



Rapid Data Delivery Engine Reference

Boca Raton Documentation Team



Rapid Data Delivery Engine Reference

Boca Raton Documentation Team

Copyright © 2015 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.

HPCC Systems® is a registered trademark of LexisNexis Risk Data Management Inc.

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.

2015 Version 5.4.2-1

Introduction	4
Roxie Overview	5
Payload INDEXes	6
Roxie Superfiles	7
How Roxie Works	8
Roxie Data Backup	12
Developing Roxie Queries	14
Development Path	14
Methods to Submit Jobs to a Roxie Cluster	15
Managing Queries	18
Adding a roxie query to the Query Set	19
Viewing Query Sets using ECL Watch	20
Using WsECL to run a roxie query	21
Packages and Packagemaps	22
Example	23
Updating Data	24
Package File Syntax	25
Working with Packages using the ecl command line tool	26
Direct Access to Roxie	35
Roxie Query Access Overview	35
Accessing your Roxie Queries	36
Deploying Data to a Roxie Cluster using DFU	40
DFU Copy	41
Remote Copy	42
Capacity Planning for Roxie Clusters	44
Capacity Planning	44
PreFlight and Roxie Metrics	46
Preflight the Roxie Cluster	47
Roxie Metrics	49
Queries Page in ECL Watch	50

Introduction

The Roxie engine - also known as the Rapid Data Delivery Engine or RDDE - uses a combination of technologies and techniques that produce extremely fast throughput for queries on indexed data housed in a dedicated High Performance Computing Cluster (HPCC).

Roxie queries can exceed several thousand a second, compared to Thor queries which tend to take from a few seconds to a few minutes each (from end to end) depending on the complexity of the query.

To fully understand this concept, it is best to look at the purpose for which each of these processes is designed:

- The Thor platform is designed to perform operations on every record (or most records) in a massive dataset.
- Queries can be run on the hThor platform if they can quickly pinpoint small sets of records within the data.
- Roxie queries are typically used to quickly pinpoint small sets of records over and over again.

If you think of all your data as an ocean, Thor would be used to perform operations on the entire ocean.

An hThor query might be used to find a single fish within that sea of data.

The query would be deployed to a Roxie cluster to be used to find hundreds or thousands of individual fish-one after another.

Roxie queries are deployed to a Roxie cluster, which pre-loads all queries into memory and prepares them to be ready to execute as soon as a query is received.

Queries are sent to Roxie via XML, SOAP, or JSON, and the results are returned in the same format. A client can communicate directly with the Roxie cluster by opening a socket to one of the servers in the cluster, or it can communicate via an ESP service such as WsECL.

Typically, Roxie results are returned to the requester rather than writing a result to a file. However, Roxie can write data files, although you generally would not want to write a file when a query is not workunit-based.

Roxie Overview

There are typically four aspects to using Roxie:

- Creating indexes on datasets
- Using indexes in queries
- Compiling and Deploying queries to the Roxie Cluster
- Providing access to those queries to client-facing interfaces via SOAP or HTTP.

When to Use Indexes

The Thor platform is designed to perform operations quickly on massive datasets without indexes, where the entire dataset (or almost all of it) is to be used. However, if only a few records are needed, an index can access them more efficiently. In the ECL language, an index behaves just like a dataset that happens to be able to implement certain functions (typically filter and count functions) much more quickly than a standard flat file or CSV dataset can.

Payload INDEXes

In conventional database systems, an index is used together with a data file to locate records in that data file. You can do the same in ECL by storing file positions in the index and using them in a `FETCH` function to retrieve the corresponding data rows from the original file.

However, because you can store whatever fields you wish in an index, it is more common in Roxie queries to design indexes that store both the search fields and the information you want to look up. This eliminates an extra disk read for the `FETCH`. Since indexes are compressed, this can also save disk space if the original data file does not need to be stored on the Roxie cluster.

Any field in an index that does not need to be searched for can be specified as being in the "payload" - such fields are stored only at the leaf nodes of the index tree which can save some space and performance in the lookups. The fields in the payload may simply be additional fields from the base dataset, but they may also contain the result of some preliminary computation (computed fields).

Roxie Superfiles

A superfile or superkey used in Roxie may contain more than a single sub-file.

However, your superfile cannot contain more than one sub-file when the superfile is used for a `FETCH` operation or in a full-keyed `JOIN`.

Even if you only have one sub-file, you can write a query that uses superfiles or superkeys (even though they contain only a single sub-file) you have the advantage of being able to update your Roxie by simply deploying new data without needing to re-compile the queries that use that data simply because the sub-file name changed. This saves compilation time, and in a production environment (which a Roxie typically is) where a data file is used by many queries, this savings can be a significant.

See the *ECL Programmer's Guide* for more details.

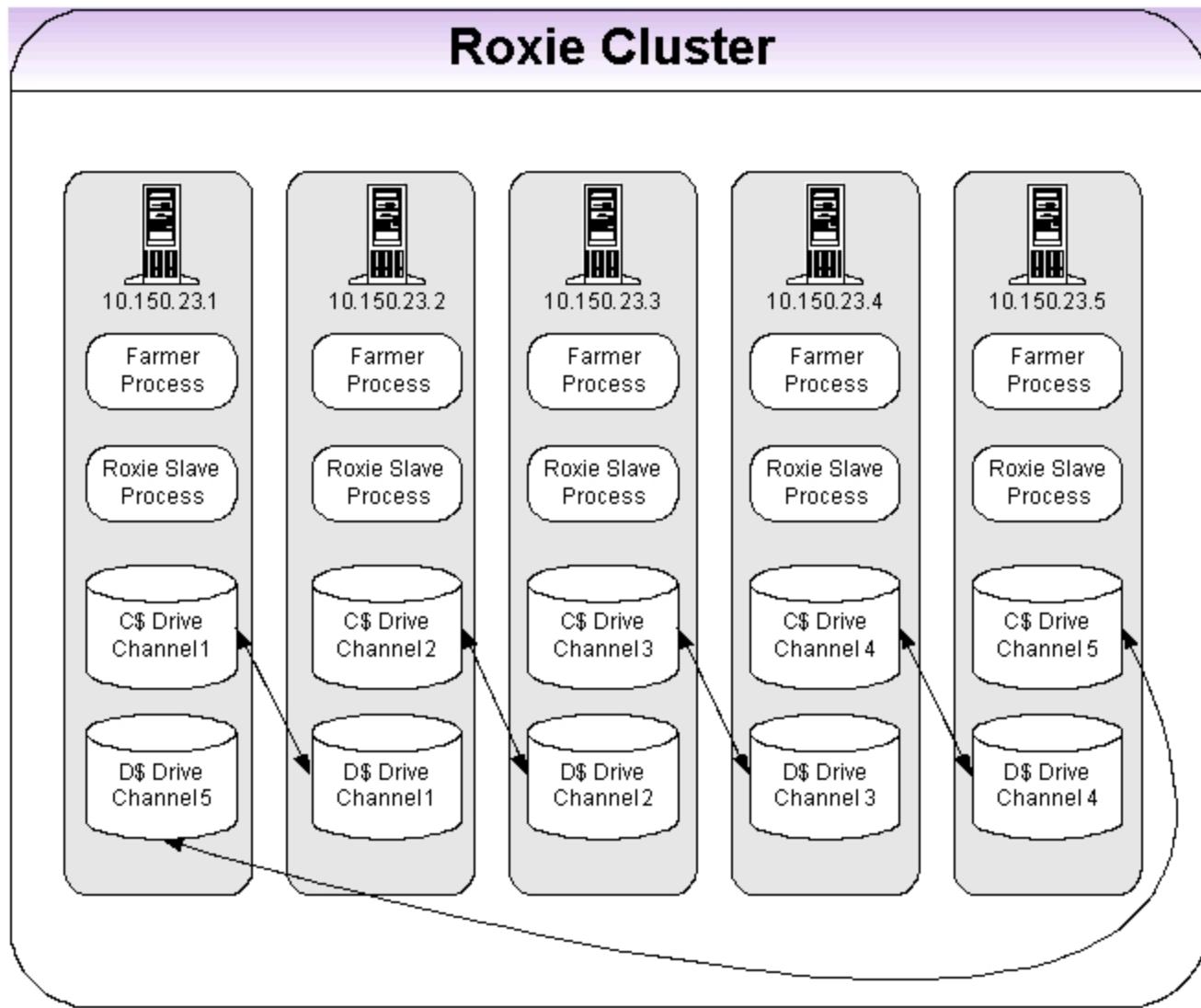
How Roxie Works

Roxie clusters consist of a number of machines connected together to function as a single entity. ECL source code for one or more queries is compiled and published to the cluster. Once published, queries can process data rapidly.

Each machine in the cluster acts in two distinct roles - these roles execute in the same process and share a lot of code - including the compiled query code - but can be thought of as logically distinct.

The **Server** process accepts incoming queries from a client, evaluates the ECL code of the query given the input that is provided in the client query, and returns the result to the client. When the Server evaluates an ECL function that requires data from disk, it determines the slave node or nodes that may have the appropriate data, and sends a request to those slave nodes to retrieve any matching data. Results from multiple slave nodes are collated and become the input for further ECL functions evaluated on the Server node. Typically, requesting applications use some form of load balancing to distribute requests evenly to the available Servers.

The **Slave** process accepts requests only from other Server nodes in the cluster. These requests correspond to a single ECL function such as a filtered index read or a disk fetch. Results are sent back to the Server that originally requested them. In order to balance performance and manage hardware failures, slaves receive the requests over multicast, and there will typically be at least two slave nodes that will receive each request from a server. The slave nodes communicate with each other to avoid duplicating effort, so that whichever slave is first able to handle the request tells the others not to bother. Each node in a cluster typically handles Slave requests on two or more multicast channels, usually one channel per disk drive. If any Slave node is not responding, the requests on that channel are handled by the other peer Slave nodes responsible for that channel.



This example shows a 5-node Roxie Cluster with each node configured to be both a **Server** and a **Slave**.

Queries that have been compiled with the target platform specified as a Roxie cluster may be published to a QuerySet using EclWatch.

Each Roxie cluster loads queries from one or more QuerySet lists.

When a query is added to the QuerySet that a Roxie is watching, Roxie will preload the query .so (or .DLL) and prepare the execution context as far as it can, so that it is ready to execute incoming requests to execute that query as soon as they arrive. This may include loading the .so (or .DLL), resolving file references, and opening files (if there are sufficient file handles available), preloading data into memory if requested, and evaluating ECL code in the query that has been marked as : ONCE.

Depending on the configuration, Roxie may read data remotely from a Thor cluster where it was prepared, or if preferred, it may be copied to the Roxie for local access.

Typically a development system might refer to data in situ on the Thor cluster, while a production system may prefer the performance benefits of copying data locally to the Roxie.

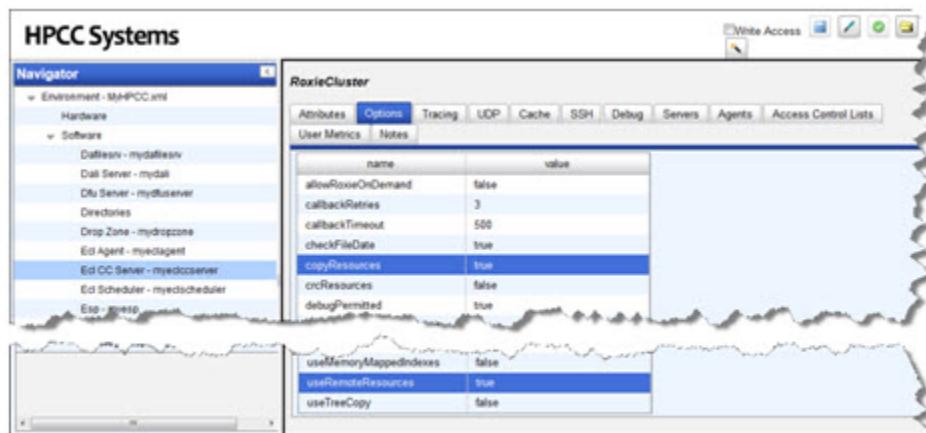
Roxie can read data remotely while it is being copied and switch to the local copy once the copy operation is complete. This provides the benefit of quick availability because the query can be active before the data is copied, while still taking advantage of the performance benefits of local data.

Queries and Data

Data files and index files referenced by a Roxie query's ECL code are made available in one of four ways, depending on the configuration of the Roxie cluster.

There are two settings in the Roxie configuration that control where Roxie looks for data and index files:

- | | |
|---------------------------|--|
| copyResources | Copies necessary data and key files from the current location when the query is published. |
| useRemoteResources | Instructs Roxie to look for data and key files in the current location after the query is published. |



These options may appear to be mutually exclusive, but the chart below shows what each of the four possible combination means.

copyResources	T	T	F	F
useRemoteResources	T	F	T	F
	Directs the Roxie cluster to use the remote instance of the data until it can copy the data locally. This allows a query to be available immediately while the data is copied.	Directs the Roxie cluster to copy the data locally. The query cannot be executed until the data is copied. This ensures optimum performance after the data is copied.	Directs the Roxie cluster to load the data from a remote location. The query can be executed immediately, but performance is limited by network bandwidth. This allows queries to run without using any Roxie disk space, but reduces its throughput capabilities.	Will use data and indexes already loaded (placed on the Roxie cluster using DFU) but will not copy or read remote data.

When **copyResources** is enabled, data files are copied from source locations to slave nodes. Index files are also copied, with an extra file part that is a “key-of-keys” or a metakey.

A copy of the metakey is always stored on each Farmer and in most cases is loaded into memory at startup to increase performance.

Command Line support

The ecl command line tool offers a means of copying queries from one Roxie to another. Typically, this means data and index files would be copied or accessed remotely following Roxie’s configuration settings.

The ecl command line tool provides an option to instruct Roxie to not copy files at the time of the query copy. This allows you to quickly copy a query without copying files.

```
ecl queries copy --no-files
```

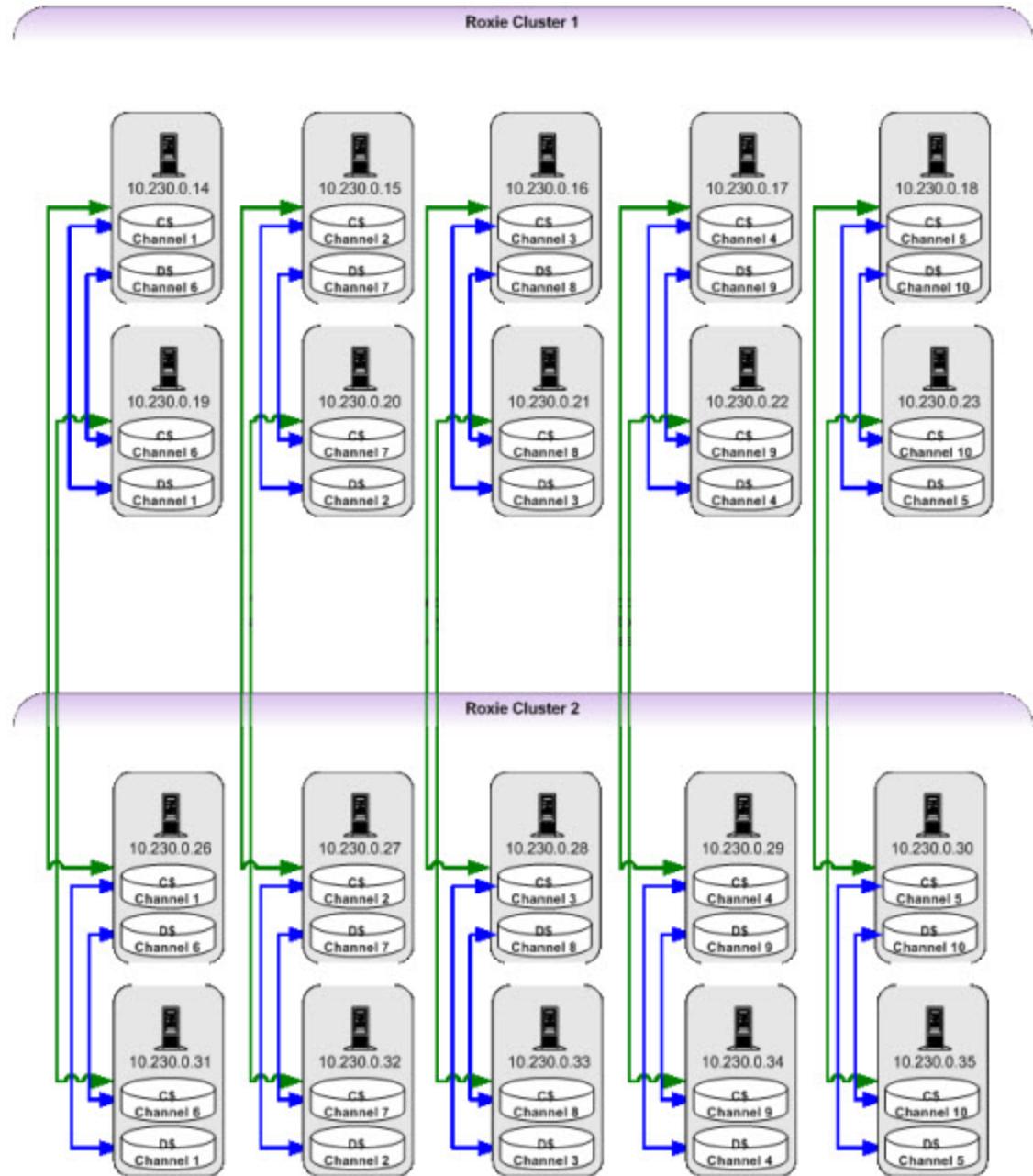
This specifies to NOT copy files referenced by the query being copied. The query cannot run until data is made available.

Roxie Data Backup

Roxie data is protected by three forms of redundancy:

- Original Source Data File Retention: When a query is deployed, the data is typically copied from a Thor cluster's hard drives. Therefore, the Thor data can serve as backup, provided it is not removed or altered on Thor. Thor data is typically retained for a period of time sufficient to serve as a backup copy.
- Peer-Node Redundancy: Each Slave node typically has one or more peer nodes within its cluster. Each peer stores a copy of data files it will read.
- Sibling Cluster Redundancy: Although not required, Roxie deployments may run multiple identically-configured Roxie clusters. When two clusters are deployed for Production each node has an identical twin in terms of data and queries stored on the node in the other cluster.

This configuration provides multiple redundant copies of data files. In this example, there are six copies of each file at any given time; eliminating the need to use traditional backup procedures for Roxie data files.



Developing Roxie Queries

Development Path

1. Determine the need.
2. Evaluate data and determine fields to index.
3. Build Index(es).
4. Create a hThor query.
5. Test and fine-tune the query (using hThor).
6. Publish the Query to a Roxie cluster.
7. Test and certify (compare results to expected results).

Note: These steps are explained in detail in the HPCC Data Tutorial and the Programmer's Guide.

Methods to Submit Jobs to a Roxie Cluster

After a query is compiled and deployed, there are several methods to submit jobs that use the query. While the most common usage is via custom applications using the XML or SOAP interface, the other methods have valid uses, too.

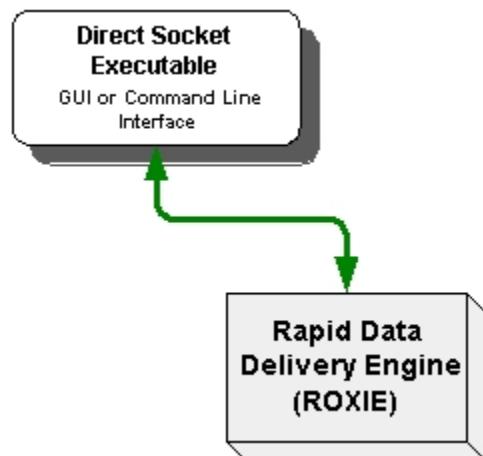
A direct socket connection can communicate directly with the Roxie cluster, eliminating all other intermediate components. This provides a means of certifying the Roxie cluster, its configuration, the deployment of the query, and the query itself.

SOAPCALL allows a Thor query to make calls to a Roxie query (See the *ECL Language Reference* for details). This provides the capability of combining Roxie results with other data processing tasks performed during ETL.

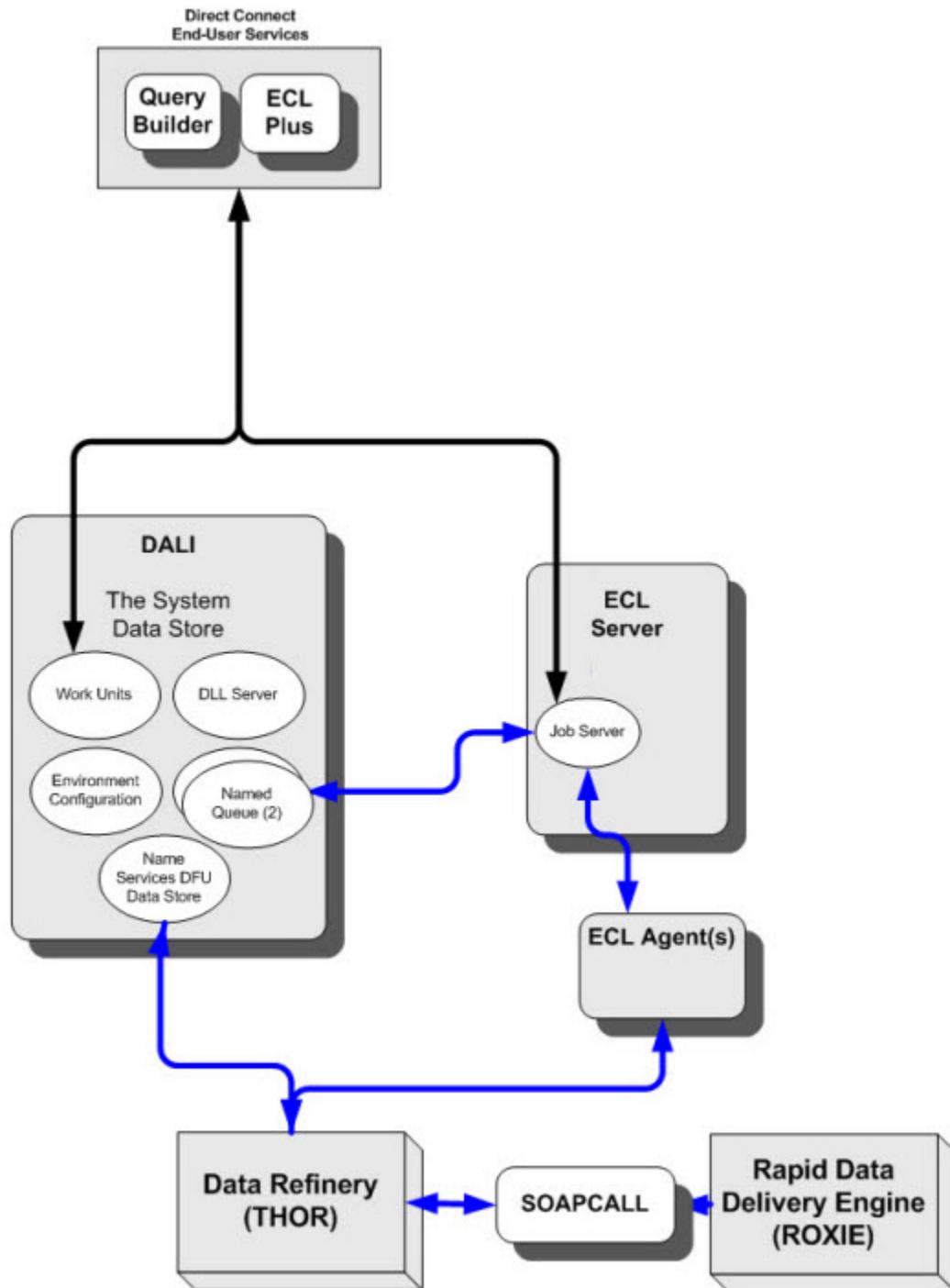
Conducting queries via an ESP service using HTTP or HTTPS allows access to queries directly from a browser. Web-based access allows you to provide easy access to anyone you wish. Using HTTPS, you can ensure data security using Secure Socket Layer (SSL) encryption. This ensures all data is encrypted as it travels across a network or the Internet. In addition, LDAP authentication is available to restrict access to a set of users.

Custom applications using SOAP provides the most flexibility and functionality. The application development process is simplified by Enterprise Services Platform's automatic Web Services Definition Language (WSDL) generation. Many development tools (such as, Microsoft's .NET Studio or NetBeans JAVA) include a tool to generate code to create proxy stubs from a WSDL document. This simplifies the development process and ensures exposure of all necessary methods and properties.

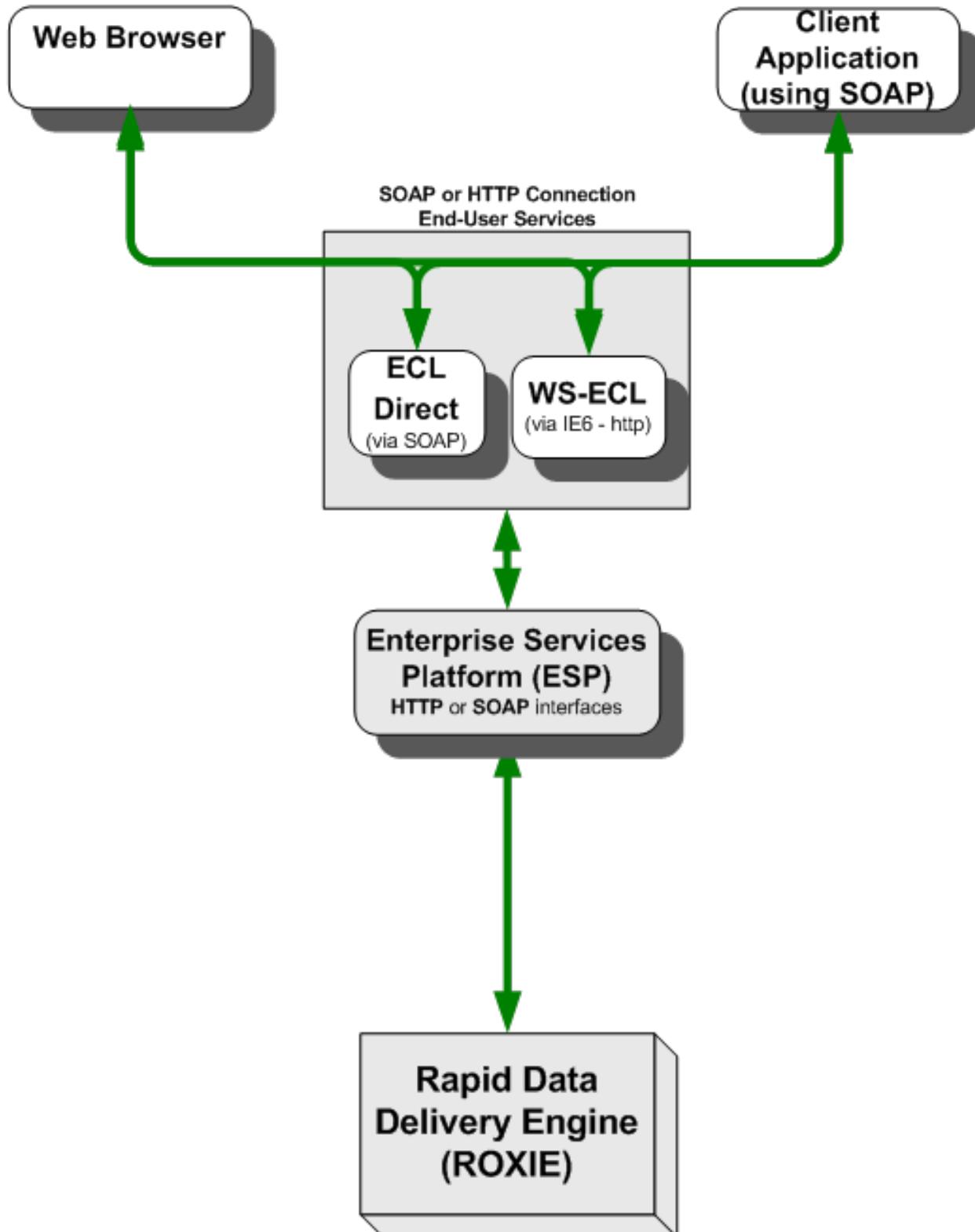
Direct Socket Connection (TCP/IP)



SOAPCALL via ECL



SOAP or HTTP/HTTPS



Managing Queries

Roxie queries are managed via Query Sets which are stored in Dali. Query Sets control which queries are loaded onto Roxie when it starts up and are manipulated by adding or removing them as required. The list of queries currently held within a QuerySet can be viewed using ECL Watch.

Once a query has been published to the Query Set, it can now be run on the roxie using a web interface.

Adding a roxie query to the Query Set

To add a roxie query to a Query Set:

1. Use ECL IDE to write your query and then compile it setting the target as the roxie cluster in your HPCC environment.
2. Go the ECL Watch tab for the compiled workunit and click the Publish button. A message is displayed indicating that your query has been published successfully.

Viewing Query Sets using ECL Watch

Using ECL Watch, you can view the query for all clusters on your HPCC. Click on the **Query Sets/Browse** menu item to see the clusters which are currently using Query Sets. Click on the **myroxie** link to view the list of queries currently available for roxie. Using this feature you can:

- View the list of currently available queries on a cluster.
- View details about each query including the ID, the name of the query, the workunit ID, the DLL(s) it uses and whether it is suspended.
- View details of the aliases that exist for each query.
- Delete a Query from the list.
- Delete an Alias.
- Toggle the suspend setting on/off

Using WsECL to run a roxie query

WsECL is the ECL Web Service interface that is provided with HPCC and is available using the following URL: <http://nnn.nnn.nnn.nnn:8002>, where nnn.nnn.nnn.nnn is the IP address of your ESP.

WsECL uses the Query Sets information to display the list of available runnable queries and you can use it, for example, to test that your query works as expected.

The web page shows all clusters using Query Sets. Expand myroxie in the tree and select the query you want to run. A default form is displayed which is generated from the input field names and types. Enter values and press submit to see the results and test your query.

Packages and Packagemaps

A Roxie query has one or more data files associated with it. At times, you may want to update data without changing the query.

Packages are a way to separate the data definition from the query.

A package defines a superkey that a query will use. A newer package that redefines the contents of that superkey can later be sent to a Roxie cluster without recompiling any queries. This allows you to refresh data and while ensuring that you are using identical code in your query. It also eliminates the need to recompile it and republish it.

This simplifies change control and allows query developers to continue development without the risk of a query being deployed to production before it is ready while still allowing data to be updated.

A packagemap provides a reference to the contents of a superkey used in queries that overrides the original definition . This affords greater flexibility and control over the data used by a query or a collection of queries.

Roxie resolves data files at query load time. First, it will look for a package defining the superkey contents. If there is no package, it will look in the Dali Server's DFU (Distributed File Utility).

The end result is quicker, more flexible process, without the need of recompiling complex queries. This allows you to update data without the chance of deploying new code that has not been approved for migration to production.

Without packages, you would do one of the following:

- Add new subfile(s) to the superkey via ECL code, ECLWatch, or DFUPlus, then reload the cluster.
- Revise the query to use a different key file, compile it, and republish it.

A packagemap file can contain one or more packages. The file is used to transfer the information to the Dali server. Once it is sent, the file is no longer used. It is a good idea to keep a copy as a backup, but it serves no other purpose.

The definition of a superfile or superkey inside of a package file overrides the definition in Dali, but does NOT change the superfile or superkey definition in Dali Server's DFU.

Package information is used as soon as it is added to Dali and the package is activated. This can be done using using **ecl packagemap add --activate** (or **ecl packagemap add & ecl packagemap activate**).

Example

In this example, we have a query named MyQuery which uses a superkey named MyDataKey that includes two subfiles:

- **~thor::Mysubfile1**
- **~thor::Mysubfile2**

If we wanted to add a third subfile, we can use a packagemap to redefine the MyDataKey superkey definition so it contains three files:

- **~thor::Mysubfile1**
- **~thor::Mysubfile2**
- **~thor::Mysubfile3**

Example 1:

```
<RoxiePackages>
  <Package id="MyQuery">
    <Base id="thor::MyData_Key" />
  </Package>
  <Package id="thor::MyData_Key">
    <SuperFile id="~thor::MyData_Key">
      <SubFile value="~thor::Mysubfile1"/>
      <SubFile value="~thor::Mysubfile2"/>
    </SuperFile>
  </Package>
</RoxiePackages>
```

Example 2:

```
<RoxiePackages>
  <Package id="MyQuery">
    <Base id="thor::MyData_Key" />
  </Package>
  <Package id="thor::MyData_Key">
    <SuperFile id="~thor::MyData_Key">
      <SubFile value="~thor::Mysubfile1"/>
      <SubFile value="~thor::Mysubfile2"/>
      <SubFile value="~thor::Mysubfile3"/>
    </SuperFile>
  </Package>
</RoxiePackages>
```

Updating Data

This section details the typical steps a query developer and a data developer follow once a query is ready for production.

- Prepare the data (in this workflow, the data is defined as a superkey)
- Write the query and test
- Publish a query using the data

Later when you want to update the data:

Deploy the data in one of the following manners:

- Create a packagemap containing a package redefining the contents of the superkey
- Add the packagemap(s) by associating the package information with a QuerySet.
- Activate the packagemap(s) with the pertinent package information targeting a QuerySet.

Use the command:

```
ecl packagemap add --activate
```

Later when new data arrives, follow these steps to update the data using packages:

- Prepare the data and create a new subfile

Note: We strongly recommend against reusing a file name. It is generally better to create new files. Roxie shares file handles so trying to change an existing file while it is loaded could cause issues.

- Create a package with a superkey definition that includes the new subfile
- Add the packagemap redefining the contents of the superkey.

Use the command:

```
ecl packagemap add --activate
```

Package File Syntax

Package files are formatted in XML using the conventions detailed in this section.

A Package file must begin with `<RoxiePackages>` and end with `</RoxiePackages>`.

Package tags have an **id** attribute that specifies what the package is referring to.

Inside the `<Package>` structure, references are made either to superfiles or other named packages. This indirect naming convention allows you to group file definitions together and reference the package containing the group.

The example shows query references first and file references following; however, this order is not required.

The lines are numbered only for reference purposes. The comments are included for clarity, but are not required.

Example:

```
1.  <RoxiePackages>
2.  <!-- Begin Queries -->
3.  <Package id="MyQuery">
4.  <Base id="thor::MyData_Key"/>
5.  </Package>
6.  <!-- End Queries -->
7.  <!-- Begin File references -->
8.  <Package id="thor::MyData_Key">
9.    <SuperFile id="~thor::MyData_Key">
10.      <SubFile value="~thor::Mysubfile1"/>
11.      <SubFile value="~thor::Mysubfile2"/>
12.      <SubFile value="~thor::Mysubfile3"/>
13.    </SuperFile>
14.  </Package>
15.  <!--End File references -->
16. </RoxiePackages>
```

In this example, we have a query: **MyQuery**. The query uses a superkey defined in a package named **thor::MyData_Key**. This is later defined on line 8. The **thor::MyData_Key** package contains one superkey definition.

The example shows query references first and file references following; however, this order is not required.

You can specify daliIP(s) and/or sourcecluster(s) to use to resolve files to be specified in the packagemap. This can be specified at different levels to affect files within a given scope.

For example:

```
<PackageMap daliip="10.239.8.10">
  <Package id="MyPackage" daliip="10.239.7.1" sourceCluster="thor1" >
    <SuperFile id="thor::Mykey1_idx">
      <SubFile value="thor::mySub1"/>
    </SuperFile>
    <SuperFile id="thor::Mykey2_idx"  daliip="10.239.8.1" sourceCluster="thor2" >
      <SubFile value="thor::mySubKeyA"/>
      <SubFile value="thor::mySubKeyB"/>
    </SuperFile>
  </Package>
</PackageMap>
```

Working with Packages using the ecl command line tool

This section contains details about the actions and options used to work with packages. The ECL command line tool is fully documented in the Client Tools Manual.

ecl packagemap add

ecl packagemap add [--daliip][options] <target> <filename>

Examples:

```
ecl packagemap add -s=192.168.1.10 roxie mypackagemap.pkg
ecl packagemap add roxie mypackagemap.pkg --overwrite
ecl packagemap add roxie mypackagemap.pkg --daliip=192.168.11.11
```

ecl packagemap add Calls the packagemap add command

Actions

Arguments

target The target to associate the packagemap with

filename The name of the file containing packagemap information.

--daliip= IP address or hostname of the remote Dali to use for logical file lookups

Options

-O, --overwrite Whether to overwrite existing information - true if present

-A, --activate Activates package map

--allow-foreign Specifies to allow the use of foreign files. If a packagemap references foreign files and this is not enabled, packagemap add will fail.

--pmid=<packagemapid> id of package map - defaults to filename if not specified

-v, --verbose Output additional tracing information

-s, --server The IP Address or hostname of ESP server running ECL Watch services

--port The ECL Watch services port (Default is 8010)

-ssl Use SSL to secure the connection to the server.

`-u, --username` The username (if necessary)

-pw, --password The password (if necessary)

ecl packagemap delete

ecl packagemap delete [options] <target><packagemap>

Examples:

```
ecl packagemap delete roxie mypackagemap
```

ecl packagemap delete

Calls the packagemap delete command

Actions

delete

Deletes a packagemap

Options

-v, --verbose

Output additional tracing information

-s, --server

The IP Address or hostname of ESP server running ECL Watch services

--port

The ECL Watch services port (Default is 8010)

-ssl

Use SSL to secure the connection to the server.

-u, --username

The username (if necessary)

-pw, --password

The password (if necessary)

ecl packagemap activate

ecl packagemap activate <target> <packagemap>

Example:

```
ecl packagemap activate roxie mypackagemap.pkg
```

ecl packagemap activate

The activate command will deactivate the currently active packagemap and make the specified packagemap active.

Arguments

target

The target containing the packagemap to activate

packagemap

name of packagemap to update

Options

-v, --verbose

Output additional tracing information

-s, --server

The IP Address or hostname of ESP server running ECL Watch services

--port

The ECL Watch services port (Default is 8010)

-ssl

Use SSL to secure the connection to the server.

-u, --username

The username (if necessary)

-pw, --password

The password (if necessary)

ecl packagemap deactivate

ecl packagemap deactivate <target> <packagemap>

Example:

```
ecl packagemap deactivate roxie mypackagemap.pkg
```

ecl packagemap deactivate

The deactivate command will deactivate the currently active packagemap.

Arguments

target

The target containing the packagemap to deactivate

packagemap

Name of packagemap to deactivate

Options

-v, --verbose

Output additional tracing information

-s, --server

The IP Address or hostname of ESP server running ECL Watch services

--port

The ECL Watch services port (Default is 8010)

-ssl

Use SSL to secure the connection to the server.

-u, --username

The username (if necessary)

-pw, --password

The password (if necessary)

ecl packagemap list

ecl packagemap list <target>

Examples:

```
ecl packagemap list roxie
```

ecl packagemap list Calls the packagemap list command

Actions

list Lists loaded packagemap names

Arguments

target The target containing the packagemap to list

Options

-v, --verbose Output additional tracing information

-s, --server The IP Address or hostname of ESP server running ECL Watch services

--port The ECL Watch services port (Default is 8010)

-ssl Use SSL to secure the connection to the server.

-u,-

-pw, --password The password (if necessary)

ecl packagemap info

ecl packagemap info [options] <target>

Examples:

```
ecl packagemap info roxie
```

ecl packagemap info Calls the packagemap info command

Actions

info returns packagemap info

Arguments

target The target containing the packagemap to retrieve

Options

-v, --verbose Output additional tracing information

-s, --server The IP Address or hostname of ESP server running ECL Watch services

--port The ECL Watch services port (Default is 8010)

-ssl Use SSL to secure the connection to the server.

-u,-

-pw, --password The password (if necessary)

ecl packagemap validate

ecl packagemap validate <target> [<filename>]

Examples:

```
ecl packagemap validate roxie mypackagemap.pkg
ecl packagemap validate roxie --active
```

The packagemap validate command verifies that :

- Referenced superkeys have subfiles defined (warns if no subfiles exist)
- All referenced queries exist in the current Roxie queryset
- All Roxie queries are defined in the package

The result will also list any files that are used by queries but not mapped in the packagemap.

Filename, --active, and --pmid are mutually exclusive. The --active or --pmid options validate a packagemap that has already been added instead of a local file.

The --queryid option checks the files in a query instead of all the queries in the target queryset. This is quicker when you only need to validate the files for a single query.

ecl packagemap validate

Calls the packagemap validate command.

Actions

validate

Validates packagemap info

Arguments

filename

The filename containing the packagemap info to validate

target

The target containing the packagemap to validate

Options

-v, --verbose

Output additional tracing information

-s, --server

The IP Address or hostname of ESP server running ECL Watch services

--active

Validates the packagemap that is active for the given target

--pmid=<packagemapid>

Validates the given packagemap

--queryid

Validate the files for the given queryid if they are mapped in the packagemap

--port

The ECL Watch services port (Default is 8010)

-ssl

Use SSL to secure the connection to the server.

-u, --username

The username (if necessary)

-pw, --password

The password (if necessary)

Tips:

- Always use superfiles or superkeys to take advantage of indirect naming and to allow the use of packages. Roxie does not require this, but it works best this way.
- Use unique filenames instead of overwriting existing files. This prevents accidental overwrites and provides an easy way to roll back.
- If you have a large collection of keys, it is easier to maintain if you use superkeys with a single subindex-file. Multiple subfiles are useful when you need to add data quickly, but if time allows, it is better to rebuild a new single key file.
- Before you delete a packagemap, make sure you have a backup copy.

Direct Access to Roxie

Roxie Query Access Overview

WsECL provides quick and easy access to your published queries, but is not optimized for the rapid throughput Roxie provides. To take full advantage of this capability, you should access Roxie directly.

WsECL still plays an important role when using this technique. It provides an immediate means of testing, provides a web interface for one-off query execution, and provides the WSDL and schema which can be used for automated code generators for SOAP access. It also provides the HTTP-GET and form encodes support. Finally, it also provides a JSON (Javascript Simple Object Notation) interface which is similar to Roxie's direct interface.

Web Services

A Web service is a standards-based software component accessible over the Internet. The service can be simple or complex.

For example, a Web service could request a number of stock quotes and return them in an XML result set.

Web Services are available to any platform, object model, or programming language. This provides access to users over the Internet, Intranet, or Extranet regardless of the user's platform.

It also simplifies distributed systems development. The use of standards-based components protects development investment regardless of future trends. Web Service Technologies are based upon HTTP.

Simple Object Access Protocol (SOAP)

SOAP is the most common XML-based protocol for sending requests to and receiving responses from Web Services. Basically, it is a protocol for communication between applications. It is designed to communicate over the Internet and is platform independent and language independent. Based on XML, SOAP is simple and extensible.

JavaScript Object Notation (JSON)

JSON is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, it is completely language independent.

Accessing your Roxie Queries

Once you have developed, compiled, and published your queries, you need to provide access to users.

ESP and WsECL

A standard installation of the HPCC platform includes an Enterprise Services Platform (ESP) with the WsECL service configured through a service binding to port 8002 (configurable).

WsECL provides a means to access all of your compiled and published queries using a Web interface or by using SOAP or JSON.

Tip: You can provide a JobName in the URL using this form:

```
http://ip:port/WsEcl/submit/query/<targetCluster>/<query>?_jobname=<jobname>
```

This is useful for testing queries. It is also suitable for providing real-time access to a users. You can also use proxy mode to bypass WsECL response formatting and get the roxie formatted SOAP response directly. See WsECL and Proxy mode

Access WsECL VIA SOAP

WSDL

The Simple Object Access Protocol (SOAP) sets a standard for communication between processes using a common xml based format. SOAP libraries are readily available for many languages and development platforms, including Microsoft .NET (accessible using C#, VB.NET, ASP.NET, and other CLR languages), Java (e.g., JAX and Apache AXIS), PERL, and many others.

Web Service Description Language (WSDL) is used to provide a structured description of a Web service interface.

Many of the libraries available for allowing applications to use the SOAP protocol also provide tools for automatically generating service specific APIs from a WSDL document.

The WSDL description is automatically available for any published query.

The following URL can be used to retrieve the WSDL description of a published query:

```
http://nnn.nnn.nnn.nnn:8002/WsEcl/definitions/query/roxie/<queryName>/main/<queryName>.wsdl
```

where *nnn.nnn.nnn.nnn* is the IP address (or DNS Name) of that ESP server with the binding to WsECL.

8002 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

The syntax and functionality of the SOAP protocol itself is also beyond the scope of this document. The following list of external resources can help in your understanding of standards-based technologies used.

SOAP

<http://www.w3.org/TR/soap12-part0>

<http://www.w3.org/TR/soap12-part1>

<http://www.w3.org/TR/soap12-part2>

Web Service Definition Language (WSDL) 1.1

<http://www.w3.org/TR/wsdl>

Direct Access VIA SOAP

1. Create and publish your query to a target cluster (Roxie).
2. Test and validate using WsECL to access the query.

Use the default interface provided to test:

<http://nnn.nnn.nnn.nnn:8002/>

where nnn.nnn.nnn.nnn is the IP address (or DNS Name) of that ESP server with the binding to WsECL.

8002 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

3. Begin development of the application that will consume the web service by importing the WSDL from:

<http://nnn.nnn.nnn.nnn:8002/WsEcl/definitions/query/roxie/<queryName>/main/<queryName>.wsdl>

where nnn.nnn.nnn.nnn is the IP address (or DNS Name) of that ESP server with the binding to WsECL.

8002 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

4. When testing is complete, change the base URL to

<http://nnn.nnn.nnn.nnn:9876/>

where nnn.nnn.nnn.nnn is the IP address (or DNS Name) of a Roxie Server.

9876 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

Direct Access VIA JSON

1. Create and publish your query to a target cluster (Roxie).
2. Test and validate using WsECL to access the query.

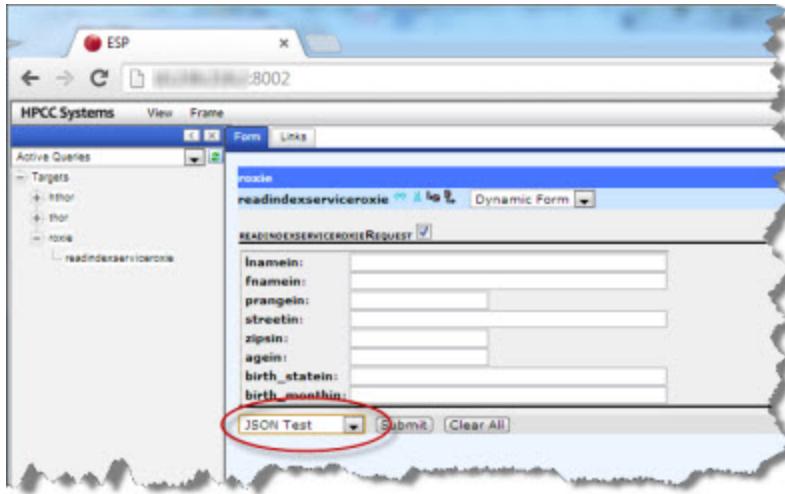
Use the default interface provided to test:

<http://nnn.nnn.nnn.nnn:8002/>

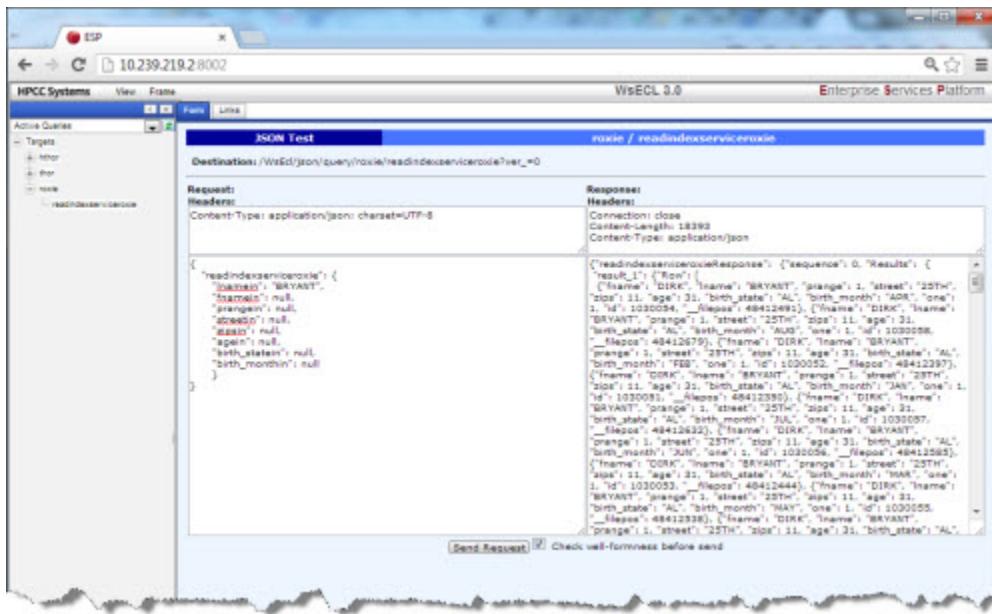
where nnn.nnn.nnn.nnn is the IP address (or DNS Name) of that ESP server with the binding to WsECL.

8002 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

3. Begin development of the application that will consume the web service using the JSON definitions available from the form in WsECL.
4. From the drop list, select JSON Test, then press the submit button.



5. Edit the query criteria on the left, then press the **Send Request** button.
6. The results display on the right side.
7. Edit and resubmit (by pressing the **Send Request** button) to see responses for different criteria.



8. Set the base URL in your application to

<http://nnn.nnn.nnn.nnn:9876>

where nnn.nnn.nnn.nnn is the IP address (or DNS Name) of a Roxie Server.

9876 is the default port. If you have a modified the port setting, use the port you have selected for that purpose.

Note: The WsECL service automatically distributes the load by sending queries to Roxie Servers in a round-robin fashion. To utilize all servers, your application should implement a similar load distribution scheme. Alternatively, you can use a load balancer and submit the query to the Virtual IP created by the load balancer.

Direct Access Via JSONP

To get a JSONP response, add the **&jsonp=<myfunction>** decoration to the URL. This returns a javascript wrapped json response:

```
myfunction("{"data_1": "hello world", "data_2": ["the", "sun", "is", "up"]});
```

WsECL and Proxy mode

For direct HTTP-GET and HTTP-FORM-POST calls to Roxie, change the URL to use proxy mode as shown below:

```
Change:  
/WsEcl/submit/query/RoxieTargetName/QueryName  
to  
/WsEcl/proxy/query/RoxieTargetName/QueryName
```

Parameterized WsEcl proxy calls to Roxie support a subset of the parameter formats supported by non-proxy mode calls.

```
/WsEcl/proxy/query/mytarget/myquery?param1=value
```

For complex parameters, use dot notation to specify nested values. For example:

```
<MyQuery>  
  <State>FL</State>  
  <Regs>  
    <Reg>  
      <Name>SOUTH</Name>  
      <Codes>  
        <Code>PBI</Code>  
        <Code>FLL</Code>  
      </Code>  
    </Reg>  
    <Reg>  
      <Name>NORTH</Name>  
      <Codes>  
        <Code>MCO</Code>  
      </Code>  
    </Reg>  
  </Regs>  
</MyQuery>
```

The URL or HTTP Form would set the following parameters:

```
State=FL  
Regs.Reg.0.Name=SOUTH  
Regs.Reg.0.Codes.Code.0=PBI  
Regs.Reg.0.Codes.Code.1=FLL  
Regs.Reg.1.Name=NORTH  
Regs.Reg.1.Codes.Code=MCO
```

Expanded Mode in WsECL

To include the schema in results, add the **expanded** decoration to the URL as shown below:

```
Change:  
/WsEcl/submit/query/RoxieTargetName/QueryName  
to  
/WsEcl/submit/query/RoxieTargetName/QueryName/expanded
```

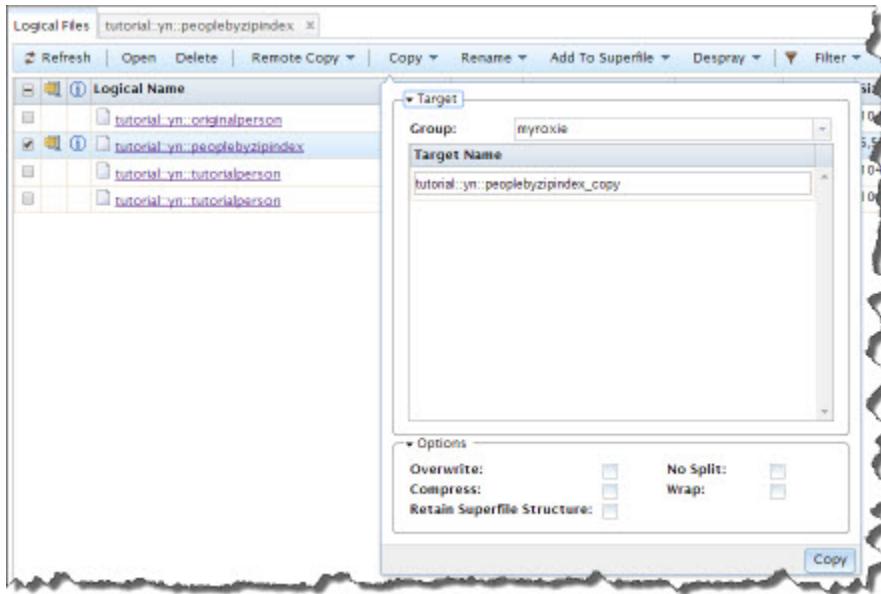
Deploying Data to a Roxie Cluster using DFU

You can use the Distributed File Utility (DFU) in ECL Watch to copy, or remote copy data files to a Roxie cluster. This allows you to copy large files to a cluster in advance of publishing a query. If data files are copied in advance, a query which requires those files will use the ones already in place. If you have large data sets, this allows you to prepare in advance of query deployment.

Note: To use this feature, the FTSlave utility must be installed to each node in the cluster. This is done automatically by the Configuration Manager Wizard.

DFU Copy

1. Open the ECLWatch web page. You can reach this page using the following URL: <http://nnn.nnn.nnn.nnn:8010>, where nnn.nnn.nnn.nnn is your node's IP address.
2. Click on the **Files** icon, then select **Logical Files**.
3. Locate the file to copy in the list of files, then mark the checkbox on the left.



4. Press the **Copy** button, then fill in **Group**, **Target Name**, and **Options** information.

Destination:

Group

Use the drop list to select the Roxie cluster to copy to.

Note: You can only choose from clusters within the current environment.

Target Name

The name of the logical file to create on the target. This is automatically filled in based upon the original logical file, but you can change it.

Options:

Overwrite

Check this box to overwrite files of the same name.

No Split

Check this box to prevent splitting file parts to multiple target parts.

Wrap

Check this box to keep the number of parts the same and wrap if the target cluster is smaller than the original.

Compress

Check this box to compress the files.

Retain SuperFile Structure

Check this box to retain the structure of a SuperFile.

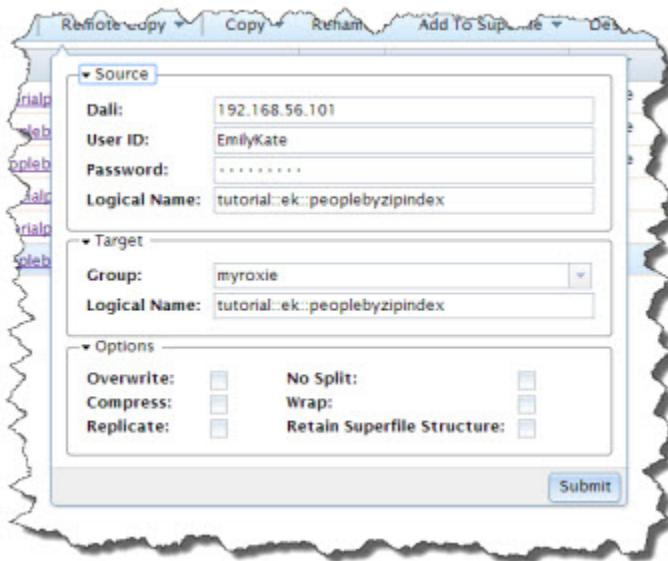
5. Press the **Copy** button.

A new tab opens displaying information about the new file.

Remote Copy

Remote Copy allows you to copy data to a Roxie from a Thor or Roxie cluster outside your environment.

1. Open the ECLWatch web page. You can reach this page using the following URL: <http://nnn.nnn.nnn.nnn:8010>, where nnn.nnn.nnn.nnn is your node's IP address.
2. Click on the **Files** icon, then select **Logical Files**.
3. Press the Remote Copy button, then fill in the details.



Source:

- Dali The IP or hostname of the Dali for the environment from which you are copying.
User ID The Username to use to authenticate on the Remote environment
Password The password to use to authenticate on the Remote environment
Logical Name The name of the logical file to copy.

Target:

- Group Use the drop list to select the Roxie cluster to copy to.
Note: You can only choose from clusters within the current environment.
Logical Name The name of the logical file to create on the target.

Options:

- Overwrite Check this box to overwrite files of the same name.
No Split Check this box to prevent splitting file parts to multiple target parts.
Wrap Check this box to keep the number of parts the same and wrap if the target cluster is smaller than the original.
Compress Check this box to compress the files.
Retain SuperFile Structure Check this box to retain the structure of a SuperFile.

4. Press the **Submit** button.

A new tab opens displaying information about the new file.

Capacity Planning for Roxie Clusters

Capacity Planning

Roxie clusters are disk-based High Performance Computing Clusters (HPCC) , typically using indexed files. A cluster is capable of storing and manipulating as much data as its combined hard drive space; however, this does not produce optimal performance.

For maximum performance, you should configure your cluster so slave nodes perform most jobs in memory.

For example, if a query uses three data files with a combined file size of 60 GB, a 40-channel cluster is a good size, while a 60-channel is probably better.

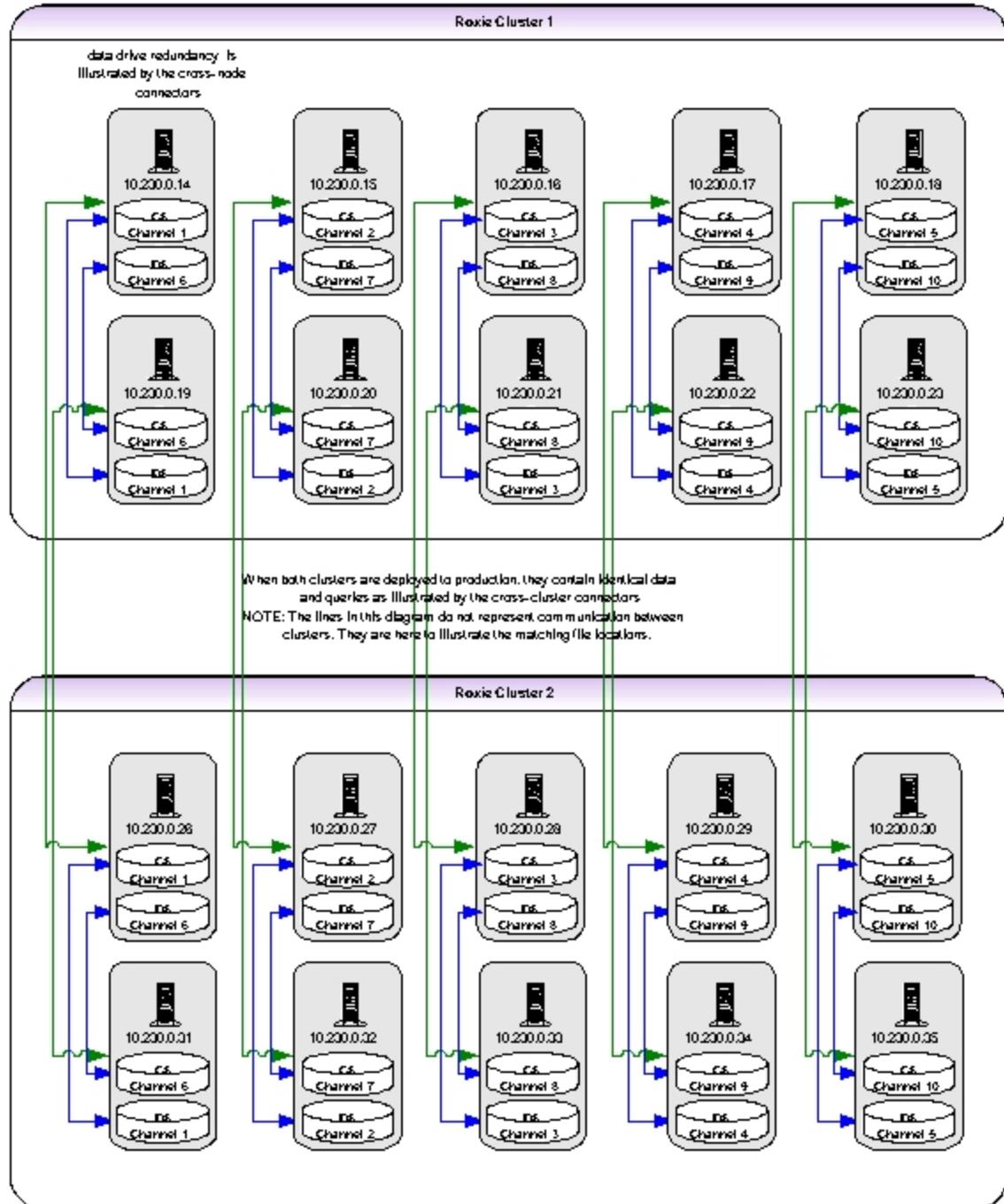
Another consideration is the size of the Thor cluster creating the data files and index files to be loaded. Your target Roxie cluster should be the same size as the Thor on which the data and index files are created or a number evenly divisible by the size of your Roxie cluster. For example, a 100-way Thor to a 20-way Roxie would be acceptable.

This is due to the manner in which data is loaded and processed by Roxie slaves. If data is copied to slave nodes, the file parts are directly copied from source location to target locations. They are NOT split or resized to fit a different sized cluster. Therefore, if you load 50 file parts onto a 40-channel cluster, part one goes to channel one, part two to channel two, etc. Parts 41-50 start at the top again so that part 41 goes to channel 1, and part 42 goes to channel 2, etc. The result is an unevenly distributed workload and would result in reduced performance. A cluster will only perform as fast as its slowest node.

The final consideration is the number of Server processes in a cluster. Each slave must also be a Server, but you can dedicate additional nodes to be only Server processes. This is useful for queries that require processing on the Server after results are returned from slaves. Those Server-intensive queries could be sent only to dedicated Server IP addresses so the load is removed from nodes acting as both Server and slave.

Configuring the Channels

In the illustration below, the nodes are configured using an N+5 scheme to share channels. Channels can be configured in many ways, this is one example.



In this depiction, each enclosure holds five Roxie slave blades (a row of servers in the picture). We will use this example for the rest of this manual.

PreFlight and Roxie Metrics

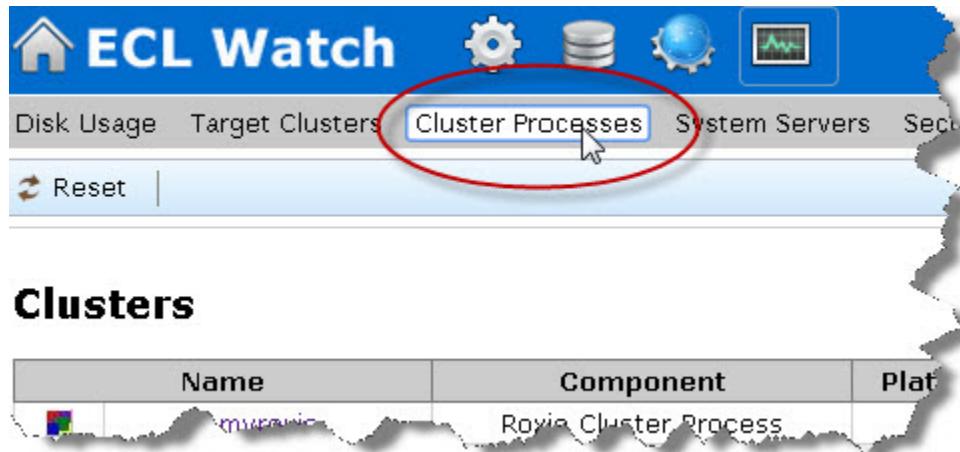
ECL Watch's Operations section provides the ability to perform Preflight activities. These preflight utilities are used for daily health checks, as well as trouble avoidance and troubleshooting. It provides a central location to gather hardware and software information from a remote set of machines and has many uses for day-to-day environment preparation.

This section contains information for performing preflight checks on a Roxie cluster. For details about other components, see the *HPCC System Administrator's Guide*.

Preflight the Roxie Cluster

1. Click on the **Operations** icon then click on the **Cluster Processes** link.

Figure 1. Cluster Processes Link

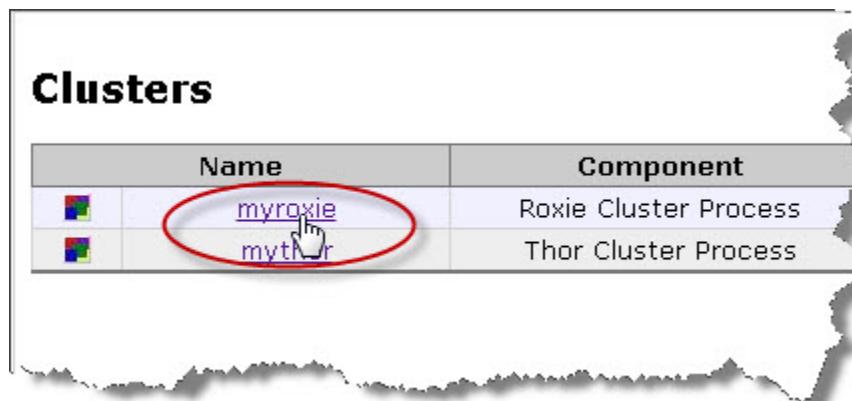


The screenshot shows the ECL Watch interface. At the top, there is a navigation bar with icons for Home, Operations (highlighted with a red circle), Target Clusters, Cluster Processes (highlighted with a red circle), System Servers, and Security. Below the navigation bar is a search bar with a 'Reset' button. The main content area is titled 'Clusters' and displays a table with two rows. The columns are 'Name' and 'Component'. The first row has a 'Name' of 'myroxie' and a 'Component' of 'Roxie Cluster Process'. The second row has a 'Name' of 'myth' and a 'Component' of 'Thor Cluster Process'.

Name	Component
myroxie	Roxie Cluster Process
myth	Thor Cluster Process

2. Click on the **myroxie** link.

Figure 2. myroxie link



The screenshot shows the ECL Watch interface, specifically the 'Clusters' section. A red circle highlights the 'myroxie' link in the 'Name' column of the table. The table has two rows: 'myroxie' (Roxie Cluster Process) and 'myth' (Thor Cluster Process).

Name	Component
myroxie	Roxie Cluster Process
myth	Thor Cluster Process

3. Press the **Submit** button to start preflight.

EXPECTED RESULTS

After pressing Submit, a screen similar to the following should display.

Figure 3. Roxie system information

Roxie Cluster 'myroxie'

	Location	Component	Condition	State	Up Time	Processes Down	/
<input checked="" type="checkbox"/>	10.239.219.5 /var/lib/HPCCSystems/myroxie	Roxie Server	Normal	Ready	6 day(s) 23:27:08	-	51%
<input checked="" type="checkbox"/>	10.239.219.4 /var/lib/HPCCSystems/myroxie	Roxie Server	Normal	Ready	6 day(s) 23:27:10	-	51%

Select All / None

Fetched: 06/13/14 12:09:27

Action: Machine Information ▾

Get processor information Warn if CPU usage is over %

Get storage information Warn if available memory is under % ▾

Local File Systems Only

Get software information Warn if available disk space is under % ▾

Show processes using filter Additional processes to filter:

Auto Refresh every mins



This indicates whether the Roxie nodes are up and running, and some information about them.



If your system has more than 1 Roxie cluster, repeat these steps for each cluster.

Roxie Metrics

HPCC Platform     Wind, User, (ecl^x, file^x, dfu^x)...  LOGGED IN AS: EmilyKate (61.68%)

Disk Usage Target Clusters Cluster Processes System Servers Security Resources

Metrics

IP Address	High Query Active	High Query Count	Last Query Date	Last Query Time	Low Query Active	Low Query Count	Retries	Needed Slaves	Active
10.239.219.4	0	0	20140630	142426	0	3	0	0	0
10.239.219.5	0	0	20140630	142232	0	1	2	1	0
Mean	0	0	20140630	142329	0	2	1	0	0
Std Dev	0	0	0	137.178716	0	1.414214	1.414214	0	0

Auto update metrics when View Columns changing.

View Columns:

<input type="checkbox"/> Aborts Sent	<input type="checkbox"/> Activities Completed	<input type="checkbox"/> Activities Completed/s	<input type="checkbox"/> Activities Started	<input type="checkbox"/> Activities Started/s
<input type="checkbox"/> All Query Active	<input type="checkbox"/> All Query Average Time	<input type="checkbox"/> All Query Count	<input type="checkbox"/> All Query Count/s	<input type="checkbox"/> All Query Failed
<input type="checkbox"/> All Query Failed/s	<input type="checkbox"/> All Query Max Time	<input type="checkbox"/> All Query Min Time	<input type="checkbox"/> Cache Adds	<input type="checkbox"/> Cache Adds/s
<input type="checkbox"/> Cache Hits	<input type="checkbox"/> Cache Hits/s	<input type="checkbox"/> Disk Read Completed	<input type="checkbox"/> Disk Read Started	<input type="checkbox"/> Get Data Buffer Pages
<input type="checkbox"/> Get Data Buffers Active	<input type="checkbox"/> Get Heap Allocated	<input type="checkbox"/> Get Heap Percent Allocated	<input type="checkbox"/> Global Locks	<input type="checkbox"/> Global Signals
<input checked="" type="checkbox"/> High Query Active	<input type="checkbox"/> High Query Average Time	<input checked="" type="checkbox"/> High Query Count	<input type="checkbox"/> High Query Count/s	<input type="checkbox"/> High Query Failed
<input type="checkbox"/> High Query Failed/s	<input type="checkbox"/> High Query Max Time	<input type="checkbox"/> High Query Min Time	<input type="checkbox"/> No Delays Prm	<input type="checkbox"/> No Delays Prm/s
<input type="checkbox"/> No Delays Sec	<input type="checkbox"/> No Delays Sec/s	<input type="checkbox"/> Packets From Self	<input type="checkbox"/> Packets From Self/s	<input type="checkbox"/> Packets Half Worked
<input type="checkbox"/> Packets Half Worked/s	<input type="checkbox"/> Packets Received	<input type="checkbox"/> Packets Received/s	<input type="checkbox"/> Packets Sent	<input type="checkbox"/> Packets Sent/s
<input type="checkbox"/> Packets Too Late	<input type="checkbox"/> Packets Too Late/s	<input type="checkbox"/> Packets Worked	<input type="checkbox"/> Packets Worked/s	<input type="checkbox"/> Index Records Read
<input type="checkbox"/> Index Records Read/s	<input checked="" type="checkbox"/> Last Query Date	<input checked="" type="checkbox"/> Last Query Time	<input type="checkbox"/> Leaf Cache Adds	<input type="checkbox"/> Leaf Cache Adds/s
<input type="checkbox"/> Leaf Cache Hits	<input type="checkbox"/> Leaf Cache Hits/s	<input checked="" type="checkbox"/> Low Query Active	<input type="checkbox"/> Low Query Average Time	<input checked="" type="checkbox"/> Low Query Count
<input type="checkbox"/> Low Query Count/s	<input type="checkbox"/> Low Query Failed	<input type="checkbox"/> Low Query Failed/s	<input type="checkbox"/> Low Query Max Time	<input type="checkbox"/> Low Query Min Time
<input type="checkbox"/> Max Queue Length	<input type="checkbox"/> Max Scan Length	<input type="checkbox"/> Max Slaves Active	<input type="checkbox"/> Mean Scan Length	<input type="checkbox"/> Node Cache Adds
<input type="checkbox"/> Node Cache Adds/s	<input type="checkbox"/> Node Cache Hits	<input type="checkbox"/> Node Cache Hits/s	<input type="checkbox"/> Nodes Loaded	<input type="checkbox"/> Nodes Loaded/s
<input type="checkbox"/> Nginx File Process	<input type="checkbox"/> Packets Abandoned	<input type="checkbox"/> Packets Abandoned/s	<input type="checkbox"/> Packets Received	<input type="checkbox"/> Packets Received/s

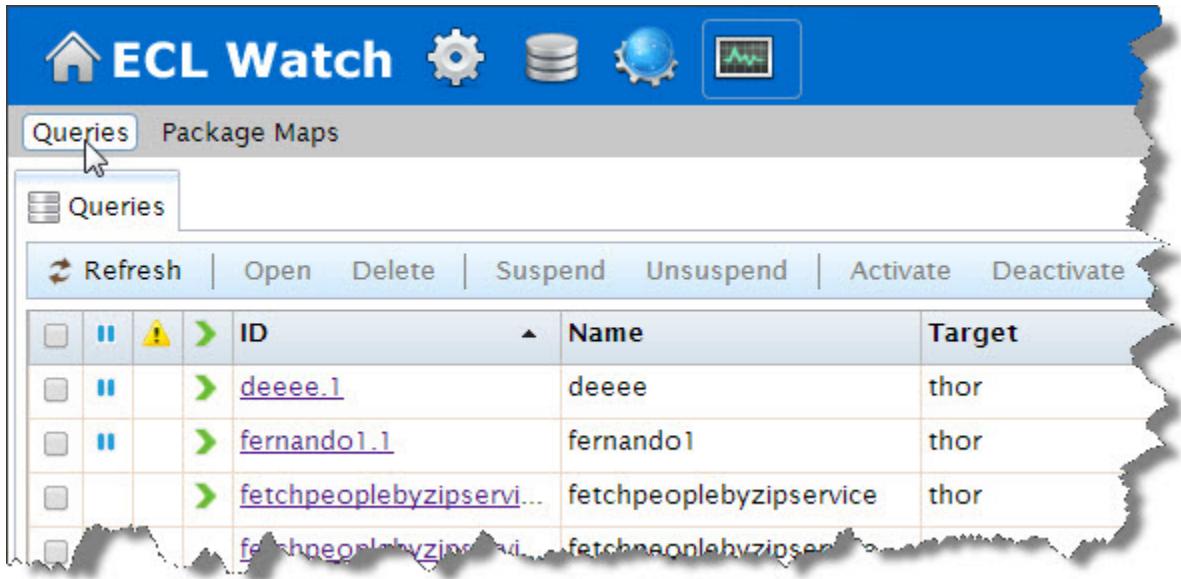
IP Address	The IP address of the Server node
slavesActive	Number of slaves active for that Server
lastQueryTime	Time stamp of most recent query executed
loQueryActive	Number of low priority queries active
loMin	Minimum time (ms) it took to run a low priority query
retriesNeeded	Count of all reply packets that arrived to a Server as a response to more than one try.
loMax	Maximum time (ms) it took to run a low priority query
hiQueryActive	Number of high priority queries active
loQueryCount	Total number of low priority queries run
loQueryAverage	Average time it took to run a high priority query
hiQueryCount	Total number of high priority queries run
hiMax	Maximum time (ms) it took to run a high priority query
hiMin	Minimum time (ms) it took to run a high priority query

heapBlocksAllocated Number of times Roxie had to allocate memory from its memory allocator

Queries Page in ECL Watch

The Queries page lists published queries for each target cluster. On this page you can see the published queries . You can also perform some actions on the selected queries.

Figure 4. Browse Query Sets



ID	Name	Target
deeee.1	deeee	thor
fernando1.1	fernando1	thor
fetchpeoplebyzipservi...	fetchpeoplebyzipservice	thor
fetchpeoplebyzipservi...	fetchpeoplebyzipser...	thor

The Queries interface provides some information at a glance, there are three columns on the left side of each listed query. These three columns provide information about these queries.

	Indicates a paused query
	Indicates an activated query
	Indicates a query suspended by the system

The queries page also provides other information at a glance:

- the query ID
- the query name
- the target
- the workunit id (WUID)
- the dll
- Published by

All the above available at a glance on the main queries page, with further actions that can be performed from the action buttons along the top of the tab. You can sort a column by clicking on the column heading. Click once for ascending, click again to toggle to descending. The direction of the arrow indicates the sort order.

To see the details page for a particular query, or to perform some action on it you must select it. You can select a query or queries by checking the check box. You can also open a particular query by double clicking on it.

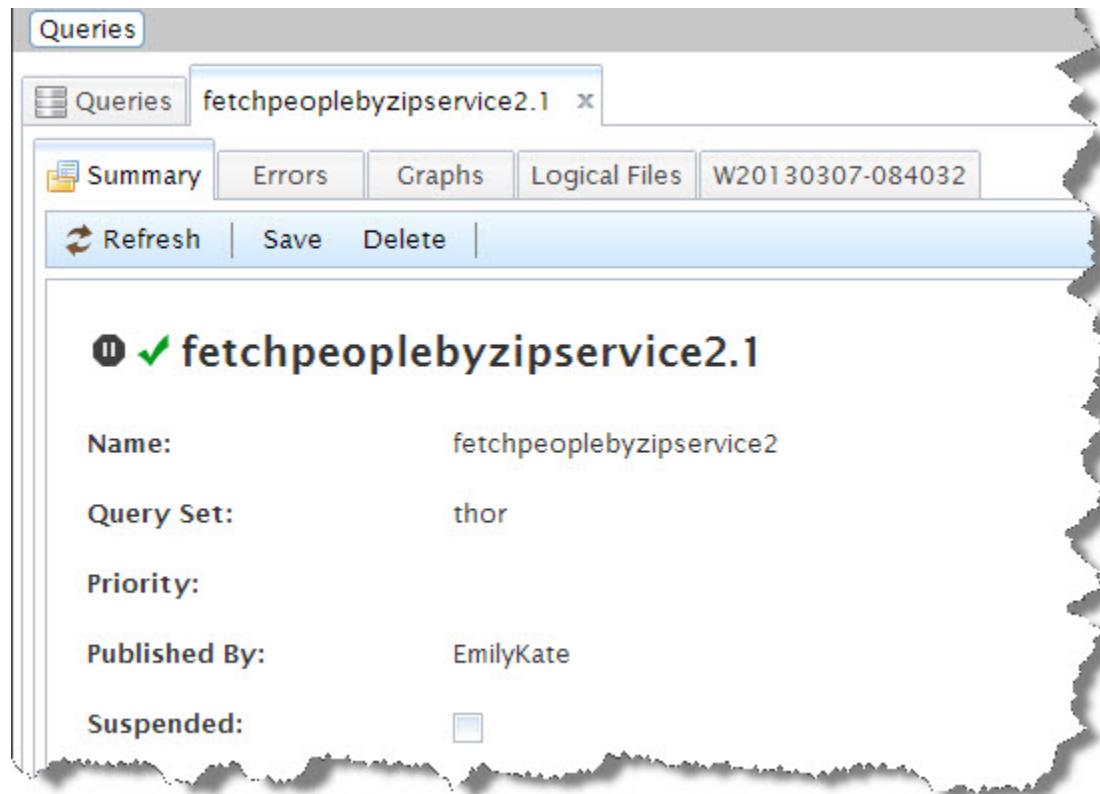
Query Details

To examine the Query Details page, you select and open the query or queries. This opens a tab containing the query details. From the query details page you can get more information about the specific query. You can also perform some actions on that query. There are several tabs with additional information about the selected query.

Query Summary Tab

The default query tab opened when you select a query is the Summary tab. The summary tab shows you some detail information about the query.

Figure 5. Query detail page



There are a few actions that you can perform on the query from this tab. Press the action buttons for the desired activity for the selected query.

Refresh Refreshes the information displayed for the selected query.

Save Saves the selected query (or queries).

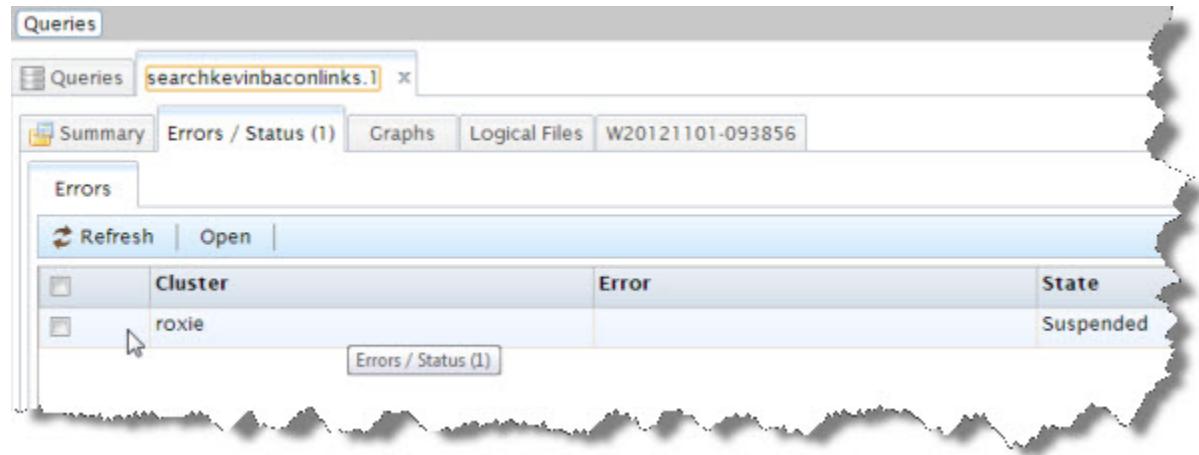
Delete Deletes the selected query (or queries).

Errors Tab

For each selected query there is an Errors tab. The Errors tab displays any errors that may have been encountered during the compiling and publishing of that query. If there aren't any errors the errors tab will be blank. If there are

errors, you can further examine any specific error by checking the box and selecting it, and then press the open action button. You could also just double click on the selected error.

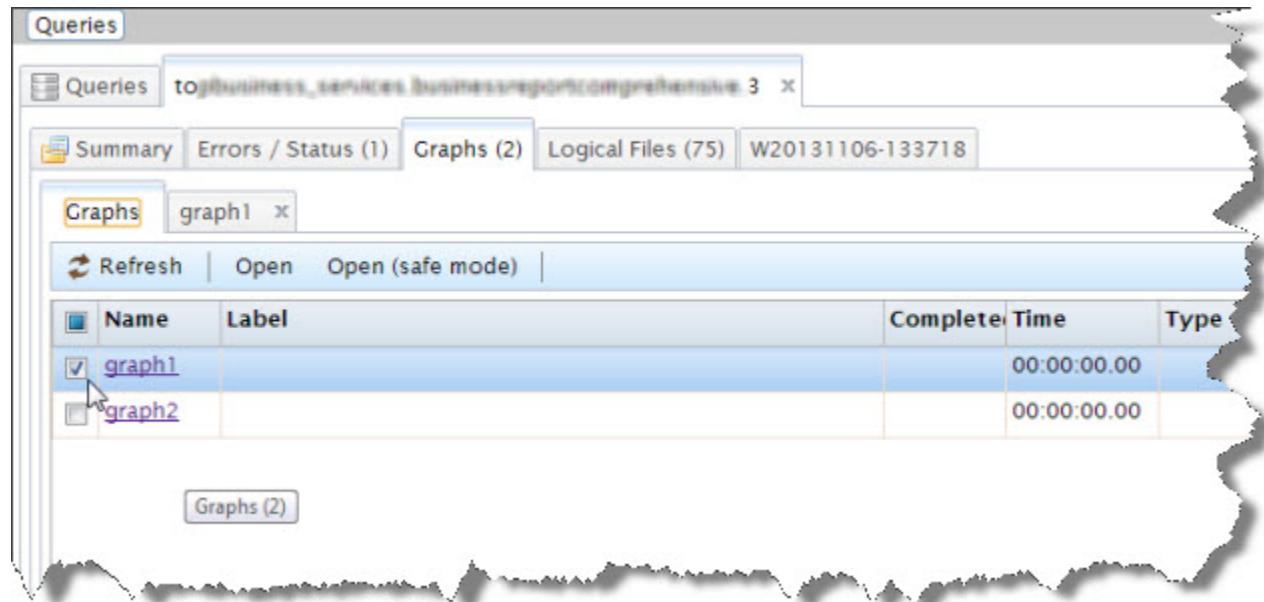
Figure 6. Query Error



Graphs Tab

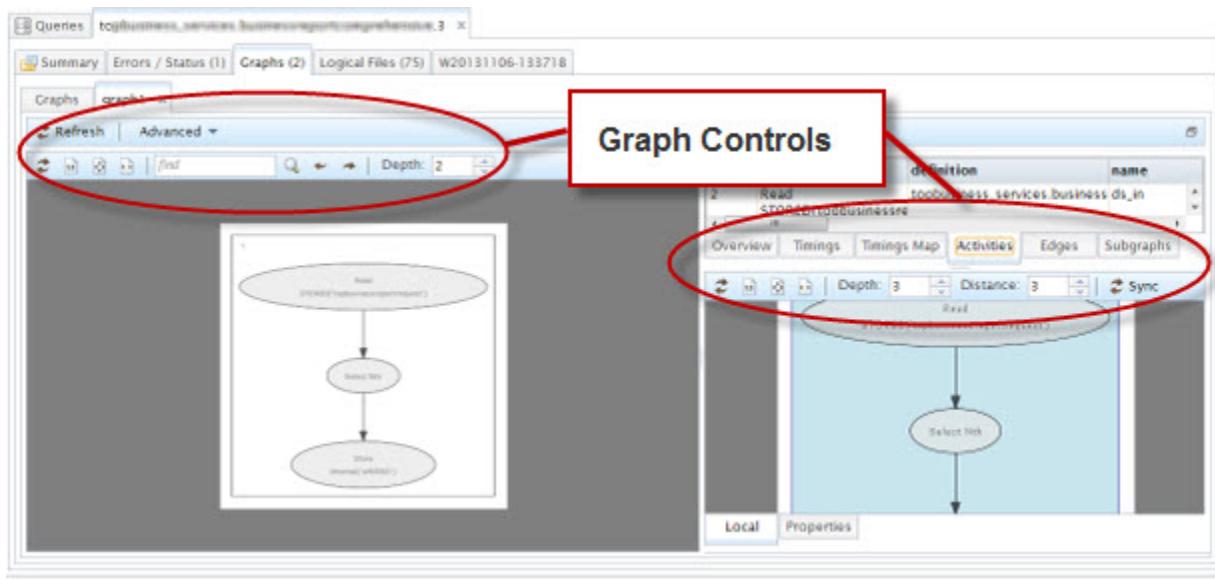
The graphs tab provides access to graphical interpretations of the query. This can be helpful in visualizing how the query ran. The graphs tab displays a list of any graphs generated by the selected query, along with some additional information like timing. To display a specific graph, you must select it, and choose to open it, or you can double click on listed graph.

Figure 7. Graphs list



Opening a graph will open a new tab showing the selected graph(s).

Figure 8. Graphs



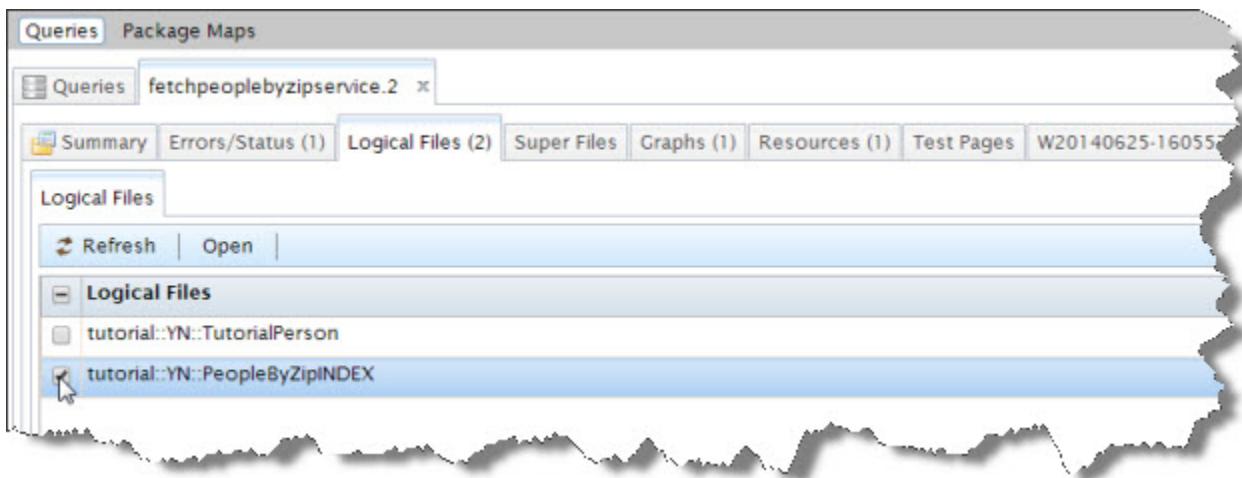
When you open a graph the visible area splits into three smaller sections each displaying some relevant component of the query graph. Notice the myriad of graph controls, and tabs in the border area of each tab. Manipulate these controls to view different aspects of the graphs.

The Advanced action button on the main graph control area, provides access to even more advanced graphing options.

Logical Files Tab

The Published queries details page provides a link to the queries Logical Files tab. The Logical Files tab shows all logical files that are used by the query. To view the logical file details for any file listed, select one or more files by checking the checkbox and press the Open action button. Tabs for each file selected opens where you can view and make changes to the file(s) without the need to go back to the logical files page.

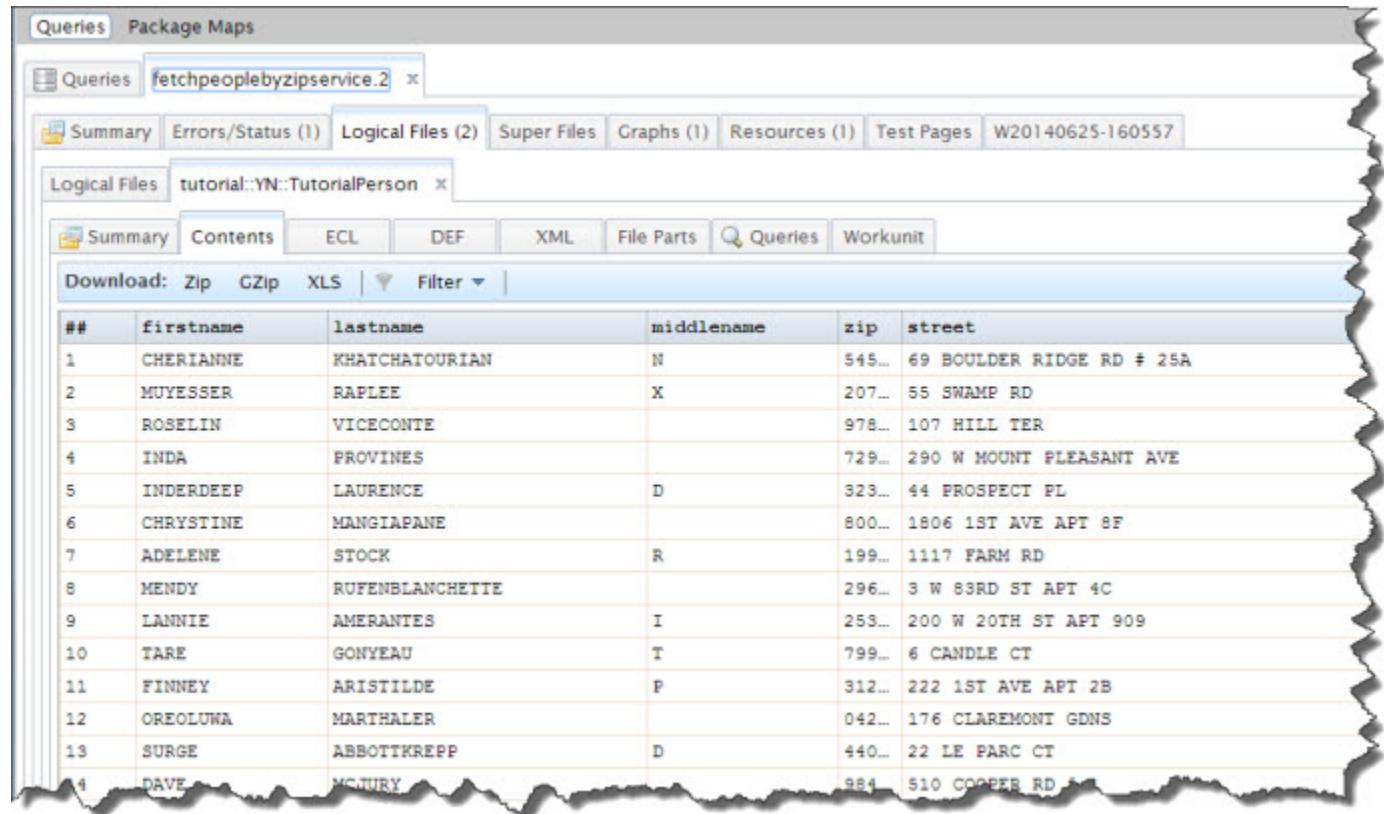
Figure 9. Queries:Logical Files Tab



The above image shows the list of Logical files on the Logical Files tab. To view more detail about a logical file listed here, check the box next to the file, and then press the **Open** action button. You can also just double click on the logical file you want to view.

Once open, you can select any of the tabs to see Summary, Contents, ECL, DEF, XML, File Parts, Queries, or the Workunit.

Figure 10. Queries:Logical Files:Contents Tab

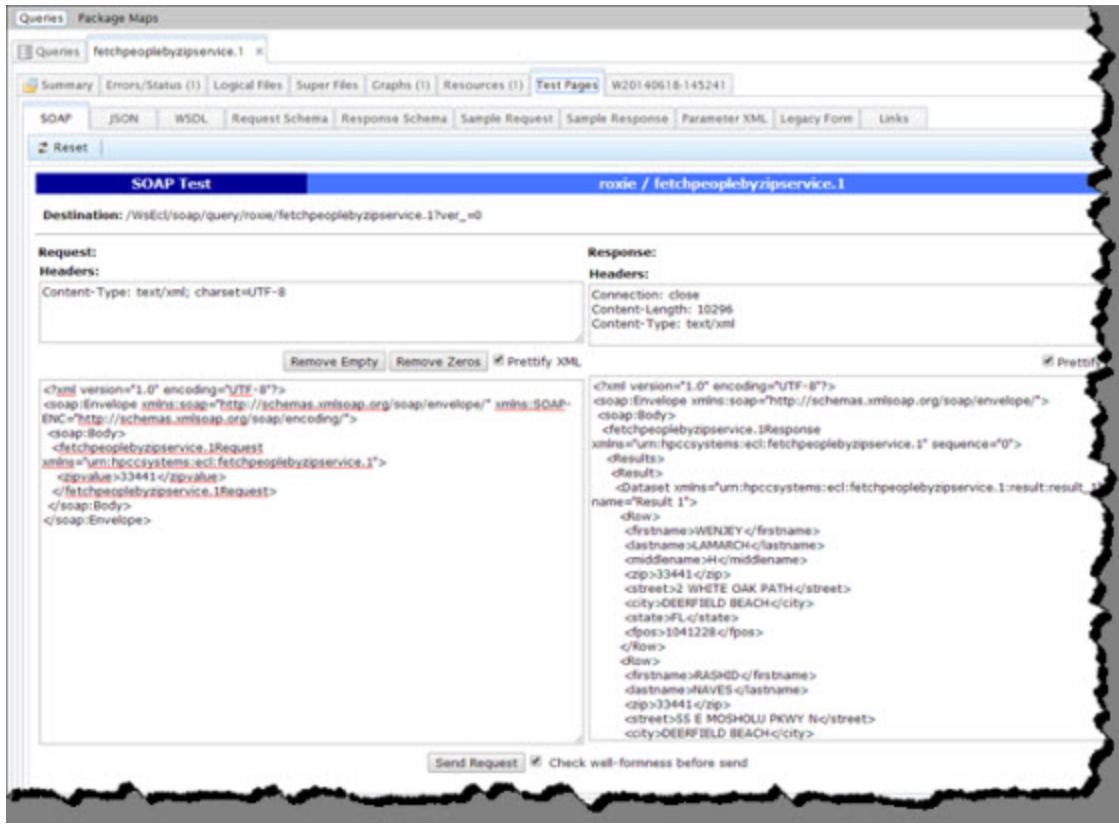


#	firstname	lastname	middlename	zip	street
1	CHERIANNE	KHATCHATOURIAN	N	545...	69 BOULDER RIDGE RD # 25A
2	MUYESSER	RAPLEE	X	207...	55 SWAMP RD
3	ROSELIN	VICECONTE		978...	107 HILL TER
4	INDA	PROVINES		729...	290 W MOUNT PLEASANT AVE
5	INDERDEEP	LAURENCE	D	323...	44 PROSPECT PL
6	CHRYSTINE	MANGIAPANE		800...	1806 1ST AVE APT 8F
7	ADELENE	STOCK	R	199...	1117 FARM RD
8	MENDY	RUFENBLANCHETTE		296...	3 W 63RD ST APT 4C
9	LANNIE	AMERANTES	I	253...	200 W 20TH ST APT 909
10	TARE	GONYEAU	T	799...	6 CANDLE CT
11	FINNEY	ARISTILDE	P	312...	222 1ST AVE APT 2B
12	OREOLUNA	MARTHALER		042...	176 CLAREMONT GDNS
13	SURGE	ABBOTTKREPP	D	440...	22 LE PARC CT
14	DAVE	MCJURY		984...	510 COOPER RD

Test Pages

The Test Pages tab provides a number of resources you can use to test your query including SOAP/JSON/WSDL and the legacy WS-ECL form, as well as other tabs showing useful information or sample details about the query.

Figure 11. Test Pages tab



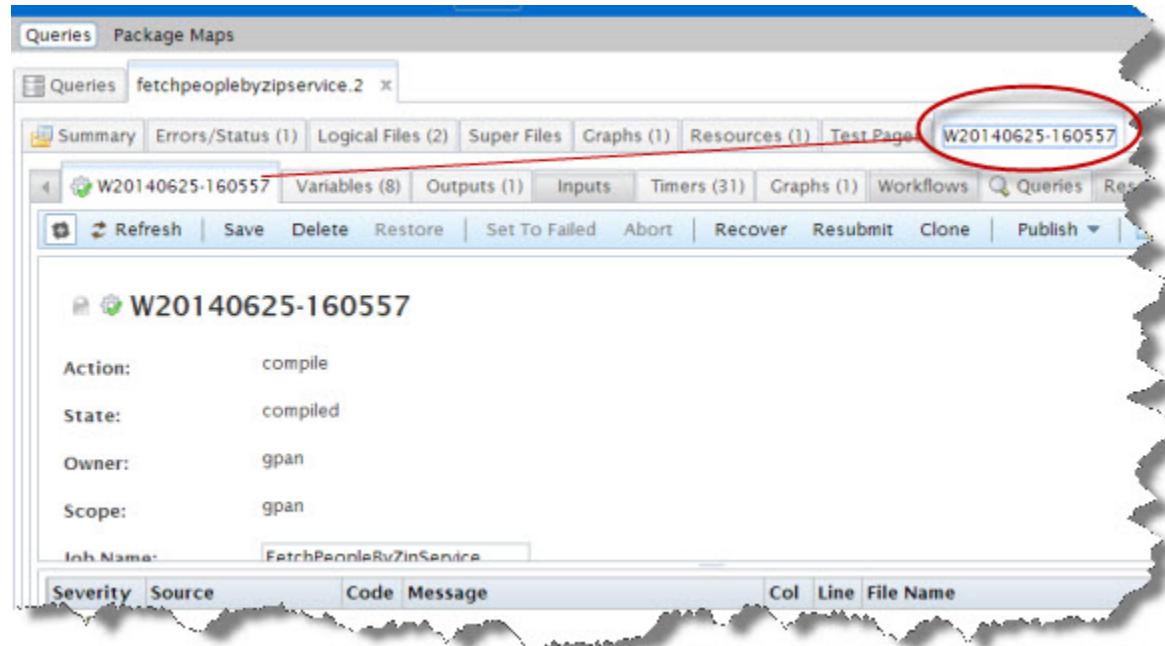
Information available from the Test pages tab.

- **SOAP** This tab provides an interactive interface to submit the query (with input data) and see the response in XML format.
- **JSON** This tab provides an interactive interface to submit the query (with input data) and see the response in JSON format.
- **WSDL** This tab provides a WSDL definition describing the functionality offered by the query (web service).
- **Request Schema** This tab provides a schema in XSD format describing a request for the query (web service).
- **Response Schema** This tab provides a schema in XSD format describing a response from the query (web service).
- **Sample Request** This tab provides a sample request for the query (web service) in XML Format.
- **Sample Response** This tab provides a sample response from the query (web service) in XML Format.
- **Parameter XML** This tab provides Parameterized XML representation of the query interface.
- **Legacy Form** This tab provides a form that can be used to submit a query and get a response. This is similar to the WsECL form.
- **Links** Provides a list of useful links such as: the Form, a sample REST URL, sample request, sample response, parameter XML, SOAP POST, WSDL, XSD, and the result schema.

The Workunits link

The Published queries details page provides a link to the workunits page. This tab is a shortcut that takes you to the same workunits tab you can get to through the ECL workunits menu.

Figure 12. Queries Workunit

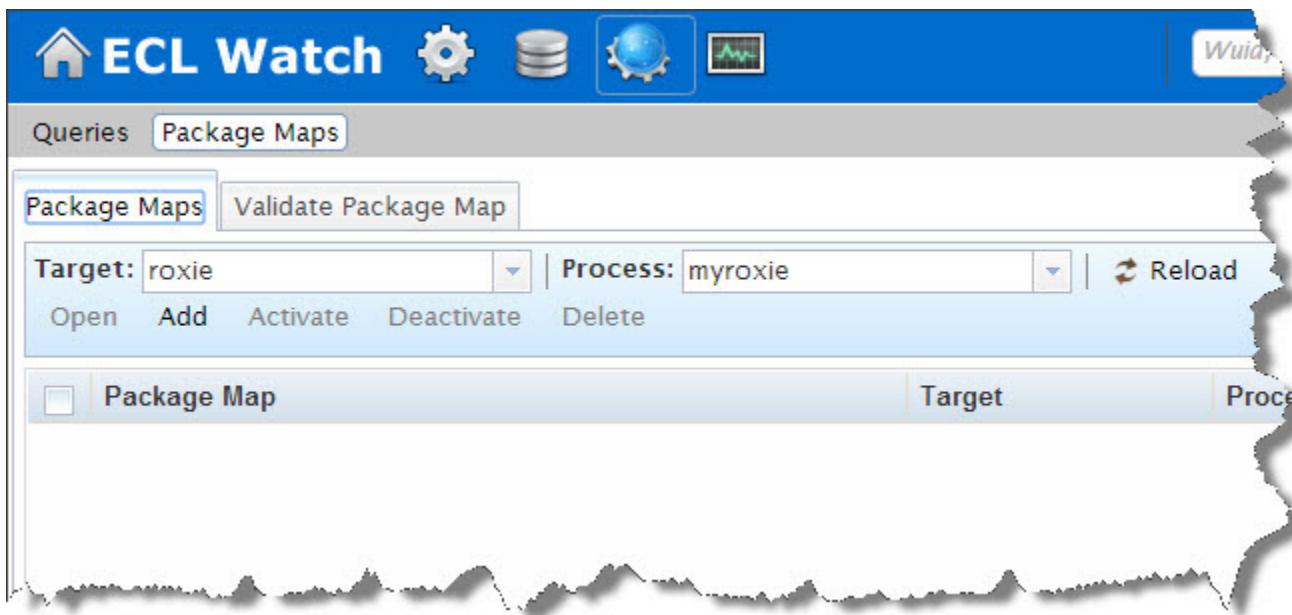


This is the same as the **ECL Workunits** page from the navigation sub-menu Workunits link. You can perform the same operations here. Notice that there are some other familiar tabs here as well, for example the Graphs tab, both from the Queries details page, and from the workunit tab nested here.

Package Maps

From the Queries icon link, you can access the package maps page. Press the **Package Maps** button on the navigation sub-menu bar, to access to the Package Maps on your cluster.

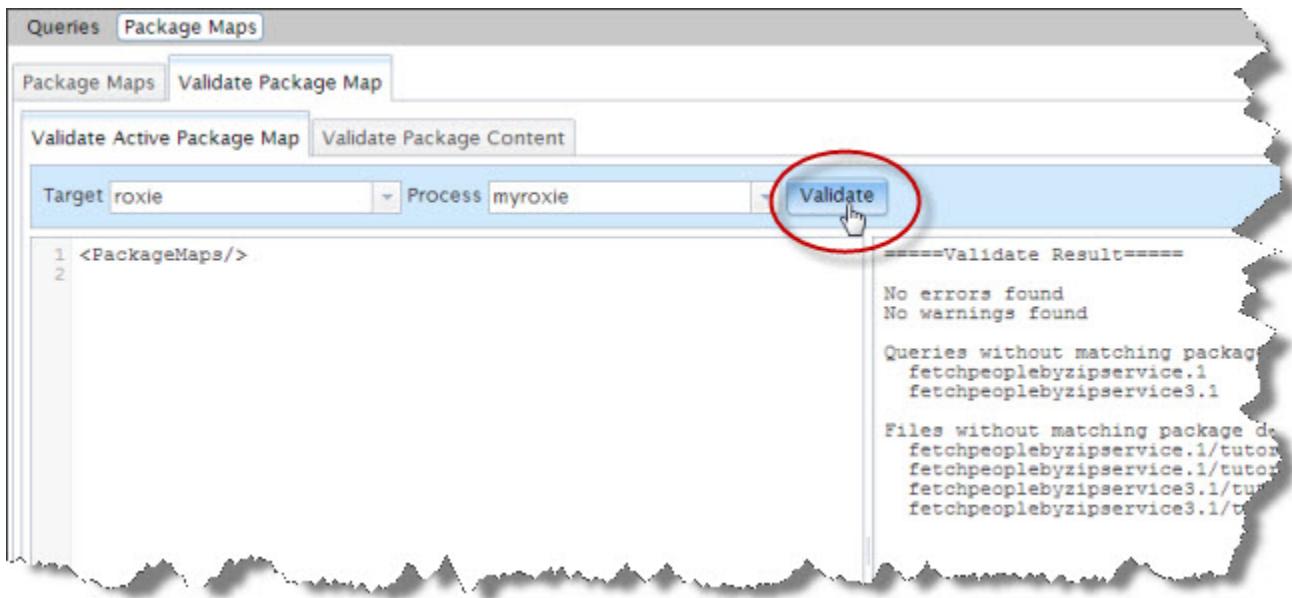
Figure 13. Package Maps



The package maps page, displays all the package maps loaded on your cluster. You can Add, Activate, Deactivate, Delete, or Open a package map. To examine a package map, select a package map from the list.

You can select the **Validate Package Map** tab to validate a package map.

Figure 14. Validate Package Maps



Choose the **Target** and **Process** from the drop lists on the Validate Package Map tab.

Press the **Validate** button to validate the package map. The result is shown on the **Validate Active Package Map** tab. You can also Validate the package content, on the **Validate Package Content** tab.