HPCC Systems® HPCC4J Project

Boca Raton Documentation Team

HPCC Systems® HPCC4J Project

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com> Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2021 Version 8.0.44-1

HPCC4J Overview

Introduction to HPCC4J

The HPCC Systems for Java is a collection of Java based APIs and tools which help developers interact with HPCC Systems servers and tools in a relatively simple, and standardized fashion.

The project houses multiple HPCC Systems centric Java based APIs and tools.

The project is available on Github in the hpcc4j repository.

https://github.com/hpcc-systems/hpcc4j

wsclient

The API which standardizes and facilitates interaction with HPCC Systems Web based Services (ESP Web services).

The project is based on stub code generated from WSDL using Eclipse tools based on Apache Axis.

Provides a mechanism for actuating HPCC Systems web service methods.

clienttools

Java based interface to HPCC Systems client tools. Currently only interfaces with the ECLCC Server.

rdf2hpcc

RDF data ingestion tool to HPCC.Systems Based on Apache Jena and is dependent on wsclient.

dfsclient

Distributed data ingestion & extraction library. Uses internal HPCC Systems binaries to efficiently read and write data remotely in parallel. Supports generic and custom dataset creation and translation through *IRecordBuilder* & *IRecordAccessor* interfaces

commons-hpcc

Set of common use libraries used in conjunction with a wide array of HPCC Systems related java clients.

NOTE:

	As is common in Java client communication over TLS, HPCC4J based clients targeting an HPCC cluster over TLS will need to import the appropriate certificates to its local Java keystore. *One way to accomplish this is to use the keytool packaged with Java installations. Refer to the keytool documentation for usage.

Use Cases

This section provides examples that illustrate typical Java client and HPCC Systems® interaction.

wsclient

Example: User wants to submit and execute an ECL query from a Java client:

Use the **wsclient** package to connect to the target HPCC system.

```
//Fetch platform object based on connection settings
//Provide the connection type, http|https, the ecl watch ip, and port,
//your ESP username and password (if required)

Platform platform = Platform.get("http", "ip", 8010, "username", "password");
HPCCWSClient connector = platform.getHPCCWSClient();
```

Create a *WorkunitInfo* object with the ECL code and submit that object to the WECL workunit web service.

The *WorkunitInfo* object contains all the information needed by HPCC to compile and execute an ECL query correctly.

```
WorkunitInfo wu=new WorkunitInfo();!
wu.setECL("output('Hello World');"); // The ECL to execute.
wu.setCluster("mythor");             // This can be hardcoded to a known cluster,
                                     // or can be selected from
                                     // valid cluster names clusterGroups[0] (above)
```

This is just one way to submit ECL, you can also submit ECL, and receive the WUID, which can later be used to fetch results. The results (if successful) are returned as a List of Object Lists.

```
List<List<Object>> results = connector.submitECLandGetResultsList(wu);

//logic to analyze results would need to be implemented.
int currentrs = 1;

for (List<Object> list : results)
{
  Utils.print(System.out, "Resultset " + currentrs +":", false, true);
  for (Object object : list)
  {
    System.out.print("[ " + object.toString() +" ]");
  }
  currentrs++;
  System.out.println("");
}
```

The preceding example shows how simple it is to code for this interface. This template can be expanded to interact with most of the ESP web services and their methods.

This connector can be used to actuate various HPCC WebService methods. For example, the client can request a list of available Target cluster names.

```
List<String> clusters = connector.getAvailableTargetClusterNames();
```

or cluster groups

```
String[] clusterGroups = connector.getAvailableClusterGroups();
```

Which can then be used as one of the required parameters for other WS actions, such as spraying a file:

```
connector.sprayFlatHPCCFile("persons", "mythor::persons", 155, clusters.get(0), true);
```

DFSClient

Example: User wants to read file "example::dataset" in a parallel fashion from HPCC Systems into a Java client.

Reading Example:

The following example is for reading in parallel from

```
HPCCFile file = new HPCCFile("example::dataset", "http://127.0.0.1:8010" , "user", "pass");
DataPartition[] fileParts = file.getFileParts();
ArrayList<HPCCRecord> records = new ArrayList<HPCCRecord>();
for (int i = 0; i < fileParts.length; i++)
{
    HpccRemoteFileReader<HPCCRecord> fileReader = null;
    try
    {
       HPCCRecordBuilder recordBuilder = new
                         HPCCRecordBuilder(file.getProjectedRecordDefinition());
       fileReader = new HpccRemoteFileReader<HPCCRecord>(fileParts[i],
                    file.getRecordDefinition(), recordBuilder);
    }
    catch (Exception e) { }
    while (fileReader.hasNext())
    {
       HPCCRecord record = fileReader.next();
       records.add(record);
    }
    fileReader.close();
}
```

Writing Example:

Example: User wants to spray their dataset into an HPCC Systems logical file named "example::dataset.

```
FieldDef[] fieldDefs = new FieldDef[2];
fieldDefs[0] = new FieldDef("key", FieldType.INTEGER, "lNTEGER4", 4, true, false,
                            HpccSrcType.LITTLE_ENDIAN, new FieldDef[0]);
fieldDefs[1] = new FieldDef("value", FieldType.STRING, "STRING", 0, false, false,
                            HpccSrcType.UTF8, new FieldDef[0]);
FieldDef recordDef = new FieldDef("RootRecord", FieldType.RECORD, "rec", 4, false, false,
                                  HpccSrcType.LITTLE_ENDIAN, fieldDefs);

String eclRecordDefn = RecordDefinitionTranslator.toECLRecord(recordDef);

// See WSClient documentation on connection / construction of WSClient
Platform platform;
HPCCWsClient wsclient;

HPCCWsDFUClient dfuClient = wsclient.getWsDFUClient();
DFUCreateFileWrapper createResult = dfuClient.createFile("example::dataset", "mythor",
                                    eclRecordDefn, 300, false, DFUFileTypeWrapper.Flat, "");
DFUFilePartWrapper[] dfuFileParts = createResult.getFileParts();
DataPartition[] hpccPartitions = DataPartition.createPartitions(dfuFileParts,
         new NullRemapper(new RemapInfo(), createResult.getFileAccessInfo()),
dfuFileParts.length, createResult.getFileAccessInfoBlob());

//------------------------------------------------------------------------------
// Write partitions to file parts
//------------------------------------------------------------------------------

ArrayList<HPCCRecord> records = new ArrayList<HPCCRecord>();
```

```
int recordsPerPartition = records.size() / dfuFileParts.length;
int residualRecords = records.size() % dfuFileParts.length;

int recordCount = 0;
int bytesWritten = 0;
for (int partitionIndex = 0; partitionIndex < hpccPartitions.length; partitionIndex++)
{
   int numRecordsInPartition = recordsPerPartition;
   if (partitionIndex == dfuFileParts.length - 1)
   {
      numRecordsInPartition += residualRecords;
   }

   HPCCRecordAccessor recordAccessor = new HPCCRecordAccessor(recordDef);
   HPCCRemoteFileWriter<HPCCRecord> fileWriter = new
   HPCCRemoteFileWriter<HPCCRecord>(hpccPartitions[partitionIndex], recordDef,
         recordAccessor, CompressionAlgorithm.NONE);
   try
   {
         for (int j = 0; j < numRecordsInPartition; j++, recordCount++)
         {
             fileWriter.writeRecord(records.get(recordCount));
         }
         fileWriter.close();
         bytesWritten += fileWriter.getBytesWritten();
   }
   catch (Exception e)
   {
   }
}

//-----------------------------------------------------------------------------
// Publish and finalize the file
//-----------------------------------------------------------------------------

dfuClient.publishFile(createResult.getFileID(), eclRecordDefn, recordCount, bytesWritten, true);
```

Buids

To build the projects using Maven, navigate to the base directory of the project and issue the following command:

```
mvn install
```

NOTE: hpcccommons, wsclient, and dfsclient are controlled via the top-level maven pom file and can be built with a single command. All sub-projects can be built individually using the pom file in each sub-project directory.

For more information on how to use Maven see http://maven.apache.org

HPCC4J Source Code

The source can be found under the HPCC Platform github account in the hpcc4j repo.

https://github.com/hpcc-systems/hpcc4j

To utilize this library as a dependency in your own maven project, add the following definition to your pom.xml.

```
<dependency>
    <groupId>org.hpccsystems</groupId>
    <artifactId>wsclient</artifactId>
    <version>7.8.2-1</version>
</dependency>
```

Contributions to source are accepted and encouraged. All contributions must follow the JAVA source format described in the HPCC-JAVA-Formatter.xml file which can be found in hpcc4j/eclipse. This formatter can be used by the Eclipse IDE to automatically format JAVA source code.

- From eclipse, choose Window->Preferences->Java->Code Stype->Formatter.

  Import the HPCC-JAVA-Formatter.xml file and set it as the Active profile.

- From the JAVA editor, choose Source->Format