z/TPFDF V1.1

TPF Users Group Fall 2008

Application Development using SDO Access to z/TPFDF – Advanced Features

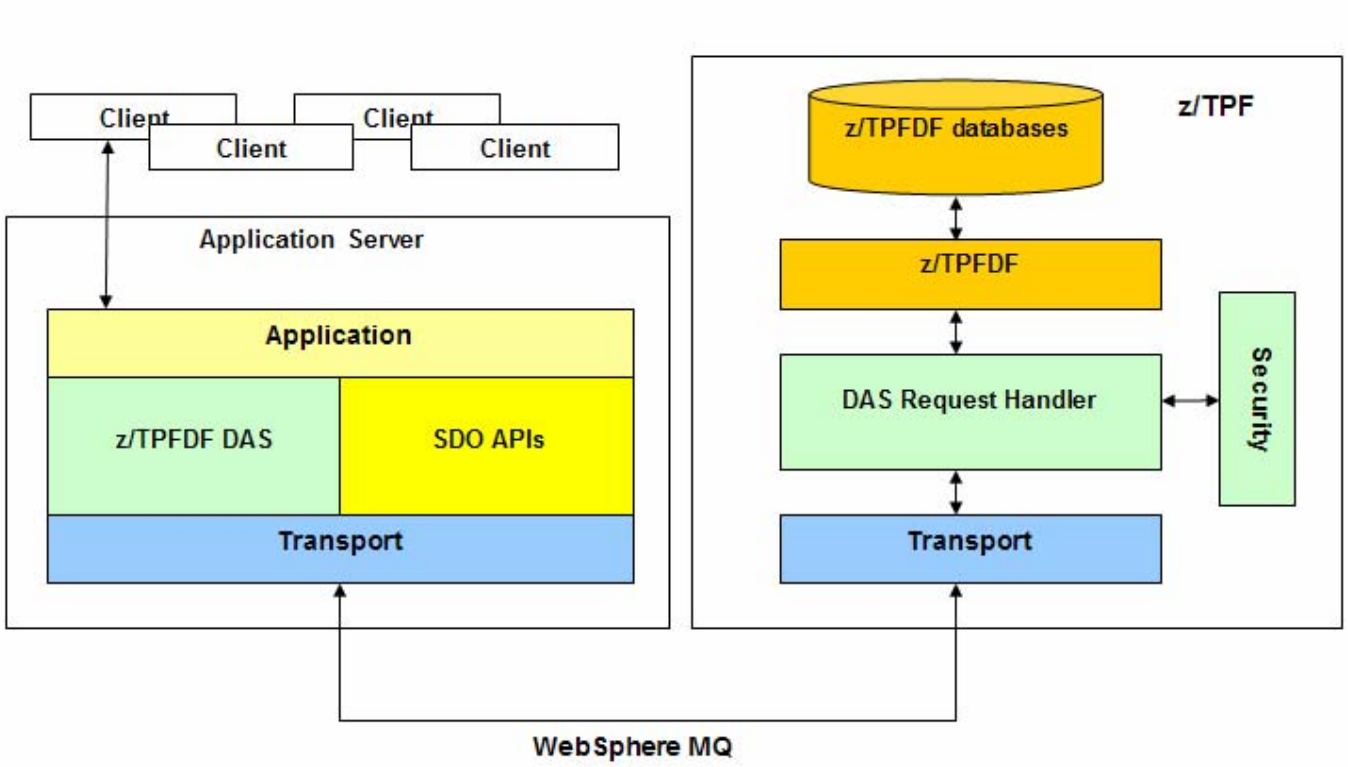
Name: Glenn Katzen
Venue: Applications Development Subcommittee

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- **SDO Access to z/TPFDF Overview**
- **Sample Scenario and Database**
- **Creating and Deleting Subfiles**
- **Indexing**
- **De-indexing**
- **Fullfile Data Requests**

SDO Access to z/TPFDF Overview



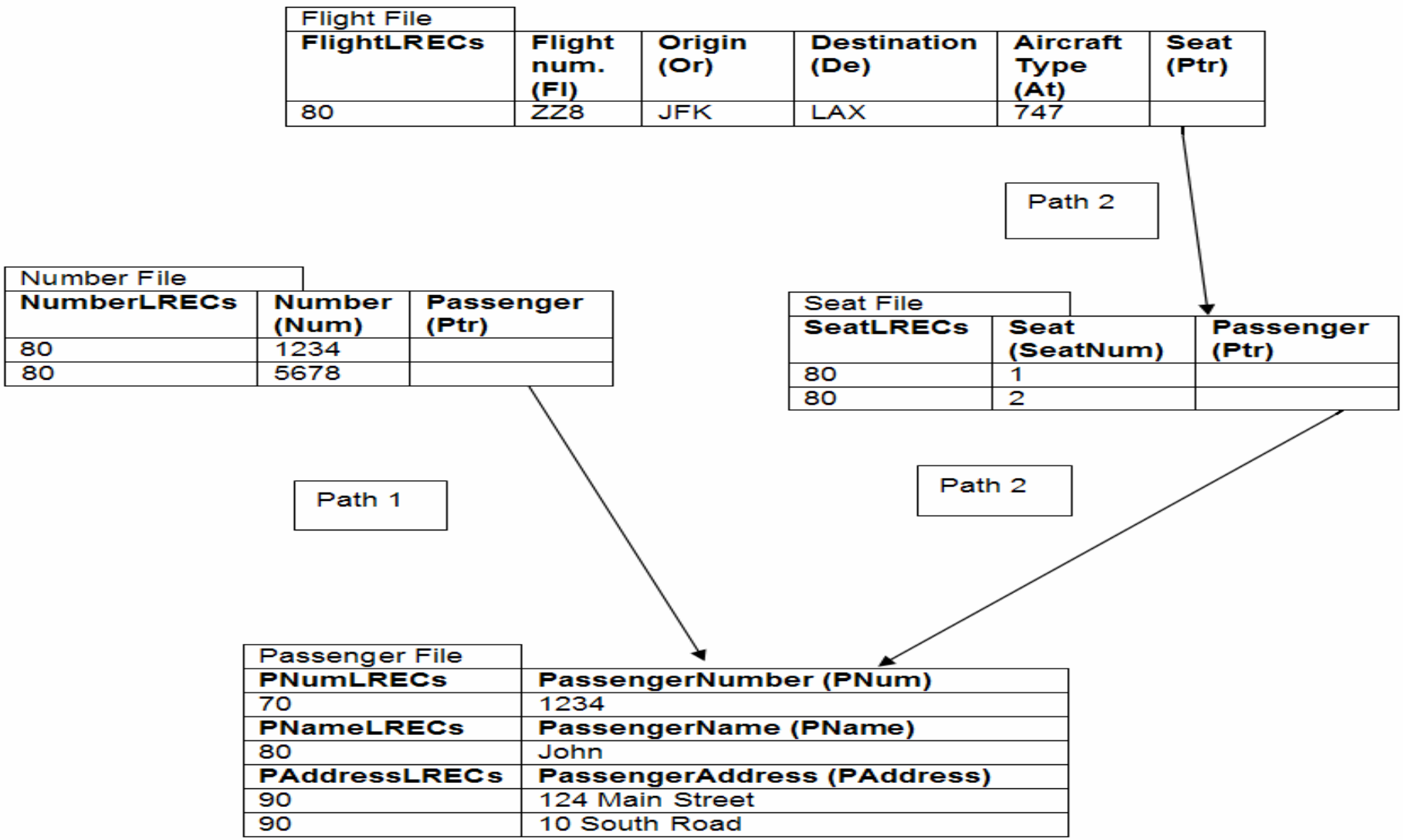
API Overview

- **z/TPFDF DAS Methods**
  - readData
  - applyChanges
  - copyDataObject
- **SDO DataObject Methods**
  - getXXX
  - setXXX
  - unset
  - createDataObject
  - delete

Scenario

- **A database containing flight information exists on z/TPFDF and metadata has been created.**
- **We will develop an application that can add and remove passenger subfiles, index and de-index passenger subfiles, and perform efficient queries for passenger data.**

Database



Instantiating a DAS

The first step in obtaining a DataGraph is to instantiate a z/TPFDF DAS.

```
DatabaseParameter databaseParam =
  new DatabaseParameter(  “Flight” );

AuthorizationModule authModule =
  new AuthorizationModule( “User1”, “Pass”, “Unencrypted”, null );

ZTPFDFDAS das =
  new ZTPFDFDAS( “ConfigFile.xml”, databaseParam, authModule );
```

Obtaining a DataGraph

The readData method of the z/TPFDF DAS is used to retrieve a DataGraph.

```
PathParameter pathParam = new PathParameter( “1”, 1234 );

SearchParameter searchParam = new SearchParameter();

PropertiesParameter propertiesParam = new PropertiesParameter( "*" );

DataGraph dataGraph = das.readData( “PassengerFile”, pathParam,
                                    searchParam, propertiesParam );
```

Creating Subfiles

A new subfile is created by calling the createDataObject method on the root DataObject of the DataGraph.

The code below creates a new passenger subfile.

```
DataObject root = dataGraph.getRootObject();

DataObject passengerSubfile =
  root.createDataObject( “PassengerFile” );
```

Deleting Subfiles

A subfile is deleted by calling the delete method on the subfile DataObject.

The code below deletes a passenger subfile.

```
passengerSubfile.delete();
```

Indexing Subfiles

A new subfile must be indexed by creating an index LREC and setting a reference to the subfile DataObject.

The code below indexes a passenger subfile in the Number File along Path 1.

```
DataObject numberSubfile =
  root.getDataObject( “NumberFile.0” );

DataObject numberIndexLREC =
  numberSubfile.createDataObject( “NumberLRECs” );

numberIndexLREC.setInt( “Num”, 1234 );
numberIndexLREC.setDataObject( “PassengerFile”,
                                    passengerSubfile );
```

Indexing Subfiles Continued

Subfiles may be indexed along additional paths using the copyDataObject method of the z/TPFDF DAS. First, a DataGraph for the desired path must be retrieved.

The code below obtains a DataGraph along Path 2 of the database.

```
PathParameter pathParam2 = new PathParameter( “2”, 1 );

SearchParameter searchParam2 = new SearchParameter();

PropertiesParameter propertiesParam2 = new PropertiesParameter( "*" );

DataGraph dataGraph2 = das.readData( “PassengerFile”, pathParam2,
                                     searchParam2, propertiesParam2 );
```

Indexing Subfiles Continued

The copyDataObject method is used to copy the subfile to be indexed over to the new DataGraph.

The code below places a passenger on a flight by copying the passenger subfile DataObject over to the DataGraph along Path 2 and indexing the subfile in the Seat File.

```
DataObject passengerSubfileCopy =
  das.copyDataObject( passengerSubfile, dataGraph2 );

DataObject seatIndexLREC =
  root.getDataObject( “FlightFile.0/FlightLRECs.0”
                      + “/SeatFile/SeatLRECs.0” );

seatIndexLREC.setDataObject( “PassengerFile”,
                             passengerSubfileCopy );
```

De-indexing Subfiles

A subfile is de-indexed by unsetting the reference to the subfile DataObject within the index LREC DataObject. A ChangeMadeDataGraphNotValidException will be thrown when applyChanges is called.

The code below removes a passenger from a flight by de-indexing the passenger subfile from the Seat File.

```
seatIndexLREC.unset( “PassengerFile” );

try {

  das.applyChanges( dataGraph2 );

} catch ( ChangeMadeDataGraphNotValidException e ) {}
```

De-indexing Subfiles Continued

If the database supports auto de-indexing, it can be activated by deleting all LREC DataObjects within a subfile DataObject and calling applyChanges. As with manual de-indexing, a ChangeMadeDataGraphNotValidException will be thrown.

The code below removes a passenger from all flights and other index files using auto de-indexing.

```
passengerSubfile.getList( “PNumLRECs” ).clear();
passengerSubfile.getList( “PNameLRECs” ).clear();
passengerSubfile.getList( “PAddressLRECs” ).clear();

try {

  das.applyChanges( dataGraph );

} catch ( ChangeMadeDataGraphNotValidException e ) {}
```

Fullfile Requests

If no top-level filter is specified on a readData request, all top-level subfiles with LRECs matching the provided search criteria will be returned in the DataGraph.

The code below performs an efficient query to retrieve all passengers with passenger numbers greater than 1234.

```
PathParameter pathParam = new PathParameter( “1” );

SearchParameter searchParam = new SearchParameter();
searchParam.addCondition( “NumberFile”, “Num”,
                                    SearchOperator.GREATER, 1234 );

PropertiesParameter propertiesParam = new PropertiesParameter( "*" );

DataGraph dataGraph = das.readData( “PassengerFile”, pathParam,
                                                searchParam, propertiesParam );
```

Fullfile Considerations

- **Indexing and de-indexing are not allowed with a DataGraph that has been retrieved through a fullfile request**
- **Search criteria is often necessary to avoid exceeding the 4 MB size limitation**
- **Fullfile may also be specified in the metadata**

Follow-up Information

- **PK60030**
  - z/TPFDF APAR
- **PJ32720**
  - Co-requisite z/TPF APAR
- **Java z/TPFDF DAS code is available for download on the TPF web site:**
  http://www-01.ibm.com/software/htp/tpf/download/ztpfsdo.htm
- **SDO libraries are freely available:**
  - http://www.eclipse.org/modeling/emf
  - http://incubator.apache.org/tuscany/

Trademarks

- IBM is a trademark of International Business Machines Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Notes
- Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
- All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
- This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
- All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
- Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
- Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
- This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.