How to use the Java-based Connector with DBCS

The z/VSE e-business Connectors component 5686-CF8-35 contains different types of connectors:

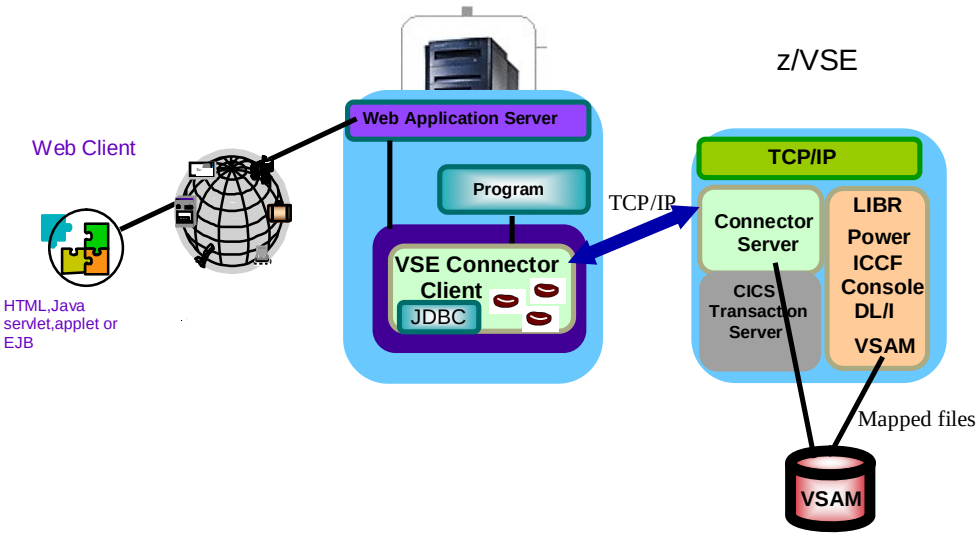
- Java based-Connector (VSE Connector Client and Server)
- VSE Script Server
- VSAM Redirector
- Web Services (SOAP)

These connectors have different options how to work with DBCS data. This document describes how to use the **Java-based Connector** with DBCS data.

Note: This document does not cover tools like VSE Navigator which are using the VSE Connector Client under the covers.

Overview

The Java-based connector consists of a VSE Connector Client and a VSE Connector Server. The VSE Connector Server runs on z/VSE. The VSE Connector Client provides a set of Java classes that you can use as programming interface (API) in your Java applications.



The API has several classes, each providing several methods/functions. Many functions take textual data as input or pass back textual data as output. Here, we have to distinguish between textual names of system resources (like member names, file names, job names, user IDs) and textual user data (like the contents of a member, the contents of a file or record).

Textual resource names

Textual resource names do not contain DBCS characters. Most resource names do only allow characters from A-Z, 0-9 and some special characters like $, %, &, #, -, _ or *.

For resource names, ASCII to EBCDIC codepage conversion automatically happens on the server side. Since the names only contain SBCS characters, you can use a SBCS codepage like IBM-1047 (US English) for that conversation.

Textual user data

Textual user data can be contained within resources like members, files or records. When accessing such user data, the user of the VSE Connector Client API can decide if the access should be performed without translation (binary mode) or with translation (text mode).

Since the codepage translation process of the VSE Connector Server does not support DBCS, you need to access the user data in binary mode to avoid any codepage translation at that time. The VSE Connectors do not display the user data itself; it's the user application that for example displays the data on a screen or on a web page. So anything that deals with correct displaying of the data (e.g. BiDi, left-to-right vs. right-to-left) is out of the scope of the VSE Connector Client. It's the user application that has to take care of this.

You can use Java's codepage translation methods to translate any (SBCS and DBCS) EBCDIC data, which you have obtained in binary mode, into its ASCII or Unicode equivalent. Java internally works with Unicode (UTF-16), so special characters (DBCS) can be handled by Java. Java supports various codepages (called Charsets in Java), including EBCDIC DBCS codepages. Please see here for a list of supported codepages/charsets: http://java.sun.com/javase/6/docs/technotes/guides/intl/encoding.doc.html

To translate binary data into a Unicode Java string, you can use the following code:

```
byte[] bindata = new byte[100];
// fill bindata with binary data

String str = new String(bindata,"Cp1047");
```

In this example the binary data will be translated from the EBCDIC codepage Cp1047 into a Java string (Unicode). You can also specify a DBCS EBCDIC codepage instead.

To translate a Java String into binary data using an EBCDIC codepage use the following code:

```
String str = "Text to translate";
byte[] bindata = str.getbytes("Cp1047");
```

Again, this example uses codepage Cp1047, but you can use any other codepage as well.

Accessing Library member contents in binary mode

The class VSELibraryMember can be used to upload (`VSELibraryMember.upload(...)`) or download (`VSELibraryMember.download(...)`) the contents of a library member. To switch to binary mode, use method `VSELibraryMember.setBinary(true)` before up- or downloading.

```
VSEConnectionSpec spec = new VSEConnectionSpec(InetAddress.getByName("1.2.3.4"),
                                                2893,"SYSA","password");
spec.setLogonMode(true);

VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);

VSELibraryMember member = new VSELibraryMember(system,"LIB1","SLIB2","MYMEMB","X");
member.setBinary(true);
member.download("C:\\myfile.txt");

member.setBinary(true);
member.upload("C:\\myfile.txt");
```

Dependent on the format of the member (fixed or string) and the logical record length, you can the split the received binary member data into lines of for example 80 bytes each and translate the lines from DBCS EBCDIC to Unicode or ASCII.

Accessing POWER entry contents in binary mode

The class VSEPowerEntry can be used to upload or submit (`VSEPowerEntry.put(...)`) or download (`VSEPowerEntry.get(...)`) the contents of a POWER entry. The POWER entry can be a job, a listing or a punch entry. To switch to binary mode, use method `VSEPowerEntry.setTransferBinary(true)` and the `VSEPowerEntry.setTransferRecordLength(record-length)` before up- or downloading.

```
VSEConnectionSpec spec = new VSEConnectionSpec(InetAddress.getByName("1.2.3.4"),
                                                2893,"SYSA","password");
spec.setLogonMode(true);

VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);

VSEPowerEntry entry = new VSEPowerEntry(system,VSEPowerEntry. QUEUE_LIST,
                              "MYJOB2",1234,0);
entry.setTransferBinary(true);
entry.setTransferRecordLength(121);
entry.get("C:\\myfile.txt");

entry.setTransferBinary(true);
entry.setTransferRecordLength(121);
entry.put("C:\\myfile.txt");
```

Dependent on the format of the entry and the logical record length, you can the split the received binary entry data into lines of for example 121 bytes each and translate the lines from DBCS EBCDIC to Unicode or ASCII.

Accessing VSAM record contents in binary mode

The class VSEVsamRecord can be used to get (`VSEVsamRecord.getField(...)`), update (`VSEVsamRecord.setField(...)` followed by `VSEVsamRecord.commit()`), or insert (`VSEVsamRecord.add(...)`) VSAM records in a VSAM cluster. To access VSAM data, you need to define a mapping of the fields within the record (see class VSEVSamMap). The mapping defines what part of the record belongs to what field. Every field has a name, an offset within the record, a length in bytes and a data type.

The data type determines if the field data is translated by the VSE Connectors or not. There are different data types like INTEGER, PACKED, or ZONED which are used to store numeric data. For textual data you typically use the type STRING. The contents of a STRING type field will be translated from EBCDIC to ASCII and vice versa when access the record. If the VSAM record contains DBCS data, you should use the data type BINARY instead. BINARY fields will not be translated.

```
VSEConnectionSpec spec = new VSEConnectionSpec(InetAddress.getByName("1.2.3.4"),
                                               2893,"SYSA","password");
spec.setLogonMode(true);

VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);

VSEVsamCluster cluster = new VSEVsamCluster(system,"MY.USER.CATALOG",
                                            "MY.SAMPLE.CLUSTER");
VSEVSamMap map = cluster.getVSEVsamMap("MYMAP");

// add a record with DBCS data
VSEVsamRecord record1 = cluster.getVSEVsamRecord(map);
record1.setKeyField(map.getIndex("KEY-FIELD"), "RECORD00");            // STRING (key)
record1.setField(map.getIndex("DBCS-FIELD"),dbcsdata.getBytes(("Cp1047")); // BINARY
record1.setField(map.getIndex("NUMERIC-FIELD"),new Integer(123));          // PACKED
record1.add();

// read a record with DBCS data
VSEVsamRecord record2 = cluster.getVSEVsamRecord(map);
record2.setKeyField("KEY-FIELD", "RECORD00");                    // STRING field (key)
String dbcsdata = new String((String)record2.getField(map.getIndex("DBCS-FIELD")),
                             "Cp1047"); // BINARY field
Integer numdata = (Integer)record2.getField(map.getIndex("NUMERIC-FIELD"));   // PACKED

// update the record
record2.setField(map.getIndex("DBCS-FIELD"),textdata.getBytes(("Cp1047")); // BINARY
record2.commit();
```

Here the DBCS data is translated with Java methods into Java's String representation (Unicode) and vice versa. The field in the VSAM record is defined in the mapping as BINARY, so no translation happens during transfer.

Recommendations

- Use a standard US English EBCDIC codepage (e.g. IBM-1047) and a standard US English ASCII codepage (e.g. IBM-850) in the configuration member for the VSE Connector Server (see skeleton SKVCSCFG in ICCF library 59). This codepages are used for translating textual resource names.

- Transfer any user data that contain DBCS data in binary mode only. Perform the codepage translation in your Java program. Java is capable of translating DBCS EBCDIC data into its ASCII or Unicode equivalent.

- For VSAM data: Define the fields containing textual DBCS data as BINARY instead of STRING and perform the codepage translation in your Java program.