Dynamic ESDL

Boca Raton Documentation Team

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.

2026 Version 10.4.18-1

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 is an integral part of ESP.

Dynamic ESDL Workflow Tutorial

Before You Begin...

You will need:

- Access to an HPCC Systems Cluster (version 7.0 or later). This can be one running in a Virtual Machine.
- Access to ECL Watch and WsECL (using a browser).

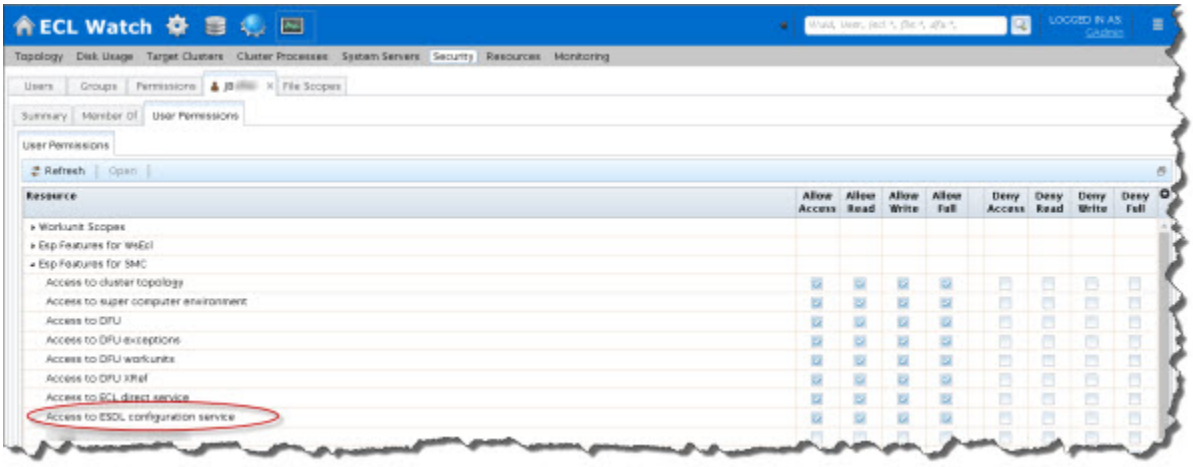
  For purposes of this tutorial, we assume that you know how to submit a published query using WsECL.
- The ECL IDE (version 7.0 or later)

  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 Systems 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:

- 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.

ESDL Configuration

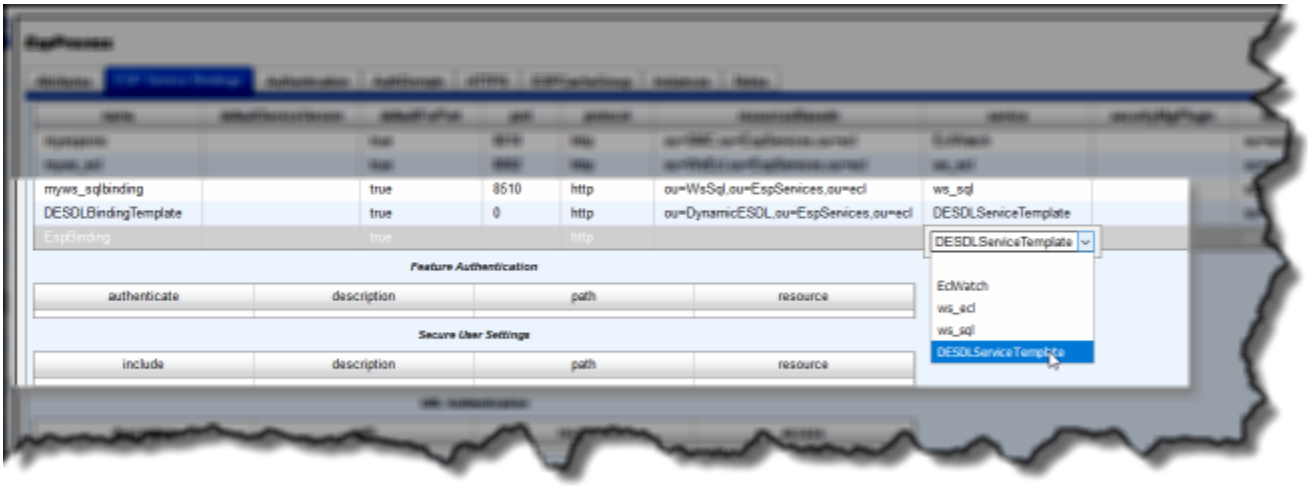
The Configuration for Dynamic ESDL is an integral part of ESP. Therefore, special configuration is not really necessary for basic ESDL definitions. In Configuration Manager there are some configuration elements that you can customize.

To customize your Dynamic ESDL set up In Configuration Manager

1. Go to the **ESP - myesp** page then select Service bindings.

2. Select the **ESP Service Bindings** Tab

The default ESP Service Bindings tab lists the service bindings. Among the bindings listed there is the DESDLBindingTemplate. This binding is a template, and provides a default configuration for all DESDL bindings that you're going to add on any port. Although the template binding is set to use port 0, there is no binding created on port 0, the template only exists to provide a default configuration.



You can customize the template to suit your needs. For example, you can set up LDAP security or logging that will apply to all bindings. If you wanted to have a custom configuration that is different from this generic template, for instance to bind to Port 8020 that will use a different LDAP server, you can add another DESDL binding template on port 8020.

1. Right-click and choose **Add** on the ESP Service Bindings tab

2. Click on the field in the **service** column to select DESDL Service Template from the drop list

3. Click on the **port** column and set the port number as desired, for example, port 8020

This template will override the template on port 0 and will provide configuration for the dynamic bindings on port 8020.

Write the ESDL Service Definition

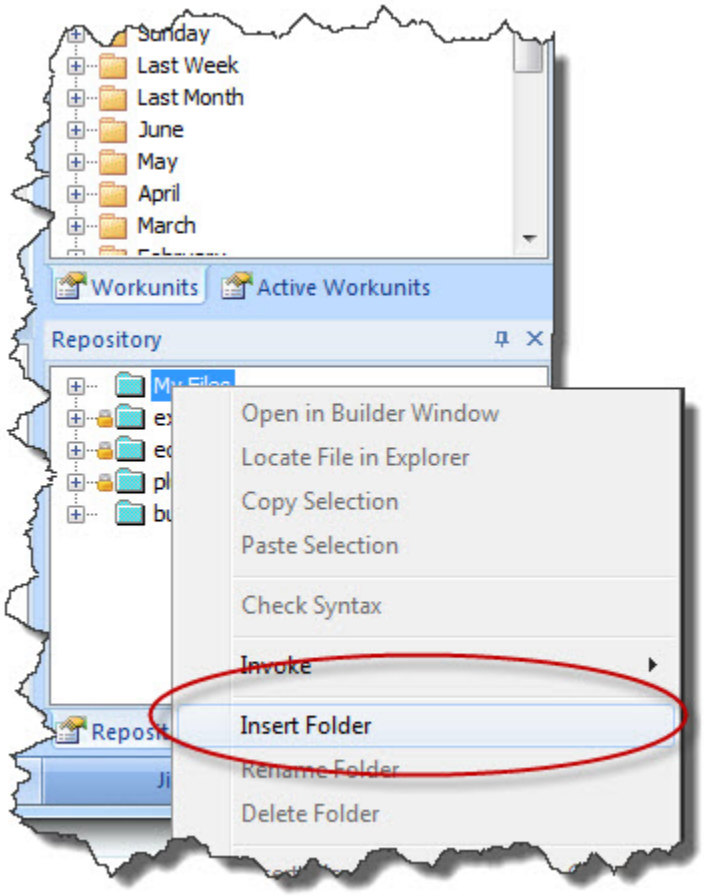
In this portion of the tutorial, we will write the Service Definitions in the ECL IDE. The program listing below shows an ESDL service called *MathService*. It contains one method, *AddThis*, with a request and a response defined.

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.

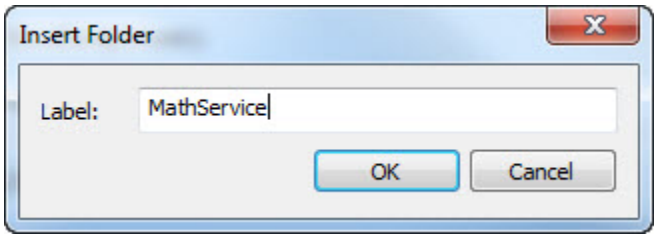
Figure 1. Insert Folder



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

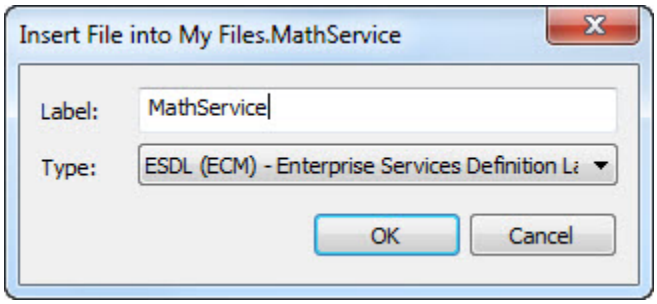
4. Enter **MathService** for the label, then press the **OK** button.

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.

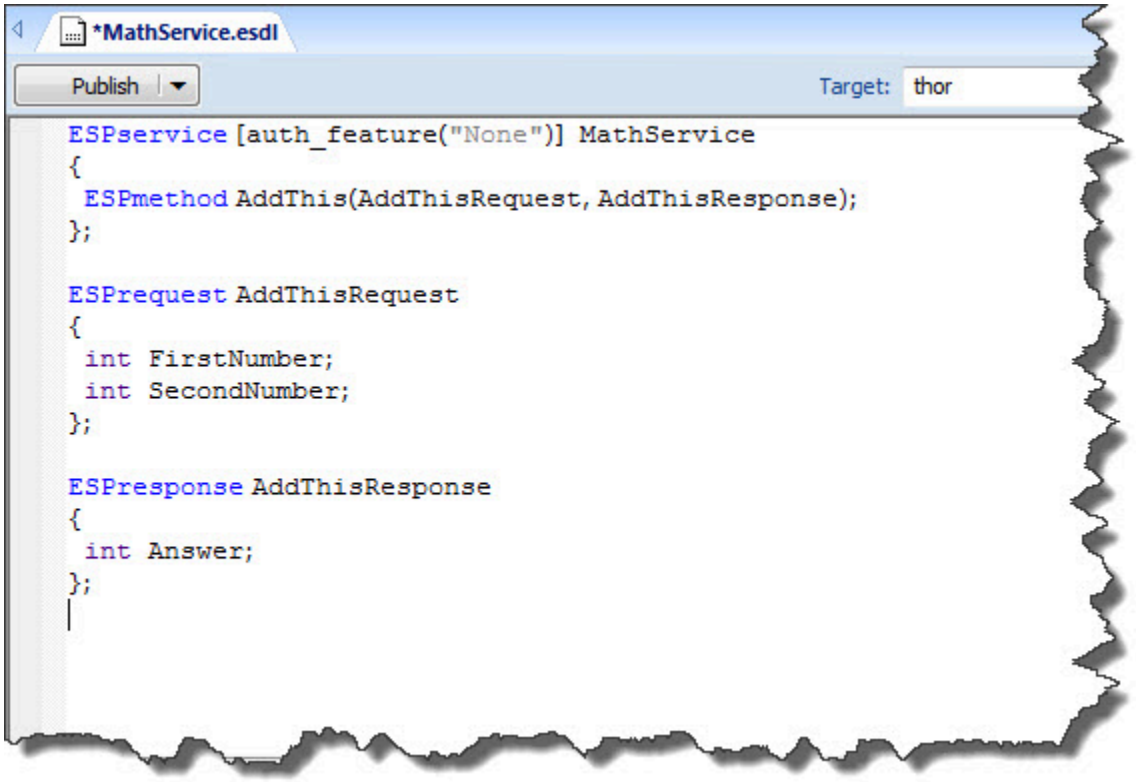
7. Write the following code, replacing what was written, in the editor workspace :

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

ESPrequest AddThisRequest
{
  int  FirstNumber;
  int  SecondNumber;
};

ESPresponse AddThisResponse
{
  int  Answer;
};
```

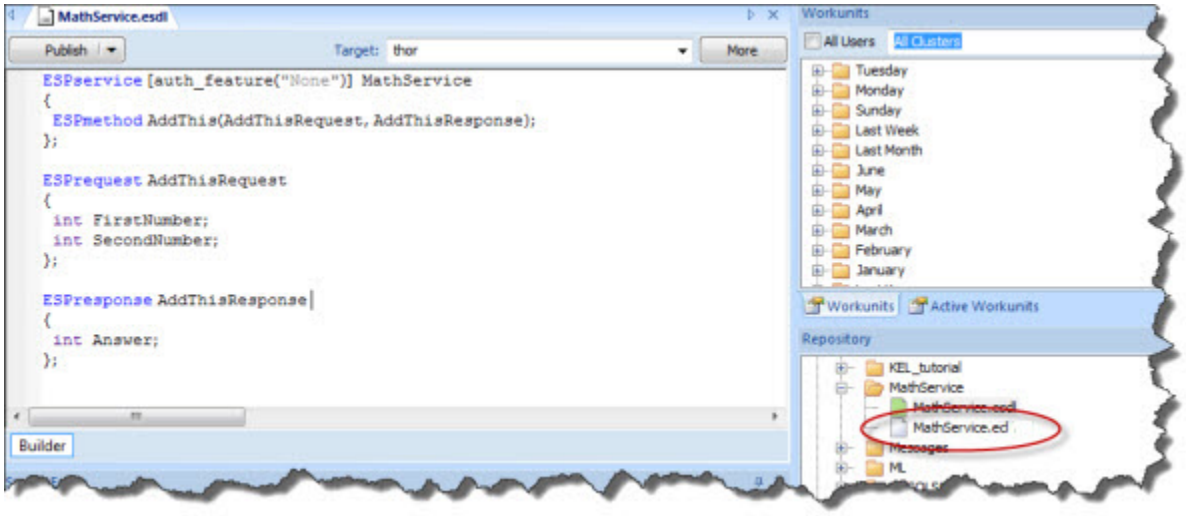
Figure 5. ESDL Code in Editor Window



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 MathService 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:

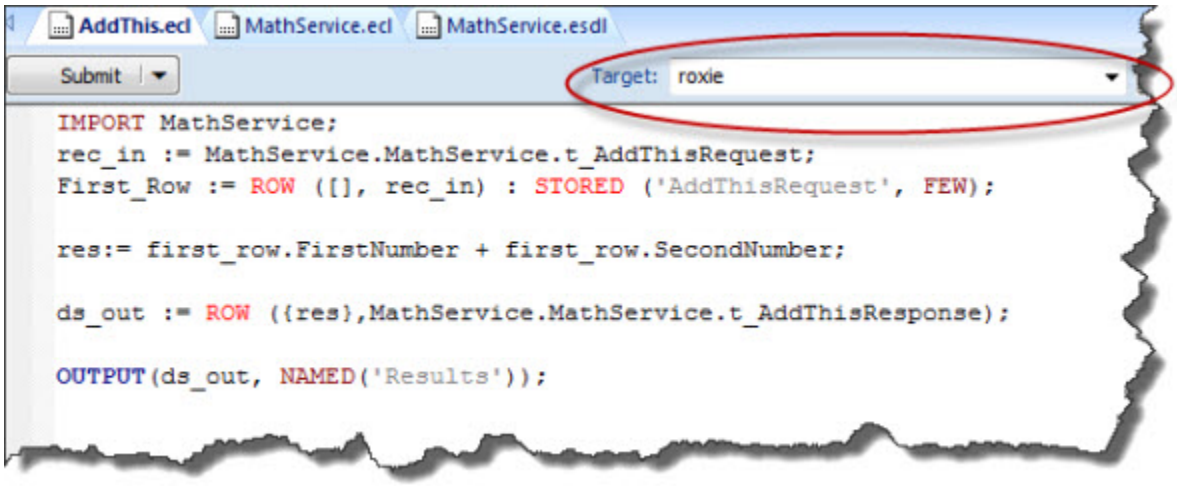
```
IMPORT MathService;
rec_in := MathService.MathService.t_AddThisRequest;

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'));
```

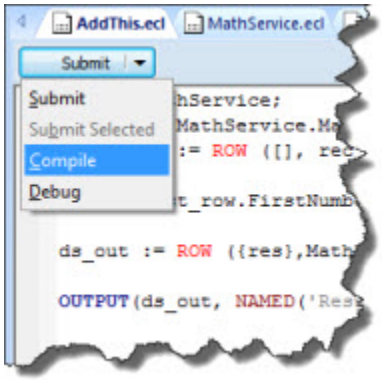
4. Using the **Target** drop list, select *Roxie* as the Target cluster.

Figure 7. Target Roxie



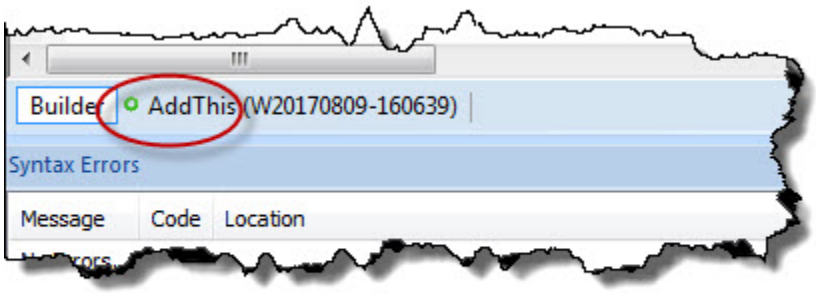
5. In the Builder window, in the upper left corner the **Submit** button has a drop down arrow next to it. Select the arrow, then select **Compile**.

Figure 8. Compile



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

Figure 9. Compiled

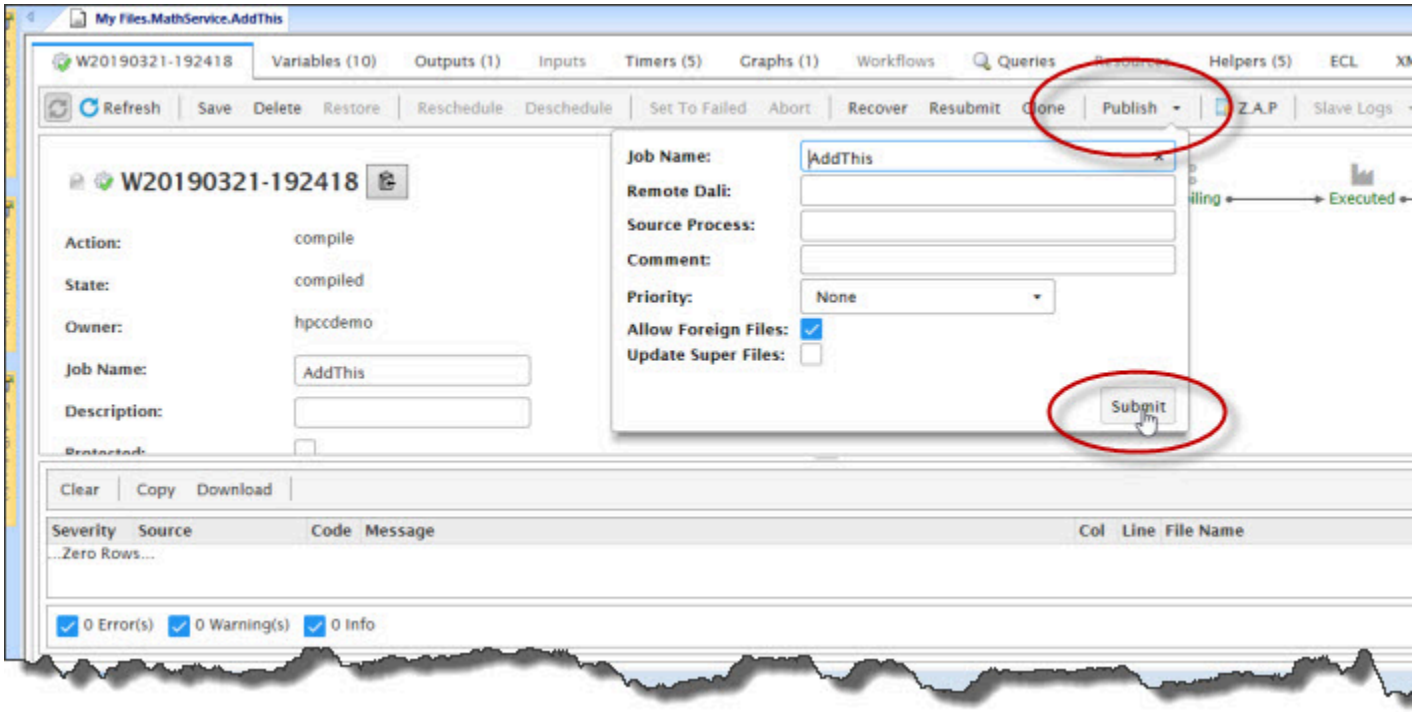


7. Select the workunit tab for the AddThis workunit that you just compiled.

   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
```

   Find the *addthis* service under the Roxie, Target, Active Queries.

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.

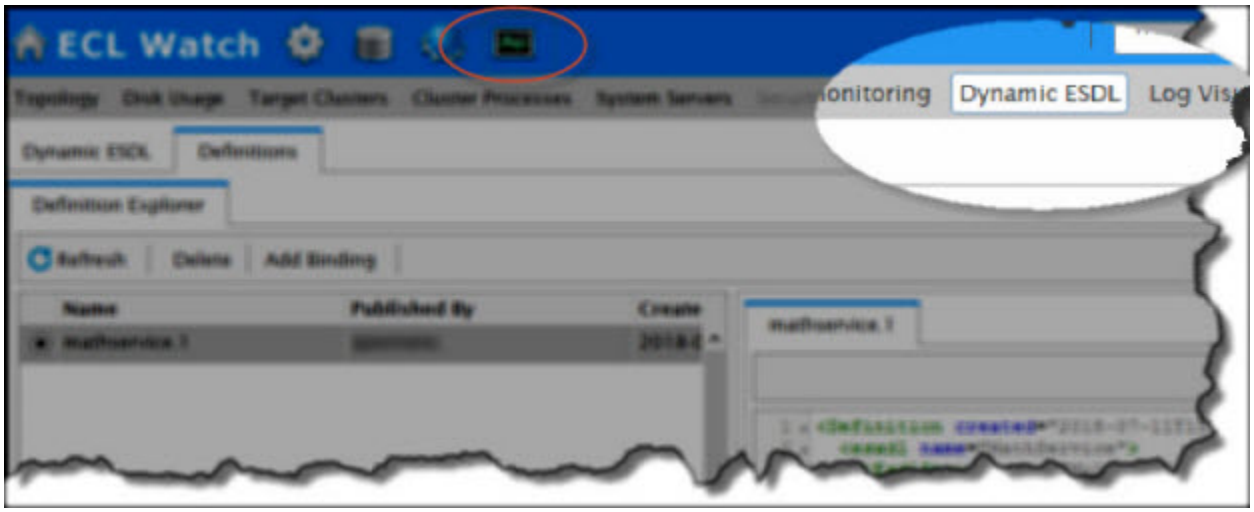
2. Press the **Publish** button.

   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)

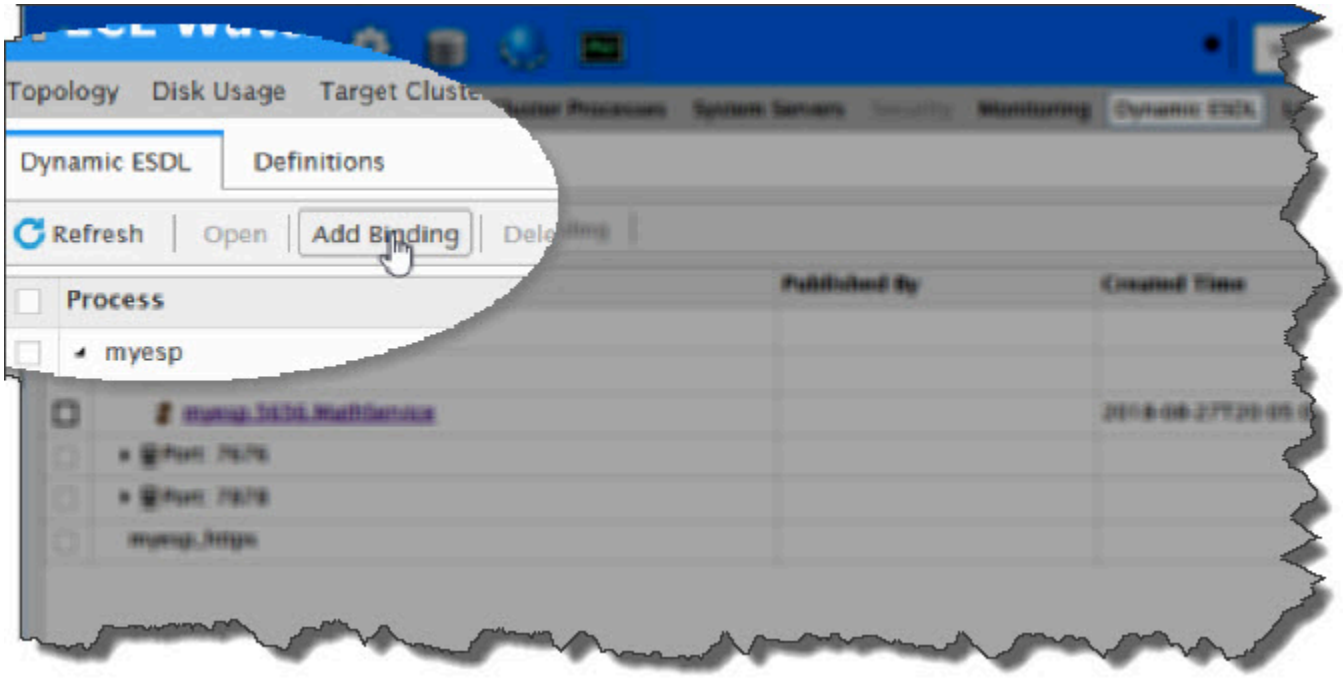
4. Select the Operations tab, then select **Dynamic ESDL**.

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.

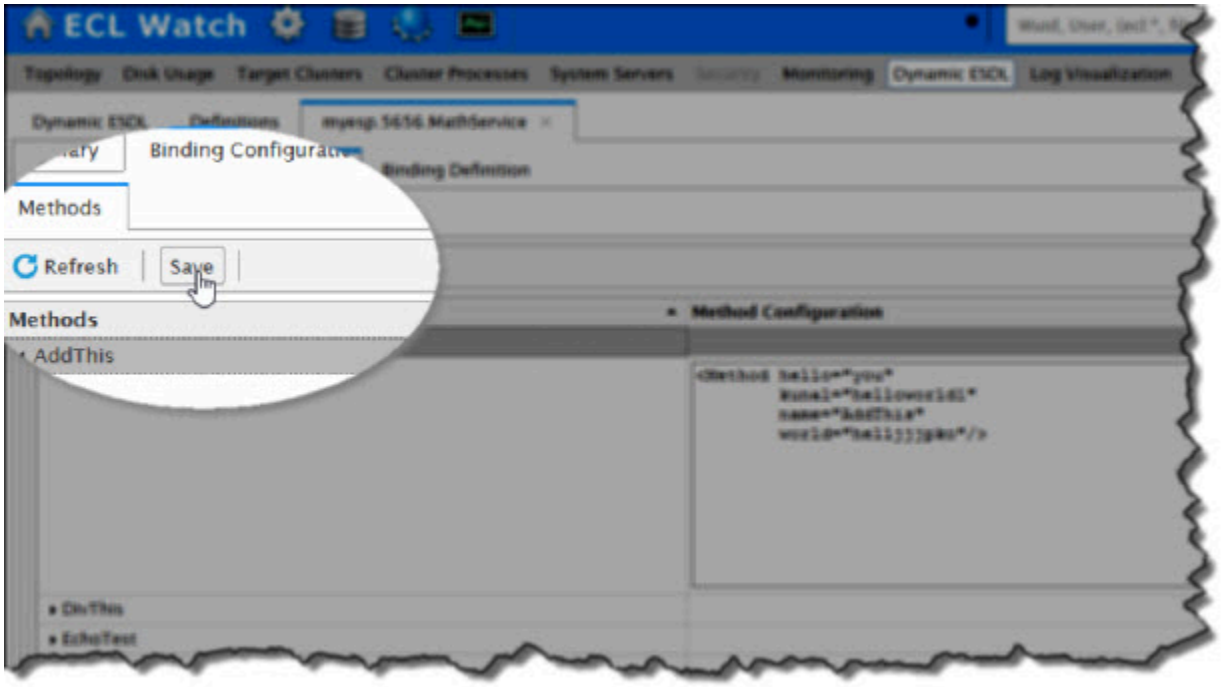
7. Select the **Binding** tab, then expand *AddThis* by clicking on the triangle next to it.

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
```

Additional examples can be found in the following places:

- In the ECL IDE in the **examples/EsdlExample** folder
- In a a folder where Client Tools was installed (**clienttools/examples/EsdlExample**)
- In a a folder on a server where the platform was installed (**opt/HPCCSystems/examples/EsdlExample**)
- In the HPCC Systems HPCC-Platform repository on GitHub:

  (https://github.com/hpcc-systems/HPCC-Platform/tree/master/initfiles/examples/EsdlExample)

ESDL Command Line Interface

The ESDL Command Syntax

The **esdl** utility tool aids with creating and managing ESDL-based and Dynamic ESDL services on an HPCC cluster. It provides commands for generating artifacts (such as XML, ECL, XSD, WSDL, Java, C++), managing and publishing ESDL definitions and bindings, and advanced configuration using manifest files and log transforms. Each command is documented in detail in the sections below.

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

Output Generating ESDL Definitions

xml	Generate XML from ESDL definition.
ecl	Generate ECL from ESDL definition.
xsd	Generate XSD from ESDL definition.
wsdl	Generate WSDL from ESDL definition.

ESDL and DESDL Service Managing

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.
get	Get ESDL definition.

esdl xml

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

filename.ecm	The file containing the ESDL definitions
-r|--recursive	process all includes
-I, --include-path	Locations to look for included ESDL files. They can be absolute or relative paths. If you need to specify multiple directories, you can use multiple -I options or use a single entry with the directories separated with the environment separator character. For Linux, use a colon (:) and for Windows, use a semi-colon (;). The paths can also be set using an environment variable--ESDL_INCLUDE_PATH.
-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 Systems 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).
-I, --include-path	Locations to look for included ESDL files. They can be absolute or relative paths. If you need to specify multiple directories, you can use multiple -I options or use a single entry with the directories separated with the environment separator character. For Linux, use a colon (:) and for Windows, use a semi-colon (;). The paths can also be set using an environment variable--ESDL_INCLUDE_PATH.
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
-I, --include-path	Locations to look for included ESDL files. They can be absolute or relative paths. If you need to specify multiple directories, you can use multiple -I options or use a single entry with the directories separated with the environment separator character. For Linux, use a colon (:) and for Windows, use a semi-colon (;). The paths can also be set using an environment variable--ESDL_INCLUDE_PATH.
-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 <ESDLDefinitionID> [options]

ESDLDefinitionID	The ID 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> <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 MathService.1 MathService --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 <ESPBindingID> [options]

ESPBindingID	The ESDL Binding ID
-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 ESDL binding ID

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

Example:

```
esdl unbind-service myesp.8003.MathService
```

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.8003.MathService AddThis --config myMethods.xml
```

esdl unbind-method

esdl unbind-method <ESDLBindingID> <MethodName> [options]

ESDLBindingID	The ID of the ESDL Binding
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 binding. To unbind a method, provide the ID of the ESDL binding and the name of the method you are unbinding.

Example:

```
esdl unbind-method myesp.8003.MathService AddThis
```

esdl get-binding

esdl get-binding <ESDLBindingId> [options]

ESDLBindingId	The target ESDL binding id <ESPProcessName>.<Port>.<ServiceName>
-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 ID of the ESDL binding, which is normally in the format <ESPProcessName>.<Port>.<ServiceName>

Example:

```
esdl get-binding  myesp.8003.MathService -s nnn.nnn.nnn.nnn -p 8010
```

esdl manifest

esdl manifest <manifest-file> [options]

-I | --include-path <path>	Search path for external files included in the manifest. Use once for each path.
--outfile <filename>	Path and name of the output file.
--output-type <type>	Overrides the value supplied in the manifest attribute Manifest/@output-Type. Allowed values are 'binding' or 'bundle'. Default is 'bundle'.
--help	Display usage information for the given command.
-v, --verbose	Output additional tracing information.
-tcat, --trace-category <flags>	Control which debug messages are output; a case-insensitive, comma-delimited combination of: dev, admin, user, err, warn, prog, info. Errors and warnings are enabled by default if not verbose, and all are enabled when verbose. Use an empty <flags> value to disable all.

For a detailed explanation of the variables and options used in the manifest command, see the: https://github.com/hpcc-systems/HPCC-Platform/blob/master/tools/esdlcmd/README.md.

For additional manifest file examples, refer to the repository at: https://github.com/hpcc-systems/HPCC-Platform/tree/master/initfiles/examples/EsdlExample/Manifest.

The **manifest** command creates an XML configuration file for an ESDL ESP from an input XML manifest file. The type of configuration output depends on the manifest file input and on command-line options.

The Manifest File

A manifest file is an XML-formatted template combining elements in and outside of the manifest's `urn:hpcc:esdl:manifest` namespace. Recognized elements of this namespace control the tool while all other markup is copied to the output. The goal of using a manifest file with the tool is to make configuring and deploying services easier:

- The manifest file format abstracts some of the complexity of the actual configuration.
- By allowing you to include external files like ESDL Scripts and XSLTs into the output, you can store and maintain them separately in your repo.

The result of running the manifest tool on a manifest file is an XML artifact suitable for use with the ESDL ESP. Supported output includes:

- *binding*: The output is an ESDL binding that may be published to dali.
- *bundle*: The output is an ESDL bundle file that may be used to launch an ESP in application mode.

Example Manifest File:

```
<em:Manifest xmlns:em="urn:hpcc:esdl:manifest">
  <em:ServiceBinding esdlservice="WsFoobar" id="WsFoobar_desdl_binding" auth_feature="DEFERRED">
    <Methods>
      <em:Scripts>
        <em:Include file="WsFoobar-request-prep.xml"/>
        <em:Include file="WsFoobar-logging-prep.xml"/>
      </em:Scripts>
      <Method name="FoobarSearch" url="127.0.0.1:8888">
```

```
        <em:Scripts>
          <em:Include file="FoobarSearch-scripts.xml"/>
        </em:Scripts>
      </Method>
    </Methods>
    <LoggingManager>
      <LogAgent transformSource="local" name="main-logging">
        <LogDataXPath>
          <LogInfo name="PreparedData" xsl="log-prep"/>
        </LogDataXPath>
        <XSL>
          <em:Transform name="log-prep">
            <em:Include file="log-prep.xslt"/>
          </em:Transform>
        </XSL>
      </LogAgent>
    </LoggingManager>
  </em:ServiceBinding>
  <em:EsdlDefinition>
    <em:Include file="WsFoobar.ecm"/>
  </em:EsdlDefinition>
</em:Manifest>
```

The tool is permissive and flexible, copying through most markup to the output. Recognized elements in the manifest namespace may be treated differently. They are only required in order to take advantage of the automated processing and simplified format of the manifest file. This example highlights the recommended usage of manifest elements to use the tool's capabilities. Although you could replace some of the elements below with verbatim bundle or binding output elements we won't cover that usage here.

Manifest Elements and Attributes

em:Manifest — Required root element. By default the tool outputs a bundle, though you may explicitly override that on the command line or by providing an `@outputType='binding'` attribute.

em:ServiceBinding — Valid for both bundle and binding output. Enables recognition of em:Scripts and em:Transform elements.

em:EsdlDefinition — Relevant only for bundle output. Enables element order preservation and recognition of em:Include as a descendant element.

em:Include — Causes external file contents to be inserted in place of the element. Processing is context dependent; the parent dictates how the file is handled. Facilitates code reuse in a configuration as code development environment.

em:Scripts / em:Transform — Trigger preservation of element order for all descendant elements and enable em:Include recognition.

Manifest Attributes

Manifest Root Attributes

Attribute	Required?	Value	Description
@outputType	No	string	A hint informing the tool which type of output to generate. The command line option --output-type may supersede this value to produce a different output.

Attribute	Required?	Value?	Description
@xmlns[:prefix]	Yes	string	The manifest namespace urn:hpcc:esdl:manifest must be declared. The default namespace prefix should not be used unless all other markup is fully qualified.

ServiceBinding Attributes

Attribute	Required?	Value?	Description
@esdlservice	Yes	string	Name of the ESDL service to which the binding is bound. Output on the Binding/Definition element. Also used to generate a value for Definition/@id for bundle type output.
@auth_feature	No	string	Used to declare authorization settings if not present in the ESDL Definition, or override them if they are.
@returnSchemaLocationOnOK	No	Boolean	When true, a successful SOAP response (non SOAP-Fault) will include the schema location property. False by default.
@namespace	No	string	String specifying the namespace for all methods in the binding. May contain variables replaced by the ESP at runtime.
@created	No	string	Timestamp of binding creation.
@espbinding	No	string	Set to match the id. Otherwise the value is unset and unused.
@espprocess	No	string	Name of the ESP process this binding is running on.
@id	No	string	Runtime name of the binding. When publishing to dali the value is [ESP Process].[port].[ESDL Service]. When not present in the manifest a default value is generated of the form [@esdlservice]_desdl_binding.
@port	No	string	Port on the ESP Process listening for connections to the binding.
@publishedBy	No	string	Userid of the person publishing the binding.