Using Ensemble as an ESB

Version 2018.1
2024-05-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using Ensemble as an ESB
Ensemble Version 2018.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Figures

About This Book

This book describes all aspects of using Ensemble as an Enterprise Service Bus (ESB). This book is intended for the following audiences:

- Developers at client sites who will be using the public Service Registry and developing applications using the services through the ESB. These developers should be familiar with the first chapter, "Enterprise Service Bus and Registry Overview", the second chapter, "Accessing the Public Service Registry through the Public REST API", and the "Service Registry Public API Reference". Developers at client sites do not need to read any of the other chapters, which describe the internals of using Ensemble as an ESB. Optionally, an ESB site can provide supplementary information to these client developers describing the conventions and practices used by the ESB.
- System architects or designers of an ESB should be familiar with all chapters of this book.
- System administrators should be familiar with all chapters of this book, but should focus on "Administering the Public Service and External Service Registries", "Configuring an ESB," and "Configuring ESB Services and Operations".
- ESB custom component developers should be familiar with all chapters of this book, but should focus on "Configuring ESB Services and Operations".

This book consists of the following chapters, appendices, and reference:

- Enterprise Service Bus and Registry Overview
- Accessing the Public Service Registry through the Public API
- Administering the Public Service and External Service Registries
- Configuring an ESB
- Configuring ESB Services and Operations
- Appendix Configuring an Ensemble System and Creating a Namespace
- Appendix Configuring a Web Application
- Appendix Pass-through Service and Operation Walkthrough
- Service Registry Public API Reference

For a detailed outline, see the table of contents.

For general information, see *Using InterSystems Documentation*.

1

Enterprise Service Bus and Registry Overview

This chapter introduces using Ensemble as an Enterprise Service Bus, describes the Ensemble ESB architecture, and provides an overview of deploying an ESB. It contains the following sections:

- Enterprise Service Bus Concepts
- Enterprise Service Bus Architecture
- Deploying an Enterprise Service Bus

1.1 Enterprise Service Bus Concepts

An Enterprise Service Bus (ESB) provides a single point to both access and govern applications that have SOAP, REST, or other network APIs. The ESB provides the following capabilities:

- Provides a centralized location to discover and access the services.
- Isolates applications from detailed knowledge of the services, allowing you to change the description of a service in a single location without having to update all applications that are dependent on it. For example, you can change the address of the server or even the API of the service and handle the changes in the ESB. This provides protocol independence to your applications. If an application is developed using a SOAP API but a new service provides additional functionality with a REST API, the ESB can make the new REST service available as both REST and SOAP APIs allowing you to add the functionality of the new service while maintaining compatibility for existing applications.
- Provides a mechanism to organize and track the applications used by the enterprise and the dependencies between these applications. For example, the ESB can provide the contact information for technical support when a user has problems with an application.

Why would you need an ESB? Consider the following scenario:

1. Your enterprise has developed many critical applications. Each application has its own environment and user interface and is independent of the other applications.
2. You face new problems and competitive requirements to increase efficiency. To solve these problems you must get these independent applications to communicate and interact with each other.
3. You modify the applications to provide access via a SOAP or REST API.

4. You start to develop new applications calling these SOAP and REST APIs. These new applications can combine the functionality of the existing applications, making it possible to combine workflows and streamline procedures.
5. But each of these new applications requires detailed knowledge of the applications it is using: the address of the server running the application and the specific API needed to access the application.
6. As the number of these new applications grow, it becomes more difficult to maintain the overall collection of applications. Simply moving a service to a new server or making any change to the functionality of a service requires you to update every application dependent on it. It can be difficult even finding out which applications are dependent on a service.
7. Users are uncertain who to contact when they run into problems.

Using Ensemble as an ESB allows you to create a unified mechanism for accessing services. This makes it easier to maintain the set of applications and provides a mechanism to track usage and identify potential blockages.

1.2 Enterprise Service Bus Architecture

The Ensemble ESB architecture has the following components:

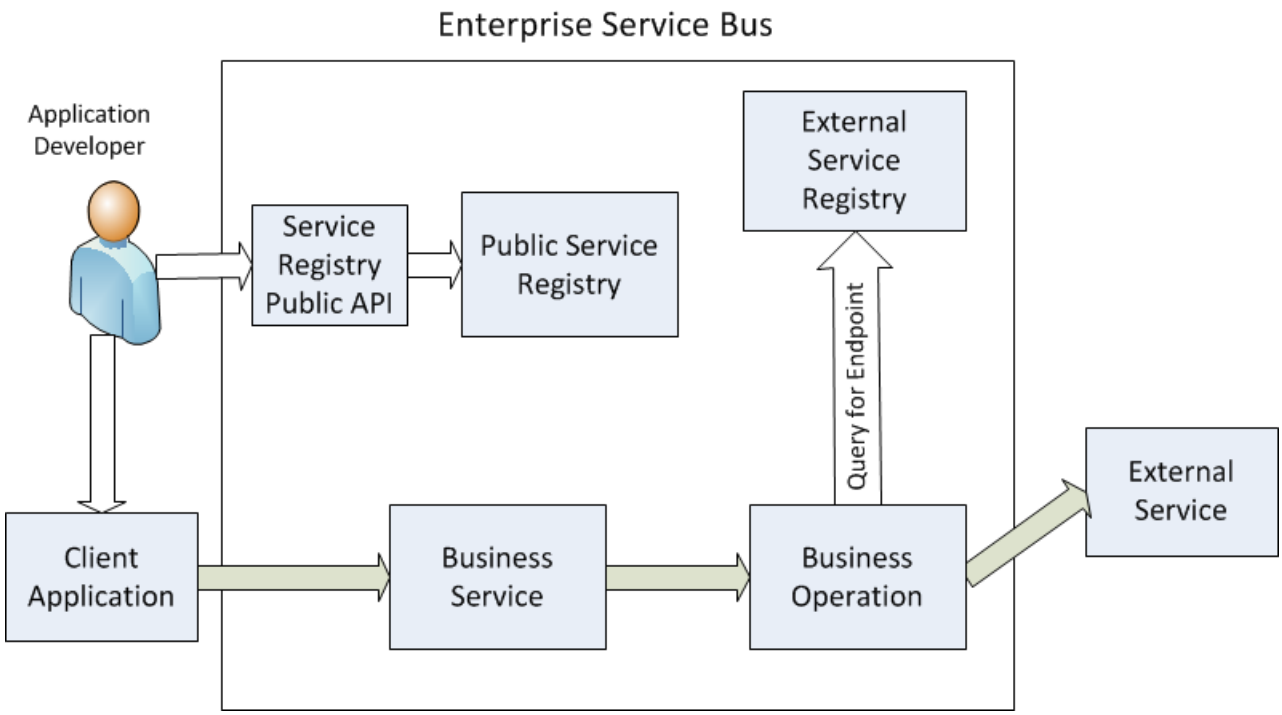
- Routing mechanism that connects the incoming requests with the server. This is implemented by a production with specialized business services and business operations, and, optionally, business processes.
- Public Service Registry—This service registry is accessible to ESB clients through the ESB public REST API. Developers use it to get information about the services available through the ESB.
- External Service Registry—This service registry is accessible only within the ESB production. It provides endpoint information to the ESB business hosts.
- SAML token validation service.

Both the ESB production and the service registries should be defined in a namespace that is used exclusively for the ESB.

Note: InterSystems recommends that only one ESB production should be running on an Ensemble instance.

In its simplest form, an ESB consists of pass-through services, pass-through operations, and the service registries. The following illustrates the architecture of a simple ESB.

Figure 1–1: Simple Enterprise Service Bus Architecture



Typically, the application developer uses a web page or application to query the ESB service registry to find out about the available services and get the URLs for the pass-through business services that provide access to the underlying services. This query and service discovery process takes place while the developer is creating the client application. Once the developer has the URLs, documentation, and other information needed to access the services, the client application does not need to access the service registry in order to call the services. In some environments, the client application may make runtime calls to the public API to ensure that the registry entry has not been modified since last accessed.

The client application calls a pass-through service. The pass-through service sends the message to the pass-through operation that is specified in its Target setting. The pass-through operation is configured to look up the endpoint URL in the External Service Registry. It then calls the external service using that endpoint URL. The external service returns the response to the pass-through operation, which directs the response to the pass-through service. The pass-through service, in turn, returns the response to the client application.

In addition to pass-through services and operations, you can define more complex services, operations, and business processes to add capabilities such as:

- Routing calls from a single incoming service to multiple external services depending on the content of the call. This allows the ESB to provide a service that is not available from a single external service. The client applications are insulated from the external services that are required to implement the expanded service.
- Modifying the parameters or protocol used for a service. If a set of applications are developed using one service, but later a superior service with a different API becomes available, rather than modifying each application, the translation can be done in the ESB. The client applications are insulated from the changes required to switch from one external service to another.
- Implementing the service directly on the ESB. If there is no external service that meets the need, one can be implemented on the ESB using ObjectScript.

But adding these more complex services to an ESB has an efficiency cost. The additional processing costs of these complex services slows the time it takes for the ESB to handle a request and reduces the throughput.

For ESB systems that require very high throughput, you can reduce the overhead of processing a request by eliminating the persistent messages. The persistent messages are the object that is sent from the pass-through service to the pass-through operation and the object returned by the operation to the service. These objects are stored in the Caché database. These persistent messages are very useful in tracking and reporting on the calls processed by the ESB and in troubleshooting any problems. But creating these objects requires resources and for systems with very high throughput, the storage required for these objects can be very large. To maintain the system, you must frequently purge these messages. You can suppress the use of persisted messages to gain efficiency at a cost of reduced flexibility. See "Suppressing Persistent Messages in Pass-through Services and Operations" for more information.

Note: If you are running a HealthShare product, the HealthShare service registry is distinct from the Ensemble service registry. The HealthShare service registry provides a similar capability to the Ensemble External Service Registry. In most cases you should continue to use the HealthShare service registry and not use the Ensemble External Service Registry.

1.3 Configuring an Enterprise Service Bus

Installing and configuring Ensemble is described in *Preparing to Use Ensemble*. This document describes the few configuration procedures that are specific to Ensemble installations that are being used as an ESB. These procedures are:

- Create an Ensemble namespace to contain the ESB production and the Service Registry.
- Configure the CSP gateway.
- Create External Service Registry entries that define the endpoints for external services used by the ESB.
- Create the ESB production, add the business services and business operations that provide the services, and start the production.
- Create the roles and users needed to access the Public Service Registry through the Ensemble portal and through the public REST API.
- Create the web applications needed to make the business services available to clients.
- Create Public Service Registry entries that describe the services accessible through the ESB.

See "Overview of Configuring Ensemble as an ESB" for details.

2

Accessing the Public Service Registry through the Public REST API

The public REST API provides read access to the public Service Registry. The API provides information about the services available through the ESB. You can make REST calls to return information about all services or can query for services that meet specified conditions. The calls return service information in JSON.

Depending on how the ESB is configured, you can access the API by with a username and password or by anonymous access. The registry returns services based on the permissions given to the user or to anonymous access. The service registry administrator controls which services are returned depending on the login information. For example, the administrator may choose to configure the registry in one of the following ways:

- Return information about all services to all users, including those logged in anonymously. With this policy, getting information about a service does not imply that you have permissions to call the service.
- Limit the services returned to the ones that the user has permission to call based on the login account or anonymous access.

Each service registry entry contains the following information:

- Service identification, which consists of a name, a domain, and a version.
- Endpoint URL, which gives the address to call the service. Typically, this address is on the ESB server. The endpoint of the external server that actually performs the service is not returned.
- Protocol used to call the service, such as REST, SOAP, HTTP, or TCP.
- Return mechanism used by the service, such as synchronous return or callback.
- Description, which provides a brief text summarizing the service.
- Topics, which are used to search for services.
- Schema, which contains a schema description of the return messages for those messages that have a schema language. Schema information includes the schema format, provide a URL for the schema defintion, or provide the content of the schema.
- Actions describe the HTTP request methods that can be used at the specified endpoint. Each action can describe the impact of the request method on the server and can specify the schema of the incoming message and response message.
- Contacts describe users or organizations that can provide information or support for the service.
- Attributes provide a mechanism for the registry administrator to extend the information provided about each service. Attributes consist of a series of name-value pairs.

- Files provide a mechanism to provide any set of files for the service. Typically, these are used to provide documentation or templates for using the service. When the public API returns information about a file, it does not include the file contents because the contents can be very large. To access the file contents, you must make an explicit request.

The following sections describe:

2.1 Summary of the Service Registry Public API

This section provides a summary of the REST calls in the public API. Full details about each call are provided in "Service Registry Public API Reference".

REST Call	URL and Description
Get About	GET `/about` Lists the public API REST calls that are available.
Get File By ID	GET `/services/id/`*name*`/`*domain*`/`*version*`/file/`*filename* Returns the specified file.
Get Services	GET `/services` Shows all services accessible to the user.
Get Services By ID	GET `/services/id/`*name*`/`*domain*`/`*version* Returns the service, if accessible to the user, that matches the specified name, domain, and version.
Get Services By Protocols	GET `/services/protocols/`*protocol-list* Shows all services accessible to the user that match a protocol in the list.
Get Services By Stages	GET `/services/stages/`*stage-list* Shows all services accessible to the user that match a stage in the list.
Get Services By Version	GET `/services/version/`*version* Shows all services accessible to the user that match the specified version.
Get Services By Word	GET `/services/includesword/`*search-text* Shows all services accessible to the user that match the specified search text.
Get Services Since Modified	GET `/services/modifiedsince/`*date-time* Shows all services accessible to the user whose registry entry has been created or updated since the specified date-time.

2.2 JSON Description of Services

The Get Services calls return a JSON message with any number of services. If no services match the Get Services request and are accessible to the user, the call returns an empty JSON message. Otherwise, the Get Services call returns a JSON message with one or more services that match the request and are accessible to the user. This section provides an example of a returned JSON message and describes the fields in the message.

The following message could be returned by any Get Services call. The WSDL schema is abbreviated in the listing but appears in full in the JSON message.

```
[
{
    "Name": "MathServiceSOAP",
    "Domain": "UnitTest",
    "Version": "1.1",
    "Stage": "Live",
    "Protocol": "SOAP",
    "Description": "Add 2 Numbers",
```

```
    "Endpoint": "https://hostname/enslatest/csp/support/Demo.SOAP.MathService.cls",
    "ResponseStyle": "Sync",
    "LastModified": "2015-03-16 19:07:47.469",
    "Topics":
    [
        "Test",
        "Maths"
    ],
    "Contacts":
    [
      {
        "Identity": "QD Developer Moon",
        "Type": "Operator",
        "Details": "Details of contact",
        "BusinessPartner": "QD",
        "Notes": "This SOAP service is designed to have minimum moving parts"
      }
    ],
    "Schema":
    {
       "Type": "Notes",
       "Ref": "https://hostname/enslatest/csp/support/Demo.SOAP.MathService.cls?wsdl=1",
       "Content": "<definitions targetNamespace='http://tempuri.org'> ...</definitions>",
       "Notes": "Some WSDL"
    },
    "Public": true,
    "Attributes":
    [
       {
          "Name": "One",
          "Value": "1"
        }
    ],
    "Files":
    [
       {
          "Filename": "SOAPMathService.WSDL",
          "FileExtention": ".WSDL",
          "MIMEType": "text/text",
          "CharEncoding": "UTF-8",
          "FileSize": "1.44 KB",
          "Contents": null
       }
    ],
    "Actions":
    [
       {
          "Name": "Sum",
          "Ref": "Sum",
          "Verb": "POST",
          "Description": "Add up 2 numbers",
          "ReadOnly": false,
          "Idempotent": true
       }
    ]
},
{
    "Name": "PublicREST",
    "Domain": "UnitTest",
    "Version": "0.9",
    "Stage": "Live",
    "Protocol": "REST",
    "Description": "REST Call for the Public Registry",
    "Endpoint": "http://hostname:57774/csp/registry/docserver/public",
    "ResponseStyle": "Sync",
    "LastModified": "2015-03-05 16:15:33.38",
    "Topics":
    [
       "Public",
       "Search",
    ],
    "Public": true,
    "Attributes":
    [
       {
          "Name": "Security",
          "Value": "Username and Password"
       }
    ],
    "Files":
    [
       {
           "Filename": "TestPlan",
           "MIMEType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
           "CharEncoding": "UTF-8",
```

```
           "FileSize": "16.95 KB",
            "Contents": null
        }
    ],
    "Actions":
    [
        {
            "Name": "public",
            "Ref": "public",
            "Verb": "GET",
            "Description": "Returns REST endpoint for public registry",
            "RequestSchema":
            {
                "Type": "Notes",
                "Notes": "This is the REST endpoint to query the public Registry"
            },
            "ResponseSchema":
            {
               "Type": "REST Information
            },
            "ReadOnly": true,
            "Idempotent": true
        }
    ]
}
]
```

The following lists and describes the fields that can appear in the service description. For all fields, the ESB site may provide guidelines and general practices, that provide information on how the fields are used in its service registry.

Name

Name identifies the service. The other attributes identifying the service are the **Domain** and **Version**. The **Domain** typically identifies the category of the service and **Version** is a string representing a version number. The combination of Name, Domain, and Version is unique for each service registry entry. For example, two or more services may have the same name as long as the domains are different or the versions are different.

Domain

Domain specifies the category of the service.

Version

Version identifies the version of the service.

Stage

Lifecycle Stage describes the development state of the service. It can have one of the following values: **Concept**, **Development**, **Test**, **Staging**, **Live**, **Deprecated**, and **Defunct**.

Protocol

Service Protocol describes the protocol used by the client to access the service. It can have any of the following values: **File**, **FTP**, **HL7**, **HTTP**, **REST**, **SOAP**, **SQL**, **X12**, and any custom value.

Description

Description provides a brief explanation of the service

Endpoint

Endpoint specifies the location of the service. This is the location you use to call the service.

Response Style

Response Style describes how the service returns a value. It may have one of the following values: **Sync**, **Callback**, **Remote Deposit**, or a blank value.

Last Modified

The date and time that the registry entry was last changed in the ESB registry. you can discover services that were added or updated since a specified date and time by calling Get Services By Modified.

Topics

Topics provide a series of terms that are useful in searching for services using the Get Services By Word call.

Contacts

Contacts specify people or organizations that support or are associated with the service. A contact can contain the following subfields:

- **Identity**—Specifies the name or other identity of the person or organization.
- **Type**—Specifies whether the contact is the service **Author**, **Consumer**, **Provider**, **Operator**, **Manager**, or **Sponsor**.
- **Details**—Specifies details about the contact.
- **Business Partner**—A field that is used internally by the service registry but that is not useful to an application using the public API.
- **Notes**—Specifies supplementary information about the contact.

Schema

The Schema describes the structure of the service messages. Schema contains subfields that describe or contain the structure definition. Schema can contain the following subfields:

- **Type**—Specifies the name of the schema definition mechanism used to describe the schema. It can bee WSDL, XSD, HL7, SEF, X12, AST, EDF, or a custom schema definition mechanism.
- **Ref**—Specifies the URL of the schema definition.
- **Content**—Provides the full text content of the schema definition.
- **Notes**—Provides supplementary information about the schema.
- **Thumbnail**—Provides a brief excerpt from the schema.

Public

Always has a true value. Used internally in the service registry.

Attributes

Attributes specify a list of Name-Value pairs that allow an ESB service registry to contain any information about a service.

Files

Files provide a way to include one or more files in a registry entry. For example, a file can contain documentation on the service. Each file in a single registry entry must have a unique filename. When you make a Get Services call, the returned JSON message does not include the file contents. To get the file contents, you make a separate **GetFileByID** call. Each File element consists of the following subfields:

- **Filename**—Provides the name of the file. Typically, this includes the file extension. You cannot have two files in a single registry entry that have the same filename.
- **File Extension**—Provides information on the format and purpose of the file. Optional.

- **MIMEType**—Provides information on the application used to access the file. Optional.
- **CharEncoding**—Provides the character encoding information. Optional.
- **File Size**—Provides the file size. Optional.
- **Contents**—This field is always null in the JSON service description. You must call the Get File By ID call to get the file contents.

Actions

Actions specify the SOAP actions or the REST HTTP request methods that you can use with the specified URL. SOAP actions provide a summary of the information defined in the WSDL. An action can contain the following subfields:

- **Name**—Specifies the name used by the Registry to identify this action.
- **Ref**—Specifies a string identifier of the action.
- **Verb**—Specifies the HTTP request method. Typically, this is GET, PUT, POST, or DELETE, but it can be any HTTP request method.
- **Description**—Specifies a text explanation of the action.
- **Request Schema**—Specifies the format of the incoming message body and consists of the following subfields: Type, Ref, Notes, Thumbnail, and Content.
- **Response Schema**—Specifies the format of the response message body and consists of the following subfields: Type, Ref, Notes, Thumbnail, and Content.
- **Read Only**—If checked, specifies that the call does not make any changes to the state of the server.
- **Idempotent**—If checked, specifies that making multiple identical calls have the same impact on the server as making a single call.

2.3 Get All Services

The Get Services REST call returns all service registry entries that you have permission to view. The call returns the complete registry entry for each service except that file contents are omitted. The registry returns all services that you have permission to view. The URL for this call is:

GET `/services`

For example, a complete URL, including the server and top-level directory is:

```
https://esb.example.com/registry/services
```

2.4 Get Service by ID

The Get Service by ID call returns information about the identified service. A service is uniquely identified by a name, domain, and version. The URL syntax for this call is:

GET `/services/id/`*`name/domain/version`*

For example, to get the service with the name `coffeemaker`, the domain `demo`, and the version `1.0`, use the following call:

```
https://esb.example.com/registry/services/id/coffeemaker/demo/1.0
```

The Get Service by ID returns the same information for the specified service as is returned for that service by Get All Services. Consequently, if you have called Get All Services, there is no need to call Get Service by ID.

Note: Although the purpose of this call is to return a single service, you can specify wildcards in the name, domain, and version parts of the URL. If you do use wildcards, the call returns all services that match the specification.

2.5 Getting Selected Services

Instead of getting all the services from the registry or getting a single service by ID, you can select a subset of services. You can specify any one the following selection criteria:

- Protocol—selects any service that has a protocol in the list. Protocols can be any of the following: **File**, **FTP**, **HL7**, **HTTP**, **REST**, **SOAP**, **SQL**, **X12**, and any custom value.
- Stage—selects any service that has a stage that is specified in the list. Stages can be any of the following: **Concept**, **Development**, **Test**, **Staging**, **Live**, **Deprecated**, and **Defunct**.
- Version—selects any service that has the specified version.
- Text search—selects any service that contains the specified text in any of the following fields: Name, Domain, Description, Endpoint, and Topics. The text comparison is not case-sensitive.
- Modification date—selects any service whose entry has been modified after the specified date.

The Get Services By Protocols call selects the services that have a protocol that matches one of the ones in the list. The URL syntax for this call is:

GET `/services/protocols/`*protocol-list*

The following call returns all services with the REST protocol:

```
https://esb.example.com/registry/services/protocols/REST
```

The following call returns all services with either the REST or SOAP protocol:

```
https://esb.example.com/registry/services/protocols/REST,SOAP
```

The Get Services By Stages call selects the services that have a stage that matches one of the stages in the list. The URL syntax for this call is:

GET `/services/stages/`*stage-list*

The following call returns all services with the Live stage:

```
https://esb.example.com/registry/services/stages/Live
```

The Get Services By Version call selects the services that have a version that matches the specified version. The URL syntax for this call is:

GET `/services/version/`*version*

For example, the following call returns all services with version 1.0:

```
https://esb.example.com/registry/services/version/1.0
```

The Get Services By Word call finds all services that contain the specified search text in any of the following fields:

- Name

- Domain
- Description
- Endpoint
- Topics

The URL syntax for the Get Services By Word call is:

GET `/services/includesword/`*search-text*

For example, the following call returns all services that have the text string “accounts, payable” in one of the fields:

```
https://esb.example.com/registry/services/includesword/accounts,%20payable
```

The Get Services Since Modified call returns all services whose registry entry has been modified after the specific date-time. The URL syntax for this call is:

GET `/services/modifiedsince/`*date-time*

For example, the following call returns all services whose entries were modified after the specific date-time:

```
https://esb.example.com/registry/services/modifiedsince/2015-02-1%2011:30
```

The date is specified as year-month-day, the URL space code %20 as a separator, and hours:minutes using a 24 hour clock.

2.6 Getting File Contents

The Get File By ID call returns the contents of the specified file. It can return either a text or a binary file. It does not return a JSON message or any file metadata, it only returns the contents of the file. The URL syntax for this call is:

GET `/services/id/`*name*`/`*domain*`/`*version*`/file/`*filename*

where *name*`/`*domain*`/`*version* identify the service and *filename* specifies the name of the file in that service.

For example, the following call returns the contents of a file:

```
https://esb.example.com/registry/services/id/coffeemaker/demo/1.0/file/HowToCallCoffeemaker.docx
```

Note: You should not use any wildcards in the Get File By ID call. Although it is possible to specify wild cards in the *name*`/`*domain*`/`*version* parts of the URL, the behavior of this call is undefined if there are multiple services that match the specification.

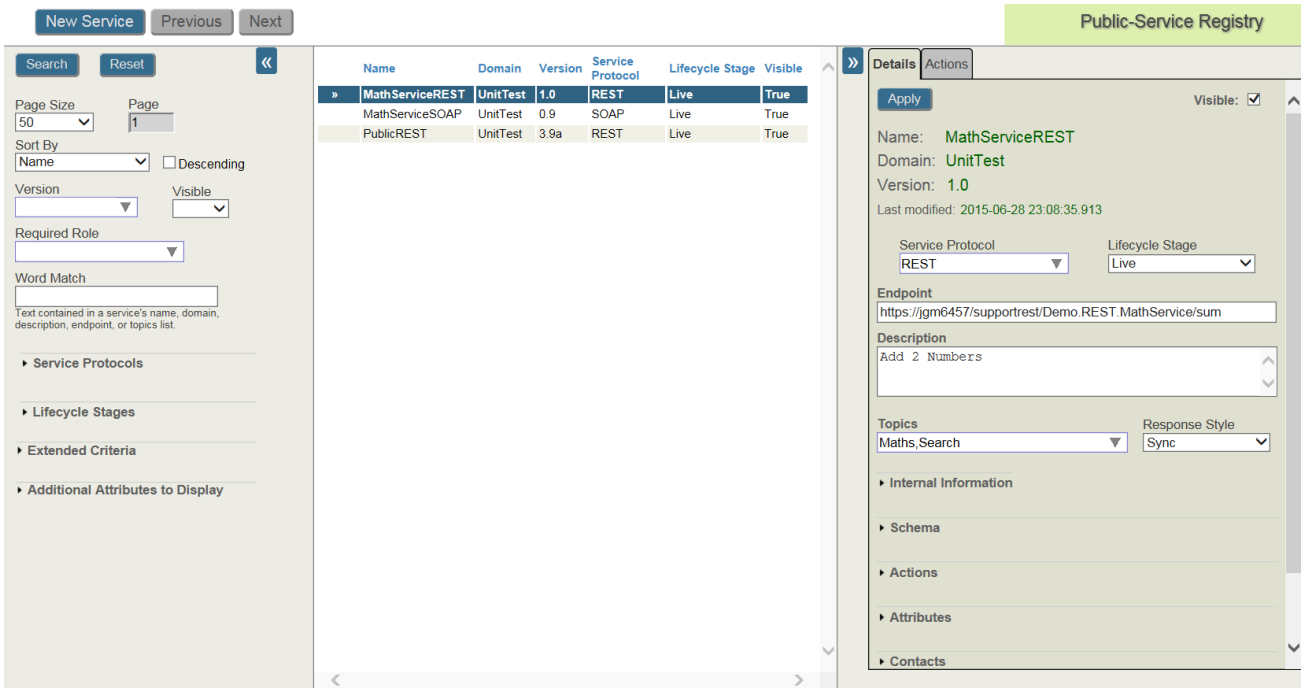
3

Administering the Public Service and External Service Registries

This chapter describes how to administer the public service registry and the external service registry. Although the registries have different purposes, they have a similar structure and user interface. This chapter contains:

3.1 Administering the Public Service Registry

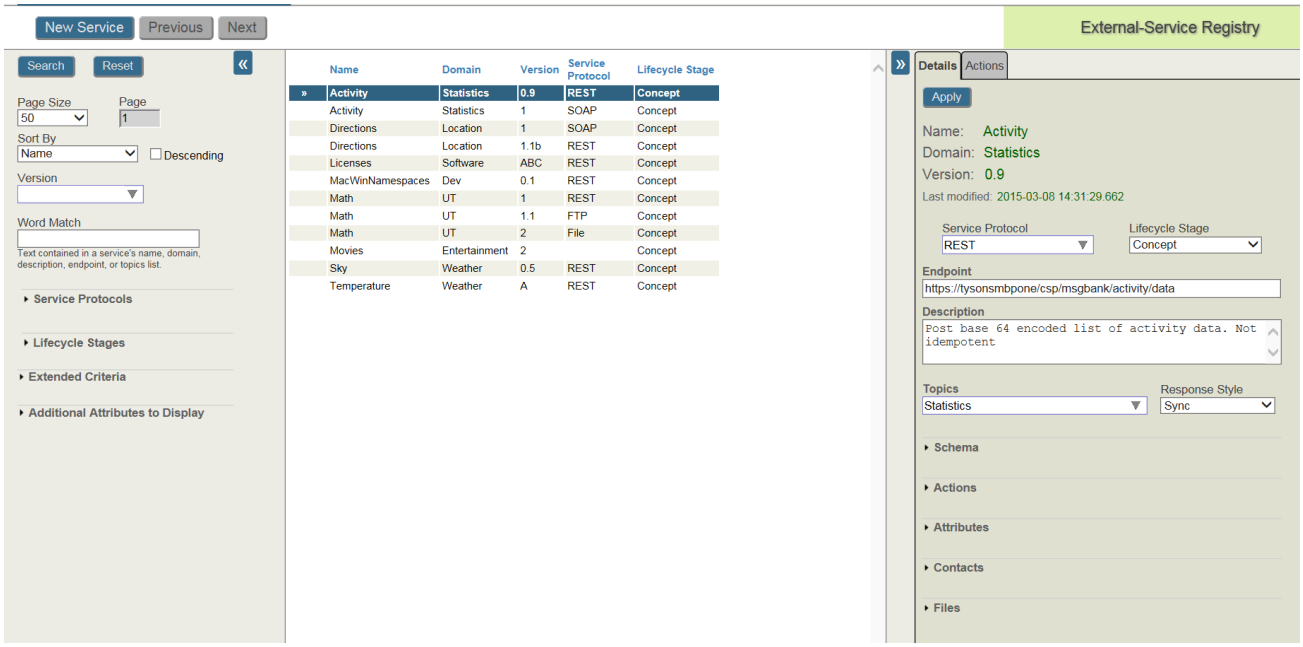
To administer the Public Service Registry, select **Ensemble** > **Configure** > **Public-Service Registry**. The Public Service Registry is protected by row-level security; consequently, you must have the `%EnsRole_RegistryManager` role in order to access the Public Service Registry.



For information on the fields in the public registry, see Fields in both the Public and the External Service Registries and Internal Fields in the Public Service Registry. For information on creating and maintaining entries in the service registries, see Creating and Maintaining a Service Registry Entry. For information on searching the service registries, see Searching and Viewing the Service Registry.

3.2 Administering the External Service Registry

To administer the External Service Registry, select **Ensemble** > **Configure** > **External-Service Registry**.



For information on the fields in the external registry, see Fields in both the Public and the External Service Registries. For information on creating and maintaining entries in the service registries, see Creating and Maintaining a Service Registry Entry. For information on searching the service registries, see Searching and Viewing the Service Registry.

3.3 Fields in both the Public and the External Service Registries

This section describes the fields in the public and external service registries. Most fields are identical in the public and external service registries. In addition to these fields that are common to both registries, the public Service Registry has some private fields. These fields are only accessible to the registry administrator and are not accessible through the public REST API.

Name

Name identifies the service. The other attributes identifying the service are the **Domain** and **Version**. Each registry should adopt conventions on how these attributes are used. The **Domain** typically identifies the category of the service and **Version** is a string representing a version number. Name in combination with Domain and Version must be unique for each entry. For example, two or more entries may have the same name as long as the domains are different or the versions are different.

Version

Version identifies the version of the service.

Domain

Domain specifies the category of the service.

Lifecycle Stage

Lifecycle Stage describes the development state of the service. It can have one of the following values: **Concept**, **Development**, **Test**, **Staging**, **Live**, **Deprecated**, and **Defunct**.

Service Protocol

Service Protocol describes the protocol used by the client to access the service. It can have any of the following values: **File**, **FTP**, **HL7**, **HTTP**, **REST**, **SOAP**, **SQL**, **X12**, and any custom value. In the External Service Registry, the Service Protocol determines how the Endpoint is used to set the business operation properties. You can explicitly enter TCP as the service protocol but it is not listed in the drop-down choices.

Description

Description provides a brief explanation of the service

Endpoint

Endpoint specifies the location of the service. On the public Service Registry, the endpoint is typically a URL of a business service on the ESB. On the External Service Registry, the endpoint provides information about the location of the data or service and is used to set the properties of the ESB host.

The format of the endpoint in the External Service Registry is dependent on the service protocol. For details on how the Endpoint is used for each type of service protocol, see "Using the External Service Registry to Configure ESB Hosts."

Response Style

Response Style describes how the service returns a value. It may have one of the following values: **Sync**, **Callback**, **Remote Deposit**, or a blank value.

Topics

Topics allow you to define search terms to aid in searching for registry entries from the administrator page and from the public API. You cannot enter values when you create a registry entry but can add them by entering the values in the **Details** panel. You can specify topics by entering a list of terms, separated by commas, or by checking one or more topics from the drop-down menu. The drop-down menu lists topics that have already been defined for registry entries.

Schema

The Schema describes the structure of the service messages. Schema contains subfields that describe or contain the structure definition. You cannot specify a schema when you create a registry entry but can add them by entering the values in the Details panel. Schema contains the following subfields:

- **Type**—Specifies the name of the schema definition mechanism used to describe the schema. You can select WSDL, XSD, HL7, SEF, X12, AST, or EDF, or you can enter the name of any other schema definition mechanism.
- **Reference**—Specifies the URL of the schema definition.
- **Notes**—Provides supplementary information about the schema.
- **Thumbnail**—Provides a brief text excerpt from the schema.
- **Content**—Provides the full text content of the schema definition.

Actions

Actions specify the SOAP actions or the REST HTTP request methods that can be used with the specified URL. SOAP actions provide a summary of the information defined in the WSDL. You cannot add an action when you create a registry entry, but you can add actions by Selecting the Actions plus sign in the Details panel. An action contains the following subfields:

- **Name**—Specifies the name used by the Registry to identify this action.
- **Reference**—Identifier for the action or web method.
- **Verb**—Specifies the HTTP request method. Typically, this is GET, PUT, POST, or DELETE, but you can enter any HTTP request method.
- **Read Only**—If checked, specifies that the call does not make any changes to the state of the server.
- **Idempotent**—If checked, specifies that making multiple identical calls have the same impact on the server as making a single call.
- **Description**—Specifies a text explanation of the action.
- **Request Schema**—Specifies the format of the incoming message body and consists of the following subfields: Type, Reference, Notes, Thumbnail, and Content.
- **Response Schema**—Specifies the format of the response message body and consists of the following subfields: Type, Reference, Notes, Thumbnail, and Content.

Attributes

Attributes specify a list of name-value pairs that allow you to specify any arbitrary field in the registry entry. You cannot add an attribute when you create a registry entry, but you can add attributes by Selecting the Attributes + (plus sign) in the Details panel. Each attribute consists of a name and a string value. Once you have defined an attribute, you can update the value or delete the attribute, but cannot change the name of the attribute.

Contacts

Contacts specify people or organizations that support or use the service. You cannot add a contact when you create a registry entry, but you can add contacts by Selecting the Contacts plus sign on the Details panel. A contact contains the following subfields:

- **Identity**—Specifies the name or other identity of the person or organization.
- **Contact Type**—Specifies whether the contact is the service **Author**, **Consumer**, **Provider**, **Operator**, **Manager**, or **Sponsor**.
- **Business Partner**—Provides a link to the Ensemble business partner object, which provides contact information such as address and phone numbers. Note that the public API only provides the business partner name and does not include any of the contact information from the business partner object.
- **Details**—Specifies details about the contact, for example you could enter phone numbers, email addresses, or other contact information that can be accessible through the public API.
- **Notes**—Specifies supplementary information about the contact.

Once you have defined a contact, you can update any of the subfields except for the **Identity**. You can also delete the contact by selecting the red X.

Files

Files provide a way to store any text or binary file in a registry entry. For example, you can store files that contain documentation on the service or large schema definitions. You cannot add a file when you create a service, but you can add a file by Selecting the Files plus sign on the **Details** panel. You can add a file either located on your local system or located on the Ensemble server. A File contains the following subfields:

- **Filename**—Provides the name of the file. The name of the file in the registry does not have to be the same as the name of the original file on your local file system or on the Ensemble server.
- **File Extension**—Provides information on the format and purpose of the file. This field is set by the last part of the file name.
- **MIMEType**—Provides information on the application used to access the file.
- **CharEncoding**—Provides the character encoding information.
- **File Size**—Provides the file size. This field is calculated from the file contents.
- **Contents**—When you create a file, this subfield is set to the contents of the file.

3.4 Internal Fields in the Public Service Registry

The following are internal attributes, which are only present on the public Service Registry. These internal attributes are only accessible to administrators who are using the management portal. These attributes are not accessible through the public API to the service registry.

Public

This checkbox controls whether the information about this service is available through the Registry's public API. If **Public** is true (the box is checked) then the API returns information about this service if the user is authorized to see it based on the **Required Roles** setting. If **Public** is false, no information about this service is returned by the public API.

Required Roles

Required Roles specifies a list of roles that provide public API access to the registry entry. In order for a user to retrieve information about a registry entry, the user must have one of the roles listed in **Required Roles**. The user may have this role by either being logged into Ensemble or by using a web application with this role. See "Defining Roles and Users for the Public Service Registry" for details on how to define these roles.

Instance

Instance identifies the Ensemble instance that is providing the service at the specified endpoint. Typically, this is the Ensemble instance used for the ESB and service registries. In cases where the endpoint service is not provided by an Ensemble business service, leave this field blank.

Namespace

Namespace identifies the Ensemble namespace that is providing the service at the specified endpoint. Typically, this is the Ensemble namespace used for the ESB and service registries. In cases where the endpoint service is not provided by an Ensemble business service, leave this field blank.

Production

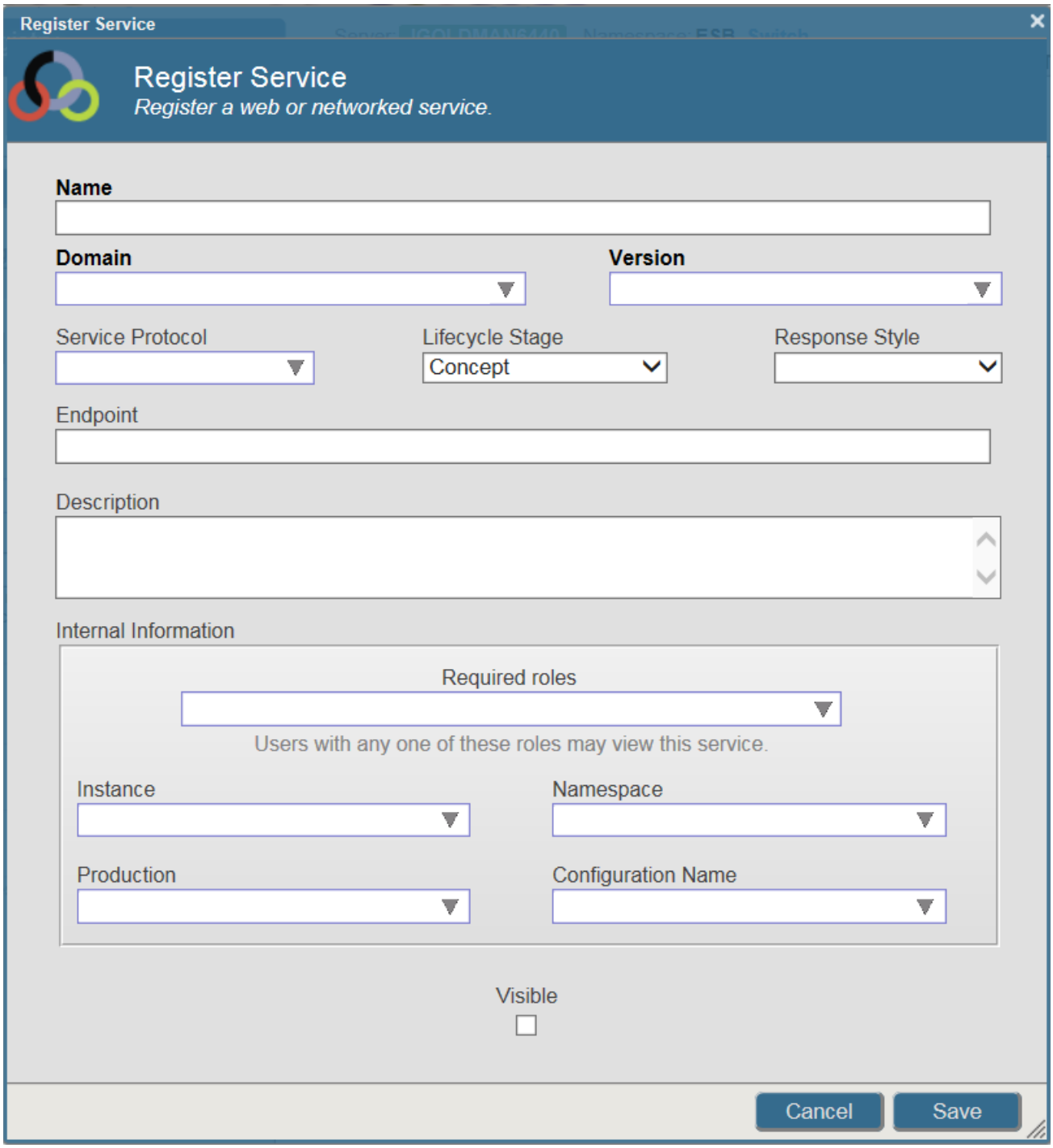
Production identifies the Ensemble production that is providing the service at the specified endpoint. Typically, this is the ESB Ensemble production. In cases where the endpoint service is not provided by an Ensemble business service, leave this field blank.

ConfigName

ConfigName identifies the Ensemble configuration item that is providing the service at the specified endpoint. Typically, this is configuration item is a business service in the ESB Ensemble production. In cases where the endpoint service is not provided by an Ensemble business service, leave this field blank.

3.5 Creating and Maintaining a Service Registry Entry

To create a new Public Service Registry or External Service Registry entry, select **New Service**. If you create a new service entry in the Public Service Registry, Ensemble displays the following Register Service form:



If you create a new service entry in the External Service Registry, Ensemble displays a similar form that omits the **Internal Information** fields and the **Visible field**.

When you are creating a new service entry, you must enter values in the **Name**, **Domain**, and **Version** required fields before selecting the **Save** button. You cannot change the values in these three fields after saving the registry entry. You can create a copy of the registry entry with new values in these fields by selecting the registry entry and then selecting **Save As** in the **Actions** panel.

You can enter values in the Register Service form for the other properties:

- **Service Protocol**

- **Lifecycle State**
- **Response Style**
- **Endpoint**
- **Description**

When creating a Public Service Registry entry, you can also enter values for the Internal Information fields:

- **Required roles**
- **Instance**
- **Namespace**
- **Production**
- **Configuration Name**
- **Visible**

Once you have saved a registry entry, you can select the registry entry from the list and use the **Details** pane to modify any of these fields except for the three required fields. You can also enter or modify the following fields on the **Details** panel:

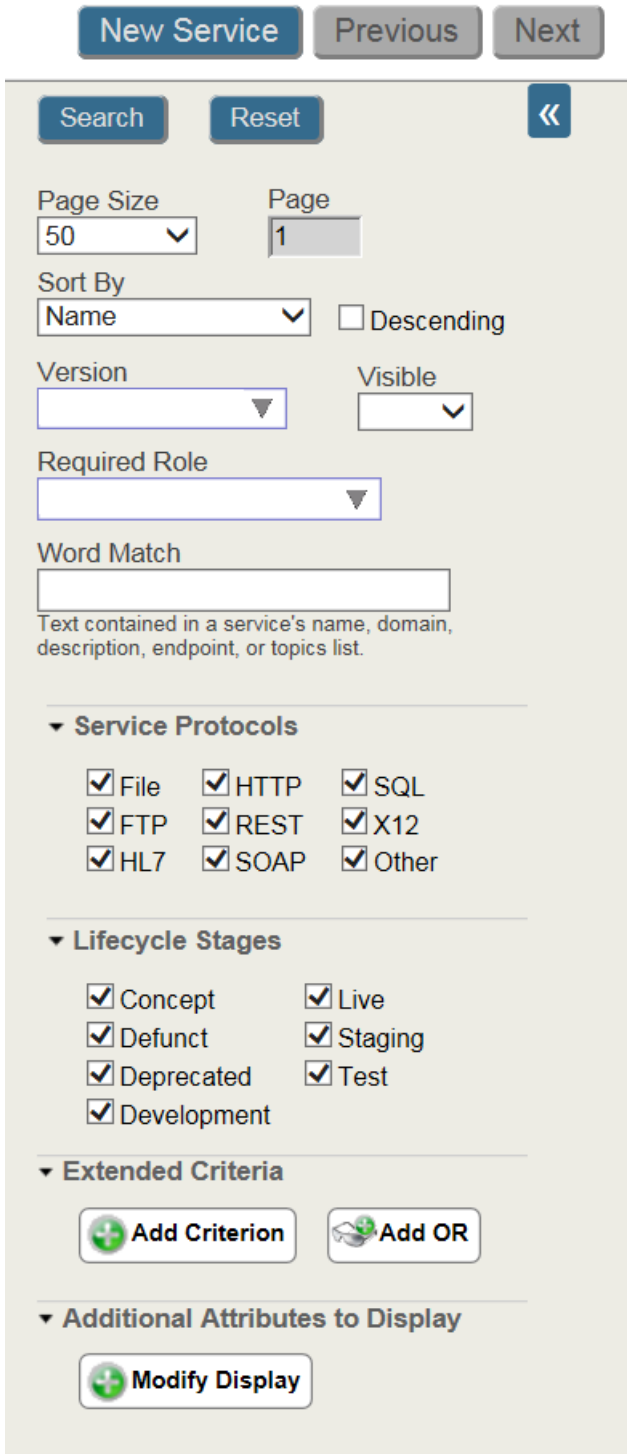
- **Topics**
- **Schema**
- **Actions**
- **Attributes**
- **Contacts**
- **Files**

The **Actions**, **Attributes**, **Contacts**. and **Files** fields take multiple values. To add a new item, select the Add Action, Add Attribute, Add Contact, or Add File plus sign. Fill in the form and then select **Apply**. You can update an existing item by selecting the clipboard for that item. You can delete an item by selecting the red X for that item.

The **Actions** panel allows you to delete a registry entry or save a copy with changes to one or more of the required fields: **Name**, **Domain**, and **Version**.

3.6 Searching and Viewing the Service Registry

The **Next** and **Previous** buttons and the search panel allow you to view the registry entries. If there are more registry entries than will fit on a single page, the **Next** and **Previous** buttons allow you to scroll through the entries. The search panel allows you to find the registry entries that match your query.



The search panel allows you to do the following:

- Specify the **Page Size**, which specifies the maximum number of registry entries that can be displayed in a page.
- Sort the registry entries by Name, Domain, Version, Service Protocol, Lifecycle Stage, and Last Time Modified. You can sort in ascending order (the default) or descending order.
- You can select the registry entries to display by specifying any of the following criteria:
  - Specified version

  - Visible or invisible entries
  - Specified Required Role
  - Text in **Word Match** in any of the following fields: Name, Domain, Description, Endpoint, or Topics
  - Any of the selected service protocols
  - Any of the selected lifecycle stages
  - Extended criteria that specify conditions based on attribute values

- You can also display the values of specific attributes in the list of services by using the **Modify Display** field.

4

Configuring an ESB

This chapter describes how to configure an Ensemble system as an ESB. This chapter contains the following sections:

4.1 Overview of Configuring Ensemble as an ESB

This section provides an overview of the tasks required to configure Ensemble as an ESB. Details of each task are provided in other sections.

- Create an Ensemble namespace to contain the ESB production and the Service Registry and configure the CSP gateway. See "Configuring an Ensemble System and Creating a Namespace" for details.
- Create External Service Registry entries that define the endpoints for external services used by the ESB. See "Administering the Public Service and External Service Registries" for details.
- Create the ESB production, add and configure the business services and business operations that provide the services, and start the production. See "Configuring ESB Services and Operations" for details on configuring services and operations.
- Create the web applications needed to make the business services available to clients. See "Configuring a Web Application" for details.
- Create the roles and users needed to access the Public Service Registry through the Ensemble portal and through the public REST API. See "Defining Roles and Users for the Public Service Registry" for details.
- Create the web application to enable the REST API for the Public Service Registry. See "Configuring a Web Application for the Public Service Registry API" for details.
- Create Public Service Registry entries that describe the services accessible through the ESB. See "Administering the Public Service and External Service Registries" for details.

4.2 Defining Roles and Users for the Public Service Registry

You can configure an ESB so that anyone can access the information in the registry or you can limit access by individual registry entries to specific accounts.

To allow open access to some or all entries in the Public Service Registry, do the following:

1. Create a role by doing the following:
   a. Select **System Administration** > **Security** > **Roles** to display the Roles portal page.
   b. Select the **Create New Role** button and name the role, for example, ServReg_Unauthenticated, and Select the **Save** button.
2. Edit or create a web application for the Public Service Registry by doing the following:
   a. Select the **Allowed Authentication Methods Unauthenticated** check box.
   b. Include the role that you defined in the previous step in the Application Roles.

   See "Configuring a Web Application for the Public Service Registry API" for details including other roles that you should include.
3. For each entry in the Public Service Registry that you want to be available to unauthenticated users, add the role that was defined in Step 1 to the **Required Roles** field, which is a comma-separated list in **Internal Information**.

To allowed unauthenticated access, you only need to define the role—you do not need to define a user.

To limit access to entries in the Public Service Registry to a one or more users, do the following:

1. Create a role by doing the following:
   a. Select **System Administration** > **Security** > **Roles** to display the Roles portal page.
   b. Select the **Create New Role** button and name the role, for example, ServReg_IDServices, and Select the **Save** button.
2. Edit or create a web application for the Public Service Registry specify the following: and select
   a. Select the **Allowed Authentication Methods Password** check box.
   b. Do not include the role that you defined in the previous step in the Application Roles.

   See "Configuring a Web Application for the Public Service Registry API" for details including other roles that you *should* include.
3. For each entry in the Public Service Registry that you want to be available to users with this role, add the role to the **Allowed Roles** field, which is a comma-separated list.
4. Create or edit a user account and assign the role to the user as follows:
   a. Select **System Administration** > **Security** > **Users** to display the Users portal page.
   b. Select the **Create New User** button, name the user, fill in the relevant fields, such as password, and Select the **Save** button or select an existing user to edit.
   c. On the **Roles** tab, select the role or roles you created, select the right-arrow button, and select the **Assign** button.

To access the Public Service Registry REST API you do not have to assign any additional roles to the user. You may need to assign additional roles to provide access to the service itself.

Note: Restricting or permitting access to a registry entry is entirely independent of restricting or permitting access to the service described by the registry entry. You can control access to the service itself by controlling access to the web application that invokes the service or by using custom code in the service.

4.3 Configuring a Web Application for the Public Service Registry REST API

This section describes how to configure a web application for the Public Service Registry REST API.

1. Define a web application that will handle calls to the Ensemble CSP port. The web application name defines the root of the URL that will call the service. A single web application can support multiple business services but they must all have a class that is the same or a subclass of the web application dispatch class.
   a. Select **System Administration** > **Security** > **Applications** > **Web Applications** to display the **Web Applications** portal page. Select the **Create New Web Application** button.
   b. Name the web application, such as `/services`. You must start the name with a / (slash) character. All alphabetic characters in the name should be in lower case.
   c. Set the **Namespace** to the namespace that the service registry is running in, such as SERVICESNS. Leave the Namespace Default Application unchecked.
   d. You can check the Application, CSP/ZEN, and Inbound Web Services checkboxes.
   e. Leave the Resource Required and Group By ID fields empty.
   f. If you want the registry to be accessed by users who log in, select the **Allowed Authentication Methods Password** check box. If you want the registry to be accessed by users without logging on, select the **Unauthenticated** check box.
   g. Set the Dispatch Class to the Ens.ServiceRegistry.API.REST.Query component class.
   h. Select Save.
   i. Select the **Application Roles** tab.
   j. In the **Available** box, select the following roles:
      - %EnsRole_RegistrySelect
      - Roles defined for the namespace's globals and routines databases, such as %DB_SERVICESNSG and %DB_SERVICESNSR.
      - Any role defined for the Public Service Registry that you want to be available to all users regardless of the account they use to log in. If you are configuring the web application to allow unauthenticated access, you must specify at least one of these roles.

      You can select multiple roles while holding the Ctrl key.
   k. After the roles are highlighted, select the right-arrow button to move them to the **Selected** text box.
   l. Then Select the **Assign** button.

Note: If your web application allows both unauthenticated and authenticated service, then it will prompt for a password, but if you do not enter a password, the web applications grants you unauthenticated access.

4.4 Using the External Service Registry to Configure ESB Hosts

If the ESB production and the External Service Registry are in the same namespace, you can use the registry to configure ESB hosts. This allows you to centralize the information about external services in the registry. In most cases, you can use this to set the properties of the business operations that are accessing the external services, but you can also use it for business services, such as services based on the FTP incoming adapter that retrieve the incoming message from an FTP server.

You can use the External Service Registry to set the properties for hosts that implement any of the following adapters:

- EnsLib.FTP.InboundAdapter
- EnsLib.FTP.OutboundAdapter
- EnsLib.File.InboundAdapter
- EnsLib.File.OutboundAdapter
- EnsLib.HTTP.OutboundAdapter
- EnsLib.SOAP.OutboundAdapter
- EnsLib.SQL.InboundAdapter
- EnsLib.SQL.OutboundAdapter
- EnsLib.TCP.OutboundAdapter

To configure a host to get its settings from the External Service Registry, set the **External Registry ID** field to the key value of the registry entry. The key value consists of the service Name, Domain, and Version concatenated together with || (two vertical bars) separating the elements. For example, to get the property values from the External Service Registry entry with the name GetDaily, the domain Weather, and Version 1.0, set the **External Registry ID** property to the following:

```
GetDaily||Weather||1.0
```

The following describes how the Endpoint value is used to set the adapter properties for each protocol:

- HTTP and REST—The Endpoint sets the HTTPServer, HTTPPort, and URL properties. For example if the Endpoint is:

  ```
  http://newsrvcs.example.com:57781/csp/samples/docserver/namespaces
  ```

  Then the business operation properties are set as follows:

  HTTPServer: newsrvcs.example.com

  HTTPPort: 57781

  URL: /csp/samples/docserver/namespaces

  If the business operation is a pass-through business operation, then the special characters * (asterisk), ^ (circumflex), and | (vertical bar) describe how to build the endpoint from the incoming URL. See “Configuring Pass-through Business Operations” for details.

- FTP—The Endpoint sets the FTPServer, FTPPort, FilePath, and SSLConfig properties. For example, if the Endpoint is:

  ```
  ftp://newftp.example.com/filestore/docs/
  ```

  Then the business operation properties are set as follows:

  FTPServer: newftp.example.com

  FTPPort: 21

  FilePath: /filestore/docs/

  If the protocol is sftp://, then SSLConfig is set to "!SFTP" and the FTPPort is set to 22.

- SQL—The Endpoint is used to set the DSN property.
- SOAP—The Endpoint is used to set the WebServiceURL property. The EnsLib.SOAP.GenericOperation and EnsLib.SOAP.GenericOperationInProc pass-through operations are considered HTTP operations, not SOAP operations for this purpose.
- File—The Endpoint is used to set the FilePath property.
- TCP—The Endpoint is used to set the IPAddress and Port properties. For example, if the Endpoint is:

  ```
  102.23.10.1:4500
  ```

  Then the business operation properties are set as follows:

  IPAddress: 102.23.10.1

  Port: 4500

- HL7 and X12—The beginning of the Endpoint string specifies the underlying protocol: File, TCP, FTP, or HTTP. For File and TCP, the file: and tcp: are removed from the string Endpoint before parsing it.

The business operation settings from the External Service Registry entry override any settings from System Default Settings, production definition, and business host class definition.

5

Configuring ESB Services and Operations

This chapter describes how to configure the services and operations in the ESB production and how to use the services provided by them. This chapter contains the following sections:

5.1 Configuring Pass-through Business Services

To add a pass-through business service, Select the + sign for adding a new service in the Production Configuration page, then:

1. Select the class based on the protocol used and whether SAML security is being used. Choose a class from the following:
   - EnsLib.HTTP.GenericService
   - EnsLib.REST.GenericService
   - EnsLib.REST.SAMLGenericService
   - EnsLib.SOAP.GenericService
   - EnsLib.SOAP.SAMLGenericService
2. Decide if the pass-through service is to be called through the CSP port or a special port. For live productions, you should use the CSP port in conjunction with a robust web server software so that you have a secure, fully configurable system that can handle heavy loads. The web server installed with Ensemble is a limited system intended for use on development systems, but not on fully-loaded live systems. The special port is a light-weight listener that provides minimal configuration and security. Although it is possible to configure a service to accept calls on both ports, we do not recommend this configuration.

   If your pass-through service is to be called on the CSP port:

   - Leave the **Port** field blank.

- Select the **Enable Standard Requests** check box in the **Connection Settings** field on the Settings tab.
- On the **Additional Settings** section, set the **Pool Size** to 0. This suppresses the pass-through service from listening on the special port. If you omit this step and leave the **Port** field blank, Ensemble displays an error message.

If your pass-through service is to be called from a special port:

- Specify a port number.
- Clear the **Enable Standard Requests** check box in the **Connection Settings** field on the Settings tab.

3. Set the **Target** to point to the pass-through operation.

To optimize performance for pass-through services that use the CSP port, you can configure the pass-through business service to keep the connection open between calls. You can do this by checking the **Keep Standard Request Partition** checkbox.

See Using SAML Validation in Pass-through Services for information on using SAML validation on SOAP and REST pass-through services.

To use the CSP port to access the pass-through business service, you also need to define a web application. See "Configuring a Web Application" for details.

5.2 Configuring Pass-through Business Operations

To add a pass-through business operation, Select the + sign for adding a new operation in the Production Configuration page, then:

1. Select the class based on the protocol used and whether you want to suppress storing the messages in the database. See Suppressing Persistent Messages in Pass-through Services and Operations for more information. Choose a class from the following:
   - EnsLib.HTTP.GenericOperation
   - EnsLib.HTTP.GenericOperationInProc—suppresses storing messages.
   - EnsLib.REST.GenericOperation
   - EnsLib.REST.GenericOperationInProc—suppresses storing messages.
   - EnsLib.SOAP.GenericOperation

2. Configure the pass-through operation **HTTP Server**, **HTTP Port**, and **URL** settings. You can either configure these settings directly or use the external service registry to configure them. To set them via the external service registry, set the value of the **External Registry ID** property. See "Using the External Service Registry to Configure ESB Hosts" for details on setting **External Registry ID**. After you apply the **External Registry ID** setting, Ensemble reads the current values from the registry and uses them to set the other properties and marks them as read-only.

   You can either explicitly set the URL or set it to be derived from the incoming URL sent to the generic service. You can do this either through the service registry or directly through the URL property. To derive it from the incoming URL, set either the URL segment of the external service registry **Endpoint** field or the **URL** property as follows:

   - empty string: Use the URL from the GenericMessage, which is typically the URL passed into the generic service. Typically, you use this if the pass-through service is called from a special port and the URL does not contain the web application name and the service component name.
   - | (vertical bar): Remove the web application name and configuration name from the URL value in the GenericMessage and use the remainder as the URL. Typically, you use this when the GenericService is called using the standard

CSP port. The web application name and the configuration name of the business service are needed to route the call to the GenericService but are not needed by the external server.

The URL is compared to web application name and the configuration name with a case-sensitive compare. All alphabetic characters in the web application name must be in lower case and the segment of the URL corresponding to the configuration name must match the case in the configuration name as defined in the production.

- ^ (circumflex): Remove the web application name from the URL value in the GenericMessage and use the remainder as the URL. Typically, this is only used if the GenericService is called using the standard CSP port and the pass-through operation component has a name that is identical to the first part of the URL expected by the external server. All alphabetic characters in the web application name in the URL must be in lower case.

If the **URL** property specifies a string that is not empty and does not contain either a vertical bar or circumflex, then the incoming URL is not used to generate the outgoing URL.

3. Although the pass-through operations typically do not change the contents of the pass-through message, you can specify that the EnsLib.HTTP.GenericOperation and EnsLib.REST.GenericOperation operations perform character set translation. For example, you could have the operation perform character translation so that accented characters are displayed correctly in a web browser. To set the HTTP or REST generic operation to perform character set translation, clear the **Read Raw Mode** check box in **Additional Settings**.

5.3 Using SAML Validation in Pass-through Services

Pass-through services validates the SAML token but does not check it to see if it provides access to the resources on the external server. The external server must check if the SAML token permits the requested access. The **Validation** field controls the level of SAML token validation done by the pass-through business service. The following flags specify the kind of validation that is done:

- **t**—Must contain an Authorization header SAML token with key 'access_token='
- **a**—Token must contain an Assertion.
- **r**—Requires Assertions to contain NotBefore/NotOnOrAfter time conditions
- **v**—Verifies Assertion signatures using a Trusted X.509 certificate and, if present, NotBefore/NotOnOrAfter conditions.
- **o**—Validates other signed nodes such as TimeStamp.

By default, validation has a value of 1, which is equivalent to specifying tarvo. If you specify a value of 0, the pass-through service performs no validation.

When checking the NotBefore and NotOnOrAfter time conditions, the default clock skew allowance is 90 seconds. To change the skew allowance, set the global `^Ens.Config("SAML","ClockSkew")`. To set the default clock skew for all components to 180 seconds, enter the following command:

```
Set ^Ens.Config("SAML","ClockSkew")=180
```

To change the skew allowance for a specific component to 180 seconds, enter the following command:

```
Set ^Ens.Config("SAML","ClockSkew",component-name)=180
```

5.4 Suppressing Persistent Messages in Pass-through Services and Operations

To obtain maximum efficiency using pass-through services and operations, you can suppress the use of persistent messages. The pass-through service sends the call directly to the pass-through operation without creating a persistent message. The operation sends the reply message to the service in the same way. This allows you to achieve high throughput levels and eliminates the need to purge messages, but has the following limitations:

- No persisted record of the pass-through call is maintained. You cannot view the message in the message viewer or view a message trace. This makes it challenging to troubleshoot problems.
- No retry mechanism is available. If the first attempt to contact the server fails, the failure is returned to the application calling the pass-through service.
- This mode is available only if a pass-through service targets a pass-through operation directly. If the message passes through any other production component, such as a business process router, then persisted messages are used throughout the process.

To suppress the use of persistent messages for a pass-through service and pass-through operation pair, choose one of the specialized classes for the pass-through operation:

- EnsLib.HTTP.GenericOperationInProc
- EnsLib.REST.GenericOperationInProc
- EnsLib.SOAP.GenericOperationInProc

In addition, to suppress persistent messages, you must clear the **Persist Messages Sent InProc** checkbox in the pass-through business service configuration.

If the pass-through service is using the standard CSP port, you can further improve efficiency by configuring the service to keep the TCP connection open between calls. To do this, in the pass-through service configuration, check the **Keep Standard Request Partition** checkbox.

5.5 Using Other Business Services, Processes, and Operations

Although the simplest ESB systems can consist of a production with only pass-through services and operations, some requirements can only be met with more complex production components. For example, if your ESB must do any of the following, you need other kinds of business services and operations:

- Route calls from a single incoming service to multiple external services depending on the content of the call.
- Modify the parameters or protocol used for a service.
- Expose a single service that is implemented by combining the capabilities of two or more external services.
- Provide the service directly on the ESB.
- Provide security other than one based on SAML tokens.

For more information on using other business services, processes, and operations to handle REST and SOAP service requests, see *Creating Web Services and Web Clients with Ensemble*.

5.6 Tracking Performance Statistics of Pass-through Services and Operations

To ensure that your ESB system continues to meet the needs of your users, it is important to monitor and track its performance. By performing this monitoring regularly, you can manage any increase in workload by adding resources to handle the increased load. If your pass-through services and operations are using persistent messages, you can use Ensemble's monitoring facilities to track and report on performance, but the Activity Volume monitoring provides a mechanism to produce summary performance statistics that are useful for the following:

- Tracking overall system performance.
- Ensuring that the system is meeting Service-Level Agreements (SLAs).
- Identifying potential problems.

If you have suppressed storing persistent messages (see "Suppressing Persistent Messages in Pass-through Services and Operations"), these summary statistics are your main tool for tracking performance.

The summary statistics consists of the following information for each pass-through service and operation:

- Total number of messages that completed during a fixed time interval (10 seconds)
- Total elapsed time to complete processing these messages

As generally true for monitoring performance, it is important to monitor and record performance statistics on a regular basis. This allows you to detect trends and compare current performance to a baseline. This can be useful in determining whether the cause of performance problems is increased load, network issues, problems with the servers providing the services, or ESB performance issues.

The summary statistics mechanism for pass-through services and operations consists of the following components:

- Configurations and Global settings to enable pass-through services and operations to generate the statistics.
- Daemons to get the statistics from the pass-through services and operations and send them to a daemon to store the statistics.
- User interface to monitor and explore the statistics.

For details, see "Monitoring Activity Volume" in *Monitoring Ensemble*.

A

Configuring an Ensemble System and Creating a Namespace

This appendix provides information on configuring Ensemble and Caché so that you can use it as an ESB or with pass-through REST and SOAP services and operations. It contains the following sections:

This appendix briefly discusses how to configure your system so that you can use HTTP and SOAP services through the Ensemble CSP port. This information is intended to help you set up a development or test system for these services. Complete information about these topics is provided in the Caché documentation. See "Configuring Caché" in the Caché System Administration Guide for more details.

To set up an Ensemble development or test system for HTTP or SOAP services, follow these steps:

1. If you have installed Ensemble in a locked down installation, Studio access is disabled. Open the Management Portal and enable Studio access:
   a. Start the Management Portal from the Ensemble cube. You will have to use your Windows login username rather than _system to access the portal. Enter the password that you specified during installation.
   b. Select **System Administration** > **Security** > **Services** to get to the Services portal page.
   c. The `%Services_Bindings` service is disabled by default. Select the service name and check the **Service Enabled** checkbox and save the setting.
2. If you are not using an existing Ensemble namespace, create a new namespace:
   a. Select **System Administration** > **Configuration** > **System Configuration** > **Namespaces** to get to the Namespaces portal page.
   b. Select the **Create New Namespace** button, specify a name for the namespace, such as SERVICESNS.
   c. Select the **Create New Database** button for the globals database.
   d. In the Database Wizard, enter a name for the globals database, such as SERVICES_GDB. The wizard uses the name to create a directory for the database.
   e. Select the **Next** button twice to get to the Database Resource form. Select the **Create a new resource** radio button. The wizard displays a Create New Resource form. Accept the suggested name, such as %DB_SERVICES_GDB and ensure that Public Permissions Read and Write checkboxes are *not* checked. Select the **Save** button on the Database Resource form and the **Finish** button on the Database Wizard form.
   f. Repeat steps c through e for the routines database.
   g. Select the **Save** button to complete creating the namespace.

h. Select **Close** to close the log.

This completes the system configuration.

B

Configuring a Web Application for a Pass-through Business Service

This appendix describes how to configure a web application for a pass-through business service. To configure a web application for the Public Service Registry REST API, see "Configuring a Web Application for the Public Service Registry API".

1. Create an empty role and assign it to the unknown user as follows:
   a. Select **System Administration** > **Security** > **Roles** to display the Roles portal page.
   b. Select the **Create New Role** button and name the role, for example, Services_Role, and Select the **Save** button.
   c. Select the **Members** tab, select the Unknown User, Select the right arrow, and Select the **Assign** button.
2. Define a web application that will handle calls to the Ensemble CSP port. The web application name defines the root of the URL that will call the service. A single web application can support multiple business services but they must all have a class that is the same or a subclass of the web application dispatch class.
   a. Select **System Administration** > **Security** > **Applications** > **Web Applications** to display the **Web Applications** portal page. Select the **Create New Web Application** button.
   b. Name the web application, such as `/restpassthrough` or `/soappassthrough`. You must start the name with a / (slash) character. All alphabetic characters in the name should be in lower case.
   c. Set the **Namespace** to the namespace that the production is running in, such as SERVICESNS or ENSDEMO. Leave the Namespace Default Application unchecked.
   d. You can check the Application, CSP/ZEN, and Inbound Web Services checkboxes.
   e. Leave the Resource Required and Group By ID fields empty.
   f. Check the Unauthenticated checkbox on the **Allowed Authentication Methods** line.
   g. Set the Dispatch Class to the component class, such as EnsLib.REST.GenericService or EnsLib.SOAP.GenericService.
   h. Select Save.
   i. Select the Matching Roles tab.
   j. In the **Select a Matching Role:** field, select the role that you created in the previous step.
   k. In the **Select target roles to add to the selected matching role** field, select the role or roles associated with the namespace globals and routines. The globals and routines may be in the same database or in separate databases. If your service, accesses another Caché database, you should also select its role. For example, if you are defining

a web application for the Demo.REST.DirectoryService class in ENSDEMO, you must also select the %DB_SAMPLES role. You can select multiple roles while holding the Ctrl key.

Note: The globals database also may have a secondary database and a corresponding role, such as %DB_GDBSECONDARY. This secondary database is used to store passwords. You don't need access to this database for pass-through services and operations, but if you create a custom web service that uses password access, you should also add the secondary database role to the target database.

l. After the roles are highlighted, select the right-arrow button to move them to the **Selected** text box.

m. Then select the **Assign** button.

Note: If your web application allows both unauthenticated and authenticated service, then it will prompt for a password, but if you do not enter a password, the web applications grants you unauthenticated access.

C

Pass-through Service and Operation Walkthrough

This walkthrough demonstrates how to use pass-through services and operations.

Note: This walkthrough makes use of some free services provided by the openweathermap.org server and the www.webservicex.net server. Neither of these servers is associated with InterSystems Corporation and these servers may not always be available. If they are not available, you can substitute any other REST or SOAP server available to you. The openweathermap.org server supplies free weather data. If you intend to use it in an application, you should get a application ID. Although the service is free, product support is available for the weather server at a fee. See http://openweathermap.com/ for more information about this server. The www.webservicex.net server is provided by Cloud Computing Technologies Ltd. For more information on the server, contact them at Info@WebserviceX.NET.

This section contains:

C.1 Using REST Pass-through Services and Operations

This section adds a REST pass-through service and a REST pass-through operation. The service listens on the CSP port and sends the HTTP REST message to the pass-through operation. The pass-through operation sends the HTTP REST message to an external server, api.openweathermap.org, that returns current weather information.

Select or create a namespace and production. In order to use the CSP port, you must define a web application in the namespace and a role. See "Configuring an Ensemble System" for step-by-step instructions to create a namespace, role, and web application. For this walkthrough, name the web application `/restpassthrough` (web application name should be lower case) and set the web application **Dispatch Class** to EnsLib.REST.GenericService.

1. On the production configuration page, select the plus sign for adding new operations.
   a. Specify the operation class: EnsLib.REST.GenericOperation
   b. Name the operation, such as WeatherRESTPassthroughOp.
   c. Don't enable it yet.

d. Select **OK**.

2. Select the Pass-through operation that you created and select the Settings tab.
   a. If you are using the external service registry, set the **External Registry ID** to identify the registry entry that sets the **HTTP Server**,**HTTP Port**, and **URL** settings and skip to Step 3. Otherwise specify these settings directly as follows:
   b. Enter the **HTTP Server**: api.openweathermap.org
   c. Set the **URL** field to: |
   d. Leave the other fields at their default values.
   e. Check the Enabled checkbox.
   f. Select **Apply**.
3. Select the plus sign for adding new services.
   a. Specify the service class: EnsLib.REST.GenericService
   b. Name the service, such as WeatherRESTPassthroughServ.
   c. Don't enable it yet.
   d. Select **OK**.
4. Select the Pass-through service that you created and select the Settings tab.
   a. Ensure that the **Port** number is blank.
   b. In the **TargetConfigName** field, choose the pass-through operation that you added in the previous step.
   c. In **Connection Settings**, check the **Enable Standard Requests** checkbox.
   d. In **Additional Settings**, set **Pool Size** to 0. This suppresses the pass-through service from listening on the special port. If do not set the pool size to 0, you must specify a port number.
   e. Check the **Enabled** checkbox.
   f. Select **Apply**.
5. Start the production.
6. Enter the URL for the pass-through service using the CSP port. The URL consist of the following parts:
   - `http://localhost:`*nnnnnn*`/` where *nnnnnn* is the default CSP port.
   - *web-application-name*`/` that you specified. See Configuring a Web Application for a Pass-through Business Service.
   - *passthrough-service-name*`/`
   - URL required by the external service, such as `data/2.5/weather?q=London,uk`.

   For example, enter the following URL in a web browser:

   ```
   http://localhost:57772/restpassthrough/WeatherRESTPassthroughServ/data/2.5/weather?q=Boston,ma
   ```

   Note that all alphabetic characters in the web application name must be in lower case and the pass-through service name must match the case of the configuration item name.

If all is working, the call returns a JSON message with the current weather in London. For example, it can return the following JSON message:

{
"coord":{"lon":-0.13,"lat":51.51},
"sys":{"message":0.0441,"country":"GB","sunrise":1399609017,"sunset":1399664214},
"weather":[{"id":802,"main":"Clouds","description":"scattered clouds","icon":"03n"}],
"base":"cmc stations",
"main":{"temp":285.32,"pressure":1008,"temp_min":284.15,"temp_max":286.48,"humidity":91},
"wind":{"speed":5.14,"gust":6.17,"deg":284},
"rain":{"3h":0},
"clouds":{"all":32},
"dt":1399600508,
"id":2643743,
"name":"London",
"cod":200
}

C.1.1 Pass-through Background Information

Notice how the URL is transformed between your call to the pass-through service and the call to the external server. The URL sent to the pass-through service is:

```
http://localhost:57772/Restpassthrough/WeatherRESTPassthroughServ/data/2.5/weather?q=Boston,ma
```

and the URL sent to the external server is:

```
http://api.openweathermap.org/data/2.5/weather?q=London,uk
```

The last parts of the URL starting with data are identical. The **HTTP Server** field sets the server in the outgoing call. The | in the URL field instructs the operation to strip the web application name and configuration name from the incoming URL and then include the remainder in the outgoind URL.

Note: The EnsLib.REST.GenericService class does not provide any additional functionality over the EnsLib.HTTP.GenericService class. Although you could use the EnsLib.HTTP.GenericService class to pass-through a REST call, you should use the appropriate subclass.

C.1.2 Pass-through Troubleshooting

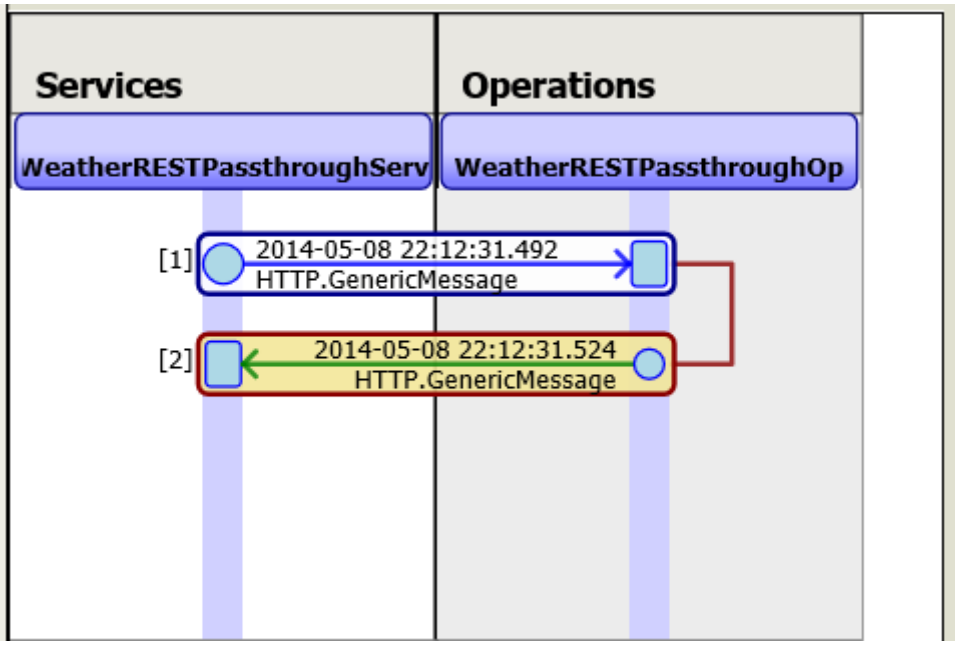
There are a few troubleshooting steps you can take if you don't get the expected results.

1. Enter the openweathermap URL directly in the web browser without using Ensemble. Enter the following URL in the web browser:

   ```
   http://api.openweathermap.org/data/2.5/weather?q=London,uk
   ```

   This ensures that the weather server is working correctly.

2. Examine the Ensemble message tracing for the production. A successful message trace looks like the following:



The trace may help you identify where the error is.

3. Run a TCP trace utility. Specify a local port to listen on, the server api.openweathermap.org, and port 80. In the pass-through operation settings, set the following properties:

   - **HTTP Server**: localhost
   - **HTTP Port**: port that TCP trace is listening on

   Then Select **Apply** and enter the pass-through service URL in a web browser:

   ```
   http://localhost:57772/Restpassthrough/WeatherRESTPassthroughServ/data/2.5/weather?q=Boston,ma
   ```

   The TCP trace should show the HTTP request being sent out by the pass-through operation and the server's response.

C.2 Using SOAP Pass-through Services and Operations

This section adds a SOAP pass-through service and a SOAP pass-through operation. The service listens on the CSP port and sends the SOAP message to the pass-through operation. The pass-through operation sends the SOAP message to an external server, www.webservicex.net, which among other services provides a currency conversion service that returns the exchange rate for two specified currencies. In order to call the SOAP passthrough services, you need to use a SOAP toolkiit, such as SOAPUI. For information on the SOAPUI toolkit, see http://www.soapui.org/.

Select or create a namespace and production. In order to use the CSP port, you must define a web application in the namespace and a role. See "Configuring an Ensemble System" for step-by-step instructions to create a namespace, role, and web application. For this walkthrough, name the web application `/soappassthrough` and set the web application **Dispatch Class** to EnsLib.SOAP.GenericService.

C.2.1 Calling the External Server from the SOAP Toolkit

Before creating the pass-through services and operations, use a SOAP toolkit to create a project and call the external server directly. Using your SOAP toolkit, do the following:

- Create a new project.
- Specify the following as the location of the WSDL file:

  ```
  http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
  ```

- Select the request generated for the project. If your toolkit generates a SOAP 1.1 and 1.2 requests, use the SOAP version 1.2 request. The toolkit displays the SOAP message that is sent to the server. For example, SOAPUI displays:

  ```
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:web="http://www.webserviceX.NET/">
    <soap:Header/>
    <soap:Body>
      <web:ConversionRate>
        <web:FromCurrency>?</web:FromCurrency>
        <web:ToCurrency>?</web:ToCurrency>
      </web:ConversionRate>
    </soap:Body>
  </soap:Envelope>
  ```

- Replace the two question marks with the abbreviations for the currencies. For example, to convert Euros to US Dollars, replace the first question mark with EUR and the second question mark with USD.
- Execute the request. Using SOAPUI, you execute the request by Selecting the green arrow. The SOAP toolkit displays the response SOAP message, such as:

  ```
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
      <ConversionRateResponse xmlns="http://www.webserviceX.NET/">
        <ConversionRateResult>1.3603</ConversionRateResult>
      </ConversionRateResponse>
    </soap:Body>
  </soap:Envelope>
  ```

C.2.2 Adding the SOAP Passthrough Service and Operation

This section adds a SOAP passthrough operation and a SOAP passthrough service that use the CSP port.

1. Go to the Production Configuration page and select the plus sign for adding new operations.
   a. Specify the operation class: EnsLib.SOAP.GenericOperation
   b. Name the operation, such as ConvertCurrencySOAPCSPOp.
   c. Don't enable it yet.
   d. Select **OK**.
2. Select the pass-through operation that you created and select the Settings tab.
   a. If you are using the external service registry, set the **External Registry ID** to identify the registry entry that sets the **HTTP Server**,**HTTP Port**, and **URL** settings and skip to Step 3. Otherwise specify these settings directly as follows:
   b. Enter the **HTTP Server**: www.webservicex.net
   c. In the URL field, enter | (vertical bar). This instructs the operation to remove the extra parts of the incoming URL that are needed by the web application but are not expected by the external service
   d. Check the Enabled checkbox.
   e. Select **Apply**.
3. Select the plus sign for adding new services.
   a. Specify the operation class: EnsLib.SOAP.GenericService

b. Name the operation, such as ConvertCurrencySOAPCSPServ.

c. Don't enable it yet.

d. Select **OK**.

4. Select the pass-through service that you created and select the Settings tab.

   a. Check the **Enabled** checkbox.

   b. Ensure that the **Port** field is blank.

   c. Set the **TargetConfigName** to the passthrough operation that you created in the previous step.

   d. In **Connection Settings**, check the **Enable Standard Requests** checkbox.

   e. In **Additional Settings**, set **Pool Size** to 0. This suppresses the passthrough service from listening on the special port.

   f. Select **Apply**.

5. Start the production.

6. In your SOAP toolkit, enter the URL for the pass-through service using the CSP port. The URL consist of the following parts:

   - `http://localhost:`*nnnnnn*`/` where *nnnnnn* is the default CSP port.
   - *web-application-name*`/` that you specified. See Configuring a Web Application for a Pass-through Business Service.
   - *passthrough-service-name*`/`
   - URL required by the external service, such as `CurrencyConvertor.asmx`.

   For example, if you specified the service name as `soappassthrough` and the passthrough business service name as `ConvertCurrencySOAPCSPServ`, enter the following URL in a web browser:

   ```
   http://localhost:57772/soappassthrough/ConvertCurrencySOAPCSPServ/CurrencyConvertor.asmx
   ```

   Note that all alphabetic characters in the web application name must be in lower case and the pass-through service name must match the case of the configuration item name.

   Then execute the request. The SOAP toolkit should return the same XML message that it previously returned.

Service Registry Public API Reference

Sample JSON Return Messages with Registry Entries

Most of the Service Registry public API calls return one or more registry entries. This section describes the format of those entries.

JSON Message with Multiple Entries


```
[
{
    "Name": "MathServiceSOAP",
    "Domain": "UnitTest",
    "Version": "1.0",
    "Stage": "Live",
    "Protocol": "SOAP",
    "Description": "Add 2 Numbers",
    "Endpoint": "https://hostname/enslatest/csp/support/Demo.SOAP.MathService.cls",
    "ResponseStyle": "Sync",
    "LastModified": "2015-03-16 19:07:47.469",
    "Topics":
    [
        "Test",
        "Maths"
    ],
    "Contacts":
    [
      {
        "Identity": "QD Developer Moon",
        "Type": "Operator",
        "Details": "Details of contact",
        "BusinessPartner": "QD",
        "Notes": "This SOAP service is designed to have minimum moving parts"
      }
    ],
    "Schema":
    {
       "Type": "Notes",
       "Ref": "https://hostname/enslatest/csp/support/Demo.SOAP.MathService.cls?wsdl=1",
       "Content": "<definitions targetNamespace='http://tempuri.org'> ...</definitions>",
       "Notes": "Some WSDL"
    },
    "Public": true,
    "Attributes":
    [
       {
          "Name": "One",
          "Value": "1"
        }
    ],
    "Files":
    [
       {
          "Filename": "SOAPMathService.WSDL",
          "FileExtention": ".WSDL",
          "MIMEType": "text/text",
          "CharEncoding": "UTF-8",
          "FileSize": "1.44 KB",
          "Contents": null
       }
    ],
    "Actions":
    [
       {
          "Name": "Sum",
          "Ref": "Sum",
          "Verb": "POST",
          "Description": "Add up 2 numbers",
          "ReadOnly": false,
          "Idempotent": true
       }
    ]
},
{
    "Name": "PublicREST",
    "Domain": "UnitTest",
    "Version": "0.9",
    "Stage": "Live",
    "Protocol": "REST",
    "Description": "REST Call for the Public Registry",
    "Endpoint": "http://hostname:57774/csp/registry/docserver/public",
```

```
    "ResponseStyle": "Sync",
    "LastModified": "2015-03-05 16:15:33.38",
    "Topics":
    [
       "Public",
       "Search",
    ],
    "Public": true,
    "Attributes":
    [
       {
          "Name": "Security",
          "Value": "Username and Password"
       }
    ],
    "Files":
    [
       {
           "Filename": "TestPlan",
           "MIMEType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
           "CharEncoding": "UTF-8",
           "FileSize": "16.95 KB",
            "Contents": null
       }
    ],
    "Actions":
    [
       {
          "Name": "public",
          "Ref": "public",
          "Verb": "GET",
          "Description": "Returns REST endpoint for public registry",
          "RequestSchema":
          {
              "Type": "Notes",
              "Notes": "This is the REST endpoint to query the public Registry"
          },
          "ResponseSchema":
          {
             "Type": "REST Information
          },
          "ReadOnly": true,
          "Idempotent": true
       }
    ]
}
]
```

Common URL Parameters

The Common URL Parameter `?format` is available in most Service Registry public API calls. It controls the line breaks and indenting in the JSON responses. This parameter is available on all Service Registry Public API calls. By default, these calls return JSON with a single space separating each elements and with no line breaks.

URL Parameters

`?format=`*format-code*

Optional parameter that formats and indents the returned JSON.

Where *format-code* contains one or more of the following codes:

- `n`—Terminate each element of the JSON response with a newline character.
- `w`—Terminate each element of the JSON response with a newline/line feed.
- `i`—Indent each level of the JSON response with four space characters or one tab. If a digit 1 through 9 is also specified, then indent by that number of spaces or tabs.
- Digits `1` through `9`—Indent each level of the JSON response with the specified number of spaces.
- `t`—Use the tab character instead of the space character for indenting.
- `u`—Convert JSON output to UTF-8 from its internal format.

Get About

Get About lists the public API REST calls that are available.

Request Method and URL

GET `/about`

URL Parameters

See Common URL Parameters.

Response

```
[
    {
        "EndPoints": [
            {
                "GetAbout": "/about"
            },
            {
                "GetServices": "/services"
            },
            {
                "GetServicesByID": "/services/id/:DATA/:DATA/:DATA"
            },
            {
                "GetFileByID": "/services/id/:DATA/:DATA/:DATA/file/:DATA"
            },
            {
                "GetServicesByWord": "/services/includesword/:DATA"
            },
            {
                "GetServicesModifiedSince": "/services/modifiedsince/:DATA"
            },
            {
                "GetServicesByProtocols": "/services/protocols/:DATA"
            },
            {
                "GetServicesByStages": "/services/stages/:DATA"
            },
            {
                "GetServicesByVersion": "/services/version/:DATA"
            }
        ]
    },
    {
        "QueryParameters": [
            {
                "format":
"1-9 = indent with this number of spaces (4 is the default with the 'i' format specifier);
i - indent with 4 spaces unless 't' or 1-9;n - newline (lf);
t - indent with tab character;
u - output pre-converted to UTF-8 instead of in native internal format;
w - Windows-style cr/lf newline"
            }
        ]
    }
]
```

Sample Call

```
https://esb.example.com/registry/about
```

Get File By ID

Get File By ID returns the specified file.

Request Method and URL

GET `/services/id/`*name*`/`*domain*`/`*version*`/file/`*filename*

URL Parameters

None.

Responses

File contents.

Sample Call

```
https://esb.example.com/registry/services/id/weather/util/1.0/file/brief.txt
```

Get Services

Get Services shows all services accessible to the user.

Request Method and URL

GET `/services`

URL Parameters

See “Common URL Parameters”.

Responses

See “Sample JSON Return Messages with Registry Entries”.

Sample Call

https://esb.example.com/registry/services

Get Services By ID

Get Services By ID shows all services accessible to the user that match the specified name, domain, and version.

Request Method and URL

GET `/services/id/`*`name/domain/version`*

URL Parameters

See "Common URL Parameters".

Responses

See "Sample JSON Return Messages with Registry Entries".

Sample Call

https://esb.example.com/registry/services/id/weather/util/1.0

Get Services By Protocols

Get Services By Protocol shows all services accessible to the user that match a protocol in the list.

Request Method and URL

GET `/services/protocols/`*protocol-list*

where *protocol-list* is a list of protocols, separated by commas.

URL Parameters

See "Common URL Parameters".

Responses

See "Sample JSON Return Messages with Registry Entries".

Sample Call

https://esb.example.com/registry/services/protocols/REST,SOAP

Notes

Protocols can be any of the following: **File**, **FTP**, **HL7**, **HTTP**, **REST**, **SOAP**, **SQL**, **X12**, and any custom value.

Get Services By Stages

Get Services By Stages shows all services accessible to the user that match a stage in the list.

Request Method and URL

GET `/services/stages/`*stage-list*

where *stages-list* is a list of stages, separated by commas.

URL Parameters

See "Common URL Parameters".

Responses

See "Sample JSON Return Messages with Registry Entries".

Sample Call

https://esb.example.com/registry/services/stages/Development,Live

Notes

Stages can be any of the following: **Concept**, **Development**, **Test**, **Staging**, **Live**, **Deprecated**, and **Defunct**.

Get Services By Version

Get Services By Version shows all services accessible to the user that match the specified version.

Request Method and URL

GET `/services/version/`*version*

where *version* is a version number.

URL Parameters

See "Common URL Parameters".

Responses

See "Sample JSON Return Messages with Registry Entries".

Sample Call

https://esb.example.com/registry/services/version/1.0

Get Services By Word

Get Services By Word shows all services accessible to the user that match the specified search text.

Request Method and URL

GET `/services/includesword/`*search-text*

URL Parameters

See "Common URL Parameters".

Responses

See "Sample JSON Return Messages with Registry Entries".

Sample Call

https://esb.example.com/registry/services/includesword/patient

Notes

The following fields are searched for the specified text:

- Name
- Domain
- Description
- Endpoint
- Topics

Get Services Modified Since

Get Services Modified Since shows all services accessible to the user whose registry entry has been created or updated since the specified date-time.

Request Method and URL

GET `/services/modifiedsince/`*date-time*

URL Parameters

See “Common URL Parameters”.

Responses

See “Sample JSON Return Messages with Registry Entries”.

Sample Call

https://esb.example.com/registry/services/modifiedsince/2015-02-1%2011:30