HPCC Systems® HPCC4J Project

Boca Raton Documentation Team



HPCC Systems® HPCC4J Project

Boca Raton Documentation Team
Copyright © 2026 HPCC Systems®. All rights reserved

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.

LexisNexis and the Knowledge Burst logo are registered trademarks of Reed Elsevier Properties Inc., used under license. Other products, logos, and services may be trademarks or registered trademarks of their respective companies. All names and example data used in this manual are fictitious. Any similarity to actual persons, living or dead, is purely coincidental.

2026 Version 10.8.14-1

HPCC4J Overview 4
Introduction to HPCC4J 4
Use Cases 5
Buids 8
Using HPCC4J with HPCC on a Kubernetes Cluster 9
Certificate Manager Setup 9
Configuring the HPCC Systems Certificates 10
Trusting Generated Certificates 11
Resolving Certificate Domain Name Locally 12
Testing 12
Using HPCC4J with HPCC on Bare Metal 13
Configuring Rowservice Signing Keys in HPCC Systems 13

© 2026 HPCC Systems®. All rights reserved

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.

© 2026 HPCC Systems®. All rights reserved

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:

© 2026 HPCC Systems®. All rights reserved

```
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
//------------------------------------------------------------------------
```

© 2026 HPCC Systems®. All rights reserved

```
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);
```

© 2026 HPCC Systems®. All rights reserved

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

© 2026 HPCC Systems®. All rights reserved

Using HPCC4J with HPCC on a Kubernetes Cluster

The following section is based on the HPCC Systems Helm documentation that can found here: https://github.com/hpcc-systems/HPCC-Platform/tree/master/helm/examples/certmanager

Certificate Manager Setup

During installation, the HPCC Systems Helm charts utilize a certificate manager to generate certificates for the cluster. These certificates are required to create secure connections to the cluster and must be configured in order to utilize HPCC4j. The following steps will setup a local certificate manager within Kubernetes:

Install JetStack Cert Manager

```
helm repo add jetstack https://charts.jetstack.io
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.crds.yaml
helm install cert-manager jetstack/cert-manager --version v1.11.0
```

Add Root Certificate Authority

Create a certificate request similar to the following example:

```
[req]
default_bits           = 2048
default_keyfile        = ca.key
distinguished_name     = dn
prompt                 = no
x509_extensions        = x509_ca

[dn]
C                      = YOUR_COUNTRY
ST                     = YOUR_STATE
L                      = YOUR_CITY
O                      = YOUR_ORGANIZATION
OU                     = YOUR_ORGANIZATION_UNIT
CN                     = Internal Cluster CA
emailAddress           = YOUR_SUPPORT_EMAIL

[x509_ca]
basicConstraints=CA:true,pathlen:1
```

Create the root certificate via OpenSSL and add it to a Kubernetes secret.

```
openssl req -x509 -newkey rsa:2048 -nodes -keyout ca.key -sha256 -days 1825 -out ca.crt -config ca-req.cfg
kubectl create secret tls hpcc-local-issuer-key-pair --cert=ca.crt --key=ca.key
kubectl create secret tls hpcc-signing-issuer-key-pair --cert=ca.crt --key=ca.key
```

© 2026 HPCC Systems®. All rights reserved

Configuring the HPCC Systems Certificates

Now that we have created a certificate authority, we need to configure HPCC to utilize the certificate manager / root certificate and enable the rowservice.

NOTE: The rowservice is an internal HPCC Systems service that HPCC4j depends on to read and write data to / from HPCC Systems clusters in a performant and secure manner.

We can change this configuration by creating and applying an override yaml file to override the default settings within the HPCC helm charts.

certificateValues.yaml:

```
certificates:
  enabled: true
dafilesrv:
  - name: rowservice
    disabled: false
    application: stream
    service:
      servicePort: 7600
      visibility: global
  - name: direct-access
    disabled: true
    application: directio
    service:
      servicePort: 7200
      visibility: local
  - name: spray-service
    application: spray
    service:
      servicePort: 7300
      visibility: cluster
```

Applying Helm Configuration Changes

Installing an HPCC cluster with configuration changes:

```
helm install myhpcc hpcc/hpcc --set global.image.version=latest -f certificateValues.yaml
```

These configuration changes can also be made after the HPCC cluster has been installed via helm upgrade:

```
helm upgrade -f certificateValues.yaml myhpcc hpcc/hpcc
```

NOTE: If you run into an issue where the HPCC Helm charts complain about the cert-manager missing make sure to apply the cert-manager.crds.yaml in the above Certificate Manager Setup step, and then verify cert-manager.io/v1 is listed in the output of `kubectl api-versions`

© 2026 HPCC Systems®. All rights reserved

Trusting Generated Certificates

The certificates that were created during the previous steps come from an unknown certificate authority (the local certificate authority we created) and are therefore not trusted by default. Since the certificates aren't trusted any attempt to connect to the cluster will fail with an error message indicating that the certificates aren't trusted and/or that building the PKIX path failed.

Example error message:

```
ERROR RowServiceOutputStream Exception occurred while attempting to connect to row service (localhost:7600):
  PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
    unable to find valid certification path to requested target

java.lang.Exception: Exception occured while attempting to connect to row service (localhost:7600):
  PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
    unable to find valid certification path to requested target
```

We can fix this issue by adding the certificates to the local trust store and adding an entry to our hosts file for the domain names associated with the certificates.

© 2026 HPCC Systems®. All rights reserved

Resolving Certificate Domain Name Locally

Certificates are attached to a particular domain name when created; by default the HPCC Helm charts will generate the certificates using the **eclwatch.default** domain name. However, your domain name server will not know that the eclwatch.default domain should point to your local IP address; So we will need to add an entry to your local host file so that eclwatch.default resolves correctly.

```
sudo -- sh -c -e "echo '127.0.0.1 eclwatch.default' >> /etc/hosts";
sudo -- sh -c -e "echo '127.0.0.1 rowservice.default' >> /etc/hosts";
sudo -- sh -c -e "echo '127.0.0.1 sql2ecl.default' >> /etc/hosts";
```

Adding Certificates to the Java Trust Store

Download TLS certificate and add it to the Java keystore.

NOTE: The path to the keystore below may need to be updated. As an example in some installations the path would instead be: `$JAVA_HOME/lib/security/cacerts`

```
openssl s_client -showcerts -connect eclwatch.default:8010 < /dev/null | openssl x509 -outform DER > cert.der
sudo keytool -import -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit -noprompt -alias eclwatch-tls -file cert.der
openssl s_client -showcerts -connect rowservice.default:7600 < /dev/null | openssl x509 -outform DER > cert.der
sudo keytool -import -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit -noprompt -alias dafilesrv-tls -file cert.der
openssl s_client -showcerts -connect sql2ecl.default:8510 < /dev/null | openssl x509 -outform DER > cert.der
sudo keytool -import -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit -noprompt -alias sqltoecl-tls -file cert.der
```

Testing

Your local cluster should now be available at https://eclwatch.default:8010, however you will likely need to tell your browser to trust the SSL certificates; as the above steps only created trust for Java applications.

The file utility within DFSClient can be used to test the certificate configuration; If you encounter a PKIX error when running the file utility command then you need to revisit the above steps.

```
java -cp dfsclient-jar-with-dependencies.jar org.hpccsystems.dfs.client.FileUtility -read existing::hpcc::file_to_read -url https://eclwatch.default:8010
```

© 2026 HPCC Systems®. All rights reserved

Using HPCC4J with HPCC on Bare Metal

Configuring Rowservice Signing Keys in HPCC Systems

This guide provides steps to generate and configure rowservice signing keys to allow for secure communication between an HPCC Systems cluster and external clients. These signing keys are required for authentication and secure reading and writing of data between HPCC4J clients and HPCC Systems clusters and must be properly configured on target HPCC Systems clusters.

Step 1: Generate Signing Keys

If signing keys do not already exist, they must be generated and placed in a directory that is accessible to the **hpcc** user. The default directory **/home/hpcc/certificate** will be used in the example configuration below, but on multi-node clusters it likely makes sense to change this directory.

Generate Signing Keys

```
sudo /opt/HPCCSystems/etc/init.d/setupPKI
```

This command generates a pair of signing keys:

- **Private Key:** `/home/hpcc/certificate/key.pem`
- **Public Key:** `/home/hpcc/certificate/public.key.pem`

Step 2: Configure Signing Keys in the HPCC Systems Environment

Once the keys are generated, they need to be referenced in the HPCC Systems environment.xml configuration file.

Add Keys Configuration Section

Modify the `environment.xml` file located at `/etc/HPCCSystems/environment.xml` to include the following under the `<EnvSettings>` node:

```
<EnvSettings>
  <Keys>
    <ClusterGroup keyPairName="mythor" name="thorcluster_1"/>
    <ClusterGroup keyPairName="mythor" name="thorcluster_2"/>
    <KeyPair name="mythor" privateKey="/home/hpcc/certificate/key.pem" publicKey="/home/hpcc/certificate/public.key.pem"/>
  </Keys>
```

Explanation:

- `<ClusterGroup>` entries define which Thor clusters will use the specified key pair.
- `<KeyPair>` defines the key pair used for signing and must reference the correct file paths.

© 2026 HPCC Systems®. All rights reserved

- Each Thor cluster in the HPCC system must have an associated <ClusterGroup> entry specifying the keyPairName.

Step 3: Synchronize Configuration and Keys Across the Cluster

After updating the configuration, ensure that all nodes within the cluster have the updated environment.xml file and the necessary key files.

Sync environment.xml to All Nodes

```
scp /etc/HPCCSystems/environment.xml hpccadmin@nodeX:/etc/HPCCSystems/environment.xml
```

Repeat this step for each node in the HPCC cluster.

Sync Signing Keys Across the Cluster

```
scp /home/hpcc/certificate/key.pem hpccadmin@nodeX:/home/hpcc/certificate/key.pem
scp /home/hpcc/certificate/public.key.pem hpccadmin@nodeX:/home/hpcc/certificate/public.key.pem
```

Step 4: Verify Key Synchronization

To ensure that the signing keys have been correctly synchronized across all nodes, compute the MD5 hash of the key files and compare them.

Check MD5 Hash of Keys

Run the following command on each node:

```
 md5sum /home/hpcc/certificate/key.pem /home/hpcc/certificate/public.key.pem
```

Compare the output across all nodes. If the MD5 hash values are identical, the keys have been correctly synchronized. If there are discrepancies, re-sync the keys and verify again.

Step 5: Restart HPCC Rowservice and ESP

Once the configuration and keys are updated across the cluster, the ESP and Dafilesrv services need to be restarted for these changes to take affect.

```
sudo /etc/init.d/dafilesrv restart
sudo /etc/init.d/hpcc-init -c myesp restart
```

© 2026 HPCC Systems®. All rights reserved

Testing Configuration

The above configuration can be tested by using the FileUtility in the HPCC4j dfsclient library to attempt to read a file from the configured HPCC Systems cluster. The latest copy of the dfsclient jar can be found here: https://mvnrepository.com/artifact/org.hpccsystems/dfsclient

The following command will attempt to read “example::hpccsystems::file” from “http://your_cluster:8010”, the should be updated to an already existing file on your target cluster and the URL of your HPCC Systems ESP respectively.

```
java
   -Dotel.service.name=DFSClient.FileUtility \
   -cp dfsclient-9.10.1-0-jar-with-dependencies.jar \
   org.hpccsystems.dfs.client.FileUtility \
   -read_test example::hpccsystems::file \
   -url http://your_cluster:8010
```

If the keys have been successfully configured, you will see a similar result to the following indicating the example file was successfully read.

```
[{
  "bytesWritten": 0, "Read Bandwidth":
  "6.70 MB/s", "Write Bandwidth": "0.00 MB/s", "warns": [],
  "recordsWritten": 0, "recordsRead": 6250000, "bytesRead": 100000000,
  "time": "14.92 s", "operation":
  "FileUtility.ReadTest_example::hpccsystems::file", "errors": [],
  "successful": true
}]
```

Conclusion

Following these steps ensures that HPCC Systems can securely authenticate HPCC4j clients and allow them to read and write data within the target HPCC systems cluster.

© 2026 HPCC Systems®. All rights reserved