

WsSQL ESP Web Service Guide

Boca Raton Documentation Team



WsSQL ESP Web Service Guide

Boca Raton Documentation Team

Copyright © 2021 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.

2021 Version 8.0.44-1

| | |
|---|----|
| Introduction | 4 |
| Supported File Types | 5 |
| Setup | 6 |
| Configuration | 6 |
| Using HPCC Systems Files as a data source | 9 |
| Index Annotations | 9 |
| Methods | 11 |
| Echo | 12 |
| GetDBSystemInfo | 13 |
| GetDBMetaData | 14 |
| ExecuteSQL | 17 |
| GetResults | 19 |
| PrepareSQL | 20 |
| ExecutePreparedSQL | 22 |
| CreateTableAndLoad | 24 |
| SetRelatedIndexes | 27 |
| GetRelatedIndexes | 28 |
| Common Structures | 29 |
| Supported SQL Grammar | 30 |
| CALL | 30 |
| SELECT | 31 |
| SELECT JOIN | 34 |
| CREATE / LOAD | 36 |
| Supported Aggregate Functions | 38 |

Introduction

The WsSQL Web service is an add-on service that provides an SQL interface into HPCC Systems®. The Web service accepts a subset of prepared and standard SQL queries. This provides access to most HPCC Systems data and published queries over HTTP or HTTPS using SOAP or REST interfaces.

The WsSQL service maps HPCC Systems logical files to RDBMS tables. HPCC Systems Published Queries are exposed as RDBMS Stored Procedures.

The WsSQL service also provides convenient methods to obtain system information, metadata, and results from previously run queries.

This service is intended to be used in a programmatic fashion (for example, via database drivers), but can also be used in an interactive fashion by users who are more comfortable using SQL than ECL. This makes it possible to submit ad-hoc queries without learning ECL.

The service exposes HPCC Systems logical files as RDB tables.

- HPCC Systems Logical File <-> RDB Table
- HPCC Systems Record Fields <-> RDB Table Columns
- HPCC Systems Published query <-> RDB Stored Procedure
- Provides HPCC Systems with system and data RDB metadata
- Supports subset of SQL syntax
- Non-transactional
- Provides means for utilizing HPCC Systems index files for faster reads.

Supported File Types

The WsSQL service supports all HPCC Systems file types except :

- XML
- Files with Nested Child Datasets
- Files without record layout in its metadata.

The WsSQL service **only supports files which contain the record definition in the logical file's metadata.** Sprayed files do not contain this metadata. This metadata exists on any file or index which is written to the HPCC Systems Distributed File System (DFS). Sprayed data files typically undergo some processing and an OUTPUT of the transformed data to disk before use, so this should not interfere with the service's usefulness. You can use the *CreateTableAndLoad* method to produce a usable file from a sprayed file or one on a landing zone. See *CreateTableAndLoad* for details.

Setup

The WsSQL service is included in the HPCC Systems platform.

Prior to version 7.0, WsSQL was an add-on product and was installed separately. If you have a system configured with WsSQL prior to version 7.0, you should uninstall WsSQL before upgrading the platform.

Configuration

We recommend taking the time to read this manual in its entirety; however, the following is a quick start summary of steps.

Configure Using Wizard

1. Once Configuration Manager is running:
2. Create a new environment using the wizard.

The WsSQL service is automatically added and its service binding is created in the ESP Server's configuration.

3. Save the environment file.
4. Copy your environment.xml file to all servers.

```
# for example
sudo -u hpcc cp /etc/HPCCSystems/source/NewEnvironment.xml /etc/HPCCSystems/environment.xml
```

5. Exit Configuration manager.
6. Restart the system.
7. Access the WsSQL interface in your browser (port 8510).
8. Run the Echo method to confirm connectivity. See [Echo].

Configure Manually

This method is useful when adding the WsSQL service to an existing system that did not have a service configured or to add an additional WsSql service.

1. Once Configuration Manager is running:
2. Open an environment file (*.xml) in Advanced Mode.

If you are adding WsSQL to an existing system, open an environment file that matches the live environment.xml. It is highlighted.

3. Check the Write Access box.
4. Right-click on the **Software** portion of the tree in the left panel, and select **New ESP Service > ws_sql**.



This adds a definition for the service.

5. Select the **ESP** component, then select the ESP Service Bindings tab.



6. Provide the name of the service to bind and the port. (Default port is 8510)
7. Save the environment file.
8. Copy your environment.xml file to all servers

```
# for example
sudo -u hpcc cp /etc/HPCCSystems/source/NewEnvironment.xml /etc/HPCCSystems/environment.xml
```

9. Exit Configuration manager.
10. Restart the system.
11. Access the WsSQL interface in your browser:

Using `http://nnn.nnn.nnn.nnn:pppp` (where `nnn.nnn.nnn.nnn` is your ESP Server's IP Address and `pppp` is the WsSQL service port. The default port is 8510)

12.Run the Echo method to confirm connectivity. See [Echo].

Using HPCC Systems Files as a data source

Once installed and configured, the WsSQL service will process submitted SQL statements and generate dynamic ECL code. The code is submitted to and executed by your HPCC Systems platform. The result set is returned to your application.

Note: The WsSQL service **only supports files which contain the record definition in the logical file's metadata.** Sprayed files do not contain this metadata. This metadata exists on any file or index which is written to the HPCC Systems Distributed File System (DFS). Sprayed data files typically undergo some processing and an OUTPUT of the transformed data to disk before use, so this should not interfere with the service's usefulness. You can use the *CreateTableAndLoad* method to produce a usable file from a sprayed file or one on a landing zone. See CreateTableAndLoad for details.

In addition, you can utilize indexes on the HPCC Systems in one of two ways:

1. Provide SQL hints to tell service to use a specific index for your query.

For example:

```
USEINDEX(TutorialPersonByZipIndex)
```

2. Specify the related indexes in the HPCC Systems logical file description.

Index Annotations

The WsSQL service attempts to perform index based reads whenever possible. However, in order to take advantage of index reads, the target HPCC Systems files need to be annotated with the pertinent index file names. This is accomplished by adding the following key/value entry on the file's description using ECL Watch.

From a logical file's details page, enter the information in the Description entry box, then press the **Save Description** button.

This information is used by the service to decide if an index fetch is possible for a query on the base file.

On source file:

XDBC:RelIndexes= [*fullLogicalFilename1*; *fullLogicalFilename2*]

Example:

```
XDBC:RelIndexes=[tutorial::yn::peoplebyzipindex;  
                 tutorial::yn::peoplebyzipindex2;  
                 tutorial::yn::peoplebyzipindex3]
```

In this example, the source file has three indexes available.

You can add annotations using the SetRelatedIndexes method.

On the index file:

XDBC:PosField=[*indexPositionFieldName*]

Example:

```
XDBC:PosField=[fpos]
```

The FilePosition field (fpos) can have any name, so it must be specified in the metadata so the service knows which field is the fileposition.

Simply enter the information in the description entry box, then press the **Save Description** button.

Note: You should enter this information BEFORE publishing any query using the data file or indexes. Published queries lock the file and would prevent editing the metadata.

Methods

These methods are included in the WsSQL service interface:

- **Echo**

This method is provided to test end-to-end communication. The Input string is echoed in response.

- **GetDBSystemInfo**

You can use this method to gather version information from the HPCC Systems platform.

- **GetMetadata**

This method provides metadata you can use to create a view or model of the target HPCC Systems platform as a SQL accessible DB. You can request Tables, Stored Procedures, and/or Targetclusters.

- **ExecuteSQL**

Use this method to submit standard (non-prepared) SQL queries. This method accepts free-hand SQL text (see supported grammar below).

- **GetRelatedIndexes**

This method retrieves information from a logical file's description about related indexes.

- **GetResults**

This method provides results from previously executed queries.

- **PrepareSQL**

This method provides the ability to submit a free-hand SQL request for later use as a parameterized query. It compiles the query and returns the Workunit ID (WUID). This WUID is later used to execute the query with provided input parameters using the ExecutePreparedSQL method. This is ideal for queries which are executed many times with different values.

- **ExecutePreparedSQL**

This method executes a previously created parameterized SQL query. The query is referenced using a Workunit ID (WUID), which is returned from the PrepareSQL method. The caller can specify sequence of input parameters as key-value pairs, which are bound to the precompiled query.

- **CreateTableAndLoad**

This method reads a logical file or a file on a landing zone and creates a new table and loads the data from the source file creating a table (logical file) that WsSQL can use. This function is intended to use when you have a logical file without metadata or to import a new file into an HPCC Systems platform.

- **SetRelatedIndexes**

This method adds information to a logical file's description that WsSQL uses as an annotation of a related index.

Echo

This function takes an input string and "echoes" the value in its result.

This function is intended for end-to-end connectivity testing. A successful response indicates a good connection to the server hosting the Ws-SQL Web service. This function is designed for connectivity testing.

We recommend using this function as a first step of the application development process.

Sample Input XML

```
<soap:Envelope>
  <soap:Body>
    <EchoRequest>
      <Request>StringToEcho</Request>
    </EchoRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|-------------|------|----------------------------------|
| EchoRequest | N | Structure containing the request |
| Request | N | String to echo in result |

Response Tag Descriptions

| Tag Name | Description |
|--------------|-------------------------------|
| EchoResponse | Structure containing response |
| Response | Response |

GetDBSystemInfo

This method allows you to get HPCC System version information.

Sample Input XML

```
<soap:Envelope>
  <soap:Body>
    <GetDBSystemInfoRequest>
      <IncludeAll>1</IncludeAll>
    </GetDBSystemInfoRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|------------------------|------|--|
| GetDBSystemInfoRequest | Y | Structure containing the request |
| IncludeAll | N | If set to 1 or true, all available information is returned |

Response Tag Descriptions

| Tag Name | Description |
|-------------------------|--|
| GetDBSystemInfoResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| Name | Name. |
| FullVersion | Full Version of the HPCC Systems platform |
| Major | Major version of the HPCC Systems platform |
| Minor | Minor version of the HPCC Systems platform |
| Point | Point of HPCC Systems platform |
| Project | Project of HPCC Systems platform |
| Maturity | Maturity of HPCC Systems platform |
| WsSQLFullVersion | Full Version of WsSQL |
| WsSQLMajor | Major of WsSQL |
| WsSQLMinor | Minor of WsSQL |
| WsSQLPoint | Point of WsSQL |
| WsSQLProject | Project of WsSQL |
| WsSQLMaturity | Maturity of WsSQL |

GetDBMetaData

This methods allows you to query the HPCC Systems platform and get metadata to use to create a view or model of the target HPCC Systems as a SQL accessible DB.

You can request one or more of the following:

- Tables (Logical files in the HPCC Systems Cluster)
- Stored Procedures (Published Queries)
- TargetClusters

Sample Input XML

```
<soap:Envelope>
  <soap:Body>
    <GetDBMetaDataRequest>
      <IncludeTables>1</IncludeTables>
      <TableFilter/>
      <IncludeStoredProcedures>1</IncludeStoredProcedures>
      <QuerySet/>
      <IncludeTargetClusters>1</IncludeTargetClusters>
      <ClusterType/>
    </GetDBMetaDataRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|-------------------------|------|---|
| GetDBMetaDataRequest | Y | Structure containing the request |
| IncludeTables | N | If set to 1 or true, available tables are included in response |
| TableFilter | N | Filter for table results |
| IncludeStoredProcedures | N | If set to 1 or true, available Stored Procedures are included in response |
| QuerySet | N | QuerySet to use as filter for Stored procedures to return |
| IncludeTargetClusters | N | If set to 1 or true, available Target Clusters are included in response |
| ClusterType | N | Cluster type to use as filter |

Response Tag Descriptions

| Tag Name | Description |
|-----------------------|--|
| GetDBMetaDataResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| TableCount | Count of available tables |
| Tables | Structure containing one or more tables |
| Table | Structure containing one table |
| Name | Table name |
| Columns | Structure containing one or more columns |
| Column | Column |
| Name | Column name |
| Type | Column data type (e.g., unsigned8, string3) |
| ECL | ECL Definition for the table |
| Format | Table format (e.g., FLAT, KEYED, etc) |
| ContentType | Content Type |
| Description | Description |
| IsKeyed | Boolean indicator: Is this an index? |
| IsSuper | Boolean indicator: Is this a superfile? |
| CsvQuote | Quote character (only valid for CSV files) |
| CsvSeparate | Separator character (only valid for CSV files) |
| CsvTerminate | Record terminator character (only valid for CSV files) |
| Group | Group |
| MaxRecordSize | Maximum record size |
| Modified | Date modified |
| NumParts | Number of file parts |
| Owner | Owner name |
| QuerySets | Structure containing one or more Query Sets |
| QuerySet | Structure containing one Query Set |
| Name | Query Name |
| QuerySetQueries | Structure containing one or more QuerySetQueries |
| QuerySetQuery | Structure containing one QuerySetQuery |
| Name | Query Name |
| Id | Query ID (for internal use and informational purposes only) |
| Wuid | Workunit ID |
| Suspended | Boolean indicator: Is the query suspended? |
| Signature | Query Signature |
| InParams | Structure containing one or more Input parameters |

WsSQL ESP Web Service Guide
Methods

| | |
|-----------------|---|
| InParam | Structure containing one Input parameter |
| Name | Parameter name |
| Type | Parameter data type (e.g., string) |
| ResultSets | Structure containing one or more Result Sets (Only the first result set is returned from a WsSQL service request) |
| ResultSet | Structure containing one Result Set |
| Name | Result set name |
| OutParams | Structure containing one or more output parameters |
| OutParam | Structure containing one output parameter |
| Name | Parameter name |
| Type | Parameter data type (e.g., string) |
| QuerySetAliases | Structure containing one or more QuerySetAliases |
| QuerySetAlias | Structure containing one QuerySetAlias |
| Name | Query name |
| Id | Query Id |
| ClusterNames | Structure containing one or more TargetClusters |
| ClusterName | Structure containing one TargetCluster |

ExecuteSQL

Use this method to submit standard (non-prepared) SQL queries. The **SqlText** input tag accepts free hand SQL text (see Supported SQL grammar below).

If you are executing SQL using a SELECT or SELECT JOIN, you must specify the **TargetCluster**.

If you are executing SQL that uses a CALL to a stored procedure, you must either fully qualify the procedure name (For example: Roxie.FindPeopleByZip) or specify the **TargetQuerySet** here. Parameters must be passed in order, not by name. You can retrieve the order using GetDBMetaData.

To exclude the result schema in the result, set the **SuppressXmlSchema** option to 1 or true.

For result set paging, you can limit the total query results and the initial page returned (**ResultWindowStart**, **ResultWindowCount**).

Sample Input XML

```
<soap:Envelope>
  <ExecuteSQLRequest>
    <SqlText>SELECT * from tutorial::yn::tutorialperson where lastname='JONES'</SqlText>
    <UserName>Emily</UserName>
    <!-- Use either TargetCluster or TargetQuerySet, not both -->
    <!-- If stored procedure was fully qualified, you can omit TargetQuerySet -->
    <TargetCluster>thor</TargetCluster>
    <TargetQuerySet></TargetQuerySet>
    <SuppressResults>1</SuppressResults>
    <SuppressXmlSchema>0</SuppressXmlSchema>
    <resultLimit>100</resultLimit>
    <!-- For page loading -->
    <ResultWindowStart>0</ResultWindowStart>
    <ResultWindowCount>50</ResultWindowCount>
  </ExecuteSQLRequest>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|-------------------|------|--|
| ExecuteSQLRequest | Y | Structure containing the request |
| SqlText | Y | Free-hand SQL text (see Supported SQL grammar below) |
| UserName | N | User Name to pass to HPCC Systems. This is used as the job owner name in HPCC Systems |
| TargetCluster | Y * | If you are executing prepared SQL using a SELECT or SELECT JOIN, you must specify the TargetCluster. |
| TargetQuerySet | Y * | If you are executing prepared SQL that uses a CALL to a stored procedure, you must either fully qualify the procedure name in the prepared SQL (For example: Roxie.FindPeopleByZip) or specify the TargetCluster here. |
| SuppressResults | N | If set to 1 or true, query results are not included in response |
| SuppressXmlSchema | N | If set to 1 or true, the query result schema is not included in response |
| Wait | N | Timeout value in milliseconds. Use -1 for no timeout |
| resultLimit | N | If set, the results can contain as many records as this limit |
| ResultWindowStart | N | For use with page-loading, the starting record to return |
| ResultWindowCount | N | For use with page-loading, the number of records to include from the ResultWindowStart |

* One or the other is required

Response Tag Descriptions

| Tag Name | Description |
|--------------------|--|
| ExecuteSQLResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| ParentWuId | The Workunit Id for the parent workunit (the one with the prepared SQL) |
| Result | The query result |
| Workunit | Structure containing Workunit details. See Workunit Structure in Common Structures |
| resultLimit | The submitted resultLimit |
| ResultWindowStart | The submitted ResultWindowStart |
| ResultWindowCount | The submitted ResultWindowCount |

GetResults

This method allows you to get results from previously executed queries. Use the **Wuid** returned from **ExecuteSQL** or **PrepareSQL**. This method is ideal for results paging.

Sample Input XML

```
<soap:Envelope>
  <soap:Body>
    <GetResultsRequest>
      <Wuid>W20140529-161930</Wuid>
      <SuppressXmlSchema>1</SuppressXmlSchema>
      <ResultWindowStart>0</ResultWindowStart>
      <ResultWindowCount>0</ResultWindowCount>
    </GetResultsRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|-------------------|------|--|
| GetResultsRequest | Y | Structure containing the request |
| Wuid | Y | Workunit Id (WUID) |
| SuppressXmlSchema | N | If set to 1 or true, the query result schema is not included in response |
| ResultWindowStart | N | For use with page-loading, the starting record to return |
| ResultWindowCount | N | For use with page-loading, the number of records to include from the ResultWindowStart |

Response Tag Descriptions

| Tag Name | Description |
|--------------------|--|
| GetResultsResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| Result | The query result |
| ResultWindowStart | The submitted starting record |
| ResultWindowCount | The submitted record count |
| Workunit | Structure containing Workunit details. See Workunit Structure in Common Structures |

PrepareSQL

Use this method to submit a free-hand SQL request for later use as a parameterized query. This compiles the query and returns the **Wuid**. This **Wuid** is later used to execute the query with provided input parameters using the **ExecutePreparedSQL** method.

This is ideal for queries which are executed many times with different values.

The prepared SQL can contain any supported SQL grammar.

If you are executing SQL using a SELECT or SELECT JOIN, you must specify the **TargetCluster**.

If you using a CALL to a stored procedure, you must either fully qualify the procedure name in the SQL (For example: Roxie.FindPeopleByZip) or specify the **TargetQuerySet** here. Parameters must be passed in order, not by name. You can retrieve the order using GetDBMetaData.

Create parameters using a ? as a placeholder.

Example:

```
select * from tutorial::yn::tutorialperson where lastname=?
```

Later you would submit a request to **ExecutePreparedSQL** providing a value to use for *lastname* as shown in the following example:

```
<soap:Envelope >
  <soap:Body>
    <ExecutePreparedSQLRequest>
      <Wuid>w20140724-135811</Wuid>
      <TargetCluster>thor</TargetCluster>
      <SuppressResults>0</SuppressResults>
      <SuppressXmlSchema>1</SuppressXmlSchema>
      <resultLimit>100</resultLimit>
      <Variables>
        <NamedValue>
          <Name>lastname</Name>
          <Value>JONES</Value>
        </NamedValue>
      </Variables>
    </ExecutePreparedSQLRequest>
  </soap:Body>
</soap:Envelope>
```

Sample Input XML

```
<soap:Envelope >
  <soap:Body>
    <PrepareSQLRequest>
      <SqlText>select * from tutorial::yn::tutorialperson where lastname=?</SqlText>
      <!-- Use either TargetCluster or TargetQuerySet, not both -->
      <!-- If stored procedure is fully qualified, you can omit TargetQuerySet -->
      <TargetCluster>thor</TargetCluster>
    </PrepareSQLRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|-------------------|------|---|
| PrepareSQLRequest | N | Structure containing request |
| SqlText | Y | Free-hand SQL text (see Supported SQL grammar below) |
| TargetCluster | Y * | If you are executing prepared SQL using a SELECT or SELECT JOIN, you must specify the TargetCluster |
| TargetQuerySet | Y * | If you are executing prepared SQL that uses a CALL to a stored procedure, you must either fully qualify the procedure name in the prepared SQL (For example: Roxie.FindPeopleByZip) or specify the TargetCluster here |
| Wait | N | Timeout value in milliseconds. Use -1 for no timeout |

** One or the other is required*

Response Tag Descriptions

| Tag Name | Description |
|--------------------|--|
| PrepareSQLResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| Workunit | Structure containing Workunit details. See Workunit Structure in Common Structures |
| Result | Structure containing result (if any) |

ExecutePreparedSQL

This method executes a previously created parameterized SQL query.

The target compiled query is referenced using a Workunit ID (**Wuid**), which is returned from the **PrepareSQL** method. The caller can specify sequence of input parameters as key-value pairs, which are bound to the precompiled query.

The prepared SQL can contain any supported SQL grammar.

If you are executing prepared SQL using a SELECT or SELECT JOIN, you can supply a **TargetCluster** to override the one specified when you submitted the PrepareSQL request; however, it must be a cluster of the same type.

If you want to limit the number of results, you must use a LIMIT clause in you SQL query.

For result set paging, you can limit the total query results and the initial page returned (**ResultWindowStart**, **ResultWindowCount**).

Sample Input XML

```
<soap:Envelope>
  <soap:Body>
    <ExecutePreparedSQLRequest>
      <WuId>W20140529-161930</WuId>
      <UserName>EmilyKate</UserName>
      <!-- You can override the TargetCluster used in original PrepareSQL query, -->
      <!-- but it must be of the same type -->
      <TargetCluster>Thor</TargetCluster>
      <SuppressResults>0</SuppressResults>
      <SuppressXmlSchema>1</SuppressXmlSchema>
      <Wait>-1</Wait>
      <!-- For page loading -->
      <ResultWindowStart>0</ResultWindowStart>
      <ResultWindowCount>50</ResultWindowCount>
      <!-- Paramaters using name/value pairs -->
      <Variables>
        <NamedValue>
          <Name>firstname</Name>
          <Value>Jim</Value>
        </NamedValue>
        <NamedValue>
          <Name>lastname</Name>
          <Value>JONES</Value>
        </NamedValue>
      </Variables>
    </ExecutePreparedSQLRequest>
  </soap:Body>
</soap:Envelope>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|---------------------------|------|--|
| ExecutePreparedSQLRequest | N | Structure containing the request |
| WuId | Y | The Workunit ID (WUID) |
| UserName | N | The username to use as the job's ownname in the HPCC Systems platform |
| TargetCluster | Y | If you are executing prepared SQL using a SELECT or SELECT JOIN, you can specify the TargetCluster, but it must be the same type as the one on which it was prepared |
| SuppressResults | N | If set to 1 or true, query results are not included in response |
| SuppressXmlSchema | N | If set to 1 or true, the query result schema is not included in response |
| Wait | N | Timeout value in milliseconds. Use -1 for no timeout. |
| ResultWindowStart | N | For use with page-loading, the starting record to return |
| ResultWindowCount | N | For use with page-loading, the number of records to include from the ResultWindowStart |
| Variables | N | If your prepared SQL has parameters, supply them as name/value pairs inside this structure |
| NamedValue | N | A structure containing one Name/Value pair |
| Name | N | Name |
| Value | N | Value |

Response Tag Descriptions

| Tag Name | Description |
|----------------------------|--|
| ExecutePreparedSQLResponse | Structure containing response |
| Exceptions | Structure containing exceptions. See Exceptions Structure in Common Structures |
| ParentWuId | The Workunit ID for the parent workunit (the one with the prepared SQL) |
| Result | The query result |
| ResultWindowStart | The submitted starting record |
| ResultWindowCount | The submitted record count |
| Workunit | Structure containing Workunit details. See Workunit Structure in Common Structures |

CreateTableAndLoad

This method creates a table (HPCC Systems logical file) which can subsequently be accessed using WsSQL. The table is assigned the layout as specified in the request ECLFields section. The table is then populated with data from an existing file. The source data file can either reside on the HPCC Systems Landing Zone after being uploaded, or it can already have been sprayed on to the system.

Sample Input XML

```
<CreateTableAndLoadRequest>
  <TableName>JIM:MyNewTable</TableName>
  <TableDescription>My Description of the new table</TableDescription>
  <Overwrite>1</Overwrite>
  <EclFields>
    <EclField>
      <FieldName>PersonName</FieldName>
      <EclFieldType>
        <!-- Valid types are: BOOLEAN, INTEGER, UNSIGNED, REAL, DECIMAL, STRING, -->
        <!-- QSTRING, UNICODE, DATA, VARSTRING, or VARUNICODE -->
        <Type>STRING</Type>
        <Locale></Locale>
        <Length>20</Length>
        <Precision></Precision>
      </EclFieldType>
    </EclField>
    <EclField>
      <FieldName>PersonID</FieldName>
      <EclFieldType>
        <Type>INTEGER</Type>
        <Locale></Locale>
        <Length>2</Length>
        <Precision></Precision>
      </EclFieldType>
    </EclField>
  </EclFields>
  <TargetCluster>thor</TargetCluster>
  <Owner>Jimmy</Owner>
  <DataSource>
    <!-- Use either SprayedFileName -->
    <SpreayedFileName>Tutorial::JD::OriginalPerson</SpreayedFileName>
    <!-- or full Landing Zone details, not both -->
    <LandingZoneIP>127.0.0.1</LandingZoneIP>
    <LandingZonePath>/var/lib/HPCCSystems/mydropzone</LandingZonePath>
    <LandingZoneFileName>OriginalPerson</LandingZoneFileName>
  </DataSource>
  <DataSourceType>
    <!-- Valid types are: FLAT, CSV, JSON, or XML -->
    <Type>FLAT</Type>
    <Params>
      <Param>
        <Name></Name>
        <Values>
          <Value></Value>
          <Value></Value>
        </Values>
      </Param>
    </Params>
  </DataSourceType>
  <Wait>-1</Wait>
</CreateTableAndLoadRequest>
```


Request Tag Descriptions

| Tag Name | Req? | Description |
|---------------------|------|--|
| TableName | Y | The name of the table to create in the HPCC Systems platform's distributed file system. |
| TableDescription | N | Your description of the file. |
| Overwrite | N | Boolean indicator to specify whether to allow the new file to overwrite an existing file of the same name. |
| EclFields | Y | Structure containing details for the file's record layout. This must match layout of the source data file |
| EclField | Y | Structure containing details for one field |
| FieldName | Y | Field Name |
| EclFieldType | Y | Structure containing Data Type details |
| Type | Y | Field type. Valid types are: BOOLEAN, INTEGER, UNSIGNED, REAL, DECIMAL, STRING, QSTRING, UNICODE, DATA, VARSTRING, or VARUNICODE |
| Locale | N | Locale |
| Length | Y* | Field length |
| Precision | N | Field precision (decimal places) |
| TargetCluster | Y | The Target cluster where the job will run. |
| Owner | N | Owner name for the file |
| DataSource | Y | Structure containing Data Source details. Either SprayedFileName or LandingZone File details must be provided, not both. |
| SprayedFileName | Y** | Logical filename of sprayed data file |
| LandingZoneIP | Y** | IP address or hostname of the HPCC Systems Landing Zone |
| LandingZonePath | Y** | Path to the datafile on the Landing Zone |
| LandingZoneFileName | Y** | Physical filename |
| DataSourceType | N | Structure containing DataSource Type details. |
| Type | Y | Valid types are: FLAT, CSV, JSON, or XML |
| Params | N | Structure containing one or more type parameters that describe the DataSource. For example, CSV Terminator = \n |
| Param | N | Structure containing one type parameter. |
| Name | N | Name |
| Values | N | Structure containing one or more values. |
| Value | N | Value |
| Wait | N | Timeout value in milliseconds. Use -1 for no timeout |

* Field length is only required for fixed width fields (FLAT files)

** Either SprayedFileName or LandingZone File details must be provided, not both

Response Tag Descriptions

The response structure contains many tags which provide useful information for development and debugging of ECL code. Most of them are outside of the scope of this document. Only those that are useful when using the WsSQL service are listed below.

| Tag Name | Description |
|----------------------------|--|
| CreateTableAndLoadResponse | Structure containing response |
| TableName | Name of the created table |
| Success | Boolean indicator of success [1 = success] |
| EclRecordDefinition | Supplied record definition in ECL format |
| Workunit | Structure containing Workunit details. See Workunit Structure in Common Structures |

SetRelatedIndexes

This function adds a description to a logical file to be used as an annotation indicating an index file which is related to a data file. This makes it available to WsSQL for use in an indexed fetch.

Sample Input XML

```
<SetRelatedIndexesRequest>
  <RelatedIndexSets>
    <RelatedIndexSet>
      <FileName>Tutorial::JD::TutorialPerson</FileName>
      <Indexes>
        <Index>Tutorial::JD::TutorialPersonByName</Index>
        <Index>Tutorial::JD::TutorialPersonByZIP</Index>
      </Indexes>
    </RelatedIndexSet>
  </RelatedIndexSets>
</SetRelatedIndexesRequest>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|--------------------------|------|---|
| SetRelatedIndexesRequest | | Structure containing request |
| RelatedIndexSets | Y | Structure containing one or more related index sets |
| RelatedIndexSet | Y | Structure containing one related index set |
| FileName | Y | Logical filename to which the annotation is added |
| Indexes | Y | Structure containing one or more indexes to add to annotation |
| Index | Y | Index to add to annotation |

Response Tag Descriptions

| Tag Name | Description |
|---------------------------|--|
| SetRelatedIndexesResponse | Structure containing response |
| RelatedIndexSets | Structure containing one or more related index sets |
| RelatedIndexSet | Structure containing one related index set |
| FileName | Logical filename to which the annotation was added |
| Indexes | Structure containing one or more indexes added to annotation |
| Index | Index added to annotation |

GetRelatedIndexes

This function retrieves information from logical file descriptions about annotations indicating an index file which is related to a data file.

Sample Input XML

```
<GetRelatedIndexesRequest>
  <FileNames>
    <FileName>Tutorial::JD::TutorialPerson</FileName>
  </FileNames>
</GetRelatedIndexesRequest>
```

Request Tag Descriptions

| Tag Name | Req? | Description |
|--------------------------|------|---|
| SetRelatedIndexesRequest | | Structure containing request |
| FileNames | Y | Structure containing one or more related index sets |
| FileName | Y | Logical filename to which the annotation is added |

Response Tag Descriptions

| Tag Name | Description |
|---------------------------|--|
| GetRelatedIndexesResponse | Structure containing response |
| RelatedIndexSets | Structure containing one or more related index sets |
| RelatedIndexSet | Structure containing one related index set |
| FileName | Logical filename to which the annotation was added |
| Indexes | Structure containing one or more indexes added to annotation |
| Index | Index added to annotation |

Common Structures

These structures are returned in several methods.

Exceptions Structure

| Tag Name | Description |
|------------|---|
| Exceptions | Structure containing one or more exceptions |
| Exception | Structure containing one exception |
| Code | Code |
| Audience | Audience |
| Source | Source component |
| Message | Error message |

Workunit Structure

The workunit structure contains many tags which provide useful information for development and debugging of ECL code. Most of them are outside of the scope of this document. Only those that are useful when using the WsSQL service are listed here.

| Tag Name | Description |
|------------------|---|
| Workunit | Structure containing Workunit details |
| Wuid | Workunit Id (WUID) |
| Owner | Job Owner (if any) |
| Cluster | Target Cluster |
| Jobname | Job Name (If any) |
| StateID | State ID |
| State | State (compiled, failed, etc) |
| Protected | Boolean indicator: Is Workunit protected? |
| Snapshot | Snapshot |
| IsPausing | Boolean indicator of pause state |
| ThorLCR | Boolean indicator |
| EventSchedule | Boolean indicator |
| TotalClusterTime | Time taken to process on cluster |
| Query | Query |
| Result | Structure containing result (if any) |

Supported SQL Grammar

CALL

Call *queryName* ([*paramList*])

| | |
|-----------|---|
| queryName | The published query name or alias |
| paramList | The parameters exposed by the published query (comma-separated) |

Call executes a published ECL query as if it were a stored procedure.

Example:

```
Call SearchPeopleByZipService ('33024')
```

SELECT

select [distinct] *columnList* **from** *tableList* [USE INDEX(*indexFileName* | NONE)]

[**where** *logicalExpression*] [**group by** *columnList*¹] [**having** *logicalExpression*²]

[**order by** *columnList*¹ [asc | desc]] [**LIMIT** *limitNumber*]

NOTE: Identifiers can be unquoted or within double quotes, literal string values must be single quoted.

| | |
|--------------------------|--|
| <i>columnList</i> | columnreference1[,columnreference2,columnreference3,...,columnreferencen] |
| | The column(s) to return (comma-separated list). In addition, these aggregate functions are supported : COUNT, SUM, MIN, MAX, and AVG. These work in a similar manner as their ECL counterparts |
| columnreference | [tablename.]columnname[[AS] alias] |
| <i>distinct</i> | [distinct] col1, col2,... coln |
| | The result set will only contain distinct (unique) values |
| <i>tableList</i> | tableref1[,tableref2,tableref3,...,tableref ⁿ] |
| | One or more tables, separated by commas. |
| | NOTE: A table list with multiple tables creates an (one or more) implicit inner join using the where clause logical expression as the join condition which must contain an equality condition |
| tableref | tableName[[AS] alias] |
| | The Name of the table as referenced, optionally defining its alias |
| <i>alias</i> | The alias used to refer to the corresponding table or field reference. |
| <i>logicalExpression</i> | Logical expression based on standard SQL filtering syntax. |
| | BOOLEAN Only supports <i>True</i> or <i>False</i> , do not use Y, N, 0, or 1 |
| | Valid operators: |
| | = Equal (e.g., age=33) |
| | <> Not equal (e.g., age <>33) |
| | > Greater than (e.g., age >55) |
| | < Less than (e.g., age < 18) |
| | >= Greater than or equal (e.g., age >=21) |
| | <= Less than or equal (e.g., age <=21) |
| | IN(value1,value2,...,valuen) where values are comma separated homogeneous types |
| | NOT IN(value1,value2,...,valuen) where values are comma separated homogeneous types |
| | LIKE <i>pattern</i> where the pattern uses SQL LIKE operators with % and _ wildcards. |
| | NOT LIKE <i>pattern</i> where the pattern uses SQL LIKE operators with % and _ wildcards. |
| <i>limitNumber</i> | The number of rows to return. This overrides the default configuration attribute (EclResultLimit) but cannot be set to ALL |

¹Aliasing not supported

²Can only contain references to aggregate functions if used with *having* clause.

Aggregate functions can only be expressed in logicalExpressions by using *Group by* and *having*

Examples:

```
Select * from tableList where Sum(F1 > 100) /* is NOT SUPPORTED */
Select * from tableList Group by F1 Having Sum (F1 > 100) /* IS SUPPORTED */
```

Example:

```
Select fname, lname, state from TutorialPerson where
    state='FL' OR (lname='Smith' and fname='Joe')
//returns data that looks like this:
John Doe FL
Jim Smith FL
Jane Row FL
Joe Smith CA

Select fname, lname, state from TutorialPerson where state='FL' AND lname <> 'Smith'
//returns data that looks like this:
John Doe FL
Jane Row FL

Select fname, lname, state from TutorialPerson where state='FL' AND lname like 'Smi%'
//returns data that looks like this:
Jim Smith FL

Select fname, lname, state from TutorialPerson where
    state='FL' OR (lname='Smith' and fname='Joe') AND fname NOT LIKE 'Ji%'
//returns data that looks like this:
John Doe FL
Jane Row FL
Joe Smith CA
```

The interface supports SQL index hints, which gives the SQL user the option to specify the most appropriate HPCC Systems index for the current SQL query. This also allows you to disable the use of an index.

select *columnList* **from** *tableName* **USE INDEX**(*hpcc::index::file::name*) **where** *logicalExprssions*

USE INDEX(none) forces the system to avoid seeking an index for the current query.

Example:

```
Select fname, lname, zip, state from TutorialPerson
USEINDEX(TutorialPersonByZipIndex)where zip='33024'

//returns data that looks like this:
John Doe FL 33024
Jim Smith FL 33024
Jane Row FL 33024
```

A Select query returns two dataset outputs per request. The second dataset (named WsSQLCount) provides the total result count. This is useful when paging results.

Example:

```
<Dataset name='WsSQLResult'>
  <Row>
    <yearbuilt>1203</yearbuilt>
    <id>4</id></Row>
  <Row>
    <yearbuilt>2003</yearbuilt>
```



```
<id>5</id></Row>
</Dataset>
<Dataset name='WsSQLCount'>
  <Row>
    <WSSQLSelectQueryResultCount>2</WSSQLSelectQueryResultCount>
  </Row>
</Dataset>
```

SELECT JOIN

select *columnList* **from** *tableName* [**as** *alias*]

[<outer | inner > **JOIN** *join tableName* [**as** *alias*] **on** *joinCondition*]

[**USE INDEX**(*indexFileName* | **NONE**)]

[**where** *logicalExpression*] [**group by** *fieldName*]

[**order by** *columnNames* [asc | desc]] [**LIMIT** *limitNumber*]

| | |
|--------------------------|---|
| <i>columnList</i> | columnreference1[,columnreference2,columnreference3,...,columnreferencen] |
| | The column(s) to return (comma-separated list). In addition, these aggregate functions are supported : COUNT, SUM, MIN, MAX, and AVG. These work in a similar manner as their ECL counterparts. |
| columnreference | [tablename.]columnname[[AS] alias] |
| <i>distinct</i> | [distinct] col1, col2,... coln |
| | The result set will only contain distinct (unique) values. |
| <i>alias</i> | The alias used to refer to the corresponding table or field reference. |
| outer inner | The type of JOIN to use. Note: The WsSQL service currently supports INNER JOIN or OUTER Joins. An OUTER JOIN is converted to a FULL OUTER JOIN internally. |
| <i>joinTableName</i> | The JOIN file to use. |
| <i>joinCondition</i> | Specifies the relationship between columns in the joined tables using logical expression. |
| <i>logicalExpression</i> | Logical expression based on standard SQL filtering syntax. |
| | BOOLEAN Only supports <i>True</i> or <i>False</i> , do not use Y, N, 0, or 1. |
| | Valid operators: |
| | = Equal (e.g., age=33) |
| | <> Not equal (e.g., age <>33) |
| | > Greater than (e.g., age >55) |
| | < Less than (e.g., age < 18) |
| | >= Greater than or equal (e.g., age >=21) |
| | <= Less than or equal (e.g., age <=21) |
| | IN(value1,value2,...,valuen) where values are comma separated homogeneous types. |
| | NOT IN(value1,value2,...,valuen) where values are comma separated homogeneous types. |
| | LIKE <i>pattern</i> where the pattern uses SQL LIKE operators with % and _ wildcards. |
| | NOT LIKE <i>pattern</i> where the pattern uses SQL LIKE operators with % and _ wildcards. |
| <i>limitNumber</i> | Optional. The number of rows to return. This overrides the default configuration attribute (EclResultLimit) but cannot be set to ALL. |

¹Aliasing not supported

²Can only contain references to aggregate functions if used with *having* clause.

Aggregate functions can only be expressed in logicalExpressions by using *Group by* and *having*

Examples:

```
Select * from tableList where Sum(F1 > 100) /* is NOT SUPPORTED */  
Select * from tableList Group by F1 Having Sum (F1 > 100) /* IS SUPPORTED */
```

Example:

```
Select t1.personname, t2.address  
  from persontable as t1 inner join adresstable as t2  
  on (t1.personid = t2.personid AND  
      (t1.firstname = 'jim' AND  
       t1.lastname  = 'smith' ))
```

The interface does not convert parameter list or column list values to string literals.

String values should be single quote encapsulated. Field identifier can be left unquoted or double quoted.

For example, the table **persons** has columns lastname(String) and Zip (numeric)

```
Select Firstname from persons where lastname = 'Jones' and zip > 33445      /* works */  
Select Firstname from persons where lastname = 'Jones' and "zip" > 33445    /* also works */  
Select Firstname from persons where lastname = Jones and zip > 33445        /* doesn't work */  
Select Firstname from persons where lastname = 'Jones' and zip > '33445'    /* doesn't work */
```

CREATE / LOAD

create table [**if not exists**] *newtablename* '('(*fieldname*(*fieldtype*[*fieldlen*[*precision*]]**[UNSIGNED]**[*,*] *)+*)'

[comment '*commenttext*']**];**

load data infile '*sourcetablename*' [**connection** '*landingzoneIP*' **directory** '*landingzonepath*']

into table *newtablename*

[(fields | columns) [terminated by '*fieldDelimiter*'] [enclosed by '*quoteChar*'] [escaped by '*escapeChar*']]

[lines [terminated by '*recordDelimiter*']]];****

The CREATE TABLE and LOAD DATA SQL statements allow you to create a file in an HPCC Systems cluster that is accessible via WsSQL. You can populate the new file from data in a logical file already on the HPCC Systems cluster or one that is on a landing zone.

All Create requests **MUST** be accompanied by a Load request. You cannot create a table with the intention to load or insert data later.

| | |
|------------------------|--|
| <i>newtablename</i> | The logical filename to create on the HPCC Systems cluster. Note: Create and Load must target the same file. |
| <i>commenttext</i> | A text comment to add to the logical file's description |
| <i>sourcetablename</i> | The source file from which to extract data to load into the new table. This can be a logical file on an HPCC Systems cluster or a physical file on a landing zone. For a landing zone file, you MUST provide a connection ' <i>landingzoneIP</i> ' and a directory ' <i>landingzonepath</i> ' (the path to the file). For a landing zone file, be sure to use matching case for the filename. |
| <i>landingzoneIP</i> | The IP Address or hostname of the HPCC Systems Landing Zone |
| <i>landingzonepath</i> | The path to the datafile on the Landing Zonebut |
| <i>fieldDelimiter</i> | String value for field delimiter |
| <i>quoteChar</i> | String Value for the quote character |
| <i>escapeChar</i> | String Value for the escape character |
| <i>recordDelimiter</i> | String value for record delimiter |

Examples:

```
CREATE TABLE newCustomerFile (id DECIMAL( 30, 5), mytint INT(9),
                                mydouble DOUBLE (5,3) UNSIGNED)
COMMENT 'this file created via WsSQL and populated from file on lz';
LOAD DATA INFILE 'CustomerData' CONNECTION '127.0.0.1'
                                DIRECTORY '/var/lib/HPCCSystems/mydropzone'
INTO TABLE newCustomerFile;

CREATE TABLE newCustomerFile2 (id DECIMAL( 30, 5), mytint INT(9),
                                mydouble DOUBLE (5,3) UNSIGNED)
COMMENT 'this file created via WsSQL and populated from sprayed file';
LOAD DATA INFILE 'thor::customerdata::customers' INTO TABLE newCustomerFile2

CREATE TABLE IF NOT EXISTS newCustomerFile3 (id DECIMAL( 30, 5), mytint INT(9),
                                                mydouble DOUBLE (5,3) UNSIGNED)
COMMENT 'won't overwrite';
LOAD DATA INFILE 'thor::customerdata::customers' INTO TABLE newCustomerFile3
```

```
CREATE TABLE newCustomerFile4CSV (id DECIMAL( 30, 5), mytint INT(9),  
                                   mydouble DOUBLE (5,3) UNSIGNED)  
COMMENT 'Loading CSV data';  
LOAD DATA INFILE 'somecsvfile.csv' FIELDS TERMINATED BY ',' ENCLOSED BY '"'  
                                   LINES TERMINATED BY 'n'  
INTO TABLE newCustomerFile4CSV
```

Supported Aggregate Functions

COUNT(*[DISTINCT] columnName*)

DISTINCT(*columnName*)

SUM(*columnName*)

MIN(*columnName*)

MAX(*columnName*)

AVG(*columnName*)

These aggregate functions are supported. They behave as their ECL counterparts. See the **ECL Language Reference** for details.

| | |
|------------|--|
| COUNT | Counts the occurrences of columnName in the result, always an integer. |
| DISTINCT | Returns only distinct values of columnName in the result, output type is dependent on input type. |
| SUM | Returns the sum of the values of columnName in the result, output type is dependent on input type. |
| MIN | Returns the minimum value for of columnName in the result, output type is dependent on input type. |
| MAX | Returns the minimum value for of columnName in the result, output type is dependent on input type. |
| AVG | Returns the average of the values of columnName in the result, always a real number. |
| columnName | The column to aggregate. |

Example:

```
Select fname, lname, state, COUNT(zip) from TutorialPerson where zip='33024'
```

Supported String Modifiers

UPPER(*columnName*)

LOWER(*columnName*)

| | |
|------------|---|
| UPPER | Returns with all lower case characters converted to upper case. |
| LOWER | Returns with all upper case characters converted to lower case. |
| columnName | The column to aggregate |

Special considerations

Since this service uses both ECL and SQL, there are a few special considerations when designing the backend databases and stored procedures.

Reserved SQL Keywords

Your HPCC Systems identifiers (field names, Stored Procedure names, etc) cannot use SQL reserved words such as:

```
ADD ALL ANY AS ASC AT AVG BETWEEN BOOL BOOLEAN BY CALL COLUMN CONTAINS COUNT  
DESC DISTINCT FALSE FOR FROM GROUP HAVING IN INDEX INNER IS JOIN KEY KEYS LAST  
LEFT LIKE LIMIT LOWER MAX MIN MOD NOT NULL OFFSET ON ORDER OUT OUTER POWER SELECT  
SUM TABLE TRUE UPPER USE WHERE XOR DIV MOD OR AND
```

Special Characters

Your HPCC Systems filenames should avoid special characters other than:

```
( 'A'..'Z' | 'a'..'z' | ' ' | '$' ) ( 'A'..'Z' | 'a'..'z' | ' ' | '$' | '0'..'9' )  
File names can be prefixed with a ~ (tilde) or .::
```

If a filename is not supported, it will not be returned when interrogating the system using GetDBMetaData.