Generating and Consuming IFS Files and Web-Services with embedded SQL

Birgitta Hauser
Diplom-Betriebswirt (BA)
Software and Database Architect
Hauser@SSS-Software.de

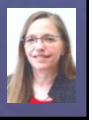
Agenda

Generating and Consuming IFS Files

- Writing Data to the IFS
- Reading IFS Files
- Consuming Data from a *.csv File

Generating and Consuming Web-Services

Work with
Large Objects and XML Documents



Large Object and XML Data Types

Large Objects = LOB

- LOB data types allow **large amounts of character data** (SBCS/DBCS) or (large) **documents in any format** to be stored within the database
  - **CLOB:** **C**haracter **L**arge **O**bject (SBCS → 1 Character = 1 Byte)
    Automatic **Code Page Conversion** supported
  - **DBCLOB:** **D**ouble **B**yte **C**haracter **L**arge **O**bject (DBCS → 1 Char = 2 Byte)
    Automatic **Code Page Conversion** supported
  - **BLOB:** **B**inary **L**arge **O**bject
    Without any conversion
- Maximum Size: **2 GB** (= 2 147 483 647 Byte)

XML Data Type

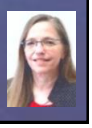
- **Special Type of a LOB** to hold **XML documents**
- Only **one data type** → SBCS/DBCS depend on the specified CCSID

Work with LOBs / XML Documents

3 Options to access LOB Values / XML Documents:

- **LOB / XML Host Variables**
  - LOB value / XML document is **copied** into the **host variable**
  - **Maximum length** on the variable depends on the **RPG restrictions**
- **LOB/XML Locators**
  - LOB/XML locator **points to** the LOB value / XML document
  - Data **remains on the sever** and is **not copied** into host variables
  - LOB/XML locators can be used within **SQL commands / functions** like **any characater variable**
  - **Commitment Control required**
- **LOB/XML Files**
  - The LOB value / XML document is **moved to** or **located in** an **IFS file**
  - A LOB/XML **locator** can be used to get **access** to the data
  - **IFS files** can be **directly accessed** with **embedded SQL**

LOB File Reference Variables



LOB/XML File Reference Variables

Are used to transfer data from and to the IFS

- Allow to **transfer IFS files** into tables (LOB/XML columns) and vice versa
- Allow to **access** and **read data** from **IFS files** with embedded SQL
- Allow to **create** and **manipulate IFS files** with embedded SQL

LOB/XML File Variables represent the IFS File

- **Point** to the **IFS files** (like LOB locators point to the data)

Defining LOB/XML File Reference Variables

- **Keyword SQLTYPE** must be used **in composition** with the **data type:**
  - `CLOB_File,` `DBCLOB_File,` `BLOB_File`
  - `XML_CLOB_File,` `XML_DBCLOB_File,` `XML_BLOB_File`
- Will be converted by the **SQL precompiler** into **a data structure**
  - Complete or relative **path name**, **length of the path name**
  - Data length is not used for input but returned when reading the data

LOB/XML Reference File Variable

Data Structure for LOB/XML File Reference Variables

- Sub-field **suffix _NL** **Length file name** `10U 0 / UNS(10)`
- Sub-field **suffix _DL** **Length IFS file data** `10U 0 / UNS(10)`
  Not used for input / set during output
- Sub-field **suffix _FO** **File options** `10U 0 / UNS(10)`
  must be specified within the source code

  SQL Precompiler integrates the **following constants:**

`SQFRD`	`= 2`	File read
`SQFCRT`	`= 8`	Create new file Return error if the file already exists
`SQFOVR`	`= 16`	Create new file or Override the exiting file
`SQFAPP`	`= 32`	Create new file or Add data to the existing file

- Sub-field **suffix _Name** **Name IFS file** `255A / CHAR(255)`

Declaring LOB/XML - Reference File Variables - Example

```
DCL-S MyCLOBFile    SQLType(CLOB_File);
DCL-S MyDBCLOBFile  SQLType(DBCLOB_File);
DCL-S MyBLOBFile    SQLType(BLOB_File);

DCL-S MyXMLCLOBFile    SQLType(XML_CLOB_File);
DCL-S MyXMLDBCLOBFile  SQLType(XML_DBCLOB_File);
DCL-S MyXMLBLOBFile    SQLType(XML_BLOB_File);
```

- Original RPGLE source

```
DCL-DS MYCLOBFILE;
  MYCLOBFILE_NL UNS(10);
  MYCLOBFILE_DL UNS(10);
  MYCLOBFILE_FO UNS(10);
  MYCLOBFILE_NAME CHAR(255) CCSID(*JOBRUNMIX);
END-DS MYCLOBFILE;
//*DCL-S MYDBCLOBFILE SQLTYPE(DBCLOB_FILE);
DCL-DS MYDBCLOBFILE;
  MYDBCLOBFILE_NL UNS(10);
  MYDBCLOBFILE_DL UNS(10);
  MYDBCLOBFILE_FO UNS(10);
  MYDBCLOBFILE_NAME CHAR(255) CCSID(*JOBRUNMIX);
END-DS MYDBCLOBFILE;
```

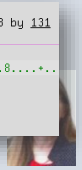
- Excerpt From the compiler listing

Write IFS File with embedded SQL – Example 1

```
DCL-Proc WriteIFS;
  DCL-PI WriteIFS;
      ParIFSFile VarChar(256) Const Options(*Trim);
  End-Pi;

  DCL-S LocClobFile SQLTYPE(Clob_File);

  DCL-S CRLF        Char(2) inz(x'0D25');
//-------------------------------------------------------------------
  Clear LocClobFile;

  LocClobFile_Name = %Trim(ParIFSFile);
  LocClobFile_NL   = %Len(%Trim(LocClobFile_Name));
  LocClobFile_FO   = SQFOVR;       //Create/Override existing IFS File

  Exec SQL Set :LocClobFile = 'Always desire to learn something useful.'
                              concat :CRLF;

  LocClobFile_FO   = SQFAPP;       //Add Data

  Exec SQL Set :LocClobFile = 'Success has a simple formula: ' concat
                              'do your best, and people may like it. '
                              concat :CRLF;
  Return;
End-Proc WriteIFS;
```

- **CLOB file** variable
- Set **CLOB file information**
  - IFS file name
  - Length IFS file name
  - File operation **Create/Replace**
- File operation **Add data**
- **Write** data to **IFS file** using **LOB reference** file variable

```
Browse : /home/Hauser/MyQuote.txt
Record :    1   of      2 by  18                Column :   1     68 by 131
Control :
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+..
 ************Beginning of data**************
Always desire to learn something useful.
Success has a simple formula: do your best, and people may like it.
 ************End of Data********************
```

Write IFS Files with embedded SQL – Example 2

```
DCL-S MyFromBLOB   SQLTYPE(BLOB_File);
DCL-S MyToBLOB     SQLTYPE(BLOB_File);

DCL-S ParToIFSFile  VarChar(256)
                    inz('/home/Hauser/Examples/BLOBFILE.txt');
DCL-S ParFromMbr    VarChar(256) inz('/QSYS.lib/HSRRRPG.lib/+
                                      QEMBSQL.file/BLOBFILE.mbr');
//*****************************************************************
  Exec SQL   Set Option  Commit=*NONE, DatFmt=*ISO, TimFmt=*ISO,
                         Naming=*SYS,  CloSQLCsr=*EndActGrp;

  //Source Member File Information
  MyFromBLOB_Name = %Trim(ParFromMbr);
  MyFromBLOB_NL   = %Len(%Trim(MyFromBLOB_Name));
  MyFromBLOB_FO   = SQFRD;                    //Read Only

  //IFS File Information
  MyToBLOB_Name = %Trim(ParToIFSFile);
  MyToBLOB_NL   = %Len(%Trim(MyToBLOB_Name));
  MyToBLOB_FO   = SQFOVR;                     //Create or Replace

  Exec SQL Set :MyToBLOB = :MyFromBLOB;
  If SQLCODE < *Zeros;
     Dsply SQLCODE;
  EndIf;

*InLR       = *On;
```

Transfer source member to IFS

- **BLOB file** variables for member and IFS file
- Initialize the variables
- Write to the IFS

Generate and write XML Document with Embedded SQL - Example

```
Select EmployeeNo, Name, FirstName, Address, ZipCode, City
  From LobStaff;
```

EMPLOYEENO	NAME	FIRSTNAME	ADDRESS	ZIPCODE	CITY
10	Meier und Sohn		Industriestr. 3-13	80333	München
20	Bauer	Herrmann	Wald-und-Wiesen-Weg. 4	63128	Dietzenbach
40	Hauser	Birgitta	Koenigsteiner Allee 59	63128	Dietzenbach
60	Lehmann	Maria	Schwarzwaldstr. 26	77880	Sasbach

```
Select XmlDocument
         (xmlgroup(EmployeeNo       as "PersNo",
                   Trim(Trim (FirstName) concat ' ' concat
                        Trim(Name)) as "PersName",
                   Address          as "Street",
                   ZipCode          as "ZipCode",
                   City             as "City"
                   Order By City Desc, Name
                   Option Row "Employee"
                   Root "Staff"))
  From LobStaff;
```

W:\home\Hauser\MyXMLDoc.xml

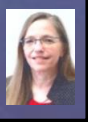
```
<?xml version="1.0" encoding="UTF-8" ?>
- <Staff>
  - <Employee>
      <PersNo>60</PersNo>
      <PersName>Maria Lehmann</PersName>
      <Street>Schwarzwaldstr. 26</Street>
      <ZipCode>77880</ZipCode>
      <City>Sasbach</City>
    </Employee>
  - <Employee>
      <PersNo>10</PersNo>
      <PersName>Meier und Sohn</PersName>
      <Street>Industriestr. 3-13</Street>
      <ZipCode>80333</ZipCode>
      <City>München</City>
    </Employee>
  - <Employee>
      <PersNo>20</PersNo>
      <PersName>Herrmann Bauer</PersName>
      <Street>Wald-und-Wiesen-Weg. 4</Street>
      <ZipCode>63128</ZipCode>
      <City>Dietzenbach</City>
    </Employee>
  - <Employee>
      <PersNo>40</PersNo>
      <PersName>Birgitta Hauser</PersName>
      <Street>Koenigsteiner Allee 59</Street>
      <ZipCode>63128</ZipCode>
      <City>Dietzenbach</City>
    </Employee>
</Staff>
```

Generate and write XML Document with Embedded SQL - Example

```
Dcl-S MyXMLDoc SQLTYPE(XML_CLOB_File);
//*****************************************************************************
Exec SQL Set Option  Commit=*None, DatFmt=*ISO, TimFmt=*ISO,
                     Naming=*SYS,  CloSQLCsr=*EndActGrp;

 Clear MyXMLDoc;
 MyXMLDoc_Name = '/home/Hauser/MyXMLDoc.xml';
 MyXMLDoc_NL   = %Len(%Trim(MyXMLDoc_Name));
 MyXMLDoc_FO   = SQFOVR;                                  //Replace if exists

  Exec SQL Select XmlDocument
                  (xmlgroup(EmployeeNo       as "PersNo",
                            Trim(Trim (FirstName) concat ' ' concat
                                 Trim(Name)) as "PersName",
                            Address          as "Street",
                            ZipCode          as "ZipCode",
                            City             as "City"
                            Order By City Desc, Name
                            Option Row "Employee"
                            Root "Staff"))
                 into :MyXMLDoc
             From LobStaffX;
  If SQLCODE < *Zeros;
     Dsply 'Error occured';
  else;
     Dsply 'XML Document generated';
  EndIf;
*InLR        = *On;
```

- **XML_CLOB file** variable
- Set **XML_CLOB file information**
  - IFS file name
  - Length IFS file name
  - File operation **Create/Replace**
- **XMLDocument** adds <?xml version="1.0" encoding="UTF-8" ?>
- Result is **directly** written into the **IFS** by using the XML_CLOB_File variable

Writing IFS Files with SQL



IFS_WRITE, IFS_WRITE_BINARY, IFS_WRITE_UTF8 – Procedures Write to the IFS

```
                          ( Path_Name    => IFSFileName
IFS_WRITE                 ( Line         => DataToBeWritten
IFS_WRITE_BINARY (        ( File_CCSID   => CCSIDForNewStreamFile )
IFS_WRITE_UTF8            ( OverWrite    => FileOperation
                          ( End_Of_Line  => EndOfLineCharacters
```

Write Data into an IFS (Integrated File System) Stream File

- **IFS_WRITE** Data is converted into a **EBCDIC** Characters
- **IFS_WRITE_BINARY** Data is **not converted**
- **IFS_WRITE_UTF8** Data is converted into **UTF-8**

New IFS File can be **created**
Data can be **replaced** in an **existing IFS File**
Data can be **appended** to an **existing IFS File**

Write Data to the IFS - Example

```
Call Qsys2.IFS_Write_UTF8(Path_Name     => '/home/Hauser/Examples/WriteWithSQL.txt',
                          Line          => 'Always desire to learn something useful.',
                          File_CCSID    => 1208,
                          OverWrite     => 'REPLACE',
                          End_Of_Line   => 'CRLF',
                          Ignore_Errors => 'NO');

Call Qsys2.IFS_Write(Path_Name   => '/home/Hauser/Examples/WriteWithSQL.txt',
                     Line        => 'Success has a simple formula: do your best, and people may like it.',
                     OverWrite   => 'APPEND',
                     End_Of_Line => 'CRLF');
```

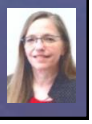
- **Write a new /home/Hauser/Examples/WriteWithSQL.txt IFS file**
- **Or Replace an existing /home/Hauser/Examples/WriteWithSQL.txt IFS file**
- **Include the text specified within the LINE parameter**

- **Add the text specified in the LINE parameter at the end of the previously generated/replaced IFS File**

WriteWithSQL.txt

```
Always desire to learn something useful.
Success has a simple formula: do your best, and people may like it.
```


Reading IFS-Files
using File Reference Variables



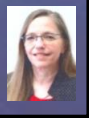
Read IFS Stream File using a File Reference Variable - Example

```
DCL-S MyCLOBFile SQLType(CLOB_File);
DCL-S MyText     Char(50);

DCL-S Start      Int(10);
DCL-S RowLen     Int(5)        Inz(50);
DCL-S Index      Uns(3);

DCL-S ParIFSFile VarChar(256)  Inz('/home/Hauser/Examples/MyText.txt');
//*****************************************************************
Exec SQL   Set Option  Commit=*NONE, DatFmt=*ISO, TimFmt=*ISO,
                       Naming=*SYS,  CloSQLCsr=*EndActGrp;

  //Retrieve Locator onto save file
  MyCLOBFile_Name = %Trim(ParIFSFile);
  MyCLOBFile_NL   = %Len(%Trim(MyCLOBFile_Name));
  MyCLOBFile_FO   = SQFRD;                          //Read Only

  For Index =  1 to 5;
      Start = (Index-1) * RowLen + 1;
      Exec SQL Set :MyText = Substr(:MyCLOBFILE, :Start, :RowLen);
      If SQLCODE = 100 or SQLCODE < *Zeros
         or %Len(%Trim(MyText)) = *Zeros;
         Leave;
      EndIf;
      Dsply MyText;
  EndFor;

*InLR       = *On;
```

- Initializing Data Structure Sub-Fields for the CLOB_File
- File Operation: Read

- Reading Data directly from the IFS with SQL Functions

Consuming IFS Files with SQL



Accessing IFS Files with SQL Functions

```
GET_BLOB_FROM_FILE   (StringExpression, Integer)
GET_CLOB_FROM_FILE   (StringExpression, Integer)
GET_DBCLOB_FROM_FILE (StringExpression, integer)
```

Data in a IFS File or in a Source Physical File Member can be accessed

- *StringExpression:* Name of the **IFS-Datei** or a **Source Physical File Member** or a **Variable** containing an IFS File Name or a Source Member Name
  Syntax Source Member: `SOURCESCHEMA/SOURCEFILE(SOURCEMEMBER)`
- *Integer* 0 = Whitespace is returned
  1 = Whitespace is removed

CCSID Conversion depends on the used (GET_LOB) Funktion

Accessing IFS Files with SQL Functions

```
GET_BLOB_FROM_FILE   (StringExpression, Integer)
GET_CLOB_FROM_FILE   (StringExpression, Integer)
GET_DBCLOB_FROM_FILE (StringExpression, integer)
```

The result of the function is a CLOB-Locator

- Can be compared with a **pointer** that points to the data
- Can be used in **SQL statements** and with **scalar functions** like any **character value**

Data is converted into the CLOB's CCSID

Must be executed under Commitment Control

- The locator is **freed** when a **COMMIT** or **ROLLBACK** is performed

```
Values(Get_CLOB_From_File('/home/Hauser/MyQuote.txt'));
Values(Get_CLOB_From_File('HSCOMMON05/QEMBSQLF(EMBLOBLOC4)'));
```

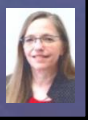
- IFS File
- Source Member

Read Data from the IFS using a File Reference Locator

```
DCL-S MyCLOBLoc   SQLType(CLOB_Locator);
DCL-S MyText      Char(50);

DCL-S Start       Int(10);
DCL-S RowLen      Int(5)          Inz(50);
DCL-S Index       Uns(3);

DCL-S ParIFSFile  VarChar(256)    Inz('/home/Hauser/Examples/MyText.txt');
//*****************************************************************
Exec SQL   Set Option Commit=*CHG, DatFmt=*ISO, TimFmt=*ISO,
                      Naming=*SYS,  CloSQLCsr=*EndActGrp;

 Commit;
 Exec SQL Set :MyCLOBLoc = Get_CLOB_From_File(:ParIFSFile);
 If SQLCODE < *Zeros;
    //Handle Error
 EndIf;

 For Index =  1 to 5;
     Start = (Index-1) * RowLen + 1;
     Exec SQL Set :MyText = Substr(:MyCLOBLoc, :Start, :RowLen);
     If %Len(%Trim(MyText)) <> *Zeros;
        Dsply MyText;
     EndIf;
 EndFor;

 Commit;
*InLR       = *On;
```

- Must be executed under Commitment Control
- Accessing the IFS File with GET_CLOB_FROM_FILE
  → Returns a LOB Locator
- Reading Data directly from the IFS with SQL Functions

Consume *.csv File from the IFS



Split Table Function

New RELEASE 7.4

```
SYSTOOLS.SPLIT(Input_List, Delimiter)
```

Table Function is located within the SYSTOOLS schema

Returns a table that contains one row for each element of the input_list

- *Input_List:* String containing the **list of elements** to be deconstructed
  Input_List elements must be **separated by the delimiter**
- *Delimiter:* **Separator** between the elements (can be more than 1 character)

Return Table consist of 2 columns

- `ORDINAL_POSITION` Integer Relative position of the element in the input string
- `ELEMENT` CLOB(2G) Value of the Element

Note: (*from documentation*) this function is provided in the SYSTOOLS schema as an **example of how to break a string apart at a delimiting character** by using an SQL table function. **Creating customized versions of this table function to better suit a specific need is encouraged**. Use the Insert Generated SQL feature in ACS to extract the source for this function. Then modify it and create a new function.

Split Table Function – Examples – Split List

```
Select *
  From Table(SysTools.Split('A, B, C, D, E, V, W, X, Y, Z', ', '));
```

ORDINAL_POSITION	ELEMENT
1	A
2	B
3	C
4	D
5	E
6	V
7	W
8	X
9	Y
10	Z

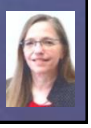
- Splits a string containing characters separated by a comma into rows

```
With x (MyList) as (Values('A, B, C, X'), ('V, W, X, Y, Z'))
Select  MyList, a.*
   from x cross join Table(SysTools.Split(MyList, ', ')) a
Order By MyList, Ordinal_Position;
```

MYLIST	ORDINAL_POSITION	ELEMENT
A, B, C, X	1	A
A, B, C, X	2	B
A, B, C, X	3	C
A, B, C, X	4	X
V, W, X, Y, Z	1	V
V, W, X, Y, Z	2	W
V, W, X, Y, Z	3	X
V, W, X, Y, Z	4	Y
V, W, X, Y, Z	5	Z

- Splits 2 Lists into Rows:
  1. A, B, C, X
  2. V, W, X, Y, Z
- The Common Table Expression X must be (cross) joined with the SPLIT() Table Function

SPLIT() Table Function within the SYSTOOLS Schema



Split Table Function – Examples – Read CSV File

```
With x as (Select Ordinal_Position as RowKey, Element as RowInfo
             from Table(SysTools.Split(Get_Clob_From_File('/home/Hauser/Employee.csv'), x'0D25')) a
             Where Trim(Element) > ''),
     y as (Select x.*, Ordinal_Position ColKey,
                  Trim(B '"' from Element) as ColInfo
             from x cross join Table(SysTools.Split(RowInfo, ',')) a)
Select RowKey,
       Min(Case When ColKey = 1 Then ColInfo End) EmployeeNo,
       Min(Case When ColKey = 2 Then ColInfo End) Name,
       Min(Case When ColKey = 3 Then ColInfo End) FirstName,
       Min(Case When ColKey = 4 Then ColInfo End) Address,
       Min(Case When ColKey = 5 Then ColInfo End) Country,
       Min(Case When ColKey = 6 Then ColInfo End) ZipCode,
       Min(Case When ColKey = 7 Then ColInfo End) City
From y
Where RowKey > 1
Group By RowKey;
```

ROWKEY	EMPLOYEENO	NAME	FIRSTNAME	ADDRESS	COUNTRY	ZIPCODE	CITY
2	10	Meier und Sohn		Industriestr. 3-13	DE	80333	Muenchen
3	20	Bauer	Herrmann	Wald-und-Wiesen-Weg. 4	DE	63128	Dietzenbach
4	40	Hauser	Birgitta	Koenigsteiner Allee 59	DE	63128	Dietzenbach
5	60	Lehmann	Maria	Schwarzwaldstr. 26	DE	77880	Sasbach

Common Table Expression X:
- Get_Clob_From_File: Access IFS file
- Split the IFS File at CRLF (=x'0D25')

Common Table Expression Y:
- Split the already in X split row data into column data

Final Select
- Put the column values into separate columns using a case clauses
- Accumulate the table on the row information

Read *.csv File with embedded SQL

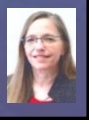
```
DCL-S DsplyText    Char(50);

DCL-DS GblDSCsrCsv01      Qualified      Inz;
   RowKey                 Int(10);
   EmployeeNo             Int(10);
   Name                   VarChar(50);
   FirstName              VarChar(50);
   Address                VarChar(50);
   ZipCode                VarChar(10);
   City                   VarChar(50);
   Country                VarChar(4);
End-Ds;
```

- Output Data Structure for the Cursor

```
Exec SQL
Declare CsrCsv01 Cursor For
   With x as (Select Ordinal_Position as RowKey, Element as RowInfo
                from Table(SysTools.Split(
                                Get_Clob_From_File(
                                        '/home/Hauser/Employee1.csv'),
                                        x'0D25')) a
                Where Trim(Element) > ''),
        y as (Select x.*, Ordinal_Position ColKey,
                     Trim(B '"' from Element) as ColInfo
                from x cross join
                     Table(SysTools.Split(RowInfo, ',')) a)
   Select RowKey,
          Min(Case When ColKey = 1 Then ColInfo End) EmployeeNo,
          Min(Case When ColKey = 2 Then ColInfo End) Name,
          Min(Case When ColKey = 3 Then ColInfo End) FirstName,
          Min(Case When ColKey = 4 Then ColInfo End) Address,
          Min(Case When ColKey = 5 Then ColInfo End) ZipCode,
          Min(Case When ColKey = 6 Then ColInfo End) City,
          Min(Case When ColKey = 7 Then ColInfo End) Country
     From y
     Where RowKey > 1
     Group By RowKey
     Order By EmployeeNo;

Exec SQL Close CsrCsv01;
Exec SQL Open CsrCsv01;

DOU 1=0;
    Exec SQL Fetch Next From CsrCsv01 into :GblDSCsrCsv01;
    If SQLCODE = 100 or SQLCODE < *Zeros;
       Exec SQL Close CsrCsv01;
       Leave;
    EndIf;

    DsplyText = %Char(GblDSCsrCsv01.EmployeeNo) + ' '  +
                %Trim(GblDSCsrCsv01.Name)       + ', ' +
                %Trim(GblDSCsrCsv01.FirstName)  + ' '  +
                %Trim(GblDSCsrCsv01.City);
    Dsply DsplyText;
EndDo;
*INLR = *On;
```

- Complete SQL Statement for reading and splitting the *.csv File
- Cursor Handling

Reading IFS Files with SQL



IFS_READ, IFS_READ_BINARY, IFS_READ_UTF8 – Table Functions Reading IFS Files

```
IFS_READ
IFS_READ_BINARY ( Path_Name            => IFSFileName
                  Maximum_Line_Length  => MaximumCharactersPerLine )
IFS_READ_UTF8     End_Of_Line          => EndOfLineCharacters
```

Read Data from an IFS (Integrated File System) Stream File

- **IFS_READ** (Read) Data is converted into a **EBCDIC** Characters
- **IFS_READ_BINARY** (Read) Data stays **unconverted**
- **IFS_READ_UTF8** (Read) Data is converted into **UTF-8**

Data can be returned as **one string** or be converted into **multiple rows**

→ depending on the End Of Line specification

Read IFS Stream File with SQL - Example

```
Select * from Table(IFS_READ_UTF8('/home/Hauser/Employee1.csv')) x;
```

LINE_NUMBER	LINE
1	"EMPLOYEENO","NAME","FIRSTNAME","ADDRESS","ZIPCODE","CITY","COUNTRY"
2	10,"Meier und Sohn","","Industriestr. 3-13","80333","Muenchen","DE"
3	20,"Bauer","Herrmann","Wald-und-Wiesen-Weg. 4","63128","Dietzenbach","DE"
4	40,"Hauser","Birgitta","Koenigsteiner Allee 59","63128","Dietzenbach","DE"
5	60,"Lehmann","Maria","Schwarzwaldstr. 26","77880","Sasbach","DE"
6	50,"Burger","Emil","Nelkenweg 21","86916","Kaufering","DE"
7	80,"Miller","Katrin","Am Lech 35","86916","Kaufering","DE"

- Read /home/Hauser/Employee1.csv file and convert the result into UTF-8
- Line Break occurs on any End_Of_Line Character

Read IFS Stream File with SQL and split it into Columns - Example

New Release 7.4 TR 3

```
With x as (Select * from Table(IFS_READ_UTF8('/home/Hauser/Employee1.csv'))),
     y as (Select x.*, Ordinal_Position ColKey, Trim(B '"' from Element) ColInfo
             from x cross join Table(SysTools.Split(Line, ',')))
Select Line_Number,
       Min(Case When ColKey = 1 Then ColInfo End) EmployeeNo,
       Min(Case When ColKey = 2 Then ColInfo End) Name,
       Min(Case When ColKey = 3 Then ColInfo End) FirstName,
       Min(Case When ColKey = 4 Then ColInfo End) Address,
       Min(Case When ColKey = 5 Then ColInfo End) Country,
       Min(Case When ColKey = 6 Then ColInfo End) ZipCode,
       Min(Case When ColKey = 7 Then ColInfo End) City
From y
Where Line_Number > 1
Group By Line_Number
Order By Line_Number
```

- IFS File

```
"EMPLOYEENO","NAME","FIRSTNAME","ADDRESS","ZIPCODE","CITY","COUNTRY"
10,"Meier und Sohn","","Industriestr. 3-13","80333","Muenchen","DE"
20,"Bauer","Herrmann","Wald-und-Wiesen-Weg. 4","63128","Dietzenbach","DE"
40,"Hauser","Birgitta","Koenigsteiner Allee 59","63128","Dietzenbach","DE"
60,"Lehmann","Maria","Schwarzwaldstr. 26","77880","Sasbach","DE"
50,"Burger","Emil","Nelkenweg 21","86916","Kaufering","DE"
80,"Miller","Katrin","Am Lech 35","86916","Kaufering","DE"
```

- CTE x: Read IFS-File with the IFS_READ_UTF8 UDTF Split it into Rows
- CTE y: Split the Rows into columns with the SPLIT UDTF
- Final Select: Accumulate the columns

LINE_NUMBER	EMPLOYEENO	NAME	FIRSTNAME	ADDRESS	COUNTRY	ZIPCODE	CITY
2	10	Meier und Sohn		Industriestr. 3-13	80333	Muenchen	DE
3	20	Bauer	Herrmann	Wald-und-Wiesen-Weg. 4	63128	Dietzenbach	DE
4	40	Hauser	Birgitta	Koenigsteiner Allee 59	63128	Dietzenbach	DE
5	60	Lehmann	Maria	Schwarzwaldstr. 26	77880	Sasbach	DE
6	50	Burger	Emil	Nelkenweg 21	86916	Kaufering	DE
7	80	Miller	Katrin	Am Lech 35	86916	Kaufering	DE

Read *.csv File with embedded SQL

New RELEASE 7.4 TR 3

```
DCL-S DsplyText    Char(50);

DCL-DS GblDSCsrCsv03       Qualified      Inz;
   RowKey                  Int(10);
   EmployeeNo              Int(10);
   Name                    VarChar(50);
   FirstName               VarChar(50);
   Address                 VarChar(50);
   ZipCode                 VarChar(10);
   City                    VarChar(50);
   Country                 VarChar(4);
End-Ds;
```

- Output Data Structure for the Cursor

```
Exec SQL
Declare CsrCsv03 Cursor For
   With x as (Select *
                 from Table(IFS_READ_UTF8(
                                    '/home/Hauser/Employee1.csv'))),
        y as (Select x.*, Ordinal_Position ColKey,
                     Trim(B '"' from Element) ColInfo
                from x cross join Table(SysTools.Split(Line, ',')))
   Select Line_Number,
          Min(Case When ColKey = 1 Then ColInfo End) EmployeeNo,
          Min(Case When ColKey = 2 Then ColInfo End) Name,
          Min(Case When ColKey = 3 Then ColInfo End) FirstName,
          Min(Case When ColKey = 4 Then ColInfo End) Address,
          Min(Case When ColKey = 5 Then ColInfo End) Country,
          Min(Case When ColKey = 6 Then ColInfo End) ZipCode,
          Min(Case When ColKey = 7 Then ColInfo End) City
   From y
   Where Line_Number > 1
   Group By Line_Number
   Order By Line_Number;

Exec SQL Close CsrCsv03;
Exec SQL Open CsrCsv03;

DOU 1=0;
    Exec SQL Fetch Next From CsrCsv03 into :GblDSCsrCsv03;
    If SQLCODE = 100 or SQLCODE < *Zeros;
       Exec SQL Close CsrCsv03;
       Leave;
    EndIf;

    DsplyText = %Char(GblDSCsrCsv03.EmployeeNo) + ' '  +
                %Trim(GblDSCsrCsv03.Name)       + ', ' +
                %Trim(GblDSCsrCsv03.FirstName)  + ' '  +
                %Trim(GblDSCsrCsv03.City);
    Dsply DsplyText;
EndDo;
*INLR = *ON;
```

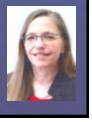
- Complete SQL Statement for reading and splitting the *.csv File
- Read IFS File with the IFS_READ_UTF8 Function
- Cursor Handling

Read *.csv File in a View with embedded SQL

```
  DCL-S DsplyText    Char(50);
//-----------------------------------------------------------------------
  Exec SQL
  Declare CsrCsv02 Cursor For
     Select EmployeeNo concat ' ' concat Trim(Name)      Concat ', '
                                  concat Trim(FirstName) Concat ' '
                                  concat Trim(City)

       from Emplcsvv02
       Order By EmployeeNo;

  Exec SQL Close CsrCsv02;
  Exec SQL Open CsrCsv02;

  DOU 1=0;
      Exec SQL Fetch Next From CsrCsv02 into :DsplyText;
      If SQLCODE = 100 or SQLCODE < *Zeros;
         Exec SQL Close CsrCsv02;
         Leave;
      EndIf;

      Dsply DsplyText;
  EndDo;

  *INLR = *On;
```

- SELECT Statement / Complexitiy hidden in the SQL View

Embedded SQL
and RESTFul Web-Services



http Functions for Db2 for i

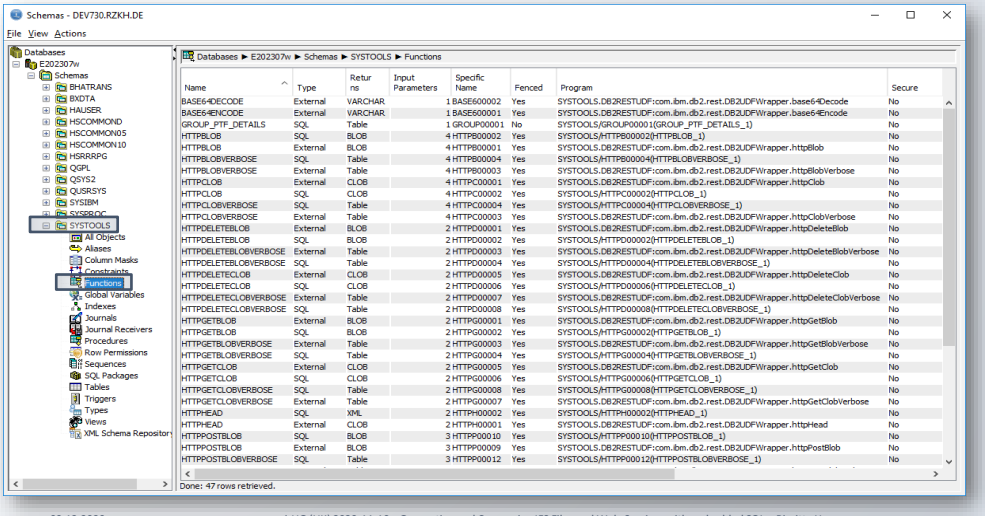
Scalar and Table Functions (names starting with http)

- Located in the **SYSTOOLS** schema

Function Names

- `http (Method) (DataType) (Verbose)`
  - **Method**: `GET, PUT, POST, DELETE, HEAD`
    Without method → Method passed as **parameter value**
  - **Data type:** of the request AND response messages
    CLOB (Character Large Object) / **BLOB** (Binary Large Object)
  - **Verbose:** with Verbose: **Table function** with 2 columns
    1st column: http header information
    2nd column: response message
    without Verbose: **Scalar Function** → response message only
- **Examples:** `httpGetClob(), httpPutBlobVerbose(), httpClob()`

Access Client Solutions (ACS)
http Functions in Schema SYSTOOLS



Consuming Web Service - Check CreditCard No

- Credit Card Checker – Bincodes: https://www.bincodes.com/api-creditcard-checker/

Attention:
- **Job CCSID must be anything else than 65535**
- **You need your own API key! In this example is the API key hidden in a SQL global variable**

```
https://api.bincodes.com/cc/?format=[FORMAT]&api_key=[API_KEY]&cc=[CC]
```

Valid Credit Card No

```
{
    "bin": "371449",
    "bank": "AMERICAN EXPRESS US CONSUMER",
    "card": "AMERICAN EXPRESS",
    "type": "CREDIT",
    "level": "PERSONAL",
    "country": "UNITED STATES",
    "countrycode": "US",
    "website": "",
    "phone": "",
    "valid": "true"
}
```

Error

```
{
    "error": "1014",
    "message": "Invalid Credit Card or Debit Card Number"
}
```

```
DCL-DS Templ_CreditCard  Template     Qualified   Inz;
  CredCardNo             VarChar(25);
  BIN                    VarChar(20);
  Bank                   VarChar(128);
  Card                   VarChar(50);
  Type                   VarChar(20);
  Level                  VarChar(20);
  Country                VarChar(128);
  CountryCode            VarChar(5);
  Website                VarChar(128);
  Phone                  VarChar(50);
  Valid                  VarChar(5);
  ErrorCode              Varchar(5);
  ErrorMsg               VarChar(128);
  ValidInd               Ind;
End-Ds;
```

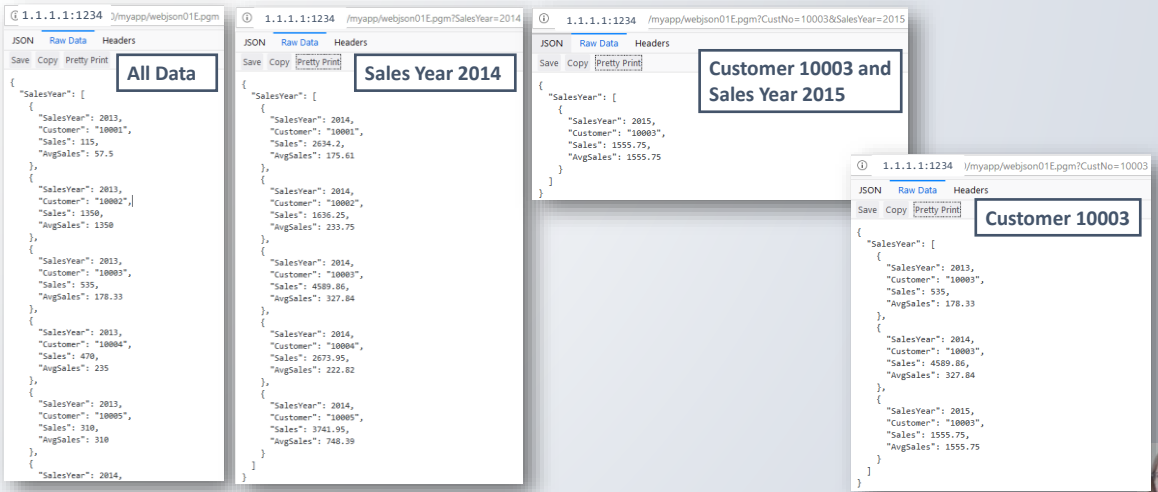

Consuming Web Service - Check CreditCard No

```
Exec SQL
  Set :LocURL = 'https://api.bincodes.com/cc/?format=json&api_key='
                concat Trim(GBLKEYBINCODES)
                concat '&cc=' concat Trim(:ParCreditCard);

Exec SQL
  Select :ParCreditCard, x.*, '0' into :RtnCreditCardInfo
    from JSON_TABLE(SysTools.HTTPGETCLOB(:LocURL, ''), '$'
           Columns(Bin          Varchar(10)   Path '$.bin'
                                              Default '' on Empty,
                   bank         VarChar(128)  Path '$.bank'
                                              Default '' on Empty,
                   card         VarChar(50)   Path '$.card'
                                              Default '' on Empty,
                   cardType     Varchar(20)   Path '$.type'
                                              Default '' on Empty,
                   cardLevel    VarChar(20)   Path '$.level'
                                              Default '' on Empty,
                   Country      VarChar(128)  Path '$.country'
                                              Default '' on Empty,
                   CountryCde   VarChar(5)    Path '$.countrycode'
                                              Default '' on Empty,
                   website      VarChar(128)  Path '$.website'
                                              Default '' on Empty,
                   phone        VarChar(50)   Path '$.phone'
                                              Default '' on Empty,
                   valid        VarChar(10)   Path '$.valid'
                                              Default '' on Empty,
                   ErrorCode    VarChar(10)   Path '$.error'
                                              Default '' on Empty,
                   ErrorMsg     VarChar(256)  Path '$.message'
                                              Default '' on Empty)
                ) x;
```

- **Build URL**

- **Receive Data from a Web service into a Data Structure with an SELECT … INTO Statement**
- **HTTPGETCLOB()**
  - **Receive (JSON/XML) Data from a Web-service**
- **JSON_TABLE()**
  - **Parse the JSON Data**

Call a RPG Program/Web Service through the browser



RESTFul Web Service RPG/Embedded SQL - H and D Specifications

```
CTL-Opt BndDir('HSBNDDIR05': 'WEBSRVUTL')
        DecEdit('0.')   Option(*NoDebugIO: *NoUnRef)  Alloc(*TeraSpace)
        Main(WEBJSON01);

        /If Defined (*CRTBNDRPG)
Ctl-Opt  ActGrp('WEBJSON01');
        /EndIf
//*************************************************************
/Include WebSrvUtl/QCPYSRC,WEBSRVUTL
/Include WebSrvUtl/QCPYSRC,APIERR
//*************************************************************
DCL-S GblJSONData      SQLType(CLOB: 1000000)  CCSID(1208);

//URL-Key Information
DCL-DS  GblDSUrlKey    Qualified       Inz;
  CustNo               VarChar(15);
  SalesYear            Packed(4);
End-DS;
```

- Calling procedures from the WEBSRVUTL open source library from Rainer Ross
- https://github.com/RainerRoss/WEBSRVUTL
- JSON Data is written into a CLOB (Character Large Object) variable with CCSID 1208 = UTF8
- The query string variables are loaded into the GblDSURLKey global data structure

Program for the RESTFul Web Service Main-Procedure

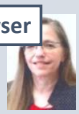
```
DCL-Proc WEBJSON01;

  DCL-S LocHeader   like(GblHeader);
  DCL-S LocErrMsg   VarChar(132);
  //-------------------------------------------------------------------------------------
  Monitor;
   Clear GblDSUrlKey;
   Clear GblJSONData;

   LocHeader = getHeader();
   getInput();
   GblDSUrlKey.CustNo = %Trim(getKeyValue('CustNo'));

   Monitor;
      GblDSUrlKey.SalesYear = %Dec(getKeyValue('SalesYear'): 4: 0);
   On-Error;
      Clear GblDSURLKey.SalesYear;
   EndMon;
```

- getHeader() – Receive the Header Information
- getInput() – Receive the URL data
- getKeyValue() – Determine the key values

- Generating the JSON Data using SQL

```
   WrtStdout(%Addr(LocHeader:*Data): %Len(LocHeader): DsApierr);
   WrtStdout(%Addr(GblJSONData_Data): GblJSONData_Len: DsApierr);

 end-proc;
```

- wrtStdOut() – Send Header and data to the browser



Program for the RESTFul Web Service
Generate the JSON Data

```
exec sql
With x as (Select Year(SalesDate) SalesYear, CustNo,
                  Sum(Amount) Sales,
                  Cast(Avg(Amount) as Dec(11, 2)) Average
             From HSRRRPG.SalesX
             Where CustNo = case When :GblDSUrlKey.CustNo > ''
                                Then :GblDSUrlKey.CustNo
                                Else CustNo End
               and Year(SalesDate)
                   = Case When :GblDSUrlKey.SalesYear > 0
                          Then :GblDSUrlKey.SalesYear
                          Else Year(SalesDate) End
             Group By Year(SalesDate), CustNo)

Select JSON_Object('SalesYear':
         JSON_ArrayAgg(
           JSON_Object('SalesYear'   : SalesYear,
                       'Customer'    : Trim(CustNo),
                       'Sales'       : Sales,
                       'AvgSales'    : Average)
                 Order By SalesYear, CustNo))
       into :GblJSONData
   From x;
If SQLCODE < *Zeros;
   Exec SQL Get Diagnostics Condition 1 :LocErrmsg=MESSAGE_TEXT;
   Exec SQL Set :GblJSONData = JSON_Object('Error': :LocErrMsg);
EndIf;
```

Common Table Expression
→ Determine the output data

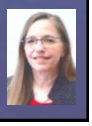
→ Based on the passed key values

Final Select:
→ Generate the JSON data

→ Predefine the sequence by adding an ORDER BY clause to the JSON_ArrayAgg() functtion

→ Write the JSON data into a CLOB Variable

Any Questions?



Special Thanks to

Holger Scherer – RZKH Rechenzentrum Kreuznach

- For providing an IBM i-System enabling the creation of the samples/code used in my presentations
- http://www.rzkh.de

Your data is save! … in the bunker

References

IBM i Information Center

- DB2 for i SQL Reference
  http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzajp/rzajppdf.pdf?lang=en
- Embedded SQL programming
  http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzpdf.pdf?lang=en
- RPG Reference
  https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzasd/sc092508.pdf?view=kc

IBM Redbooks

- Who Knew You Could Do That with RPG IV? Modern RPG for the Modern Programmer
  http://www.redbooks.ibm.com/abstracts/sg245402.html?Open
- Modernizing IBM eServer iSeries Application Data Access – A Roadmap Cornerstone
  http://www.redbooks.ibm.com/abstracts/sg246393.html?Open
- Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between
  http://www.redbooks.ibm.com/abstracts/sg248185.html?Open

Speaker‘s Biography

Birgitta Hauser
Diplom-Betriebswirt (BA)
Database and Software Architect

Birgitta Hauser worked on the IBM i and its predecessors since 1992. She graduated with a business economics diploma, and started programming on the AS/400 in 1992. She worked and works as traditional RPG Programmer but also as Database and Software Engineer, focusing on IBM i application and database modernization.

Currently she is self-employed and works in Consulting and Application and Database Modernization on IBM i and Db2 for i. Since July, 2019 she is occasionally working for Fresche Solutions Inc. (Montréal) as a contractor.

She also works in education as a trainer for RPG and SQL developers.

Since 2002 she has frequently spoken at the COMMON User Groups and other IBM i and Power Conferences in Germany, other European Countries, USA and Canada.

In addition, she is co-author of two IBM Redbooks and also the author of several articles and papers focusing on RPG and SQL for the ITP Verlag (a German publisher), IT Jungle Guru and IBM DeveloperWorks.

In 2015 she received the John Earl Speaker Scholarship Award. In 2018 she received the Al Barsa Memorial Scholarship Award.

IBM Champion 2020

Thank You!

Advanced Embedded SQL? Now i know!

If you are interested in more detailed individual Workshops on-site or remote, or support in Application and Database Modernization on IBM i Please contact me directly

Birgitta Hauser
Diplom-Betriebswirt (BA)
Database and Software Architect

Hauser@SSS-Software.de