HPCC Systems®

Dynamic ESDL

Boca Raton Documentation Team

LexisNexis® Risk Solutions

Dynamic ESDL

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com>

Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2018 Version 7.0.4-1

Dynamic ESDL 4
Dynamic ESDL Workflow Tutorial 5
Before You Begin... 5
Overview 6
Configure and Bind the dESDL Service 7
Write the ESDL Service Definition 11
Writing the ECL 15
Publish the ESDL Service Definitions and Bind the ESDL Service 18
ESDL Command Line Interface 21
The ESDL Command Syntax 21

Dynamic ESDL

Dynamic ESDL (Enterprise Service Description Language) is a methodology that helps you develop and manage web-based query interfaces quickly and consistently.

Dynamic ESDL takes an interface-first development approach. It leverages the ESDL Language to create a common interface "contract" that both Roxie Query and Web interface developers will adhere to. It is intended to allow developers to create production web services, with clean interfaces that can evolve and grow over time without breaking existing applications.

ESDL's built-in versioning support helps ensure compiled and deployed applications continue to operate while changes are made to the deployed service's interface for new functionality.

ESDL's ability to define and reuse common structures helps maintain consistent interfaces across methods.

The Dynamic ESDL service is built to scale horizontally, and hooks are provided to add custom logging and security to help create fully "productionalized" web services.

Once a service is deployed, application developers and end-users can consume the service using REST, JSON, XML, SOAP, or form encoded posts. Dynamic ESDL provides quick and easy access to a WSDL, live forms, sample requests and responses, and testing interfaces to allow developers to test logic changes, data changes, or new features, as well as to interact with the service directly using SOAP, XML, or JSON.

Dynamic ESDL Workflow Tutorial

Before You Begin...

You will need:

- Access to an HPCC Cluster (version 6.4.2 or later) that you can reconfigure using Configuration Manager. This can be one running in a Virtual Machine.

  You should have a basic understanding of Configuration Manager and how to use it. You also need access to the server(s) with sufficient rights to modify the environment.

- Access to ECL Watch and WsECL (using a browser).

  For purposes of this tutorial, we assume that you know how to submit a published using WsECL.

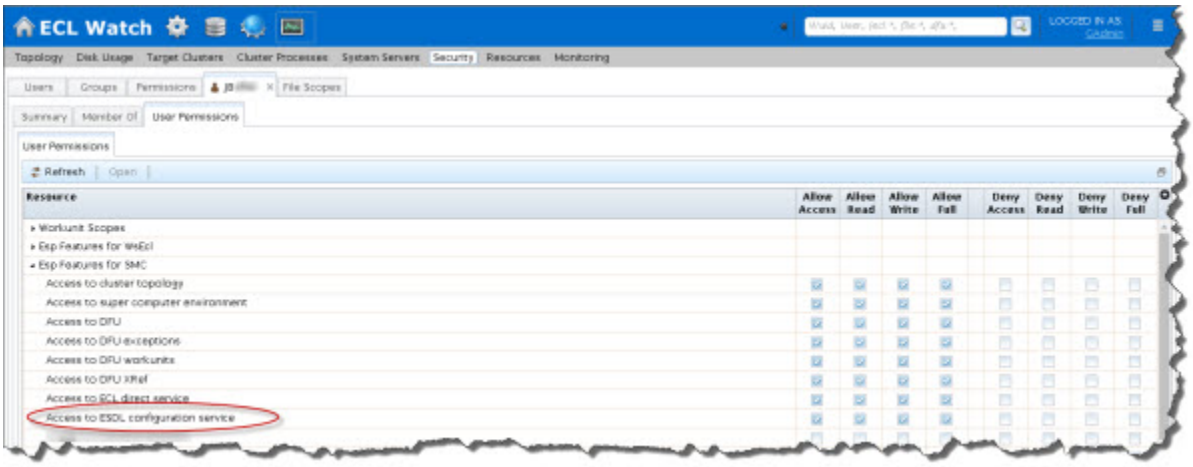
- The ECL IDE (version 6.4.2 or later)

  You won't need to know the ECL or ESDL languages to follow the steps in this book, but you will need to understand both to develop dESDL-based applications.

  For purposes of this tutorial, you should know the basics of using the ECL IDE including how to add files to your repository, how to compile ECL Code, and how to publish a compiled query.

dESDL and LDAP Security

If your HPCC platform is configured to use LDAP security, you must ensure any user who will be publishing ESDL Definitions has Access to **ESDL configuration service** set to **Allow Full**, as shown below.



Overview

In this tutorial, we will:

- Use Configuration Manager to add a Dynamic ESDL-based ESP Service and bind it to a port on an ESP server.
- Write an ESDL Service Definition using the editor in the ECL IDE.
- Generate ECL from the ESDL Service Definition within the ECL IDE.

  This step automatically generates an ECL file in your ECL repository. You will use the definitions in that ECL file when you write the ECL query that will deliver the result (the business logic).
- Write the ECL for the business logic of the query, compile it, and then publish the query to a Roxie cluster.
- Publish the Dynamic ESDL definition from the ECL IDE.
- Bind the service methods to the Roxie queries in ECLWatch using an XML formatted configuration.

Configure and Bind the dESDL Service

In this portion of the tutorial, we will add an ESP service and a service binding that reserves a port for the dynamic ESDL service.

This step is independent of the development and publishing of the actual Roxie query, so you can set it up before or after your query is ready.

1. If it is running, stop the HPCC system, using this command in a terminal window:

   ```
   sudo systemctl stop hpccsystems-platform.target
   ```

   You can use this command to confirm HPCC processes are stopped:

   ```
   sudo systemctl status hpccsystems-platform.target
   ```

2. Start the Configuration Manager service.

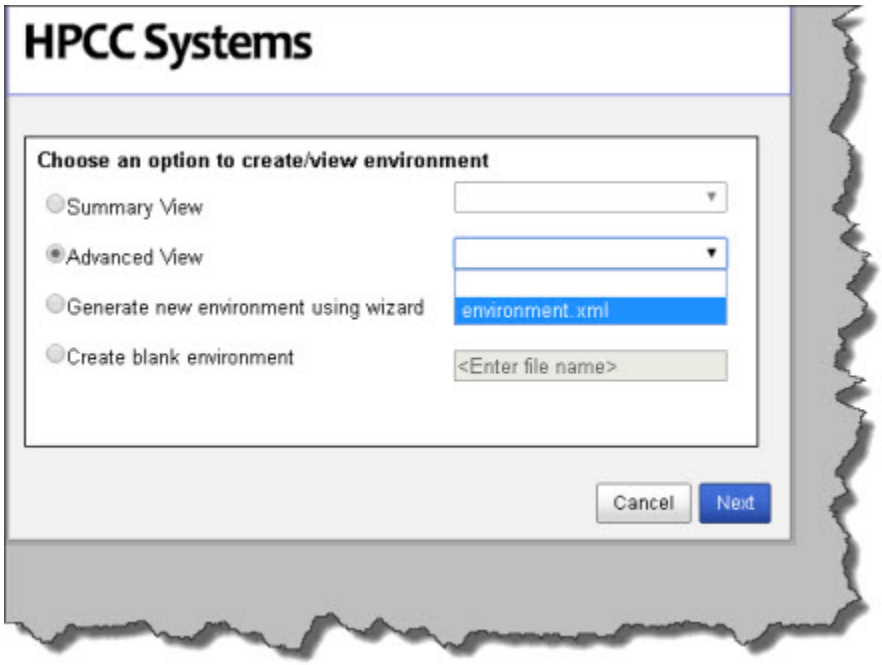
   ```
   sudo /opt/HPCCSystems/sbin/configmgr
   ```

3. Using a Web browser, go to the Configuration Manager's interface:

   ```
   http://<node ip >:8015
   ```

   The Configuration Manager startup wizard displays.

4. Select **Advanced View** and then select the source environment XML file to edit.

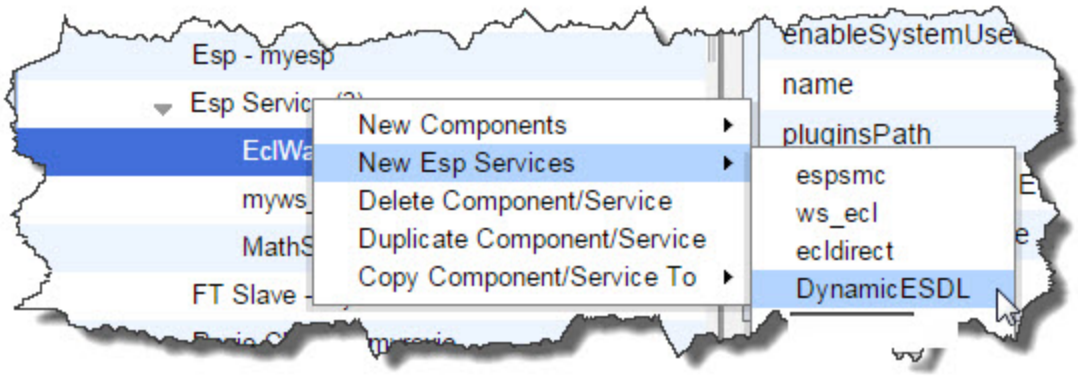


5. Press the **Next** button.

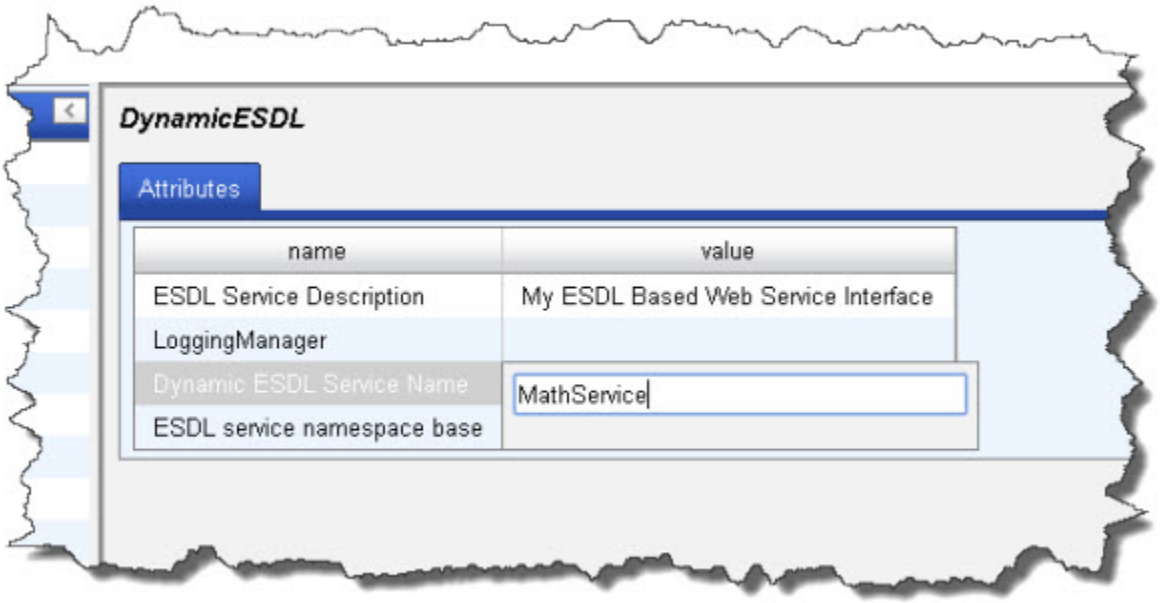
6. Check the **Write Access** box at the top of the page.

   Default access is read-only. Many options are only available when write-access is enabled.

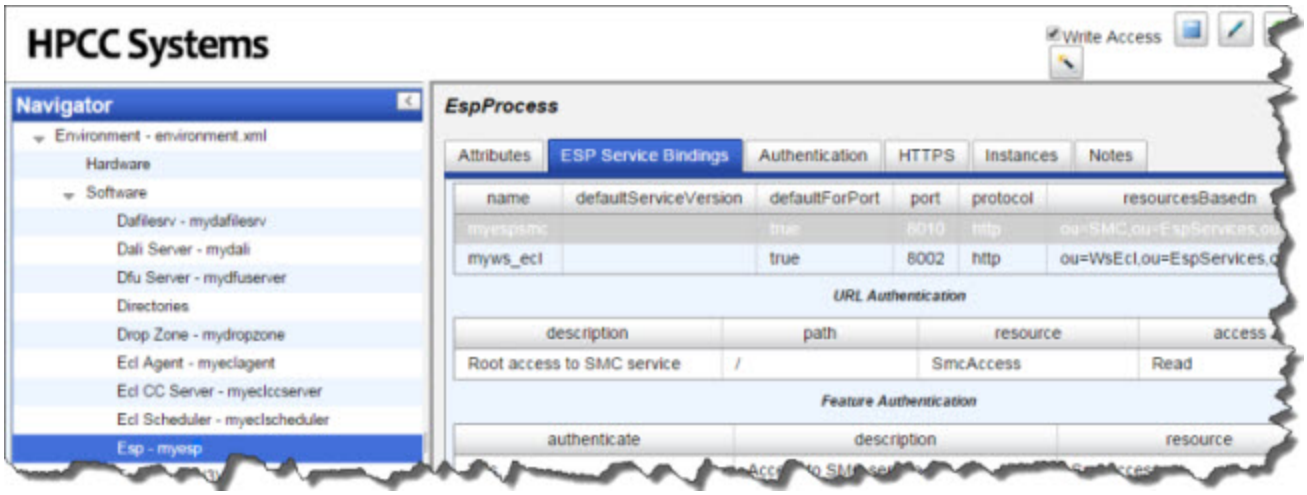
7. Right-click on **Esp Services** and select **Dynamic ESDL** from **New Esp Services.**



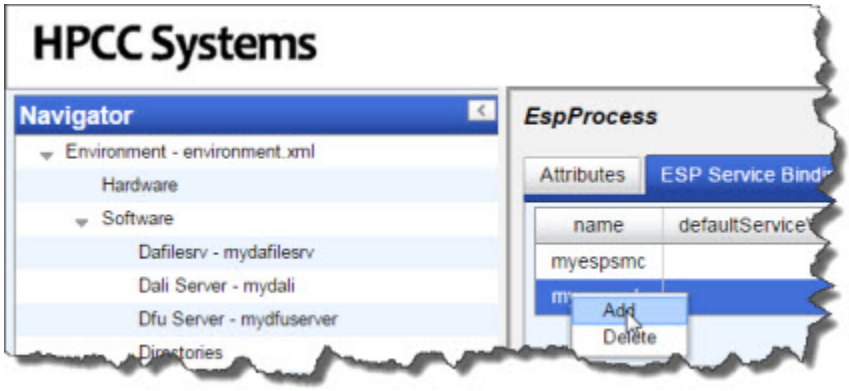
8. Change the name for the service. For this tutorial, let's name it **MathService**.



9. Select your ESP, then select the **ESP Service Bindings** tab.



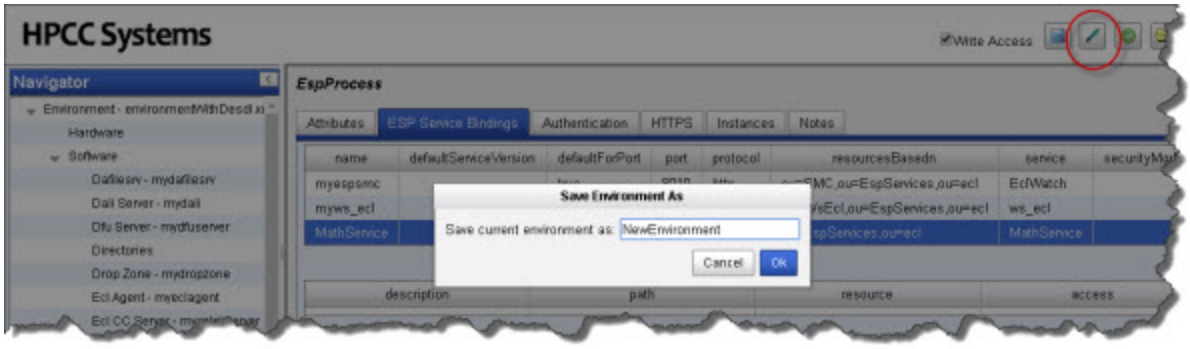
10. Right-click in the list of bindings and select **Add**



11. Provide a name, port, and then select the service from the drop list.

    For this tutorial, name it **MathService**, set the port to **8003**, and select **MathService** from the service drop list. This is service definition you added in an earlier step.

12. Press the Save As button and name your new XML file **NewEnvironment**.



13.Return to the terminal window and press **ctrl+c** to close the Configuration Manager.

14.Copy the NewEnvironment.xml file from the source directory to the /etc/HPCCSystems and rename the file to environment.xml

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

Make sure that you have sufficient privileges to write file(s) to the destination directory before attempting to copy. If prompted to overwrite the destination file, you should answer **yes**.

15.Copy the **/etc/HPCCSystems/environment.xml** to **/etc/HPCCSystems/** on **every** node.

```
sudo /opt/HPCCSystems/sbin/hpcc-push.sh -s <sourcefile> -t <destinationfile>
```

16.Restart the HPCC system on **every** node. The following command starts the HPCC system on an individual node:

```
sudo systemctl start hpccsystems-platform.target
```

You may want to create a script to push this command out to every node. A sample script is provided with HPCC. Use the following command to start HPCC on all nodes:

```
sudo systemctl start hpccsystems-platform.target
```

Write the ESDL Service Definition

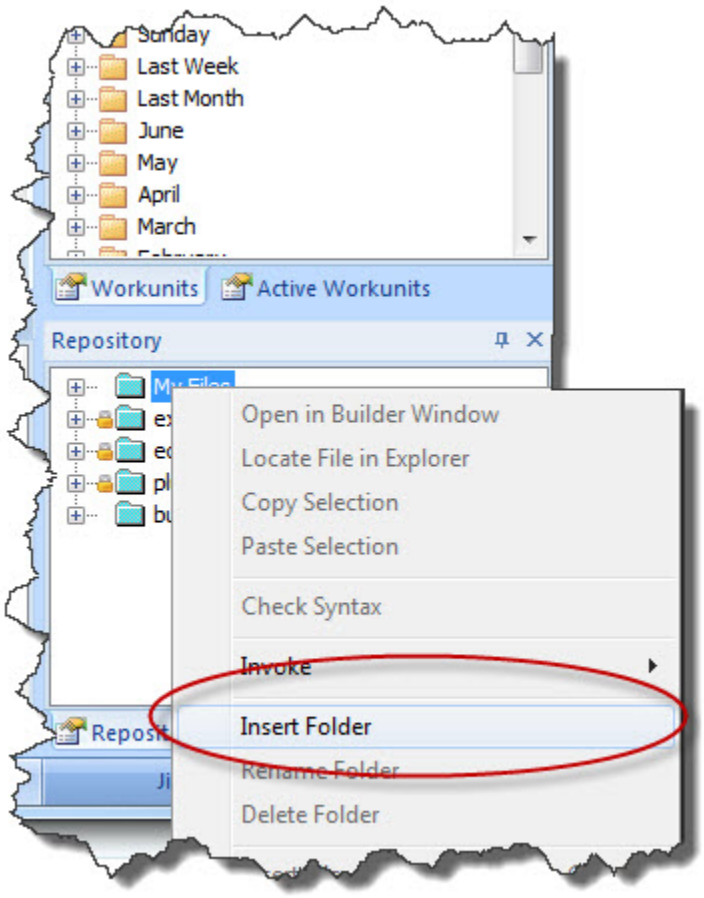
In this portion of the tutorial, we will write the Service Definitions in the ECL IDE. The program listing below shows an ESDL service called *MathService*. It contains one method, *AddThis*, with a request and a response defined.

1. Start the ECL IDE (Start >> All Programs >> HPCC Systems >> ECL IDE)

2. Log in to your environment

3. Right-click on the **My Files** folder in the Repository window, and select **Insert Folder** from the pop-up menu.

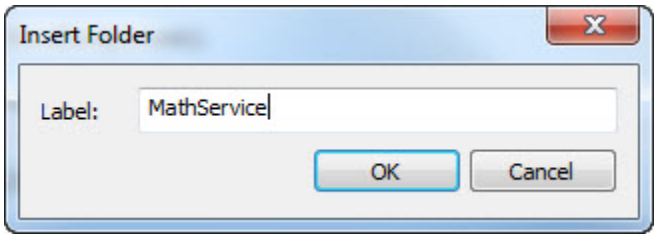
Figure 1. Insert Folder



For purposes of this tutorial, let's create a folder called **MathService**.

4. Enter **MathService** for the label, then press the **OK** button.

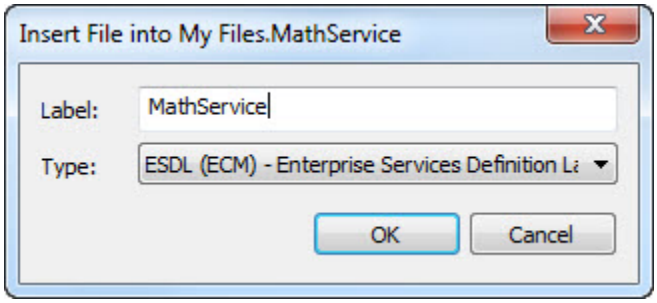
Figure 2. Enter Folder Label



5. Right-click on the **MathService** folder, and select **Insert File** from the pop-up menu.

6. Enter **MathService** for the label, select *ESDL* as the **Type**, then press the **OK** button.

Figure 3. Insert File



An Editor Window opens.

Figure 4. ESDL file

Notice that some text has been written for you in the window.

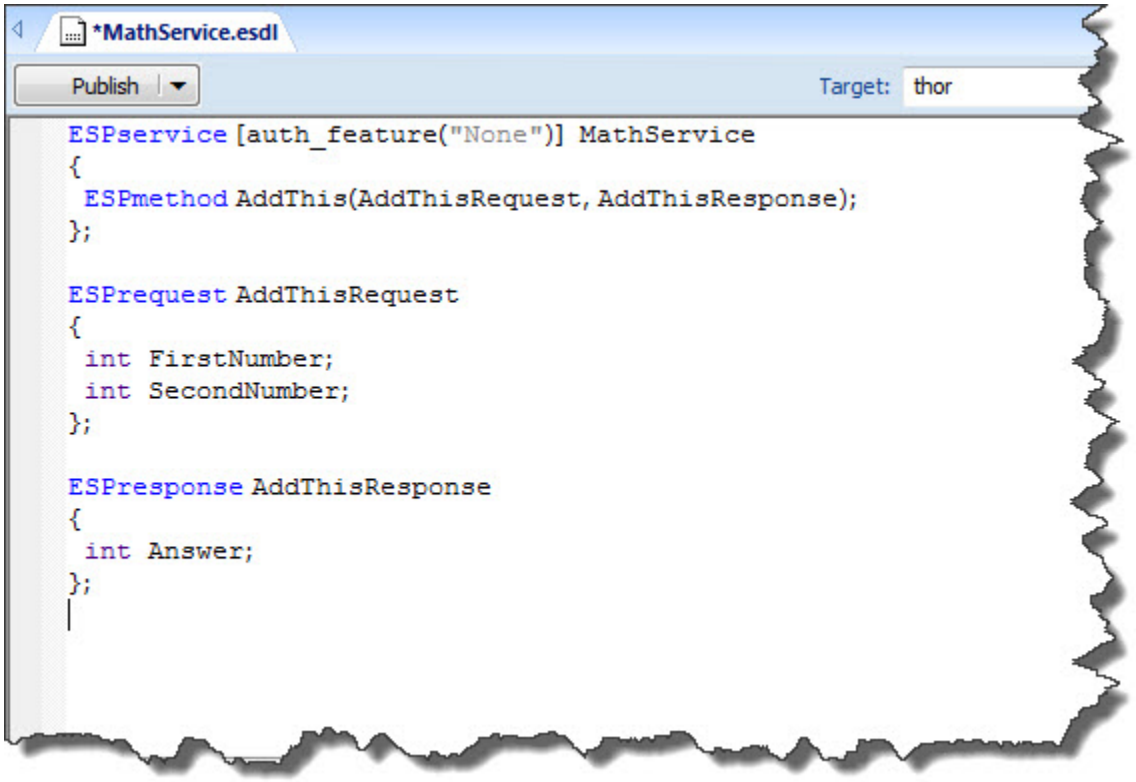
7. Write the following code, replacing what was written, in the editor workspace :

```
ESPservice [auth_feature("None")]  MathService
{
  ESPmethod AddThis(AddThisRequest, AddThisResponse);
};

ESPrequest AddThisRequest
{
  int  FirstNumber;
  int  SecondNumber;
};

ESPresponse AddThisResponse
{
  int  Answer;
};
```

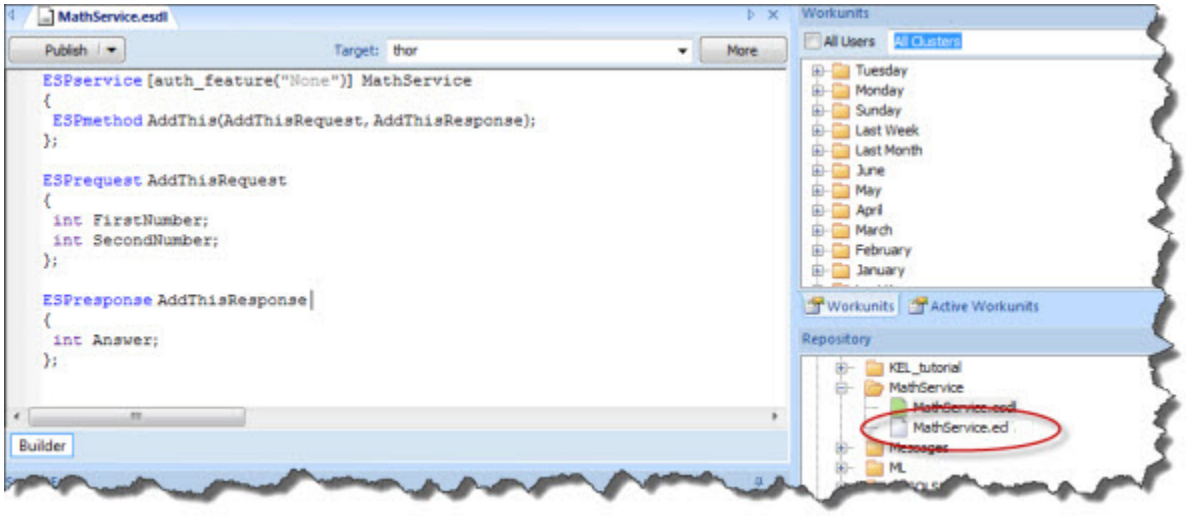
Figure 5. ESDL Code in Editor Window



8. Save the file using **ctrl+s** or the **File** >> **Save** menu option.

   Notice that a new ECL file is now in the repository folder. Saving the ESDL file automatically generated a file named *MathService.ecl* in the current directory. You could generate this ECl using the dropdown button and selecting **Generate ECL**.

Figure 6. Saving an ESDL file generates ECL



This provides the ECL Structures you will IMPORT and use in the ECL code you write to support the service method.

Writing the ECL

First, let's examine the generated ECL code in MathService.ecl.

```
/*** Not to be hand edited (changes will be lost on re-generation) ***/
/*** ECL Interface generated by esdl2ecl version 1.0 from MathService.xml. ***/
/*=================================================*/

export MathService := MODULE

export t_AddThisRequest := record
  integer FirstNumber {xpath('FirstNumber')};
  integer SecondNumber {xpath('SecondNumber')};
end;

export t_AddThisResponse := record
  integer Answer {xpath('Answer')};
end;
end;

/*** Not to be hand edited (changes will be lost on re-generation) ***/
/*** ECL Interface generated by esdl2ecl version 1.0 from MathService.xml. ***/
/*=================================================*/
```

Notice it created a file named MathService.ecl which has defined a MODULE named MathService. Remember in ECL, the name of the file (MathService) *must always exactly match* the name of the single EXPORT definition (MathService) contained in that file.

Next, we will write the ECL code to support the functionality of the AddThis method. We will IMPORT the Math-Service module and reference the request and response structures.

1. Right-click on the **MathService** Folder, and select **Insert File** from the pop-up menu.

2. Enter **AddThis** for the label, select *ECL* as the **Type**, then press the **OK** button.

   An Editor Window opens.

3. Write ECL to support the service:

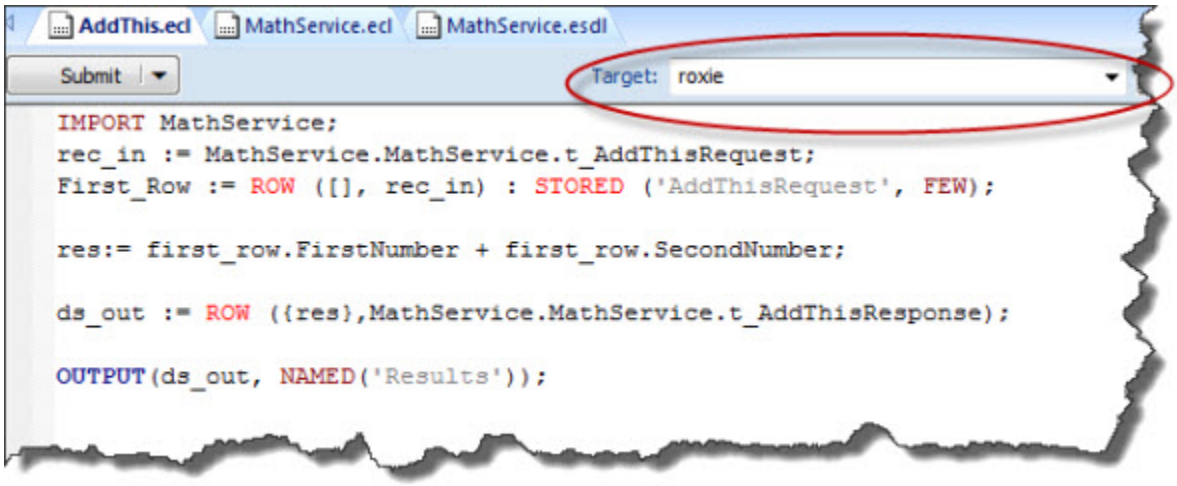
```
IMPORT MathService;
rec_in := MathService.MathService.t_AddThisRequest;

First_Row := ROW ([], rec_in) : STORED ('AddThisRequest', FEW);

res:= first_row.FirstNumber + first_row.SecondNumber;
ds_out := ROW ({res},MathService.MathService.t_AddThisResponse);
OUTPUT(ds_out, NAMED('AddThisResponse'));
```

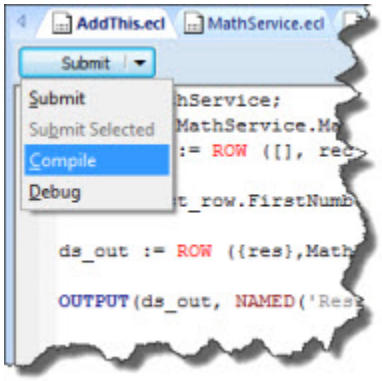
4. Using the **Target** drop list, select *Roxie* as the Target cluster.

Figure 7. Target Roxie



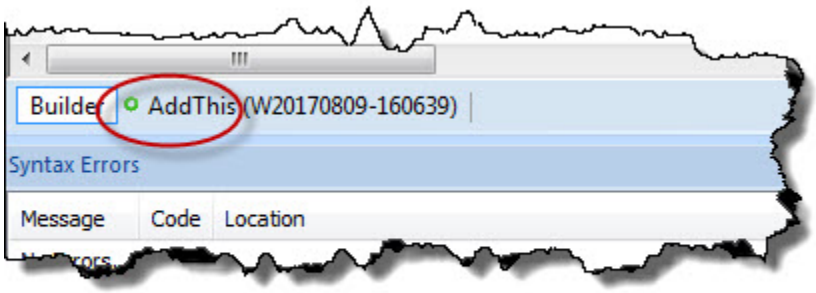
5. In the Builder window, in the upper left corner the **Submit** button has a drop down arrow next to it. Select the arrow, then select **Compile**.

Figure 8. Compile



6. When the workunit finishes, it will display a green circle indicating it has compiled.

Figure 9. Compiled

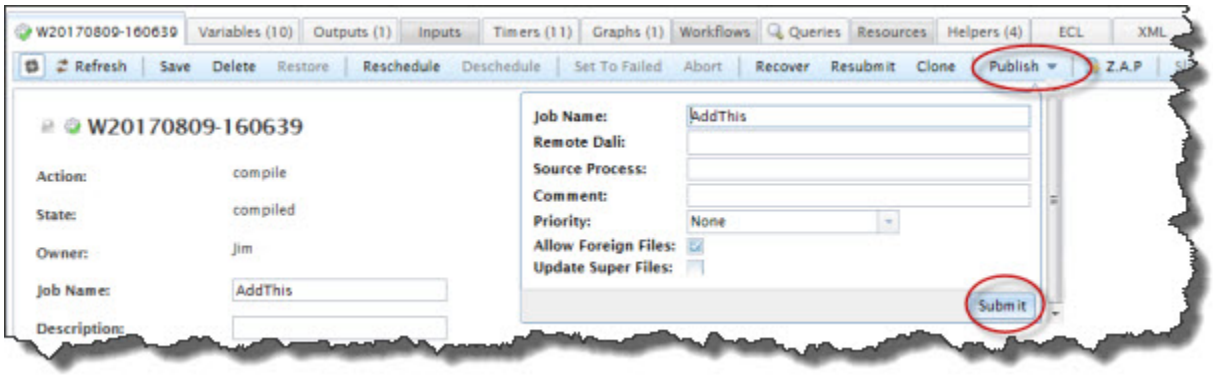


7. Select the workunit tab for the AddThis workunit that you just compiled.

   This opens the workunit in an ECL Watch tab.

8. Press the **Publish** action button, then verify the information in the dialog and press **Submit**.

Figure 10. Publish Query



   This publishes the AddThis query to the Roxie.

9. Test the service using WsECL :

```
http://<esp ip >:8002
```

Publish the ESDL Service Definitions and Bind the ESDL Service

In this portion of the tutorial, we will publish the ESDL Service definitions to the System Data Store and bind the methods to the published Roxie query.

1. Open the Dynamic ESDL definition file (MathService.esdl) in the ECL IDE.

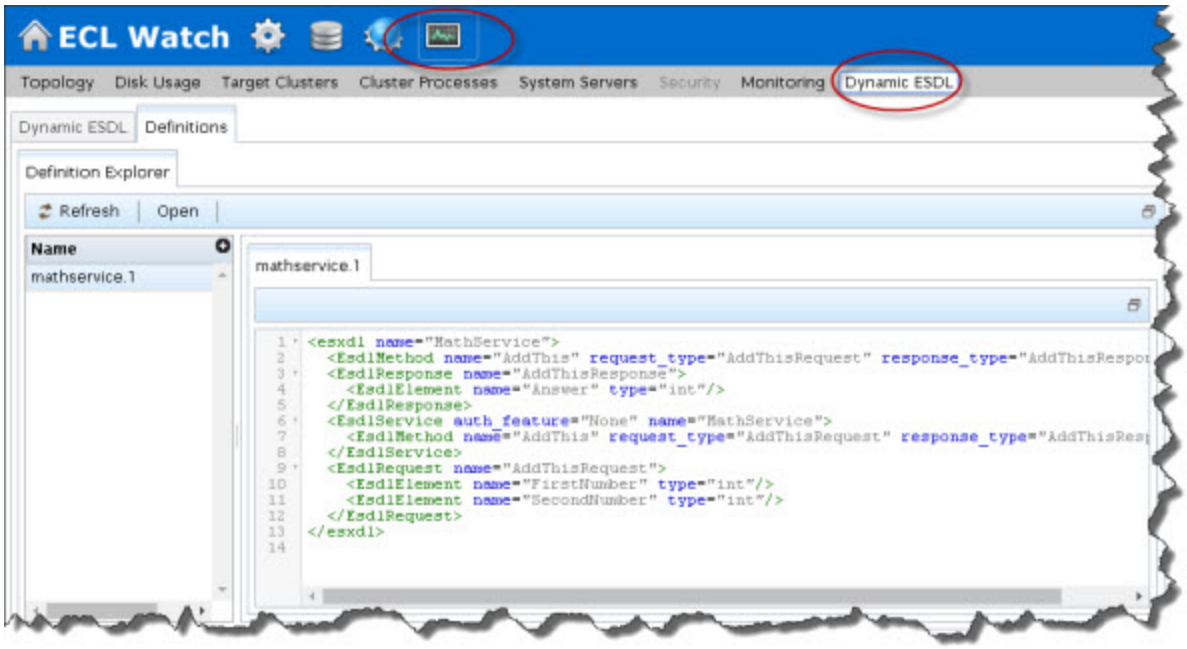
2. Press the **Publish** button.

   This publishes the ESDL Service definition to the ESP Server. Next we will bind the *AddThis* method to the *AddThis* published query.

3. Open ECL Watch in your browser (<your ESP ip>:8010)

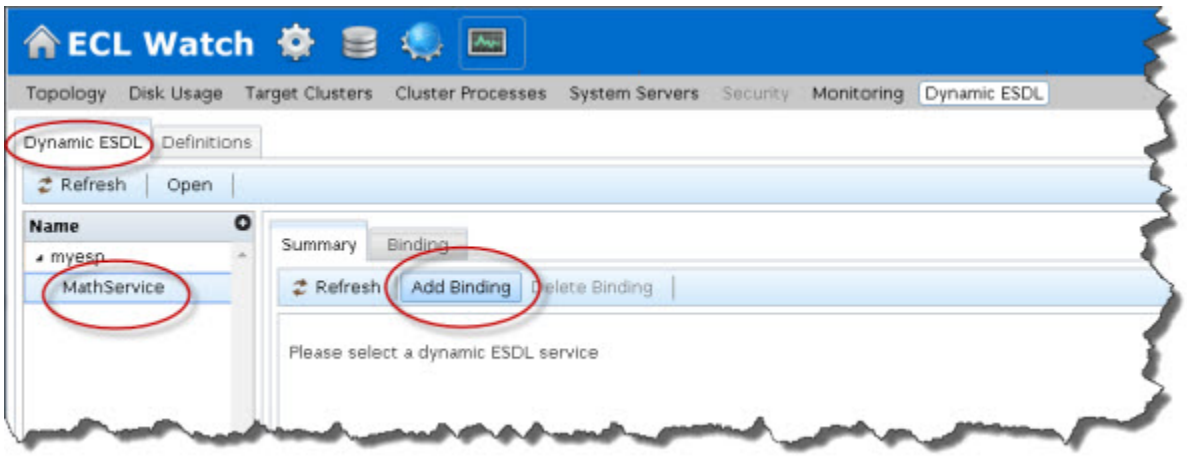
4. Select the Operations tab, then select **Dynamic ESDL**.

Figure 11. Dynamic ESDL in ECLWatch



5. Select the **Dynamic ESDL** Tab, then expand *myesp* by clicking on the triangle next to it.

Figure 12. Add Binding



6. Press the **Add Binding** button, then select *MathService.1* from the drop list and press the **Apply** button.

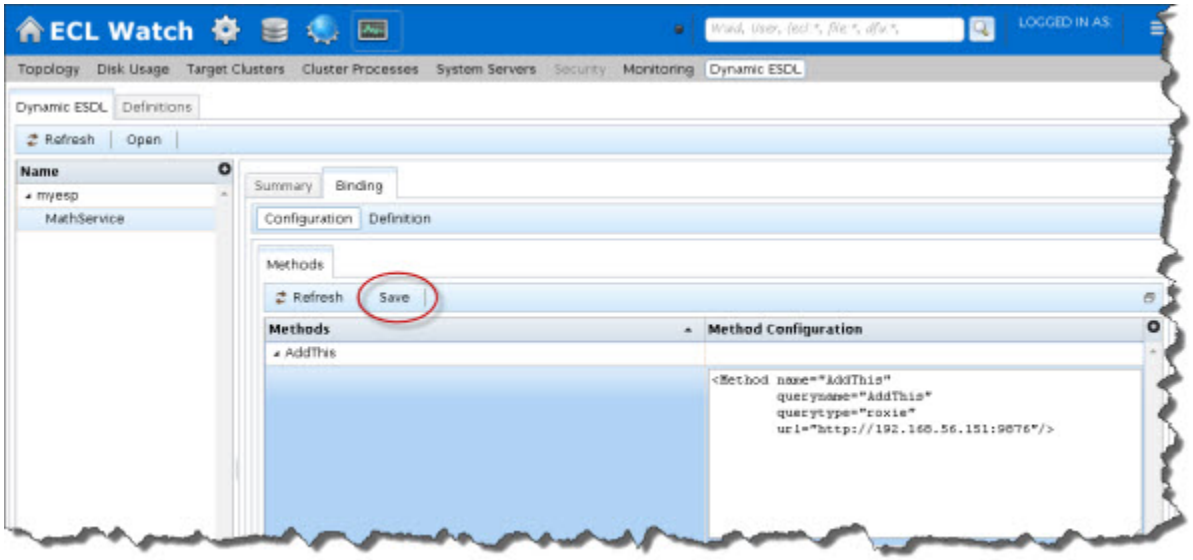
7. Select the **Binding** tab, then expand *AddThis* by clicking on the triangle next to it.

8. Provide the Method Configuration (in XML format) in the text box.

   Note: You must replace <RoxieIPRange> with the IP Range of your Roxie servers.

```
<Method name="AddThis"
        queryname="AddThis"
        querytype="roxie"
        url="http://<RoxieIPRange>:9876"/>
```

Figure 13. Method Configuration



9. Press the **Save** button.

10.Test the service using the new interface:

```
http://<node ip >:8003
```

ESDL Command Line Interface

The ESDL Command Syntax

esdl [--version] <command> [<options>]

--version	displays version info.
help <commmand>	displays help for the specified command.
xml	Generate XML from ESDL definition.
ecl	Generate ECL from ESDL definition.
xsd	Generate XSD from ESDL definition.
wsdl	Generate WSDL from ESDL definition.
publish	Publish ESDL Definition for ESP use.
list-definitions	List all ESDL definitions.
delete	Delete ESDL Definition.
bind-service	Configure ESDL based service on target ESP (with existing ESP Binding).
list-bindings	List all ESDL bindings.
unbind-service	Remove ESDL based service binding on target ESP.
bind-method	Configure method associated with existing ESDL binding.
unbind-method	Remove method from an ESDL binding on a target ESP.
get-binding	Get ESDL binding.

esdl xml

esdl xml [options] filename.ecm [<outdir>]

filename.ecm	The file containing the ESDL definitions
-r|--recursive	process all includes
-v|--verbose	display verbose information
-?/-h/--help	show usage page
Output	(srcdir|<outdir>)/filename.xml

This generates XML from the ESDL definition. This XML is an intermediate entity used by the ESDL Engine to create the runtime service definitions. This command is rarely used by itself.

Examples:

```
esdl xml MathService.ecm .
```

esdl ecl

esdl ecl sourcePath outputPath [options].

sourcePath	The absolute path to the ESDL Definition file containing the EsdlService definition for the service.
outputPath	The absolute path to the location where ECL output is to be written.
-x, --expandedxml	Output expanded XML files.
--includes	If present, process all included files.
--rollup	If present, rollup all processed includes to a single ECL output file.
-cde	Specifies the HPCC Component files directory (location of xslt files).
--ecl-imports	Comma-delimited import list to be attached to the output ECL. Each entry generates a corresponding IMPORT statement.
--ecl-header	Text to include in header or target (generated) file (must be valid ECL).
Output	(sourcePath|outputPath>)/filename.ecl

This generates ECL structures from ESDL definition. These structures create the interface (entry and exit points) to the Roxie query.

Examples:

```
esdl ecl MathService.ecm .
```

esdl xsd

esdl xsd sourcePath serviceName [options]

sourcePath	The absolute path to the ESDL Definition file containing the EsdlService definition for the service.
serviceName	Name of ESDL Service defined in the given ESDL file.
--version <version number>	Constrain to interface version
*--method <method name>[;<method name>]**	Constrain to list of specific method(s)
--xslt <xslt file path>	Path to '/xslt/esxdl2xsd.xslt' file to transform EsdlDef to XSD
--preprocess-output <raw output directory> :	Output preprocessed XML file to specified directory before applying XSLT transform
--annotate <all | none>	Flag turning on either all annotations or none. By default, annotations are generated for Enumerations. Setting the flag to 'none' will disable those as well. Setting it to al' enables additional annotations such as collapsed, cols, form_ui, html_head and rows.
--noopt	Turns off the enforcement of 'optional' attributes on elements. If no -noopt is specified then all elements with an 'optional' are included in the output. By default 'optional' filtering is enforced.
-opt,--optional <param value>	Value to use for optional tag filter when gathering dependencies. For example, passing 'internal' when some ESDL definition objects have the attribute optional("internal") ensures they appear in the XSD, otherwise they'd be filtered out
-tns,--target-namespace <target namespace>	The target namespace passed to the transform via the parameter 'tnsParam' used for the final output of the XSD.
-n <int> .	Number of times to run transform after loading XSLT. Defaults to 1
--show-inheritance	Turns off the collapse feature. Collapsing optimizes the XML output to strip out structures only used for inheritance, and collapses their elements into their child. That simplifies the stylesheet. By default this option is on
--no-arrayof	Supresses the use of the arrrayof element. arrayof optimizes the XML output to include 'ArrayOf...' structure definitions for those EsdlArray elements with no item_tag attribute. Works in conjunction with an optimized stylesheet that doesn't generate these itself. This defaults to on.
-v|--verbose	display verbose information
-?/-h/--help	show usage page
Output	(srcdir|<outdir>)/filename.ecl

This generates XSD from the ESDL definition.

Examples:

```
esdl xsd MathService.ecm MathService
```

esdl wsdl

esdl wsdl sourcePath serviceName [options]

sourcePath	The absolute path to the ESDL Definition file containing the EsdlService definition for the service.
serviceName	Name of ESDL Service defined in the given ESDL file.
--version <version number>	Constrain to interface version
*--method <method name>[;<method name>]**	Constrain to list of specific method(s)
--xslt <xslt file path>	Path to '/xslt/esxdl2xsd.xslt' file to transform EsdlDef to XSD
--preprocess-output <raw output directory> :	Output preprocessed XML file to specified directory before applying XSLT transform
--annotate <all | none>	Flag turning on either all annotations or none. By default, annotations are generated for Enumerations. Setting the flag to 'none' will disable those as well. Setting it to al' enables additional annotations such as collapsed, cols, form_ui, html_head and rows.
--noopt	Turns off the enforcement of 'optional' attributes on elements. If no -noopt is specified then all elements with an 'optional' are included in the output. By default 'optional' filtering is enforced.
-opt,--optional <param value>	Value to use for optional tag filter when gathering dependencies. For example, passing 'internal' when some ESDL definition objects have the attribute optional("internal") ensures they appear in the XSD, otherwise they'd be filtered out
-tns,--target-namespace <target namespace>	The target namespace passed to the transform via the parameter 'tnsParam' used for the final output of the XSD.
-n <int> .	Number of times to run transform after loading XSLT. Defaults to 1
--show-inheritance	Turns off the collapse feature. Collapsing optimizes the XML output to strip out structures only used for inheritance, and collapses their elements into their child. That simplifies the stylesheet. By default this option is on
--no-arrayof	Supresses the use of the arrrayof element. arrayof optimizes the XML output to include 'ArrayOf...' structure definitions for those EsdlArray elements with no item_tag attribute. Works in conjunction with an optimized stylesheet that doesn't generate these itself. This defaults to on.
--wsdladdress	Defines the output WSDL file's location address
-v|--verbose	display verbose information
-?/-h/--help	show usage page
Output	(srcdir|<outdir>)/filename.ecl

This generates WSDL from ESDL definition.

Examples:

```
esdl wsdl MathService.ecm MathService
```

esdl publish

esdl publish <filename.(ecm|esdl|xml)> <servicename> [options]

filename	The ESDL (*.ecm, *.esdl, or *.xml) file containing the service definitions.
servicename	The name of the service to publish. Optional if the ESDL definition contains only one service.
--overwrite	Overwrite the latest version of this ESDL Definition
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Publishes an ESDL service definition to the system datastore.

Examples:

```
esdl publish mathservice.ecm mathservice -s nnn.nnn.nnn.nnn --port 8010
```

esdl list-definitions

esdl list-definitions [options]

-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

This command lists published definitions

Example:

```
esdl list-definitions -s nnn.nnn.nnn.nnn --port 8010
```

esdl delete

esdl delete <ESDLServiceDefinitionName> <ESDLServiceDefinitionVersion> [options]

ESDLServiceDefinitionName	The name of the ESDL service definition to delete
ESDLServiceDefinitionVersion	The version of the ESDL service definition to delete
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to delete an ESDL Service definition. If the Service definition is bound, you must first unbind it.

Example:

```
esdl delete mathservice 2 -s nnn.nnn.nnn.nnn --port 8010
```

esdl bind-service

esdl bind-service <TargetESPProcessName> <TargetESPBindingPort | TargetESPServiceName> <ESDLDefinitionId> (<ESDLServiceName>) [command options]

TargetESPProcessName	The target ESP Process name
TargetESPBindingPort | TargetESPServiceName	Either target ESP binding port or the target ESP service name
ESDLDefinitionId	The Name and version of the ESDL definition to bind to this service (must already be defined in Dali)
ESDLServiceName	The Name of the ESDL Service (as defined in the ESDL Definition) Required if ESDL definition contains multiple services
--config <file | XML>	Configuration XML (either inline or as a file reference)
--overwrite	Overwrite the latest version of this ESDL Definition
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to bind a Dynamic ESDL-based ESP service to an ESDL definition.

To bind an ESDL Service, provide the target ESP process name (ESP Process which will host the ESP Service as defined in the ESDL Definition.)

You must also provide either the port on which this service is configured to run (ESP Binding) or the name of the service you are binding.

Optionally provide configuration information either directly inline or using a configuration file XML in the following syntax:

```
<Methods>
  <Method name="myMthd1" url="<RoxieIPRange>:9876/path?param=value" user="me" password="mypw"/>
  <Method name="myMthd2" url="<RoxieIPRange>:9876/path?param=value" user="me" password="mypw"/>
</Methods>
```

Example:

```
esdl bind-service myesp 8003 MathSvc.1 MathSvc --config MathSvcCfg.xml
                  -s nnn.nnn.nnn.nnn -p 8010
```

Configuring ESDL binding methods

The DESDL binding methods can optionally provide context information to the target ECL query. The way this information is configured, is by appending child elements to the Method (<Method>...</Method>) portion of the ESDL Binding.

For example, the following XML provides a sample ESDL Binding.

```
<Methods>
  <Method name="AddThis" url="<RoxieIPRange>:9876" querytype="roxie" queryname="AddThis"/>
</Methods>
```

If this Method requires context information, for example about gateways, then you could include the Gateways Structure (<Gateways>...</Gateways>) depicted as follows.

```
<Methods>
  <Method name="AddThis" url="<RoxieIPRange>:9876" querytype="roxie" queryname="AddThis">
    <!--Optional Method Context Information start-->
    <Gateways>
      <Gateway name="mygateway" url="1.1.1.1:2222/someservice/somemethod/>
      <Gateway name="anothergateway" url="2.2.2.2:9999/someservice/somemethod/>
    </Gateways>
    <!--Optional Method Context Information end-->
  </Method>
</Methods>
```

The DESDL ESP does not pose any restrictions on the layout of this information, only that it is valid XML. This provides the flexibility to include context information in any valid XML format.

Roxie (query) ECL developers need to decide what information they will need from the ESP request and design how that information is laid-out in the ESP request and ESDL binding configuration.

In the following example, every "AddThis" request processed by the ESP and sent to Roxie would contain the sample gateway information in the request context.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <roxie.AddThis>
   <Context>
    <Row>
     <Common>
      <ESP>
       <ServiceName>wsmath</ServiceName>
       <Config>
        <Method name="AddThis" url="<RoxieIPRange>:9876" querytype="roxie" queryname="AddThis">
          <Gateways>
            <Gateway name="mygateway" url="1.1.1.1:2222/someservice/somemethod/>
            <Gateway name="anothergateway" url="2.2.2.2:9999/someservice/somemethod/>
          </Gateways>
        </Method>
       </Config>
      </ESP>
        <TransactionId>sometrxid</TransactionId>
     </Common>
    </Row>
   </Context>
   <AddThisRequest>
    <Row>
     <Number1>34</Number1>
     <Number2>232</Number2>
    </Row>
   </AddThisRequest>
  </roxie.AddThis>
</soap:Body>
</soap:Envelope>
```

The ECL query consumes this information and is free to do whatever it needs to with it. In some instances, the query needs to send a request to a gateway in order to properly process the current request. It can interrogate the context information for the appropriate gateway's connection information, then use that information to create the actual gateway request connection.

Configuring ESDL binding for Proxy Mode methods

You can specify that ESDL service methods be proxied to another ESP instance. Set up the proxy in the dynamic configuration associated with the dESDL service.

Under the Methods tag where you would add Method tags, you can also add Proxy tags, as shown here:

```
<Methods>
  <Method name="myMethod" url="http://10.45.22.1:292/somepath"/>
  <Method name="myMethod2" url="http://10.45.22.1:292/somepath"/>
  <Proxy method="myMethod3" forwardTo="http://10.45.22.1:292" />
  <Proxy method="myWild*" forwardTo="http://10.45.22.1:292" />
</Methods>
```

The Proxy tag also supports wildcards:

```
<Proxy method="myWild*" forwardTo="http://10.45.22.1:292" />
```

This example binds all methods matching the pattern: myWild*

esdl list-bindings

esdl list-bindings [options]

-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to list bindings on a server.

Example:

```
esdl list-bindings -s nnn.nnn.nnn.nnn -p 8010
```

esdl unbind-service

esdl unbind-service <ESPProcessName> <ESPBindingName> [options]

ESPProcessName	The ESP Process name
ESPBindingName	The ESP Binding name
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to unbind ESDL service based bindings.

To unbind a given ESDL binding, provide the ESP process name and the ESP binding which make up this ESDL binding.

Available ESDL bindings to unbind can be found using the "esdl list-bindings" command

Example:

```
esdl unbind-service myesp myServiceBinding
```

esdl bind-method

esdl bind-method <TargetESDLBindingID> <TargetMethodName> [options]

TargetESDLBindingID	The id of the target ESDL binding (must exist in Dali)
TargetMethodName	The name of the target method (must exist in the service ESDL definition)
--config <file | XML>	Configuration XML (either inline or as a file reference)
--overwrite	Overwrite the latest version of this ESDL Definition
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to publish ESDL Service based bindings.

To bind an ESDL Service, provide the target ESP process name (ESP Process which will host the ESP Service as defined in the ESDL Definition.)

You must also provide the port on which this service is configured to run (ESP Binding), and the name of the service you are binding.

Optionally provide configuration information either directly inline or using a configuration file XML in the following syntax:

```
<Methods>
  <Method name="myMthd1" url="http://<RoxieIPRange>:9876/path?param=value" user="me" password="mypw"/>
  <Method name="myMthd2" url="http://<RoxieIPRange>:9876/path?param=value" user="me" password="mypw"/>
</Methods>
```

Example:

```
esdl bind-method myEsp myespbinding mySvc mySvc.1 myMthd1 --config myMethods.xml
```

esdl unbind-method

esdl unbind-method <ESPProcessName> <ESPBindingName> <ESDLServiceName> <MethodName> [options]

ESPProcessName	The target ESP Process name
ESPBindingName	The target ESP binding name associated with this service
ESDLServiceName	The name of the ESDLService associated with the target method.
MethodName	The name of the target method (must exist in the service ESDL definition)
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to unbind a method configuration associated with a given ESDL binding0.

To unbind a method, provide the target ESP process name (the ESP which hosts the service.)

You must also provide the ESP binding on which this service is configured to run, the name of the ESDL service, and the name of the method you are unbinding.

Example:

```
esdl unbind-method myesp myespbinding WsMyService mymethod
```

esdl get-binding

esdl get-binding <ESDLBindingId> [options]

ESDLBindingId	The target ESDL binding id <espprocessname>.<espbindingname>
-s, --server	The IP Address or hostname of ESP server running ECL Watch services
--port	The ECL Watch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)
--version <ver>	ESDL service version
--help	display usage information for the given command
-v, --verbose	Output additional tracing information

Use this command to get DESDL Service based bindings.

To specify the target DESDL based service configuration, provide the target ESP process (esp process name or machine IP Address) which hosts the service.

You must also provide the Port on which this service is configured to run and the name of the service.

Example:

```
esdl get-binding  myesp.dESDL_Service -s nnn.nnn.nnn.nnn -p 8010
```