

# Dynamic ESDL

Boca Raton Documentation Team



## Dynamic ESDL

Boca Raton Documentation Team

Copyright © 2022 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.

2022 Version 8.6.54-1

Dynamic ESDL .....	4
Dynamic ESDL Workflow Tutorial .....	5
Before You Begin... ..	5
Overview .....	6
ESDL Configuration .....	7
Write the ESDL Service Definition .....	8
Writing the ECL .....	12
Publish the ESDL Service Definitions and Bind the ESDL Service .....	15
ESDL Command Line Interface .....	18
The ESDL Command Syntax .....	18

# 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**.



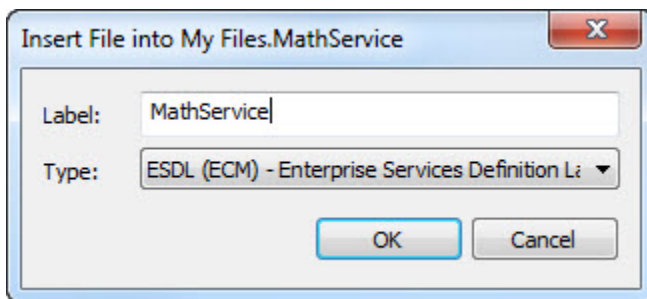
4. Enter **MathService** for the label, then press the **OK** button.

**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'));
```

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

**Figure 7. Target Roxie**



- 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**



- 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
```

# ESDL Command Line Interface

## The ESDL Command Syntax

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

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

## esdl xml

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

<i>filename.ecm</i>	The file containing the ESDL definitions
<i>-r/--recursive</i>	process all includes
<i>-v/--verbose</i>	display verbose information
<i>-?/-h/--help</i>	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].**

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

<i>sourcePath</i>	The absolute path to the ESDL Definition file containing the EsdlService definition for the service.
<i>serviceName</i>	Name of ESDL Service defined in the given ESDL file.
<i>--version &lt;version number&gt;</i>	Constrain to interface version
<i>--method &lt;method name&gt;[;&lt;method name&gt;]*</i>	Constrain to list of specific method(s)
<i>--xslt &lt;xslt file path&gt;</i>	Path to '/xslt/esxd2xsd.xslt' file to transform EsdlDef to XSD
<i>--preprocess-output &lt;raw output directory&gt; :</i>	Output preprocessed XML file to specified directory before applying XSLT transform
<i>--annotate &lt;all   none&gt;</i>	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 'all' enables additional annotations such as collapsed, cols, form_ui, html_head and rows.
<i>--noopt</i>	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.
<i>-opt,--optional &lt;param value&gt;</i>	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
<i>-tns,--target-namespace &lt;target namespace&gt;</i>	The target namespace passed to the transform via the parameter 'tnsParam' used for the final output of the XSD.
<i>-n &lt;int&gt; .</i>	Number of times to run transform after loading XSLT. Defaults to 1
<i>--show-inheritance</i>	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
<i>--no-arrayof</i>	Supresses the use of the arrayOf 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.
<i>-v/--verbose</i>	display verbose information
<i>-?/-h/--help</i>	show usage page
Output	(srcdir <outdir>)/filename.ecl

This generates XSD from the ESDL definition.

Examples:

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

## esdl wsd

**esdl wsd** *sourcePath* *serviceName* [*options*]

<i>sourcePath</i>	The absolute path to the ESDL Definition file containing the EsdlService definition for the service.
<i>serviceName</i>	Name of ESDL Service defined in the given ESDL file.
<i>--version</i> <version number>	Constrain to interface version
<i>--method</i> <method name>[;<method name>]*	Constrain to list of specific method(s)
<i>--xslt</i> <xslt file path>	Path to '/xslt/esxd2xsd.xslt' file to transform EsdlDef to XSD
<i>--preprocess-output</i> <raw output directory> :	Output preprocessed XML file to specified directory before applying XSLT transform
<i>--annotate</i> <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 'all' enables additional annotations such as collapsed, cols, form_ui, html_head and rows.
<i>--noopt</i>	Turns off the enforcement of 'optional' attributes on elements. If no <i>--noopt</i> is specified then all elements with an 'optional' are included in the output. By default 'optional' filtering is enforced.
<i>-opt,--optional</i> <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
<i>-tns,--target-namespace</i> <target namespace>	The target namespace passed to the transform via the parameter 'tnsParam' used for the final output of the XSD.
<i>-n</i> <int> .	Number of times to run transform after loading XSLT. Defaults to 1
<i>--show-inheritance</i>	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
<i>--no-arrayof</i>	Supresses the use of the arrayOf 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.
<i>--wsdladdress</i>	Defines the output WSDL file's location address
<i>-v/--verbose</i>	display verbose information
<i>-?/-h/--help</i>	show usage page
Output	(srcdir <outdir>)/filename.ecf

This generates WSDL from ESDL definition.

Examples:

```
esdl wsd 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** <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 ESDL 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>sometrxd</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
```