Technical Insights into File Access and Optimization in CICS

Chandra mouli Yalamanchili

chandu85@gmail.com

Abstract

IBM's z/OS and CICS platforms are widely recognized for their unmatched efficiency in high-volume transactional file I/O, enabled by decades of architectural refinement and deep integration with z/OS storage services. This paper explores how file access operations are managed within the IBM CICS Transaction Server on z/OS. Focusing on VSAM dataset handling, it examines foundational topics such as local and remote file access, File Control Tables (FCTs) role, and EXEC CICS macros. Advanced optimization techniques like Local Shared Resource (LSR) pools, data tables, and Record-Level Sharing (RLS) architecture are also discussed.

The paper highlights real-world best practices for error-handling techniques, performance tuning, and system design considerations. Practical code examples complement the discussion, making this paper valuable for developers and system programmers working in CICS environments.

Keywords: CICS; File access; VSAM; EXEC CICS; LSR pool; Data table; RLS; FCT; Remote file access; Coupling Facility; SMS VSAM; File Optimization; Mainframe applications

1. Introduction

CICS Transaction Server on z/OS provides a robust and scalable platform for online transaction processing, enabling enterprises to manage high volumes of mission-critical workloads. [1] Central to this capability is the structured handling of file input/output operations. By abstracting low-level dataset management and concurrency control, CICS allows developers to focus on business logic while the system ensures transactional consistency and performance. [1]

File operations in CICS—such as read, write, update, delete, and browse—are standardized through the EXEC CICS programming model, making applications interoperable across languages like COBOL, PL/I, Java, and assembler [1]. Key system components, like File Control Tables (FCTs), LSR pools, Data Tables, and Record-Level Sharing (RLS) with Coupling Facility integration, work together to optimize file access. [2]

This paper offers technical insights into these critical components, covering setup, operational behavior, performance tuning, and best practices. Special emphasis is placed on practical system programming considerations to ensure that applications built for CICS are efficient, scalable, and maintainable. [2]

2. CICS File Access Model Overview

This section explores the fundamental components required to access and manage files within the CICS environment. We will discuss how datasets are defined, how File Control Table (FCT) entries are used to configure file access properties, the types of access permissions that can be granted, the relevant system and program-level setup steps, and the general responsibilities split between system configuration and

application programming. This foundational understanding is essential for developing, configuring, and troubleshooting CICS applications that rely on efficient file handling.

Accessing a file in CICS requires several key components working together at both the system and program level:

- **Dataset Definition:** A VSAM file (e.g., KSDS, ESDS) must exist and be properly cataloged within z/OS. The dataset must be allocated to a storage volume accessible to the CICS region. [1]
- **DDNAME Association:** Each dataset is associated with a DDNAME in the CICS startup JCL or through dynamic allocation. The DDNAME acts as a symbolic link between the operating system and CICS. From the application's perspective, the DDNAME is abstracted behind the logical file name defined in the FCT. CICS handles translating the program's logical file name reference to the appropriate DDNAME and, thus, to the dataset. [1]
- **File Control Table (FCT) Entry:** CICS uses FCT entries to define the logical properties of a file, such as the logical file name (used within application programs), DDNAME (linking back to the dataset), record format, and service requests permitted (e.g., READ, WRITE, DELETE). The FCT acts as a bridge between programmatic file references and the actual underlying dataset. [2][3]

Sample FCT Entry:

```
DFHFCT TYPE=DATASET,DSNAME=CUSTOMER.KSDS,DDNAME=CUSTDD,
    SERVREQ=(ADD,UPDATE,READ,DELETE,BROWSE),
    FILE=CUSTFILE,RECORDFORMAT=FIXED
```

- **System Initialization Table (SIT) Settings:** While basic file access does not always require explicit SIT parameters, settings such as automatic file open at startup (AUTOOPEN=YES on the file resource definition) and global resource sharing parameters (e.g., resource timeout settings) can be influenced through SIT overrides. [3] Additionally, options for enabling Record-Level Sharing (RLS) or file recovery behaviors can be controlled via SIT settings.
- **Resource Definition (RDO/CSD entries):** In modern CICS systems, resource definitions for files may be managed dynamically through the Resource Definition Online (RDO) system using the CICS System Definition (CSD) file. This method provides flexibility over static FCT generation.

Access Types:

- **READ-ONLY:** Only retrieval operations are permitted; the application cannot update or delete the file.
- **WRITE-ONLY:** Typically used for output files where writing is allowed but reading is prohibited.
- **READ/WRITE:** Full access for reading, updating, or deleting records.

From a **programmer's perspective**, the application only needs to refer to the logical file name (specified by FILE= in the FCT entry) when issuing EXEC CICS file control commands. The physical dataset name (DSNAME) and operating system file management are abstracted and managed entirely by CICS, shielding the application from complexity.

The file must be opened automatically at the region startup if AUTOOPEN is specified or via an EXEC CICS OPEN FILE request issued during program execution. Conversely, files can be explicitly closed via EXEC CICS CLOSE FILE to release system resources when no longer needed. [1][2]

Step-by-Step Flow for File Access Setup and Use:

1. **Dataset Creation:** A VSAM file (e.g., KSDS) is created on z/OS and cataloged.
2. **Startup JCL Configuration:** A DDNAME is defined in the CICS startup JCL, pointing to the dataset.
3. **FCT Entry Definition:** An FCT entry is created, associating a logical file name (used in programs) with the DDNAME and specifying allowed operations.
4. **(Optional) RDO Definition:** In newer systems, files can be defined using CEDA or RDO instead of static FCT.
5. **SIT Configuration (Optional):** Auto-open options or RLS settings may be specified at region startup.
6. **CICS Startup:** At startup, CICS processes the JCL and FCT entries, establishing access paths to the datasets.
7. **Programmatic Access:** The application issues EXEC CICS commands referencing the logical file name for reads, writes, updates, or deletions.
8. **File Access Execution:** CICS transparently manages locking, data buffering, and recovery handling as the transaction executes.

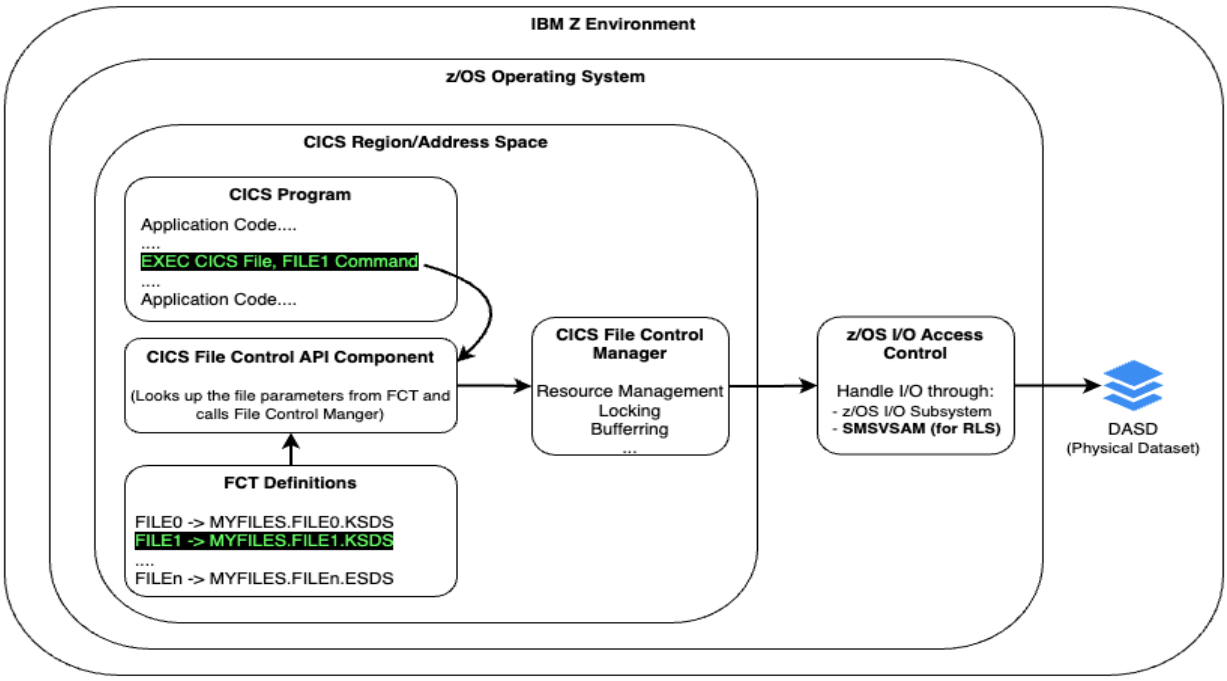

Figure 1: High-level CICS file access operations flow, from application code to physical dataset I/O management.

Understanding this foundational relationship between datasets, DDNAMEs, FCTs, SIT options, RDO management, and program-level references is critical to ensuring correct and efficient file access in CICS applications without duplication across other specialized sections such as RLS, Data Tables, or Queue Handling.

3. **Local vs. Remote Files**

This section explores how CICS handles local versus remote file access, the internal mechanisms that enable seamless file operations across regions, and the design considerations developers and system programmers must consider when building applications that interact with distributed file resources.

Local Files: Local files are datasets that are physically available within the same CICS region where the transaction is executing. All dataset I/O, locking, and buffering operations are handled internally within the region, ensuring minimal latency and high-speed access. The application program interacts with the local file through standard EXEC CICS file control commands, referencing the logical file name as defined in the File Control Table (FCT). [1]

Characteristics of local file access:

- Direct connection between the program and the dataset.
- The local CICS kernel fully manages locking and recovery.
- Optimized for performance with low communication overhead.
- Typically associated with files defined with local DDNAMEs and datasets cataloged on disks accessible to the region.

Remote Files: Remote files reside in a different CICS region but are made accessible to application programs as if they were local. CICS facilitates remote file access using inter-region communication mechanisms, allowing programs to remain unaware of the physical location of the data.

Remote file access is enabled through:

- **Distributed Program Link (DPL):** Enables a transaction in one CICS region to invoke a program in another region, which accesses the file locally.
- **Mirror Transactions**: CICS uses special system transactions, typically CSMI by default, to process file access requests on behalf of a remote region. The associated mirror program, DFHMIRS, is executed under the mirror transaction to perform the dataset operation in the remote region. [2][4]

Characteristics of remote file access:

- Involves network communication (IRC, IPIC, or TCP/IP protocols).
- Introduces network latency and additional failure points.
- It requires the configuration of remote system definitions (Connection, Session, and File Resource Definitions).
- Error handling becomes complicated due to potential communication failures, remote system downtime, or network partitioning.

- Security considerations such as transaction routing security and resource authorization checks come into play.

Impact on Application Design:

- **Performance Sensitivity:** Programs interacting with remote files must be designed with timeouts, retries, and fallback logic to handle transient network issues gracefully.
- **Atomicity and Integrity:** Distributed transactions must account for potential partial failures; techniques like two-phase commit (where applicable) may be needed for coordinated updates.
- **Resource Definitions:** System programmers must ensure that resource definitions for remote files match between the requesting and owning CICS regions to avoid mismatches.

While remote file access adds flexibility and scalability to CICS architectures, it also requires careful planning and robust error-handling practices to ensure application reliability and performance. [2]

Understanding the distinction between local and remote file handling is critical when designing CICS applications, particularly in systems that span multiple logical regions or data centers.

4. Record-Level Sharing (RLS) and Coupling Facility

This section explores the role of Record-Level Sharing (RLS) in enabling concurrent access to VSAM datasets across multiple CICS regions, the critical infrastructure components involved, such as the Coupling Facility (CF) and SMSVSAM, and the key design considerations programmers and system engineers must understand when building RLS-based systems.

Record-Level Sharing (RLS): Record-Level Sharing (RLS) was introduced to address the need for high-volume, highly available transactional access to VSAM datasets across multiple CICS regions without locking entire datasets. Instead of dataset-level locking, RLS introduces fine-grained record-level locks, enabling true concurrent data access among distributed CICS systems. [1]

Key Components of RLS Architecture:

- **SMS-Managed VSAM Datasets:** Only datasets managed by IBM's System Managed Storage (SMS) environment are eligible for RLS access. Datasets must be defined with the LOG and RECOVERY attributes if recovery and logging support are required. [1]
- **RLS-Enabled Data Management (SMSVSAM):** The SMSVSAM address space operates system-wide and provides caching, locking, and buffer management services for all RLS-accessing regions.
- **Coupling Facility (CF):** The Coupling Facility acts as a central coordination point, managing shared locks, cache consistency, and system-wide communication for RLS datasets. CF structures maintain lock information and enable high-speed, synchronized access. [2]
- **RLS Buffer Manager:** Each CICS region configured for RLS maintains its own buffer pool but coordinates with the CF and SMSVSAM for consistency and lock management.

Benefits of Using RLS:

- **Horizontal Scalability:** Allows multiple CICS regions to access and update the same dataset simultaneously without traditional dataset enqueues.

- **Reduced Contention:** Record-level locking minimizes unnecessary serialization, improving throughput for high-volume transaction systems.
- **Improved Availability:** With proper configuration, system failures in one region do not necessarily impact other regions accessing the same dataset.
- **Centralized Data Integrity:** By coordinating locks through the CF and SMSVSAM, RLS ensures data consistency even in complex distributed environments. [2]

Limitations and Design Considerations:

- **Dataset Type Restrictions:** RLS is supported for KSDS and ESDS datasets but not for RRDS datasets.
- **Application Awareness:** Applications must be designed to handle partial failures and avoid assumptions about exclusive control over records.
- **Configuration Complexity:** RLS requires carefully defining SMS classes, VSAM dataset attributes, and CF structures. A misconfiguration can lead to performance degradation or system contention.
- **Failure Handling:** Although CF and SMSVSAM provide high availability, system designers must plan for failover scenarios, including Coupling Facility structure rebuilds and SMSVSAM recovery processes.

The following diagram illustrates a simple Sysplex setup where multiple CICS regions across different MVS systems access shared VSAM datasets using SMSVSAM address spaces and coordinate locking and consistency via the Coupling Facility.

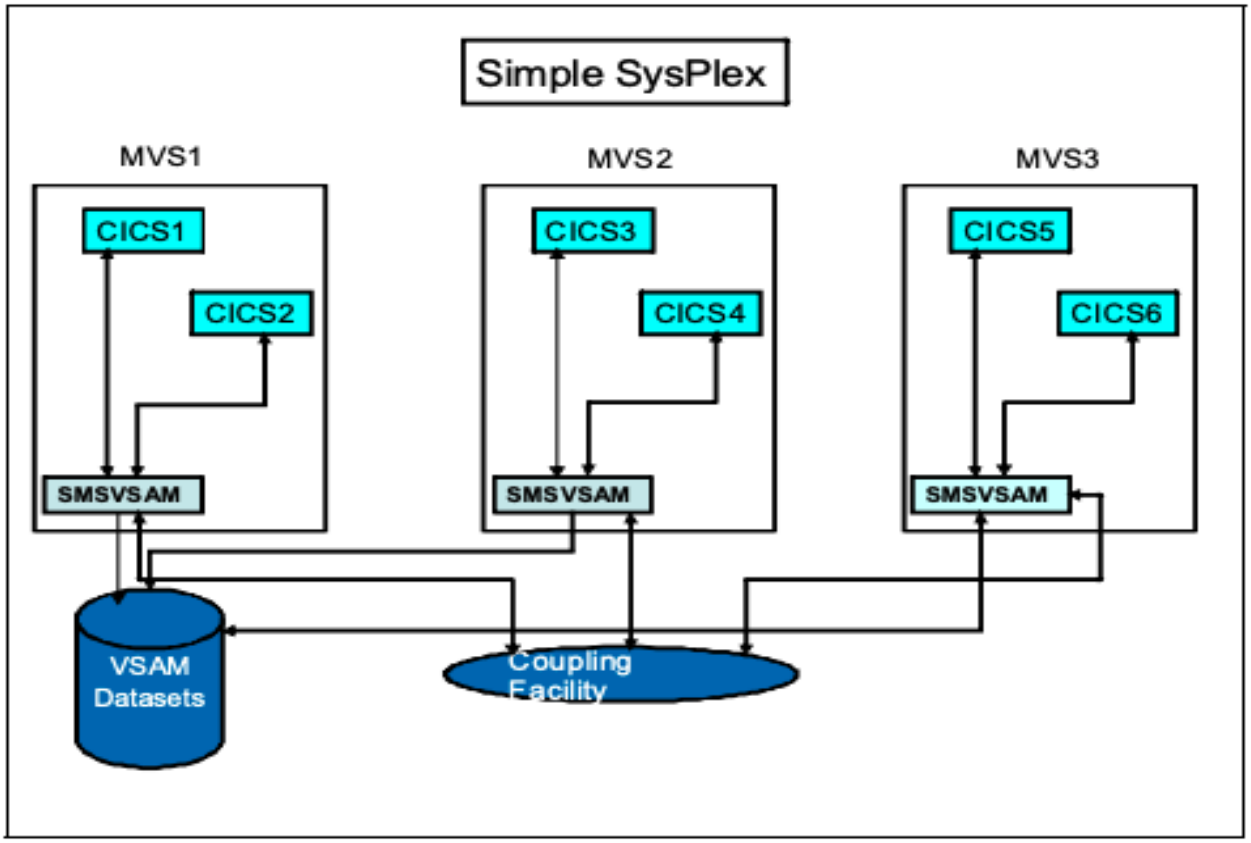

Figure 2: Illustrating the connection between CICS, VSAM/SMSVSAM, and the coupling facility. [2]

Understanding how RLS and the Coupling Facility work together is critical for designing scalable, reliable multi-region CICS applications that demand high concurrency and strong data integrity.

5. **Using EXEC CICS Macros for File Operations**

This section explores how CICS standardizes file interactions using EXEC CICS macros. These macros provide a consistent, language-independent interface for programs to perform file operations without managing low-level dataset handling. Understanding the available commands, key parameters, and typical usage patterns is critical for building robust CICS applications.

CICS provides the following primary file control operations:

- **READ:** Retrieve a specific record from a dataset.
    - **Key Parameters:** FILE, INTO, LENGTH, RIDFLD, RESP
    - Reads a record based on a supplied key (for KSDS) or directly (for ESDS).

    Sample READ Operation:

```
      EXEC CICS READ FILE('ORDRFILE')
          INTO(ORDERREC)
LENGTH(100)
          RIDFLD(ORDERKEY)
          RESP(RESPCD)
      END-EXEC
```

- **WRITE:** Create a new record in a dataset.
    - **Key Parameters:** FILE, FROM, LENGTH, RIDFLD (optional for some datasets), RESP
    - Adds a new record, ensuring key uniqueness for KSDS if required [1].

    Sample WRITE Operation:

```
      EXEC CICS WRITE FILE('ORDRFILE')
          FROM(ORDERREC)
LENGTH(100)
          RESP(RESPCD)
      END-EXEC
```

- **REWRITE:** Update the contents of an existing record.
    - **Key Parameters:** FILE, FROM, LENGTH, RESP
    - Requires that a successful READ (with UPDATE option) or a successful READNEXT/READPREV has occurred before issuing the REWRITE.

    Sample REWRITE Operation:

```
       EXEC CICS REWRITE FILE('ORDRFILE')
            FROM(ORDERREC)
LENGTH(100)
            RESP(RESPCD)
       END-EXEC
```

- **DELETE:** Remove a record from a dataset.
  - **Key Parameters:** FILE, RIDFLD, RESP
  - Deletes the specified record. Like REWRITE, it usually follows a READ UPDATE or BROWSE operation.

  Sample DELETE Operation:

```
       EXEC CICS DELETE FILE('ORDRFILE')
            RIDFLD(ORDERKEY)
            RESP(RESPCD)
       END-EXEC
```

- **BROWSE:** Sequentially navigate through records within a dataset.
  - **Key Parameters:** FILE, RIDFLD (start point optional), RESP
  - Browsing is a two-step process: initiating the browse and then reading records sequentially.

  Typical Browse Pattern (HLASM Example):

```
       EXEC CICS STARTBR FILE('ORDRFILE')
            RIDFLD(STARTKEY)
            RESP(RESPCD)
       END-EXEC

READNEXT-LOOP.
       EXEC CICS READNEXT FILE('ORDRFILE')
            INTO(ORDERREC)
            RESP(RESPCD)
       END-EXEC

       CLC   RESPCD, DFHRESP(NORMAL)
       BE    PROCESS-RECORD
       B     CLOSE-BROWSE

PROCESS-RECORD.
       * Processing logic here
       B     READNEXT-LOOP
```

```
CLOSE-BROWSE.
      EXEC CICS ENDBR FILE('ORDRFILE')
         RESP(RESPCD)
      END-EXEC
```

In this example:

- STARTBR positions the browse at a specified key or at the beginning.
- READNEXT retrieves records sequentially.
- CLC and BE check if the response is NORMAL; if so, processing continues; otherwise, the browse ends.
- ENDBR explicitly closes the browse, releasing internal CICS resources.

Important Considerations:

- Always check the RESP (response) field after each file operation to appropriately handle errors or end-of-browse conditions.
- Browses must be explicitly ended with ENDBR to release internal CICS resources.
- When using REWRITE or DELETE, the previous READ must have been issued with UPDATE intent, or an error will occur.

These macros abstract away complex dataset handling and enable CICS to automatically manage buffering, locking, logging, and recovery, ensuring data integrity and transaction consistency. [1]

6. **LSR Pools and Data Table Optimization**

This section explores two powerful optimization techniques available within CICS for improving file access performance: Local Shared Resource (LSR) Pools and Data Tables. Both mechanisms reduce physical I/O overhead, minimize locking contention, and boost transaction throughput. Understanding when and how to use each technique is essential for designing scalable and efficient CICS applications.

Local Shared Resource (LSR) Pools: LSR pools are designed to optimize environments with high transaction volumes and frequent file access. Rather than each file maintaining its own private buffers, multiple files can share a common set of data and index buffers coordinated through the LSR pool. This sharing reduces the overhead of buffer allocation and open/close processing. [1][4]

How LSR Pools Work:

- A set of VSAM files are grouped into an LSR pool.
- Files within the pool share data buffers (BUFND) and index buffers (BUFNI).
- CICS manages buffer reuse dynamically to optimize hit ratios and minimize physical I/O. [1]
- LSR improves concurrency by reducing contention for physical access to records already cached in the pool. [1]

Use Cases for LSR:

- High-frequency reference files (e.g., customer master files accessed during every transaction).
- Large datasets where browsing and lookup operations dominate workload.
- Environments where minimizing dataset open/close processing is critical to transaction response times. [1]

LSR Pool Tuning Considerations:

- Properly sizing BUFND (data buffers) and BUFNI (index buffers) is critical. [1]
- Files sharing similar access patterns should be grouped in the same LSR pool. [1]
- Monitoring buffer hit ratios and tuning buffer counts can significantly improve performance. [1]

Data Tables: Data tables are an alternative optimization where the contents of a VSAM dataset are loaded into memory during file opening, allowing CICS to satisfy reads from memory instead of disk. [2]

Types of Data Tables:

- **Unmaintained Data Table (Also known as User Data tables):**
  - The data is loaded once into memory.
  - Subsequent updates to the VSAM dataset are not reflected until the file is closed and reopened.
  - Best suited for static reference data. [2]
- **Maintained Data Table (Also known as CICS Data tables):**
  - Updates made during CICS execution are reflected both in the in-memory table and back to the VSAM dataset. [2]
  - Suitable for datasets requiring occasional updates.
- **Coupling Facility (CF) Data Table:**
  - Data tables can also be stored in the Coupling Facility for sharing across CICS regions.
  - CF-based data tables offer high availability and cross-region consistency but are limited to 16-byte maximum key lengths. [2][3]

Use Cases for Data Tables:

- Currency code mappings.
- Product catalog lookups.
- Geographic lookup tables (e.g., ZIP code to city mapping).
- Configuration tables that rarely change but are read frequently [2].

By strategically using LSR pools and data tables, CICS developers and system programmers can significantly reduce file access overhead and deliver faster transaction processing while balancing dataset volatility and update frequency considerations. [1][2]

7. Error Handling and Debugging Techniques

This section explores essential error handling and debugging strategies when working with CICS file operations. Proper handling of return codes, consistent logging practices, and the use of built-in diagnostic tools are critical to ensure reliable transaction processing and simplifying problem resolution. [1]

Key Practices for Error Handling:

- **Always Check RESP and RESP2 Fields:** Every EXEC CICS file operation should immediately check the RESP field for the operation's status. The RESP2 field can provide additional detail when an error occurs. [1]

Example for checking RESP after a READ:

```
      EXEC CICS READ FILE('ORDRFILE')
           INTO(ORDERREC)
LENGTH(100)
           RIDFLD(ORDERKEY)
           RESP(RESPCD)
      END-EXEC

      CLC   RESPCD,DFHRESP(NORMAL)
      BNE   HANDLE-ERROR
```

If the operation did not complete normally, control is transferred to an error handling routine.

- **Consider EIBRESP for Implicit Errors:** Apart from explicit RESP codes, the EIBRESP field in the CICS Execution Interface Block (EIB) can also capture implicit errors that occur outside the scope of a specific command. [1]
- **Standardize Application-Level Error Responses:** Create reusable macros or routines to standardize how file operation errors are handled across all programs. For instance, deciding when to retry, rollback, or abend based on the RESP codes. [1]

Using CICS Diagnostic Tools:

- **CEDF (CICS Execution Diagnostic Facility):** CEDF provides interactive debugging for CICS transactions. When active, CEDF pauses execution at every EXEC CICS command, allowing developers to inspect the current RESP code, EIB block, and program variables before proceeding. [1]

To activate CEDF, issue:

```
CEDF START
```

Alternatively, start a transaction with CEDF enabled to trap and diagnose issues automatically.

- **CICS Message Logs and Traces:** In production systems, enable system logs and user trace points to capture error events, response codes, and unexpected conditions without relying solely on interactive debugging. [1]

General Debugging Tips for File Operations:

- Validate that the file is OPEN before performing operations. [1]
- Always use STARTBR properly before issuing READNEXT or READPREV during browse operations. [1]
- Ensure that UPDATE intent is specified during a READ if a subsequent REWRITE or DELETE is planned. [1]
- In cases of frequent NOTFND or DUPREC errors, review the key and RIDFLD values being passed. [1]
- Monitor performance for symptoms of resource contention (e.g., buffer shortages in LSR pools). [1]

By rigorously implementing these error handling and debugging strategies, CICS applications can achieve higher reliability, faster issue resolution, and improved user satisfaction even in complex file access scenarios. [1]

8. Best Practices and Considerations

This section will summarize practical best practices for designing efficient, scalable, and reliable CICS applications that perform file operations. These practices are grounded in both IBM recommendations and real-world operational experience. [1]

General File Access Best Practices:

- **Define Appropriate Access Permissions (SERVREQ):** Carefully specify the allowed operations in the File Control Table (FCT) entry. If a file only needs to be read, configure it as READ-only to avoid unintentional updates or deletes. [1]
- **Use Auto-Open Where Appropriate:** For frequently accessed files, enabling AUTOOPEN ensures they are ready immediately after CICS region initialization, reducing transaction delays. [1]
- **Leverage LSR Pools Strategically:** Group files with high read-access rates and similar access patterns into LSR pools to optimize buffer sharing and minimize I/O bottlenecks. [1]
- **Consider Data Tables for Static Reference Files:** For heavily read but rarely updated datasets, use data tables to eliminate unnecessary disk I/O and significantly improve transaction throughput. [1]
- **Manage VSAM File Sizes and Extents Carefully:** Ensure that VSAM datasets are sized appropriately with correct Control Interval (CI) and Control Area (CA) sizing to minimize the likelihood of datasets taking excessive extents during growth. Excessive extents can lead to access delays and performance degradation when CICS transactions access files. [1] When datasets involve frequent deletions, consider enabling the CA Reclaim option to allow VSAM to reuse free control areas, improving space management and reducing fragmentation over time. [1]

Programming Practices for Reliable File Operations:

- **Check RESP Codes Immediately:** Never assume a file operation has succeeded; always verify the RESP (and RESP2) values after each EXEC CICS command. [1]
- **Use SYNCPOINT for Consistency:** For applications that perform multiple updates, issuing an explicit SYNCPOINT ensures that all file changes are committed atomically, preserving transactional integrity. [1]
- **Efficient Browse Operations:** Always issue ENDBR after completing a browse to release system resources promptly. [1]
- **Control Program Size and Complexity:** Modularize file-handling logic, especially when working with large numbers of files or complex transaction flows, to ease debugging and maintenance. [1]

Performance and Reliability Considerations:

- **Monitor Buffer Usage:** Use CICS performance classes and monitoring tools to assess buffer hit ratios and tune LSR pools accordingly. [1]
- **Anticipate Remote File Access Latency:** If remote files are part of the design, implement timeouts and fallback mechanisms to handle network delays gracefully. [1]
- **Plan for RLS and CF Usage:** In multi-region environments, ensure that datasets are SMS-managed and properly configured for Record-Level Sharing (RLS) to avoid serialization bottlenecks. [1]
- **Manage RLS Locking Carefully:** RLS record locks are shared across multiple regions, so careful design is needed to avoid prolonged lock holding. If one region acquires a lock and the corresponding task encounters long delays, other regions attempting to access the same record may experience significant wait times. Applications should be coded to minimize record lock durations and use timeout handling where appropriate to avoid transaction build-ups. [1]

Operational Best Practices:

- **Implement Structured Logging:** Maintain standardized logging of file errors, responses, and recovery actions for auditing and troubleshooting. [1]
- **Use CEDF and Tracing Tools Judiciously:** While CEDF is powerful, use it carefully in production systems and rely more heavily on tracing and message logging for continuous monitoring. [1]
- **Document File Resource Definitions Clearly:** Ensure system documentation captures DDNAME associations, FCT definitions, and expected access patterns for easy support and onboarding of new team members. [1]

By following these best practices, CICS developers and system programmers can design applications that perform efficiently and handle operational complexity gracefully, leading to greater system stability, maintainability, and user satisfaction. [1]

Conclusion

Handling file operations effectively is fundamental to achieving the high performance and reliability that CICS-based applications demand. Techniques such as careful file sizing, strategic use of LSR pools, data

table optimizations, and Record-Level Sharing (RLS) allow developers to design solutions that minimize overhead and maximize throughput. [1][2] Proper error-handling practices, structured logging, and proactive resource planning enhance application robustness. [1]

IBM’s platform capabilities, particularly its deep integration with z/OS and mature support for VSAM and SMS-managed datasets, make it exceptionally well-suited for high-volume transactional environments. Features such as RLS and coupling facility-based access coordination demonstrate IBM's continued investment in resilient, high-performance file I/O management. [1][2]

Organizations may consider exploring lightweight workload tuning strategies, the gradual introduction of cloud-enabled resilience, and the use of AI-assisted monitoring tools to complement existing file optimization techniques—all while preserving the trusted transactional semantics of CICS. [2]

References

[1] IBM Corporation, "CICS Transaction Server for z/OS V5.6 Documentation," IBM Documentation, 2020. [Online]. Available at: https://www.ibm.com/docs/en/cics-ts/5.6.

[2] IBM Redbooks, "CICS Transaction Server from Start to Finish," SG24-7952-00, IBM Corporation, 2011. [Online]. Available at https://www.redbooks.ibm.com/abstracts/sg247952.html.

[3] C. Yalamanchili, "CICS Transaction Processing on z/OS: Core Concepts and Workflow," International Journal Research of Leading Publication, Volume 2, Issue 3, March 2021. DOI 10.5281/zenodo.15154786.

[4] IBM Corporation, "Introduction to the New Mainframe: z/OS Basics", SG24-6366-01, IBM Redbooks, 2011. [Online]. Available at: https://www.redbooks.ibm.com/abstracts/sg246366.html.