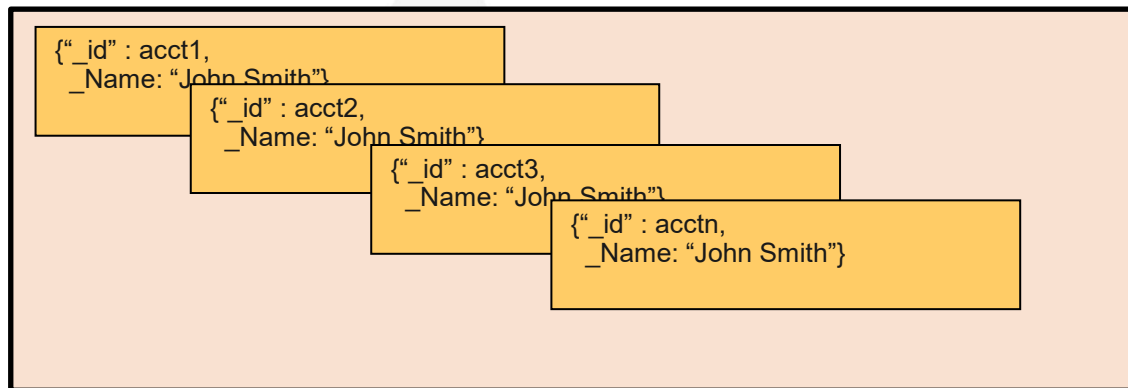


EzNoSQL Key:Value JSON Store for z/OS Session #10440



Terri Menendez
STSM
zSystems
IBM Corp
terriam@us.ibm.com
Aug 2024

Disclaimer

***Information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at IBM's sole discretion.*

Agenda

- ☐ Overview ***
- ☐ JSON Documents
 - Why JSON?
- ☐ Content Solution Website
 - Getting Started
 - API Documentation
 - Getting Started
 - ✓ Executables and Side Decks
 - ✓ Sample Programs
- ☐ EzNoSQL Indexes
 - Index Examples
- ☐ Recoverable vs Non-Recoverable Databases
- ☐ C APIs
- ☐ Java APIs
- ☐ Python APIs ***
- ☐ API Examples
- ☐ Performance Proof Points ***
- ☐ Futures

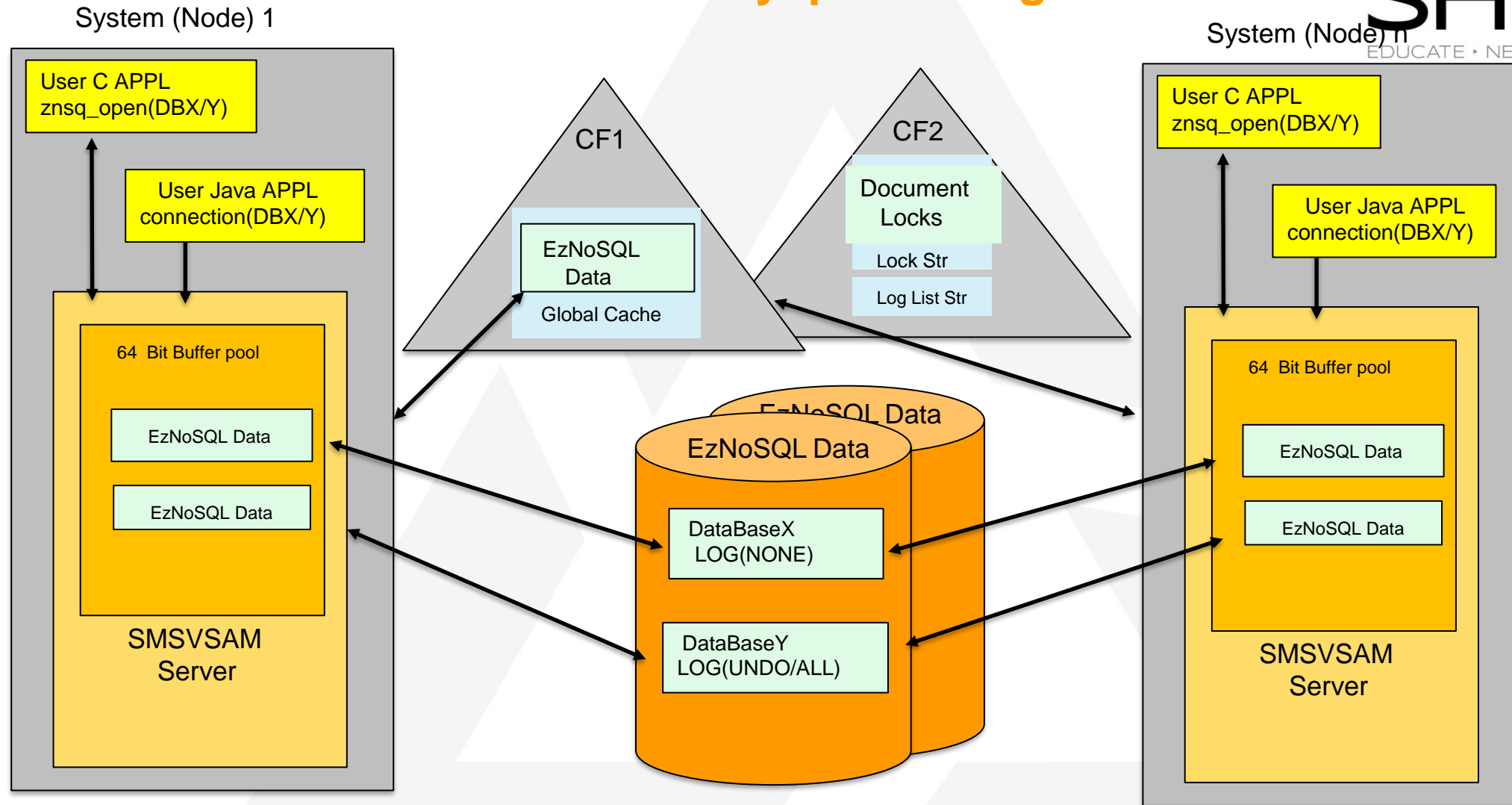
*** Python updates

EzNoSQL Overview

EzNoSQL Overview

- ❑ EzNoSQL on z/OS provides a **comprehensive set of C, Java, and Python based Application Programmer Interfaces (APIs)**:
 - enables applications to access JSON (UTF-8) documents
 - while utilizing the full data sharing capabilities of IBM's Parallel Sysplex Coupling Facility (CF) technology and System z operating system (z/OS).
- ❑ IBM's CF technology provides shared memory between processors which
 - enables separate processors to **share a single instance of a data base** (collection of documents)
 - without the need for **data sharding, replicating** the updates, or programming for **eventual consistency**.
- ❑ Sysplex allows for easy **horizontal scalability** by adding additional processors (or z/OS instances) as required.
- ❑ Implementing EzNoSQL on z/OS will inherit many of the desired functions provided by z/OS such as **system managed storage, data encryption and compression**.
- ❑ Available on z/OS 2.4 and above with APAR **OA62553 (C APIs)**, **OA64018/OA64811 (Java APIs z/OS 2.4/2.5/3.1)**, and **Python APIs**.

EzNoSQL Sysplex Design



JSON Documents

JSON (UTF-8) Documents:

Document:

{ element, element,... }

- ! One document (object) = one VSAM record
- ! Elements can vary in location, number, contents, and size.

Where an element is:

"key" : value

- ! Keys are character strings,
Values can be strings, numbers, arrays,
etc. JSON supports 5 types.

JSON Example:

```
{ "_id" : "00000001",  
  "Name" : "John Smith" }
```

JSON UTF-8 representation:

```
7B225F6964223A223030303030303031222C2224E616D65223A2224A6F686E20536D697468227D  
{ " _ i d " : " 0 0 0 0 0 0 0 1 " , " N a m e " : " J o h n   S m i t h " }
```

For complete specification on JSON:
www.json.org

Why JSON?

NoSQL (e.g. unstructured) key:value data bases have the following characteristics:

- ❑ The data (objects) may have widely varying formats:
 - {"Name" : "John Smith", "Address" : "24 First St", "Email" : "JohnSmith@gmail.com"}
 - {"Address" : "1 North St", "Name" : "Joe Jones" , "Gender" : "M" , "Married" : "Y"}
- ❑ Data format not directly related to the file definition. Allows for rapid application changes.
- ❑ No need for a DBA to manage the data base.

VSAM data bases (e.g. “semi-structured”) have the following characteristics:

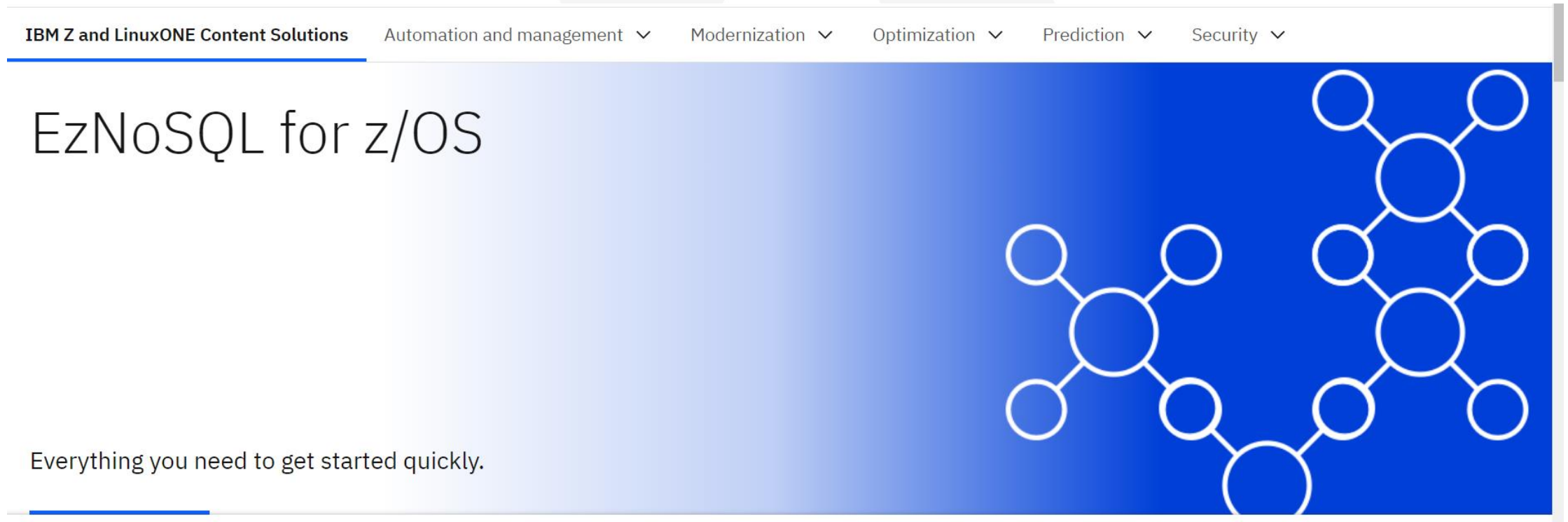
- ❑ The data (records) are partly structured and partly varying:
 - #1256789 **John Smith** (flags) 24 First St JohnSmith@gmail.com
 - #3763521 **Joe Jones** (flags) M Y
- ❑ Data format must have **primary** and **alternate** keys in the same location and length.
- ❑ Changes to the application related to the key's location or length require the data base to be redefined.

Content Solution Web Site

Getting Started...

Website:

<https://www.ibm.com/support/z-content-solutions/eznosql>



The screenshot shows the IBM Z and LinuxONE Content Solutions website. The navigation bar includes links for Automation and management, Modernization, Optimization, Prediction, and Security. The main heading is "EzNoSQL for z/OS". Below the heading, there is a blue banner with a white network diagram consisting of interconnected circles. The text "Everything you need to get started quickly." is displayed at the bottom of the banner.

IBM Z and LinuxONE Content Solutions Automation and management Modernization Optimization Prediction Security

EzNoSQL for z/OS

Everything you need to get started quickly.

Getting Started...

Documentation:

IBM Z and LinuxONE Content Solutions

Automation and management ▾

Modernization ▾

Optimization ▾

Prediction ▾

Security ▾

Introduction

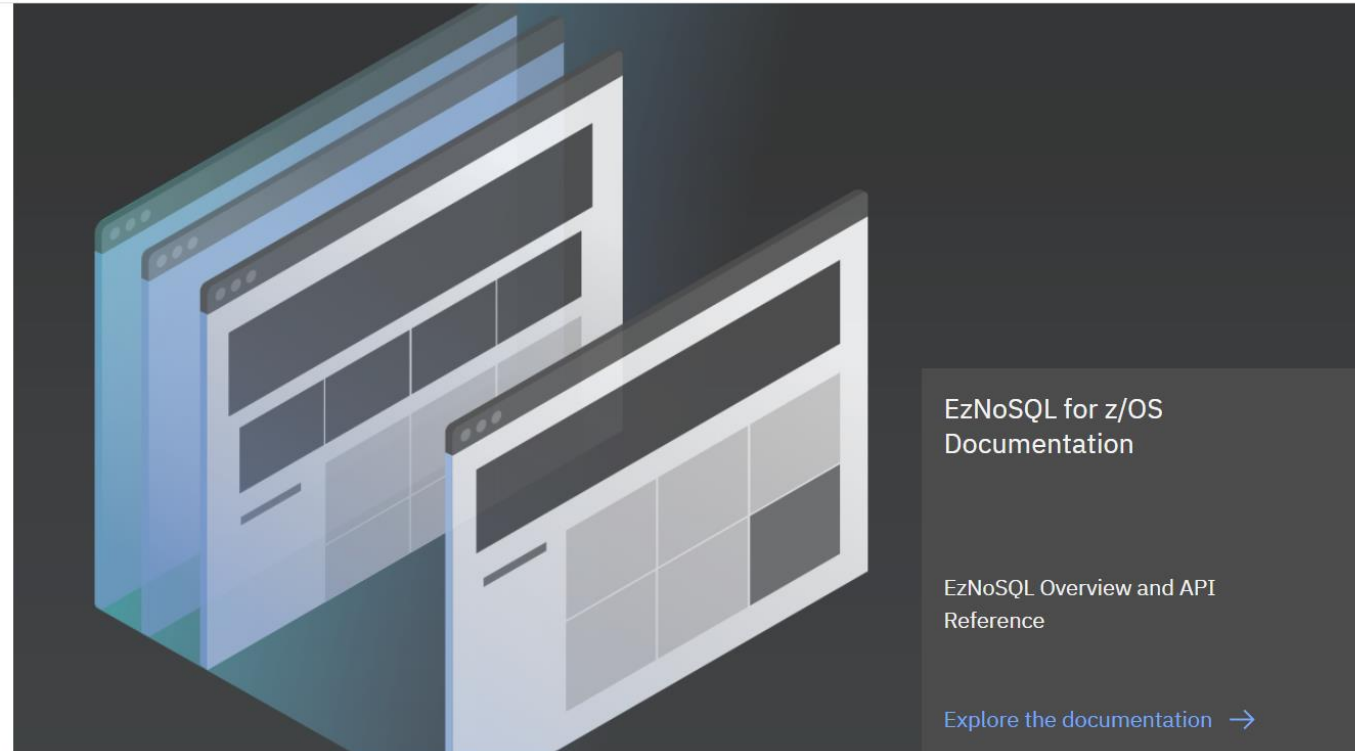
Big picture

Get started

Documentation

Technical resources

Rate this content



Explore the Documentation

EzNoSQL Documentation Reference

Maintained by VSAM RLS

Welcome to the EzNoSQL landing page.

- [EzNoSQL Documentation](#)
- [Javadoc API Reference](#)



EzNoSQL Documentation and C APIs

Documentation in GitHub:

GitHub
EzNoSQL for z/OS
Documentation



EzNoSQL C API Technical Documentation

Table of Contents

Introduction and Concepts:

- [Introduction to EzNoSQL](#)
- [JSON Documents](#)
- [Primary Keys](#)
- [Secondary Indexes](#)
- [Active Secondary Indexes](#)
- [Non-Unique Secondary Indexes](#)
- [Multi Level Keys](#)
- [Document Retrieval](#)
- [Recoverable Databases](#)

System Requirements

- [System Requirements](#)
- [Hardware/Software Requirements](#)
- [Storage Administration Requirements](#)
- [Application Requirements](#)

Performance Considerations:

- [In Memory Caching](#)

Getting Started:

- [Getting Started with EzNoSQL](#)
- [EzNoSQL Executables and Side Decks](#) ←
- [Sample Application Program](#)
- [Compile and Link Procedure](#)

Application Programming Interfaces (APIs)

- [Application Programming Tiers](#)

Data Management APIs:

- [znsq_create](#)
- [znsq_create_index](#)
- [znsq_destroy](#)
- [znsq_add_index](#)
- [znsq_drop_index](#)
- [znsq_report_stats](#)

Connection Management APIs:

- [znsq_open](#)
- [znsq_close](#)

Document Retrieval APIs:

- [znsq_read](#)
- [znsq_position](#)
- [znsq_next_result](#)
- [znsq_close_result](#)

Document Management APIs:

- [znsq_write](#)
- [znsq_delete](#)
- [znsq_delete_result](#)
- [znsq_update](#)
- [znsq_update_result](#)
- [znsq_commit](#)
- [znsq_set_autocommit](#)
- [znsq_abort](#)

Diagnostic Management APIs:

- [znsq_last_result](#)

Return and Reason Codes:

Return and Reason Codes:

- [Return Code 0](#)
- [Return Code 4](#)
- [Return Code 8](#)
- [Return Code 12](#)
- [Return Code 16](#)
- [Return Code 36](#)

Getting Started...

Executables and Side Decks

The following table shows the names and locations of the EzNoSQL executables, side decks, and sample program:

Member	Location	Description
<code>libigwznsqd31.so</code>	<code>/usr/lib/</code>	31-bit API Library DLL
<code>libigwznsqd31.x</code>	<code>/usr/lib/</code>	31-bit x side deck
<code>libigwznsqd64.so</code>	<code>/usr/lib/</code>	64-bit API Library DLL
<code>libigwznsqd64.x</code>	<code>/usr/lib/</code>	64-bit APIs
<code>igwznsqdb.h</code>	<code>/usr/include/zos/</code>	EzNoSQL Header File
<code>igwznsqsamp1.c</code>	<code>/samples/</code>	Sample 31-bit application program

Getting Started...

Java JAR and JNI library

Member	Location	Description
<code>libigwznsqj.so</code>	<code>/usr/lib/</code>	JNI Shared Library
<code>igwznsq.jar</code>	<code>/usr/include/java_classes/</code>	Java API JAR file
<code>Igwznsqsamp1.java</code>	<code>/samples/</code>	Sample Java program

Example C Program:

Executable example located in USS: /samples/igwznsqsamp1.c or /samples/IBM/igwvrvip
(Minor customization required (HLQ and STORCLAS) documented in prolog)

```
//  
// Program Name: igwznsqsamp1.c  
//  
// Function: Verify the EzNoSQL API's provided in z/OS 2.4 by:  
//           Use the API's to create a JSON database and index,  
//           write, position and read three JSON documents using an  
//           alternate key, close and destroy the JSON database.  
//  
//           Note: Before running this program, be sure the PTF for  
//           the EzNoSQL enabling APAR OA62553 has been  
//           installed on your system.  
//  
// To compile and link this sample program:  
// xlc -c -qDLL -qcpluscmt -qLSEARCH="//'SYS1.SCUNHF'" igwznsqsamp1.c  
// xlc -o igwznsqsamp1 igwznsqsamp1.o -W l,DLL /usr/lib/libigwznsqd31.x  
//
```

Getting Started...

Compile and Link the Sample C Program:

Compile and Link Procedure for C Sample Program

To compile and link the sample program `/samples/igwznsqsamp1.c` :

```
xlc -c -qDLL -qcpluscmt -qLSEARCH="//'SYS1.SCUNHF'" igwznsqsamp1.c  
xlc -o igwznsqsamp1 igwznsqsamp1.o -W l,DLL /usr/lib/libigwznsqd31.x
```

Example C Program:

```
# igwznsqsample
Database HL1.NOSQLDB created
Index HL1.NOSQLDB.AIX created
Database opened
Index for "Author" added
Document 1 written
Document 2 written
Document 3 written
Position to first document
Return code received from next_result was: X'34'
Verify first document
Return code received from next_result was: X'34'
Verify second document
Return code received from next_result was: X'0'
Verify third document
Database closed
Database destroyed
#
```

Example Java Program:

Change these variables at the top of the source file:

```
public class lgwznsqsamp1 {  
    private static final String DATA_SET_NAME = "YOUR.DATA.SET"; // EzNoSQL Database name  
    private static final String ALT_PATH_NAME = "YOUR.DATA.SET.TITLE.PATH"; // Alternate key PATH  
    private static final String ALT_AIX_NAME = "YOUR.DATA.SET.TITLE.AIX"; // Alternate key AIX  
    private static final boolean IS_PRIMARY_KEY_DB = true; // false for autogenerated key databases  
    private static final String AUTO_KEY = "\"znsq_id\""; // Reserved key for autogenerated databases  
    private static final String PRIMARY_KEY = "\"_id\""; // Custom key id for the primary index  
    private static final String ALTERNATE_KEY = "\"Title\""; // Custom key id for a secondary index  
    private static final String STORAGE_CLASS = "SXPXS01"; // SMS-managed Storage class
```

Getting Started...

To compile the sample java program `/samples/Igwznsqsamp1.java` :

```
cd /samples
javac -cp /usr/include/java_classes/igwznsq.jar Igwznsqsamp1.java
java -cp /usr/include/java_classes/igwznsq.jar:. Igwznsqsamp1
```

Example Java Program:

Create Database request successful!
Create secondary index request successful!
Add secondary index request successful!

<read/write messages>

Drop Secondary Index request successful!
Delete database request successful!
Sample Program Completed!

Explore the Documentation

EzNoSQL Documentation Reference

Maintained by VSAM RLS

Welcome to the EzNoSQL landing page.

- [EzNoSQL Documentation](#)
- [Javadoc API Reference](#)



Javadoc API Reference

OVERVIEW PACKAGE CLASS TREE INDEX HELP

eznosql API

Packages

Package	Description
<code>com.ibm.eznosql</code>	
<code>com.ibm.eznosql.exception</code>	
<code>com.ibm.eznosql.options</code>	
<code>com.ibm.eznosql.result</code>	



Javadoc API Reference

OVERVIEW **PACKAGE** CLASS TREE INDEX HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH:

Package com.ibm.eznosql

package com.ibm.eznosql

Related Packages

Package	Description
com.ibm.eznosql.exception	
com.ibm.eznosql.options	
com.ibm.eznosql.result	

Classes

Class	Description
Connection	The Connection class represents a connection between the user's task and the database.
Database	The Database class represents an EzNoSQL database identified by its data set name.
ReadUpdateResultSet	This class represents a direct read and exclusive document lock into the database, obtained as a result of a Connection.readOneForUpdate(String, String) .
ResultSet	This class represents a positioning into the database.

EzNoSQL Indexes

EzNoSQL Indexes

□ Indexes:

- All key names must be <256 characters
- Indexes can be read forward or backward
- Primary Indexes (unordered or ordered):
 - User supplied key values must be unique
 - If a key name is not supplied on the create, a primary key of “znsq_id” and unique key values will be auto-generated and pre-appended to the user document
 - No length restriction for the key value (unordered index)
 - Sequential inserts not allowed (unordered index)
 - Sequential reads will not maintain order (unordered index)
 - ORDEREDINDEX support provided in OA64954
- Secondary Indexes:
 - Optionally added by the application
 - Dynamically enabled/disabled however physically created only when the database is closed.
 - Key values maybe unique or nonunique, 4 gig duplicate keys allowed.
 - No length restriction, however, only the first 251 bytes are used as the value, otherwise treated as a non-unique key value
 - Sequential inserts allowed
 - Sequential reads will maintain order
- Multikeys:
 - Can be used as a primary or secondary key
 - Multiple key names are concatenated via a reverse solidus character: \
 - Allows key values to be used within imbedded documents or arrays (secondary indexes only)

EzNoSQL Index Examples

```
{  
  "Customer_id": "4084",  
  "Address": { "Street": "1 Main Street", "City": "New York", "State": "NY" }  
  "Accounts": ["Checking", "Savings"]  
}
```

- ❑ Unique primary key “**Customer_id**”:
 - Key Value: “4084”
 - “4084” is encrypted and randomized into an internal “derived key”
 - Document can be retrieved by with the argument “Customer_id” and “4084”
- ❑ Secondary key “**Address**”:
 - Key value: { "Street": "1 Main Street", "City": "New York", "State": "NY" }
 - Document can be retrieved with the argument “Address” and { "Street": "1 Main Street", "City": "New York", "State": "NY" }
- ❑ Secondary key “**Accounts**”:
 - Key values: “Checking” and “Savings”
 - Document can be retrieved by “Address” and either “Checking” or “Savings”
- ❑ Secondary multikey “**Address\Street**”:
 - Key Value: “1 Main Street”
 - Document can be retrieved with the argument “Address\Street” and “1 Main Street”

EzNoSQL Recoverable vs Non-Recoverable Databases

Recoverable vs Non-Recoverable Databases

- ❑ Determined during create with **the znsq_log_options** parameter:
 - **NONE** – represents a non-recoverable data base
 - **UNDO** or **ALL*** – represents a recoverable data base
- ❑ The recoverability of a database determines the duration of the locking and the transactional (atomic) capabilities:
 - Non-recoverable:
 - Exclusive document level locks (obtained for all writes) are **held only for the duration of the write request**
 - Shared locks (optionally obtained for reads) are held only for the duration of the read request
 - Recoverable:
 - **Exclusive document level locks** (obtained for all write requests) are **held for the duration of the transaction.**
 - Shared locks (optionally obtained for reads) have two options:
 - ✓ Consistent Reads (CR) are held for the duration of the read request
 - ✓ Consistent Read Extended (CRE) are held for the duration of the transaction
 - A transaction ends following a commit or backout.
 - Auto commits are issued by default after every write request. May be disabled with the `znsq_set_autocomit`

* CICSVR currently does not support EzNoSQL forward recoveries.

EzNoSLQ C APIs

C Level APIs **

Four Elements

DB Management

Create DB / Destroy DB
Add Index / List Indices

znsq_create
znsq_create_index
znsq_add_index
znsq_drop_index
znsq_destroy

Connection Management

Open / Alt Open
Close

znsq_open
znsq_close
znsq_report_stats

Document Management

Add Document
Delete Document

znsq_update_result
znsq_delete_result
znsq_delete
znsq_update
znsq_write
znsq_commit
znsq_abort
znsq_last_result
znsq_set_autocomit

Document Retrieval

Search
Next Result
Close Result Set

znsq_position
znsq_read
znsq_next_result
znsq_close_result

EzNoSLQ Java APIs

Three Java Classes

Database

**Define, Delete, Create Index,
Add Index, Drop Index**

```
fromDbParameters()  
createIndex()  
dropIndex()  
addIndex()  
delete()
```

Connection

**Open, Close, Read, Search,
Insert, Replace, Delete,
Commit, Abort, Report**

```
fromDataSetName()  
deleteOne()  
readOne()  
read()  
readOneForUpdate()  
replaceOne()  
readGeneric()  
insert()  
replace()  
commit()  
setAutoCommit()  
abort()  
close()  
getStatistics()
```

ResultSet

**Search Next, Replace,
Delete, Close Search**

```
fromSearchParameters()  
next()  
delete()  
update()  
close()
```

EzNoSLQ Python APIs

Three Python Classes

Database

**Define, Delete, Create Index,
Add Index, Drop Index**

```
znsq_create()  
znsq_create_index()  
znsq_add_index()  
znsq_drop_index()  
znsq_destroy()
```

EzNoSQLClient

**Open, Close, Read, Search,
Insert, Replace, Delete,
Commit, Abort, Report**

```
znsq_open()  
znsq_close()  
znsq_commit()  
znsq_set_autocommit()  
znsq_abort()  
znsq_write()  
znsq_delete()  
znsq_replace()  
znsq_read()  
znsq_position()  
znsq_report_stats()
```

ResultSet

**Search Next, Replace,
Delete, Close Search**

```
znsq_close_result()  
znsq_write_result()  
znsq_delete_result()  
znsq_replace_result()  
znsq_next_result()
```

Example: **znsq_last_result** API – Returned by C and Java APIs for Exceptions

```
znsq.last.result Report 2022.042 15:48:52  
API Name: znsq_create      RC: 0000000C  RS: 82030004
```

Diagnostic Data:

IDCAMS SYSTEM SERVICES

TIME: 15:48:51

03/03/22

```
DEFINE CLUSTER          -  
  (NAME(MY.JSON.DATA) -  
  CYLINDERS(1000000 1)  -  
  RECORDSIZE(500000 500000) -  
  SHAREOPTIONS(2 3)      -  
  STORCLAS(SXPXXS01)     -  
  DATACLAS(KSX00002)    -  
  LOG(NONE)              -  
  SPANNED                 -  
  DATABASE(JSON)         -  
  KEYNAMEU(_id)          -  
  VOLUME (XP0201)        -  
  FREESPACE (50 5))      -  
  DATA(NAME(MY.JSON.DATA.D) -  
  CONTROLINTERVALSIZE (32768)) -  
  INDEX(NAME(MY.JSON.DATA.I) -  
  CONTROLINTERVALSIZE (32768))  
IGD01007I DATACLAS ACSRTN VRS 02/19/92-1 ENTERED
```

Example: **znsq_last_result** API (cont.)

```
IGD01007I DATACLAS ACS EXECUTOR TEST ROUTINE IS ENTERED
IGD01007I DATACLAS DEFAULTING TO JCL OR TSO BATCH ALLOCATE
IGD01008I THE STORCLX ACSRTN VRS 07/06/99-1 ENTERED FOR SYSTEM X
IGD01008I STORCLAS ACS EXECUTOR TEST ROUTINE IS ENTERED
IGD01008I STORCLAS DEFAULTED TO JCL OR TSO BATCH ALLOCATE
IGD01009I MGMTCLAS ACSRTN VRS 07/28/88-1 ENTERED FOR SYSTEM X
IGD01009I  MGMTCLAS BEING DEFAULTED VIA JCL
IGD01010I STORGRP ACSRTN VRS 08/06/98-1 ENTERED FOR SYSPLEX
IGD01010I STORGRP BEING SET VIA STORCLAS UNLESS MULTIPLE GRPS SET
IGD01010I STORGRP BEING SET VIA STORCLAS UNLESS MULTIPLE GRPS SET
IGD17226I THERE IS AN INSUFFICIENT NUMBER OF VOLUMES IN THE ELIGIBLE
STORAGE GROUP(S) TO SATISFY THIS REQUEST FOR DATA SET
HL1.JSONWS01.BASE.KSDS1
IGD17290I THERE WERE 2 CANDIDATE STORAGE GROUPS OF WHICH THE FIRST 2
WERE ELIGIBLE FOR VOLUME SELECTION.
THE CANDIDATE STORAGE GROUPS WERE:SXP01 SXP02
IGD17279I 1 VOLUMES WERE REJECTED BECAUSE THEY WERE NOT ONLINE
IGD17279I 1 VOLUMES WERE REJECTED BECAUSE THE UCB WAS NOT AVAILABLE
IGD17279I 2 VOLUMES WERE REJECTED BECAUSE OF INSUFF TOTAL SPACE
IGD17219I UNABLE TO CONTINUE DEFINE OF DATA SET
HL1.JSONWS01.BASE.KSDS1
IDC3003I FUNCTION TERMINATED. CONDITION CODE IS 12

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 12
```

Example: **znsq_report_stats / getStatistics** API

```
{
  "name": "HL1.JSON.KSDS1",
  "version": 1,
  "documentFormat": "JSON",
  "keyname": "_id",
  "logOptions": "UNDO",
  "readIntegrity": "NRI",
  "readOnly": false,
  "writeForce": true,
  "autoCommit": false,
  "descendingKeys": false,
  "timeout": 5,
  "avgDocumentSize": 1000,
  "blockSize": 26624,
  "avgElapseTime": 150,
  "avgCPUTime": 8,
  "statistics": {
    "numberBlocksAllocated": 1234,
    "numberBlocksUsed": 124,
    "numberExtents": 1,
    "numberRecords": 2,
    "numberDeletes": 5,
    "numberInserts": 10,
    "numberUpdates": 4,
    "numberRetrieves": 13,
    "numberIndices": 1,

```

"indices":

```
[{
  "name": "HL1.JSON.AIX.KSDS1",
  "keyname": "Firstname",
  "pathname": "HL1.JSON.PATH1",
  "active": true,
  "unique": true,
  "descendingKeys": false,
  "blockSize": 26624,
  "statistics": {
    "numberBlocksAllocated": 1234,
    "numberBlocksUsed": 124,
    "numberExtents": 1,
    "numberRecords": 2,
    "numberDeletes": 5,
    "numberInserts": 10,
    "numberUpdates": 4,
    "numberRetrieves": 13,
    "numberCompoundKeys": 1,
    "compoundKeys": [
      {"descendingKeys": "T/F", "name": "altkeyname"},
      {...}
    ]
  }
}]
}
```


Performance Proof Points (C and Java)

Claim:

With the **Java API** to EzNoSQL for IBM z/OS, process up to **280,000 read** transactions per second or up to **32,200 insert** transactions per second to a NoSQL database on an IBM z16 z/OS LPAR with 8 CPs.

DISCLAIMER:

Performance results are based on IBM internal tests running a 64-bit Java application designed to only drive I/O requests executing read or insert requests to a NoSQL Database on z/OS using the Java API to EzNoSQL. Java 8 SR8 (JRE 1.8.0 z/OS s390x-64-Bit) was used and exploited compressed references. The measurement environment consisted of a single IBM z16 LPAR with 8 CPs running IBM z/OS 3.1. Eight copies of the application were run concurrently accessing the same database. The database contained 10 million documents, each of which was 121 bytes long, and used a 100 GB buffer pool. The CF cache size was set to 100 GB. The VSAM RLSFIXEDPOOLSIZE value was set to 50 GB. Read tests had a 100% buffer hit ratio. All writes are written through to disk. FICON Express32S channels running at 32G speeds and DS8950 storage with SSD were used. Your results may vary.

Claim:

With the **C language API** to EzNoSQL for IBM z/OS, process up to **300,000 read** transactions per second or up to **32,600 insert** transactions per second to a NoSQL database on an IBM z16 z/OS LPAR with 8 CPs.

DISCLAIMER:

Performance results are based on IBM internal tests running a 64-bit C application designed to only drive I/O requests executing read or insert requests to a NoSQL Database on z/OS using the C API to EzNoSQL. The measurement environment consisted of a single IBM z16 LPAR with 8 CPs running IBM z/OS 3.1. Eight copies of the application were run concurrently accessing the same database. The database contained 10 million documents, each of which was 121 bytes long, and used a 100 GB buffer pool. The CF cache size was set to 100 GB. The VSAM RLSFIXEDPOOLSIZE value was set to 50 GB. Read tests had a 100% buffer hit ratio. All writes are written through to disk. FICON Express32S channels running at 32G speeds and DS8950 storage with SSD were used. Your results may vary.

Performance Proof Points (Python)

Claim:

With the **Python language API** to EzNoSQL for IBM z/OS, process up to **260,000 read** transactions per second or up to **32,300 insert** transactions per second to an EzNoSQL database on an IBM z16 z/OS LPAR with 8 CPs.

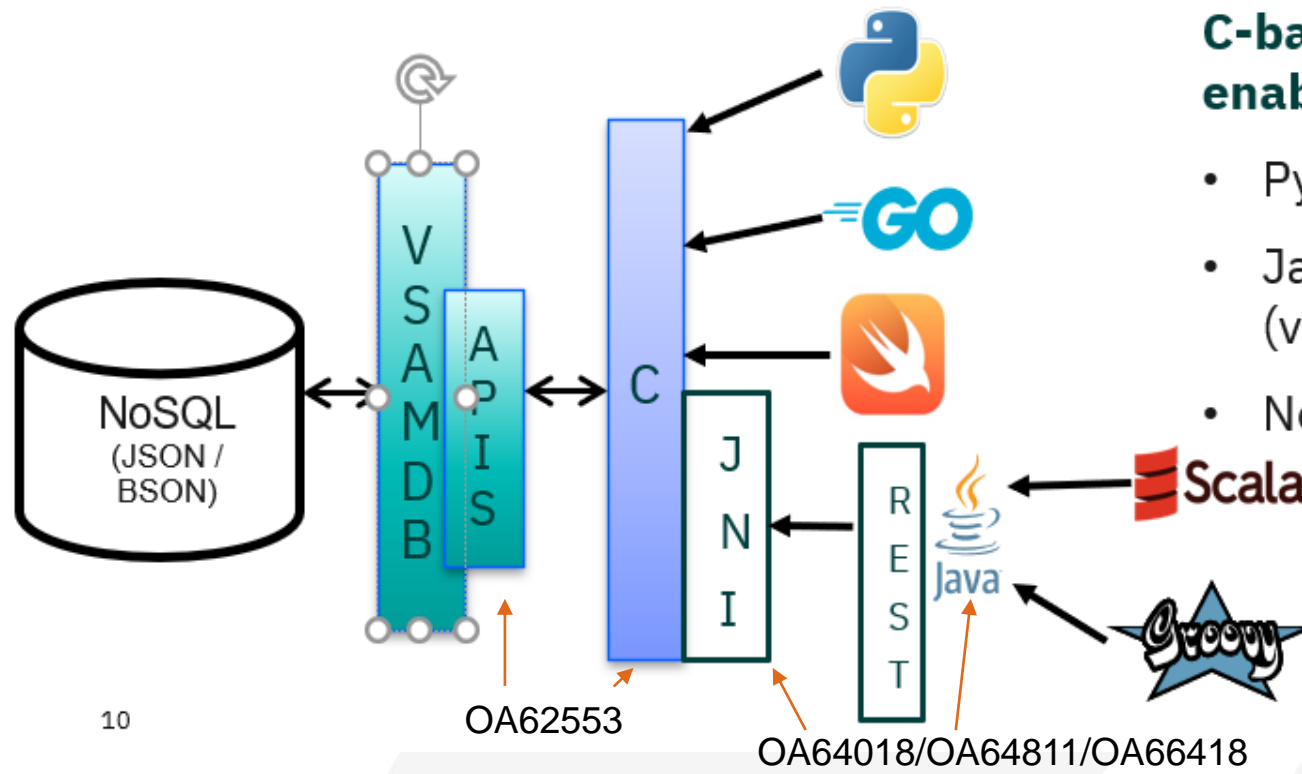
DISCLAIMER:

Performance results are based on IBM internal tests running a 64-bit Python application designed to only drive I/O requests executing read or insert requests to a EzNoSQL Database on z/OS using the Python API to EzNoSQL. The measurement environment consisted of a single IBM z16 LPAR with 8 CPs running IBM z/OS 3.1. Eight copies of the application were run concurrently accessing the same database. The database contained 10 million documents, each of which was 121 bytes long, and used a 100 GB buffer pool. The CF cache size was set to 100 GB. The VSAM RLSFIXEDPOOLSIZE value was set to 50 GB. Read tests had a 100% buffer hit ratio. All writes are written through to disk. FICON Express32S channels running at 32G speeds and DS8950 storage with SSD were used. Your results may vary.

EzNoSQL Futures

Futures

FUTURE Options**: Data stored in a platform independent format with full data sharing capabilities



C-based key-value interface to a NoSQL database enables higher level languages and interfaces

- Python, Go, Swift (interfaces to call directly)
- Java-based languages, like Groovy and Scala (via JNI)
- Network NoSQL interfaces

Your feedback is important!

Submit a session evaluation for each session you attend:

www.share.org/evaluation

