

HPCC Systems®

Dynamic ESDL

Boca Raton Documentation Team

Dynamic ESDL

Boca Raton Documentation Team

Copyright © 2017 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.

DEVELOPER NON-GENERATED VERSION

| | |
|--|----|
| Dynamic ESDL | 4 |
| Workflow Tutorial | 5 |
| Overview | 5 |
| Configure and Bind a Dynamic ESDL service | 6 |
| Write the ESDL Service Definition | 11 |
| Generate ECL definitions from the ESDL Service Definition | 12 |
| Publish the ESDL Service Definitions and Bind the ESDL Service | 13 |
| ESDL Command Line Interface | 14 |
| The ESDL Command Syntax | 14 |
| ESDL Language Reference | 28 |
| ESDL Structures | 28 |
| ESDL Datatypes | 31 |
| ESDL Attributes | 33 |

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.

Workflow Tutorial

Overview

In this section, we will:

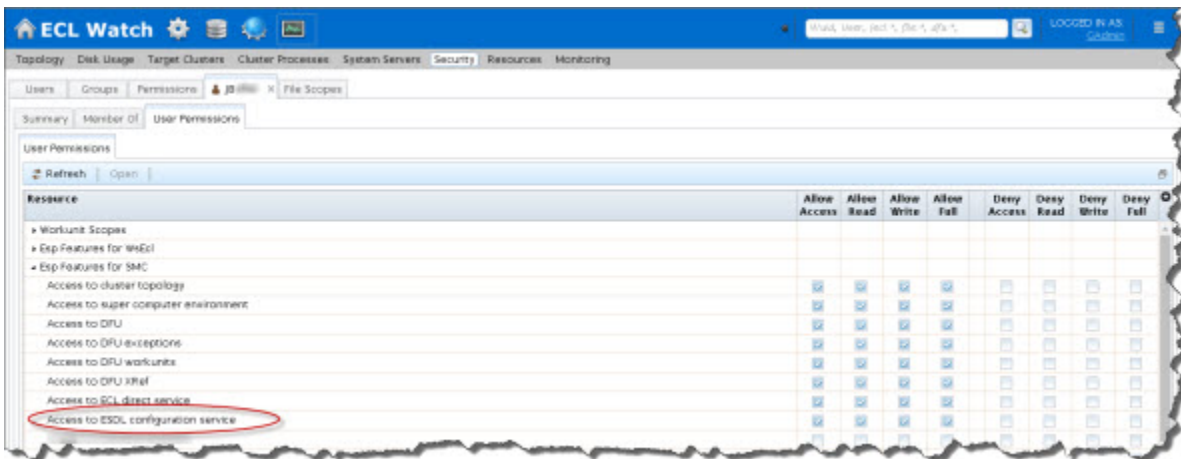
- Use Configuration Manager to add a Dynamic ESDL-based ESP Service and bind it to a port on an ESP server.
- Write an ESDL Service Definition.
- Generate ECL from the ESDL Service Definition, copy it to your ECL repository, and then write the ECL query that will deliver the result (the business logic).
- Compile the ECL business logic query and publish it to a Roxie cluster.

At this point you can test the query using WsECL.

- Publish the Dynamic ESDL definition.
- Bind the service methods to the Roxie queries using a configuration XML file.

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.



Configure and Bind a Dynamic ESDL service

This step adds 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:

Centos/Red Hat


```
sudo /sbin/service hpcc-init stop
```

Ubuntu

```
sudo service hpcc-init stop
```

Debian 6 (Squeeze)

```
sudo /etc/init.d/hpcc-init stop
```

| | |
|---|--|
|  | You can use this command to confirm HPCC processes are stopped (on Centos/Red Hat): |
| | <pre>sudo /sbin/service hpcc-init status</pre> |
| | For Ubuntu |
| | <pre>sudo service hpcc-init status</pre> |
| | For Debian 6 (Squeeze) |
| | <pre>sudo /etc/init.d/hpcc-init status</pre> |

2. Start the Configuration Manager service.

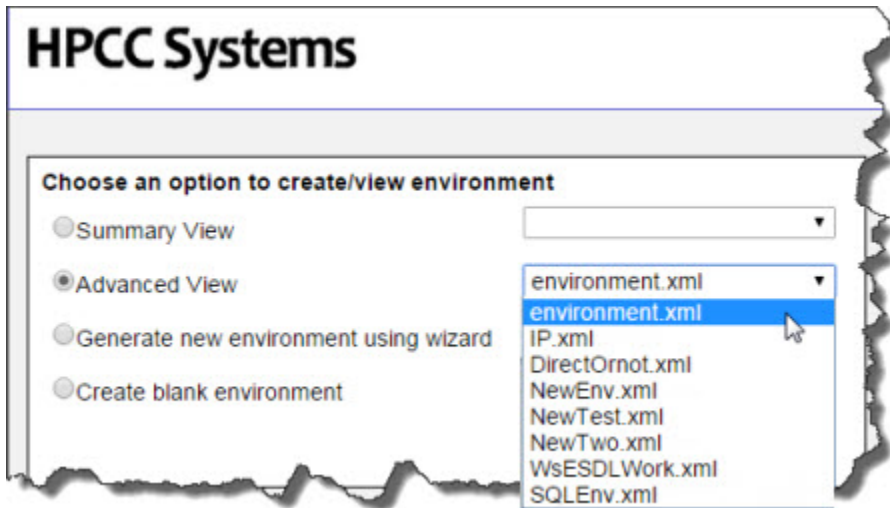
```
sudo /opt/HPCCSystems/sbin/configmgr
```

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

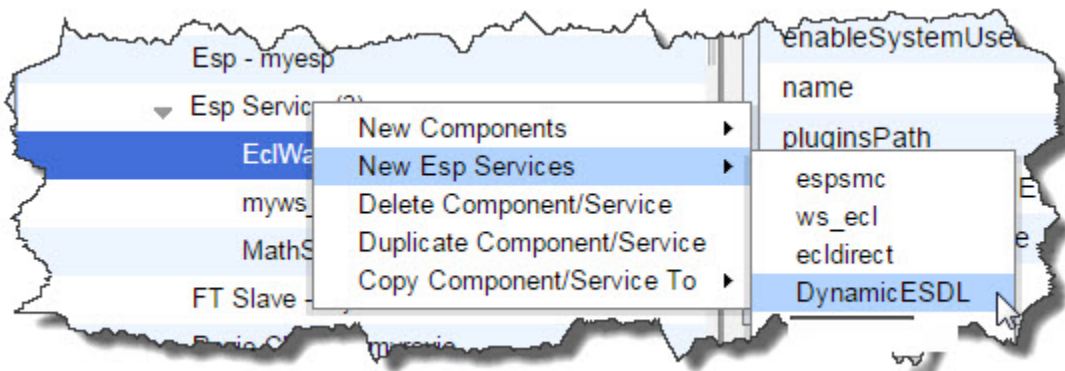
```
http://<node ip >:8015
```

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



7. Provide a name for the service.

DynamicESDL

Attributes

| name | value |
|-----------------------------|-------------------------------------|
| ESDL Service Description | My ESDL Based Web Service Interface |
| LoggingManager | |
| Dynamic ESDL Service Name | MyDynamicESDL_Service |
| ESDL service namespace base | |

8. Select your ESP , then select the ESP Service Bindings tab.

HPCC Systems

Write Access

Navigator

- Environment - environment.xml
 - Hardware
 - Software
 - Dafiesrv - mydafiesrv
 - Dali Server - mydali
 - Dfu Server - mydfuserver
 - Directories
 - Drop Zone - mydropzone
 - Ecl Agent - myeclagent
 - Ecl CC Server - myeclccserver
 - Ecl Scheduler - myeclscheduler
 - Esp - myesp**

EspProcess

Attributes | **ESP Service Bindings** | Authentication | HTTPS | Instances | Notes

| name | defaultServiceVersion | defaultForPort | port | protocol | resourcesBasedn |
|-----------|-----------------------|----------------|------|----------|-----------------------------|
| myespismc | true | | 8010 | http | ou=SMC,ou=EspServices,ou= |
| myws_ecl | | true | 8002 | http | ou=WsEcl,ou=EspServices,ou= |

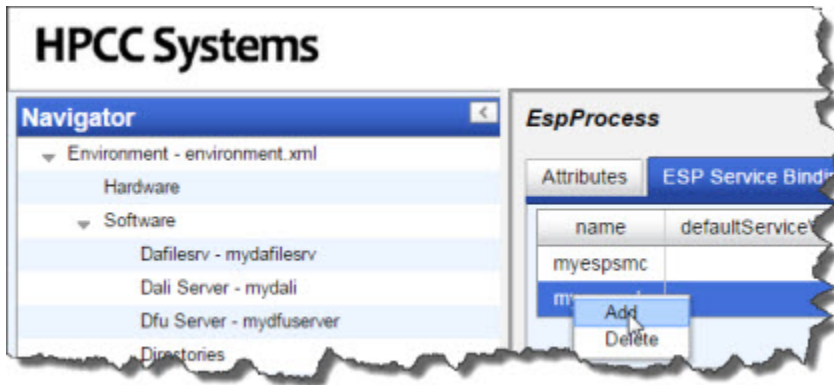
URL Authentication

| description | path | resource | access |
|----------------------------|------|-----------|--------|
| Root access to SMC service | / | SmcAccess | Read |

Feature Authentication

| authenticate | description | resource |
|-----------------------|----------------------------|-----------|
| Access to SMC service | Root access to SMC service | SmcAccess |

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



10. Provide a name, port, and then select the service. The service definition you added will display in the list of available services.



11. 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**.

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

You may want to create a script to push out the XML file to all nodes. A sample script is provided with HPCC. The following command copies the XML files out to all nodes as required:

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

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

Centos/Red Hat

```
sudo /sbin/service hpcc-init start
```

Ubuntu

```
sudo service hpcc-init start
```

Debian 6 (Squeeze)

```
sudo /etc/init.d/hpcc-init start
```



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 /opt/HPCCSystems/sbin/hpcc-run.sh -a hpcc-init start
```

Write the ESDL Service Definition

In this section, we will write the Service Definitions. The program listing below shows a service called *MathService*. It contains one method, *AddThis*, with a simple request and a simple response defined.

```
ESPservice MathService
{
    ESPmethod AddThis(AddThisRequest, AddThisResponse);
};

ESPrequest AddThisRequest
{
    int    FirstNumber;
    int    SecondNumber;
};

ESPresponse AddThisResponse
{
    int    Answer;
};
```

1. Save the file as MathService.ecm.

Generate ECL definitions from the ESDL Service Definition

In this section, we will generate ECL from this ESDL Service Definition file. This will use the esdl executable that is installed with Client Tools.

You will find this in C:\Program Files (x86)\HPCCSystems\5.2.0\clienttools\bin on a Windows machine

or /opt/HPCCSystems/bin/ on a Linux machine.

1. From the command line, run:

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

This produces a file named MathService.ecl in the current directory

```
/** 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. **/  
/*=====*/
```

2. Copy the MathService.ecl file to a module in your ECL Repository. For example, myMathService.

3. Write ECL to support the service:

```
IMPORT MyMathService;  
rec_in := MyMathService.MathService.t_AddThisRequest;  
  
First_Row := ROW ([], rec_in) : STORED ('AddThisRequest', FEW);  
c:= first_row.FirstNumber + first_row.SecondNumber;  
ds_out := ROW ({c},MyMathService.MathService.t_AddThisResponse);  
OUTPUT(ds_out, NAMED('AddThisResponse'));
```

4. Compile and Publish to your Roxie cluster.

5. Test the service using WsECL :

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

Publish the ESDL Service Definitions and Bind the ESDL Service

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

1. Publish the Dynamic ESDL definition

```
esdl publish MathService MathService.ecm -s nnn.nnn.nnn.nnn -p 8010 --version 1
```

Replace nnn.nnn.nnn.nnn with your ECL Watch ESP server's IP address.

2. Create the configuration XML file

```
<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>
```

Where name is the name of your method(s) and url is the Roxie server's IP address and port, finally queryname is the published name (alias) of the query. For a multi-node Roxie, you can use a range in the form of nnn.nnn.nnn.nnn. The ESP schedules requests to the target Roxie in a round-robin fashion.

Optionally, your method could include context information as illustrated in the above example. The context information should be formatted such that it is consumable by the target ECL query. The HPCC DESDL ESP does not impose any restrictions on the context information passed in the configuration file other than it must be valid XML.

3. Save the file as MathSvcCfg.xml
4. Bind the service methods to the queries using the configuration XML.

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

Where myesp is the name of your ESP process, 8003 is the port you reserved for your Dynamic ESDL service.

5. 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 <command></i> | 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. |
| get-binding | Get ESDL binding. |

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 filename.ecm [<outdir>] [options]

| | |
|---------------------------|--|
| <i>filename.ecm</i> | The file containing the ESDL definitions. |
| <i>-x, --expand-edxml</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 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 | (srcdir <outdir>)/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 [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.ecl |

This generates XSD from the ESDL definition.

Examples:

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

esdl wsd

esdl wsd [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.ecl |

This generates WSDL from ESDL definition.

Examples:

```
esdl wsd MathService.ecm .
```

esdl publish

esdl publish <servicename> <filename.ecm>][options]

| | |
|-----------------|---|
| servicename | The name of the service to publish |
| filename.ecm | The ESDL (*.ecm) file containing the service definitions. |
| --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 |

Publishes an ESDL service definition to the system datastore.

Examples:

```
esdl publish mathservice mathservice.ecm -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]

| | |
|-------------------------------|---|
| ESDLServiceDefinition-Name | The name of the ESDL service definition to delete |
| ESDLServiceDefinition-Version | 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

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) |
| --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 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="<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 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>sometrxdid</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.

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 <TargetESPProcessName> <TargetESPBindingPort | TargetESPServiceName> [options]

| | |
|---|---|
| TargetESPProcessName | The target ESP Process name |
| TargetESPBindingPort TargetESPServiceName | Either target ESP binding port or the target ESP service 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 unpublish ESDL Service based bindings.

To unbind 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 (the ESP Binding), and the name of the service you are unbinding.

Example:

```
esdl unbind-service myesp 8003
```

esdl bind-method

esdl bind-method <TargetESPProcessName> <TargetESPBindingName> <TargetServiceName> <TargetServiceDefVersion> <TargetMethodName> [options]

| | |
|-------------------------|---|
| TargetESPProcessName | The target ESP Process name |
| TargetESPBindingName | Either target ESP binding name |
| TargetServiceName | The name of the Service to bind (must already be defined in dali.) |
| TargetServiceDefVersion | The version of the target service ESDL definition (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-service myesp 8003 MathSvc.1 MathSvc --config MathSvcCfg.xml -s nnn.nnn.nnn.nnn -p 8010
```

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
```

ESDL Language Reference

ESDL Structures

ESPstruct

ESPstruct is a set of elements grouped together under one name. These elements, known as *members*, can have different types and different lengths. Structures can be nested and support inheritance.

Example:

```
ESPstruct NameBlock
{
    string FirstName;
    string MiddleName;
    string LastName;
    int Age;
};
```

ESPrequest

The request structure for a method.

Example:

```
ESPrequest MyQueryRequest
{
    string FirstName;
    string MiddleName;
    string LastName;
    string Sortby;

    bool Descending(false);
};
```

ESPresponse

The response structure for a method.

Example:

```
ESPresponse MyQueryResponse
{
    string FirstName;
    string MiddleName;
    string LastName;
};
```

ESParray

A structure for unbounded arrays. Arrays support inheritance and can be nested.

Example:

```
ESPstruct NameBlock
{
    string FirstName;
    string MiddleName;
    string LastName;

    int Age;
};

ESParray<ESPstruct NameBlock, Name> Names;
```

This results in something like:

```
<Names>
  <Name>
    <FirstName>James</FirstName>
    <MiddleName>Joseph</MiddleName>
    <LastName>Deerfield</LastName>
    <Age>42</Age>
  </Name>
  <Name>
    <FirstName>Emily</FirstName>
    <MiddleName>Kate</MiddleName>
    <LastName>Constance</LastName>
    <Age>33</Age>
  </Name>
</Names>
```

ESPenum

A structure containing an enumerated value.

Example:

```
ESPenum EyeColors : string
{
    Brn("Brown"),
    Blu("Blue"),
    Grn("Green"),
};

ESPstruct Person
{
    string FirstName;
    string MiddleName;
    string LastName;

    ESPenum EyeColors EyeColor("Brown"); //provides a default value
};
```

ESPinclude

ESPinclude allows you to include an external ESDL file. This is similar to the #include statement.

Example:

```
ESPinclude(commonStructures);
```

ESPservice

This defines an ESP web service interface. Once defined, this interface definition can be assigned (bound) to a Dynamic ESDL-based ESP Service.

An ESPservice should contain one or more method definitions.

Example:

```
ESPservice MyService
{
    ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
    ESPmethod MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

ESPmethod

This defines a method definition you can reference in an ESPservice structure. The method definition should contain references to a previously defined ESPrequest and ESPresponse.

Example:

```
ESPservice MyService

{
    ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
    ESPmethod
    [
        description("MyMethod Two"),
        help("This method does everything MyMethod1 does plus a few extra features"),
        min_ver("1.2")
    ]
    MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

ESDL Datatypes

boolean bool

A boolean or logical data type having one of two possible values: true (1) or false (0).

Example:

```
boolean includeFlag;  
bool includeMore;
```

string

A data type consisting of sequence of alphanumeric characters.

Example:

```
string FirstName;
```

int

An integer value.

Example:

```
int Age;
```

int64

A 64-bit integer value

Example:

```
int64 Iterations;
```

float

A 4-byte floating point or real number.

Example:

```
float Temperature;
```

double

An 8-byte floating point or real number.

Example:

```
double Temperature;
```

binary

A data type containing binary data, similar to a BLOB .

Example:

```
binary RetinaScanSample;
```

ESDL, XSD, and ECL Type Mapping

| ESDL | XSD | ECL |
|--------------|--------------|-------------------------|
| Bool boolean | bool | boolean |
| Binary | Base64Binary | String (base64 encoded) |
| Float | float | REAL4 |
| Double | double | REAL8 |
| Int | int | INTEGER |
| Int64 | long | INTEGER8 |
| String | string | String |

ESDL Attributes

You can use ESDL attributes to extend and override the default behavior of an ESDL definition. For example, adding a `max_len(n)` to a string defines the string will only need to store a certain number of characters.

Many attributes are treated as hints that may have more effect on some implementations than others. For example, `max_len(n)` will affect generated ECL code, but is ignored when generating C++.

max_len (n)

The `max_len` attribute specifies the field length for ECL string field.

Example:

```
[max_len(20)] string City;
```

It means that in ECL, City field is a fixed length of 20 chars. For integer type, the `max_len` means the maximum size in bytes for the integer (8*`max_len` bits integer).

Example:

```
[max_len(3)] int Age;
```

This generates ECL code:

```
integer3 Age(xpath('Age'));
```

This attribute works for ESPenum type, too. The ECL type is also string.

```
[max_len(2)] ESPenum StateCode State;
```

Here the StateCode is 2-char state code enumeration.

This attribute can also be used for ESPstruct, ESPrequest, ESPresponse:

```
ESPstruct [max_len(1867)] IdentitySlim : Identity  
{  
    ...  
};
```

This generates ECL code:

```
export t_MyQuery := record (share.t_Name), MAXLENGTH(1867)  
{  
};
```

The ECL option `MAXLENGTH` helps ECL engine better manage memory.

This does not affect the type in the XSD/WSDL. ESP ignores this attribute when generating the schema. The type for a string is `xsd:string` which has no length limit. Therefore, the schema stays the same if the field length changes in the Roxie query.

ecl_max_len (n)

This `ecl_max_len` attribute tells the ECL generator to use ECL `maxlength` instead of the regular field length.

Example:

```
[ecl_max_len(50)] string CompanyName;  
[max_len(6)] string Gender;
```

The generated ECL code is:

```
string CompanyName { xpath("CompanyName"),maxlength(50) };  
string6 Gender { xpath("Gender") };
```

Note: when both *max_len* and *ecl_max_len* are specified, *ecl_max_len* is used and *max_len* is ignored.

ecl_name ("name")

The *ecl_name* attribute specifies the field name in generated ECL code. By default, the field name in ECL is the same as the name defined in ECM. However, in some cases, the name could causes issues in ECL. For example keywords in ECL cannot be used as a field name.

Example:

```
[ecl_name("_export")] string Export;  
[ecl_name("_type")] string Type;
```

Here, both **EXPORT** and **TYPE** are ECL keywords and cannot be used as ECL field names. We use *ecl_name* to tell the esdl2ecl process to generate acceptable names.

counter and count_val

These two attributes are used to help ESP calculate the record count of the response.

counter counts the number of children of the nodes. When the node is an array, it is the same as the number of items in the array.

count_val will use the value of the node as record count. Field **RecordCount** is implicitly marked as *count_val*.

When an response has multiple counter, count_val, the sum of the values is returned as record-count.

Example:

```
[counter] ESParray<MyRecord, Record> Records;  
[count_val] int TotalFound;
```

max_count_var

The *max_count_var* attribute is used to specify the expected max items in a dataset (ESParray).

Example:

```
[max_count_var("iesp.Constants.JD.MaxRecords")] ESParray <ESPstruct MYRecord, Record> Records;
```

Roxie will define the constant `iesp.Constants.JD.MaxRecords` and change it at will without affecting ESP.

ecl_null (n | string)

The *ecl_null* attribute tells ESP to remove the field altogether if the field's value is *n* or *string*. This provides a means to remove a field completely when there is no data for it.

Example:

```
[ecl_null(0)] int Age;  
[ecl_null("false")] bool IsMatch;
```

Age 0 means there is no Age data for this person. So, if Age is 0, the <Age> tag is not returned.

Without this attribute, <Age>0</Age> would be returned.

For the second example, a bool value of *false*, returned as a string, is treated as null and therefore the tag is not returned.

leading_zero(n)

The *leading_zero* attribute adds zero(s) to the field value so that the total length is *n*.

Example:

```
ESPstruct Date
{
    [leading_zero(4)] Year;
    [leading_zero(2)] Month;
    [leading_zero(2)] Day;
};
```

So the Date will always have a 4-digit Year and a 2-digit Month and a 2-digit Day.

ecl_hide

The *ecl_hide* attribute hides the field from ECL (that is, the field is removed when generating the ECL code). This is used for some special cases.

Example:

```
ESPstruct Relative
{
    [ecl_hide] ESParray<ESPstruct Relative, Relative> Relatives;
    "
};
```

In this case, the Relative structure is defined in a recursive manner, and ECL does not support such a construct. Therefore, we use *ecl_hide* to avoid the recursive definition in ECL.

Sometimes a field is hidden from ECL for other reasons. In these cases, *ecl_hide* is not needed.

ecl_type ("type")

The *ecl_type* attribute defines the field type in ECL.

Example:

```
[ecl_type("Decimal10_2")] double RetailPrice;
```

ESDL does not have a monetary type, so we use *ecl_type* to define it.

ecl_keep

The *ecl_keep* attribute keeps the field in the generated ECL even though this field would have been hidden without this attribute.

min_ver

The *min_ver* attribute allows you to define the minimum (earliest) version where a field is visible. Requests using a prior version will not have access to the field.

Example:

```
[min_ver("1.03")] bool IsValid;
```

max_ver

The `max_ver` attribute allows you to define the maximum (latest) version where a field is visible. Requests using a later version will not have access to the field.

Example:

```
[max_ver("1.04")] bool IsValid;
```

depr_ver

The `depr_ver` attribute allows you to declare a field's end of life version. The field is deprecated at the specified version number. Requests using that version or any subsequent version will not have access to the field.

Example:

```
[depr_ver("1.04")] bool IsValid;
```

help

The `help` attribute (valid only for an `ESPMETHOD`) allows you to specify some additional text to display on the form that is automatically generated to execute a method.

Example:

```
ESPService MyService

{
    ESPMethod MyMethod1(MyMethod1Request, MyMethod1Response);
    ESPMethod
    [
        description("MyMethod Two"),
        help("This method does everything MyMethod1 does plus a few extra features"),
        min_ver("1.2")
    ]
    MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

description

The `description` attribute (valid only for an `ESPMETHOD`) allows you to specify some additional text to display on the form that is automatically generated to execute a method.

Example:

```
ESPService MyService

{
    ESPMethod MyMethod1(MyMethod1Request, MyMethod1Response);
    ESPMethod
    [
        description("MyMethod Two"),
        help("This method does everything MyMethod1 does plus a few extra features"),
        min_ver("1.2")
    ]
    MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

```
} ;
```