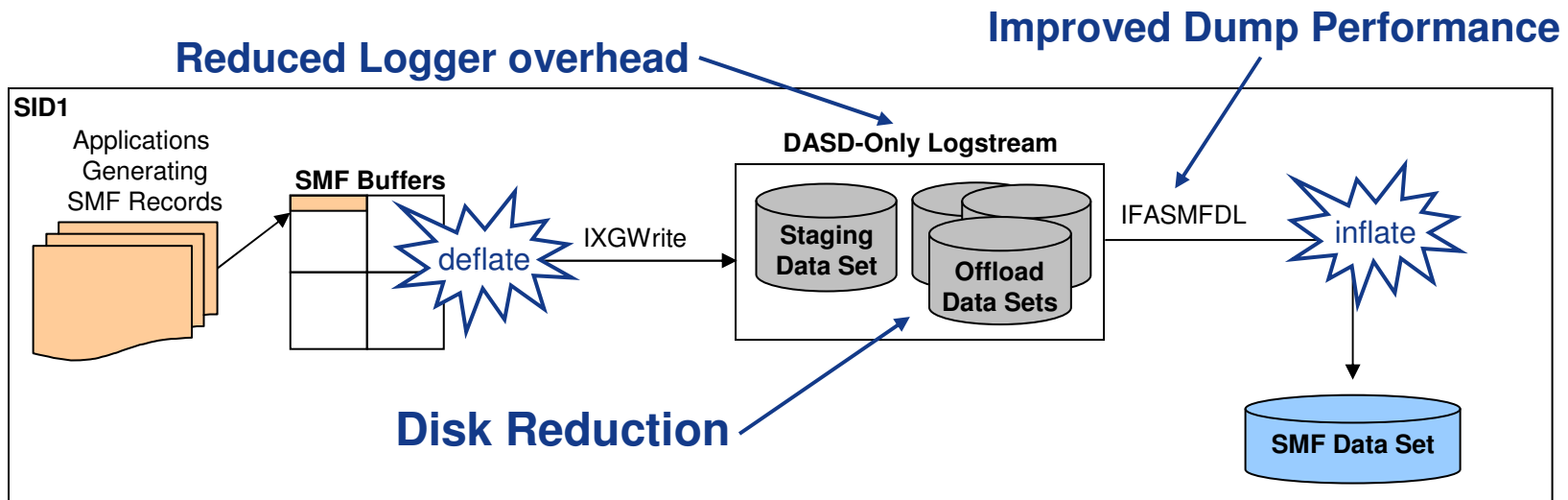
4Kb output buffer for deflate()

Data is read from an uncompressed file and written to a compressed file

When run on Java 7.1 the IBM Encryption Facility can use zEDC to perform compression.

SMF with zEDC

This example shows a DASD-Only logstream used for SMF recording



- Compression SMF logstreams reduce the amount of data in System Logger up to **4x** and reduce the elapsed time to extract IFASMF DL data up to **15%**
- zEDC compression must be available on all systems that will access zEDC compressed SMF logstreams
- Setup from SMFPRMxx either globally or per Logstream

```
SMFPRMxx in SYS1.PARMLIB  
DEFAULTLSNAME(DEFAULT,...,COMPRESS)  
LSNAME(SMF30,TYPE(30),...,COMPRESS(PERMFIX(10M)))  
LSNAME(RMF,TYPE(70:79)...,COMPRESS)
```

zEDC with IBM Sterling Connect:Direct for z/OS



High performance data compression for Managed File Transfer

- Sterling Connect:Direct can automatically leverage zEDC Express Accelerator for file compression and decompression as files are transferred
- Works with various dataset and file types on z/OS
- File transfers can be z/OS to z/OS, or z/OS to Distributed (UNIX, Linux, zLinux, and Windows)
- Fully compatible with zlib compression used in IBM Sterling Connect:Direct today – no changes required at end points

Up to **5%** compression improvement with zEDC over zlib software compression

44-82% reduction in elapsed time to transfer a file from z/OS to z/OS (results vary by dataset type and characteristics of the data)

Significant improvement in CPU time with zEDC over zlib software compression (sender TCB time and receiver TCB time)

zEDC RMF Reporting



New RMF report shows the utilization of each device.

RMF Postprocessor Interval Report : PCIE Activity Report

RMF Version : z/OS V2R1 SMF Data : z/OS V2R1
Start : 02/24/2014-05.48.00 End : 02/24/2014-05.48.44 Interval : 00:45:000 minutes

Hardware Accelerator Activity

| Function ID ↓↑ | Time Busy % ↓↑ | Request Execution Time ↓↑ | Std Dev for Request Execution Time ↓↑ | Request Queue Time ↓↑ | Std Dev for Request Queue Time ↓↑ | Request Size ↓↑ | Transfer Rate T |
|----------------|----------------|---------------------------|---------------------------------------|-----------------------|-----------------------------------|-----------------|-----------------|
| 0013 | 0.689 | 7.78 | 0.417 | 15.0 | 0.953 | 24.3 | 21.5 |

Hardware Accelerator Compression Activity

| Function ID ↓↑ | Compression Request Rate ↓↑ | Compression Throughput ↓↑ | Compression Ratio ↓↑ | Decompression Request Rate ↓↑ | Decompression Throughput ↓↑ | Decompression Ratio ↓↑ | B |
|----------------|-----------------------------|---------------------------|----------------------|-------------------------------|-----------------------------|------------------------|----|
| 0013 | 885 | 11.6 | 1.14 | 0 | 0 | | 64 |

The percent of this interval where this specific zEDC Express device was executing requests

Compression ratio of all requests serviced by zEDC. This will span all users of this device.

Average request queue time in Microseconds for this device.



Thank You!



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

Appendix

What is DEFLATE (an example)



This DEFLATE file example shows how each symbol is encoded include back references

Compressed Data

bar bar bar bar

boo boo boo boo

blah blah blah blah

foo boo blah

blah foo blah

hello

Byte # Codes and Values

```
00000000 00b2 = 0082 ('b')
00000001 00b1 = 0081 ('a')
00000002 0199 = 0099 ('r')
00000003 0070 = 0040 (' ')
00000004 00b2 = 0082 ('b')
00000005 <000a, 0004> = 0081 0099 0040 0082 0081 0099 0040 0082 0081 0099
00000015 0045 = 0015 ('\n')
00000016 00b2 = 0082 ('b')
00000017 0196 = 0096 ('\o')
00000018 0196 = 0096 ('\o')
00000019 0070 = 0040 (' ')
00000020 <000b, 0004> = 0082 0096 0096 0040 0082 0096 0096 0040 0082 0096 0096
00000031 0045 = 0015
00000032 00b2 = 0082
00000033 0193 = 0093
00000034 00b1 = 0081
00000035 00b8 = 0088
00000036 0070 = 0040
00000037 <000e, 0005> = 0082 0093 0081 0088 0040 0082 0093 0081 0088 0040 0082 0093 0081 0088
00000051 0045 = 0015
00000052 00b6 = 0086
00000053 <0008, 0020> = 0096 0096 0040 0082 0096 0096 0040 0082
00000061 <0004, 000d> = 0093 0081 0088 0015
00000065 <0005, 0017> = 0082 0093 0081 0088 0040
00000070 00b6 = 0086
00000071 <0008, 000e> = 0096 0096 0040 0082 0093 0081 0088 0015
00000079 00b8 = 0088
00000080 00b5 = 0085
00000081 0193 = 0093
00000082 0193 = 0093
00000083 0196 = 0096
00000084 0045 = 0015
```

