



Ostia Portus

Reference

Version 2012-12-17

December 2012

This document applies to Ostia Portus 2012-12-17 15:48:15 (MET) and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© Copyright Ostia 2012.

All rights reserved.

The name Ostia Software Solutions and/or all Ostia Software Solutions product names are either trademarks or registered trademarks of Ostia Software Solutions. Other company and product names mentioned herein may be trademarks of their respective owners.

Table of Contents

Reference	v
1 Portus Resource Access	1
Overview	2
2 SOAP	15
SOAP Headers	16
Soap Operations for Server Configuration	19
3 REST	23
Introduction	24
4 Frequently Asked Questions	41
How do I active the Software AG sagenv file post Portus Installation?	42
How do I modify the machine identifier in the JESMSGGLG?	42
5 Performance Hints	43
6 Internationalization	45
Setting codepages	46
Which codepage do I use?	46
SOAP versus REST differences	46
Troubleshooting	47
7 Creating a Stylesheet for your Portus Data	49
Create HTML page from City XML	50
8 Language Structure Support	59
The Portus Representation of Data	60
'Tuple' Based Databases	60
Record Based Databases	61
Calling Application Programs	61
Representing Individual Fields	62
Representing Structures	63
Representing Arrays	65
Representing Redefines	67
Summary	70
9 Data Masking	71
Example	72

Reference

The following reference materials are available for Portus:

- Glossary
- [Portus Resource Access](#)
- [SOAP](#)
- [REST](#)

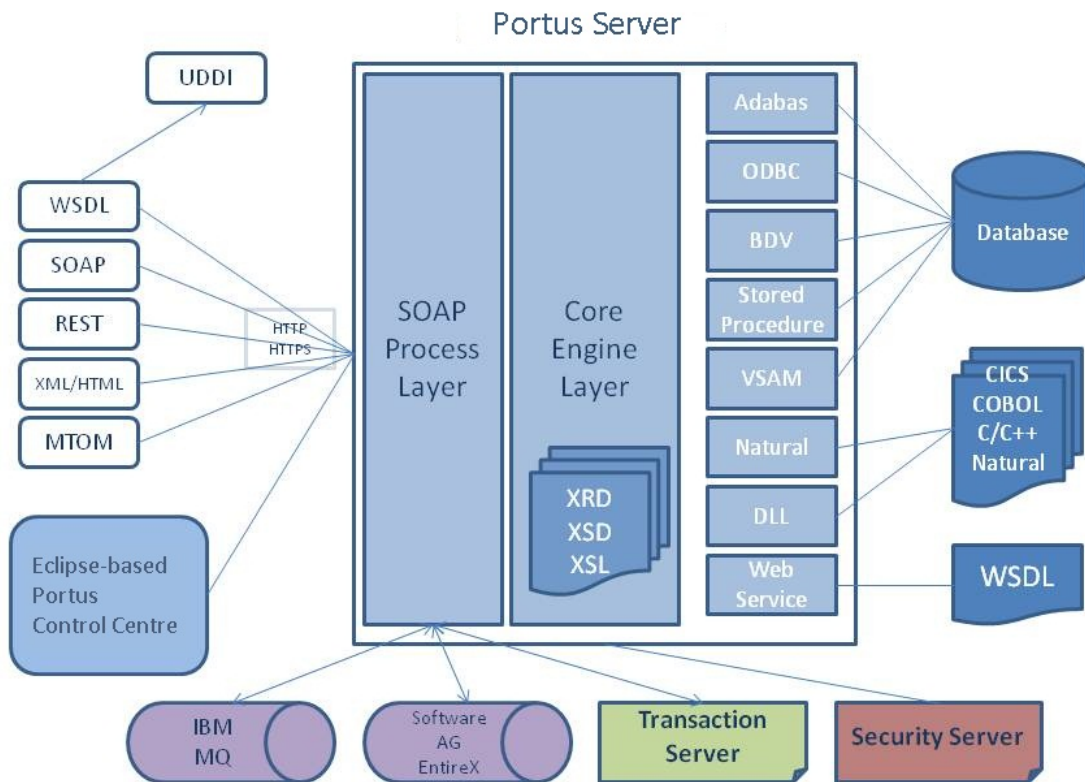
1 Portus Resource Access

- Overview 2

Overview

This section describes the operations exposed by Portus to access data sources. These operations are described in a WSDL which is defined for each web service. A client program connecting to a web service can read the WSDL to determine what operations are available on the server. These operations can be invoked using **SOAP** or **RESTful** query.

Portus also supports MTOM for binary data, raw XML, and HTML. WSDLs can be registered in a UDDI server for service look-up.



Supported protocol Versions

WSDL 1.1

SOAP 1.1

MTOM 1.0

HTTP 1.1

ODBC Version 3

Portus can also connect to the Messaging systems such as IBM MQ and Software AG EntireX via their C-interfaces.

Web Service Security can be handled by HTTP Basic Profile authorization (via Apache), or connectivity with an external security manager, such as RACF.

Further Reading

Prerequisites

Retrieve WSDL

Web Service Operations

Providing Key Information

Prerequisites

At this point your Portus Server should be installed, configured and started.

If you have not yet configured any web services, please refer to the Portus Control Center section and add at least one web service

Retrieving the WSDL for your resource.

All web services defined will have a WSDL associated with them. This is the starting point for using the operations provided by Portus. The WSDL describes the operations that may be carried out, and how they are used. This includes a description of valid parameters, data and responses for each of the operations.

In order to get the WSDL for a particular resource you simply issue a standard http request, specifying:

- the server name or *IP-address* of the server where Portus is installed [and running]
- the *TCP Port number* that Portus is listening on (as provided in the installation)
- the *name* of the service. (this is the value of the "Name" field in the web service properties)

The following example shows the URL required to retrieve the WSDL for an Adabas "Employees" file.

Server Name	PortusGateway
Port Number	56000
Name	adabas_Employees
URL for WSDL	http://PortusGateway:56000/adabas_Employees?WSDL

Portus Web Services Operations for Data Resources

The operations provided by Portus for accessing data resources [files, databases, programs, etc.] are now explained.

Parameters are required unless otherwise stated.

Operation	<p><i>list</i> (SOAP)</p> <p><i>LIST</i> (REST)</p>
Description	<p>The <i>list</i> operation returns a list of records or rows from your data source.</p> <p>The data returned can be limited or restricted by providing <i>key</i> information.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Key data</i> <p>Key data must be entered for at least one of the fields defined as a key.</p> <p>Refer to the section Specifying Key Data for more information.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	<p><i>select</i> (SOAP)</p> <p><i>SELECT</i> (REST)</p>
Description	<p>The <i>select</i> operation returns a list of records or rows from your data source. The maximum number of rows/records returned can be set via the Portus Control Center.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p> <p>The data returned can be limited or restricted by providing <i>key</i> information. The select operation extends the capability of the list operation by enabling searches on a larger set of criteria.</p> <p>The key information for a select is wrapped in a condition block and can be repeated several times within that block. Each key entry represents an 'AND' condition. Condition blocks can also be repeated several times. Each condition block represents an 'OR' condition. The condition block accepts the following specifiers:</p>
Conditions	<ul style="list-style-type: none"> ■ <i>Less than a specific value (LT).</i> ■ <i>Less than or equal to a specific value (LE).</i> ■ <i>Equal to a specific value (EQ).</i> ■ <i>Greater than a specific value (GT).</i> ■ <i>Greater than or equal to a specific value (GE).</i> ■ <i>Not equal to a specific value (NE).</i> ■ <i>Starting with a specific value (START). Character based fields only.</i> ■ <i>Ending with a specific value (ENDS). Character based fields only.</i> ■ <i>Containing a specific value (CONTAINS). Character based fields only.</i>
Example	<p>SOAP</p> <pre style="background-color: #f0f0f0; padding: 10px;"> <soapenv:Body> <nos:adabasEmployeeSelectElement> <!--1 or more repetitions:--> <condition> <!--Zero or more repetitions:--> <personnel_id Condition="GT">50012100</personnel_id> <personnel_id Condition="LE">50012700</personnel_id> </condition> <condition> <!--Zero or more repetitions:--> <personnel_id Condition="EQ">50012900</personnel_id> </condition> </nos:adabasEmployeeSelectElement> </soapenv:Body> </pre> <p>REST</p> <pre style="background-color: #f0f0f0; padding: 10px;"> http://localhost:56005/adabas_Employees_9? SELECT &condition[1].personnel_id>50012100 </pre>

	<pre>&condition[1].personnel_id<=50012700 &condition[2].personnel_id=50012900</pre> <p>The example above specifies 2 condition blocks. This will return data where the (personnel_id > 50012100 and personnel_id <= 50012700) or personnel_id = 50012900</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Key data</i> <p>Key data must be entered for at least one of the fields defined as a key.</p> <p>Refer to the section Specifying Key Data for more information.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	selectNext
Description	<p>The <i>selectNext</i> operation returns a list of records or rows from your data source. A selectNext operation may only be called following a select and subsequently other selectNext calls. For this functionality the initial select operation has to initiate a new Conversation. See Conversational Processing. The resultant conversation id must be passed in any associated selectNext calls. selectNext calls may be issued until end of file is reached or may be terminated by a selectEnd call. The maximum number of rows/records will be that set for set via the Portus Control Center.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p> <p>There is no key information for a selectNext operation as this will have been passed in by the initiating select operation.</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Key data</i> <p>None</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>

Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>Not available</p>
--------	---

Operation	selectEnd
Description	<p>The <i>selectEnd</i> operation terminates a sequence of select and/or selectNext calls with a conversation. A selectEnd operation may only be called following select or selectNext operations. For this functionality the select operation has to initiate a new Conversation. The resultant conversation id must be passed in the selectEnd call.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p> <p>There is no key information for a selectEnd operation as this will have been passed by the initiating select operation.</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Key data</i> None ■ <i>Options</i> None
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>Not available</p>

Operation	selectCount
Description	<p>The <i>selectCount</i> operation returns a count of the records or rows that match the criteria set in the condition block(s).</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p> <p>The selectCount operation is identical to that of the select operation in terms of its search capabilities.</p>

	The key information for a selectCount is wrapped in a condition block and can be repeated several times within that block. Each key entry represents an 'AND' condition. Condition blocks can also be repeated several times. Each condition block represents an 'OR' condition. The condition block accepts the following specifiers:
Conditions	<ul style="list-style-type: none"> ■ <i>Less than a specific value (LT).</i> ■ <i>Less than or equal to a specific value (LE).</i> ■ <i>Equal to a specific value (EQ).</i> ■ <i>Greater than a specific value (GT).</i> ■ <i>Greater than or equal to a specific value (GE).</i> ■ <i>Not equal to a specific value (NE).</i> ■ <i>Starting with a specific value (START). Character based fields only.</i> ■ <i>Ending with a specific value (ENDS). Character based fields only.</i> ■ <i>Containing a specific value (CONTAINS). Character based fields only.</i>
Parameters	<ul style="list-style-type: none"> ■ <i>Key data</i> Key data must be entered for at least one of the fields defined as a key. Refer to the section Specifying Key Data for more information. ■ <i>Options</i> None
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	<p><i>get</i> (SOAP)</p> <p><i>GET</i> (REST)</p>
Description	<p>The <i>get</i> operation returns a single record or row from your data source.</p> <p>The data returned is specified by providing <i>unique key</i> information identifying a single record / row.</p>

	This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL
Parameters	<ul style="list-style-type: none"> ■ <i>Key Data</i> <p>Key data must be entered for at least one of the fields defined as a primary key field, so that a single record can be identified.</p> <p>Refer to the section Specifying Key Data for more information.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	<p><i>add</i> (SOAP)</p> <p><i>ADD</i> (REST)</p>
Description	<p>The <i>add</i> operation adds a single record or row of data to your data source.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Add Data</i> <p>Provide values for each of the fields defined. These are the fields / columns of the data on your data source.</p> <p>You must add data for at least one of the fields specified as being <i>primary key</i> fields.</p> <p>Refer to the section Specifying Key Data for more information.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p>

	<ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message
--	---

Operation	<p><i>update</i> (SOAP)</p> <p><i>UPDATE</i> (REST)</p>
Description	<p>The <i>update</i> operation updates a single record or row of data.</p> <p>This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL</p>
Parameters	<ul style="list-style-type: none"> ■ <i>Update Data</i> <p>Provide values (NULL or otherwise) for each of the fields defined. These are the fields / columns of the data on your data source.</p> <p>Key data must be entered for at least one of the fields defined as a <i>primary key</i> field.</p> <p>Refer to the section Specifying Key Data for more information.</p> <p>Fields that have been left empty or NULL, e.g. <data></data> will be set to this value accordingly.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	<i>delete (SOAP)</i> <i>DELETE (REST)</i>
Description	The <i>delete</i> operation deletes a single record or row of data from the data source. This operation will only be available when the web service is a "database" type, for example, Adabas or MySQL
Parameters	<ul style="list-style-type: none"> ■ <i>Key Data</i> <p>Key data must be entered for at least one of the fields defined as a <i>primary key</i> field, so that a single record can be identified.</p> <p>Refer to the section Specifying Key Data for more information.</p> <ul style="list-style-type: none"> ■ <i>Options</i> <p>None</p>
Result	<p>SOAP</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message

Operation	<i>invoke (SOAP)</i> <i>INVOKE (REST)</i>
Description	The <i>invoke</i> operation makes a call to a function or program. This operation will only be available when the web service is a "program" type, for example, NATURAL, COBOL or when the web service describes a database stored procedure.
Parameters	<ul style="list-style-type: none"> ■ <i>Parameters</i> <p>Each of the fields defined can be input, output, or input+output.</p> <p>Provide values for each of the fields defined as input or input+output. These match the function/program parameters that are required on the call.</p>
Result	<p>SOAP</p> <p>The result will be either :</p>

	<ul style="list-style-type: none"> ■ an XML document wrapped in a SOAP message and containing the requested data ■ a SOAP fault message <p>REST</p> <p>The result will be either :</p> <ul style="list-style-type: none"> ■ an XML document containing the requested data ■ a fault message
--	---

Specifying Key Data

Keys or *Key Fields* in Portus terms are simply fields or columns on your data source (file, table, etc) that may be used to narrow a search for a record or row of data.



Note: *key fields* may or may not be *indexed* at your data source. Indexing provides better response times from your data source. If you are unsure what, if any, fields are key fields (in Adabas terms: Descriptors) or allowed to be used for searching, contact your data source administrator (DBA, etc.).

Key fields are specified to Portus by setting the appropriate value for the attribute "*key type*" in the Resource Description.

Key Types

There are two types of *key fields* for data sources:

Primary Key Fields

Primary Key Fields are fields in a data source that must always contain unique values.

They are required on all data source operations except the *list* operation, where they are optional. Secondary keys may be used instead or in addition on a list request.

Their values will not be altered in the *update* operation.

Primary Keys cannot contain *wild card* symbols, except on the *list* operation.

Secondary Key Fields

Secondary Key Fields are fields that may be used in narrowing a search.

They are not required in any operation, and their values can be updated.

They may contain wild card symbols.

Using Wild card Symbols and other Generic Search criteria

Wild cards are used where you do not wish to specify an exact value in a key field, but use a generic specification that will match for a range of different values.

These search modifiers may only be used on the *list* operation.

Portus currently supports the following generic search criteria :

Wild card for one or more characters

The character "*" may be used as a wild card for all characters.

It may only be used on fields defined as "*string*" fields.

Where it appears, it will match any character or group of characters.



Note: for Adabas resources, this wildcard may only appear at the *end* of the string data supplied.

Example

"Ga*" would match "Gat", "Gate", "Gateway", etc.

2 SOAP

- SOAP Headers 16
- Soap Operations for Server Configuration 19

SOAP Headers

In Portus, the SOAP Headers are used for versioning, the support of conversational SOAP processing, support of transactions, and specific settings on the datasource you are accessing. By default all elements are "empty". To get the default behaviour, all header elements should be left blank, or removed altogether. Example:

```
<soap:Envelope xmlns:rapdv="http://www.risaris.com/namespaces/xmiddle" ↵
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" ↵
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <rapdv:AdabasEmployeeHeader>
      <Version/>
      <ConversationState/>
      <ConversationId/>
      <TransactionState/>
      <TransactionId/>
    </rapdv:AdabasEmployeeHeader>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

Conversational Processing

Every time a SOAP request is made to Portus, this request must be associated with a specific context. By default, a new context is created and destroyed for every SOAP Request.

The user may also use the SOAP Headers to re-use and re-connect to a specific context.

When a user starts, re-uses, and ultimately finishes with their context, the process is known as a "conversation". In the SOAP Header, the "ConversationState" and "ConversationId" are used for conversational processing, see below for more information.

The Version Element

The 'Version' element is currently unused. It will be brought into use in future versions of Portus.

The ConversationState Element

The 'ConversationState' element is used to control conversation processing. It should be one of the following:

State	Description
New	A new conversation is being started. In this case, the ConversationID (see below) value must be NULL or an error will occur.
Old	An existing conversation is active with which the current SOAP message should be associated. When the SOAP request is processed, the conversation must remain active as there are further SOAP requests to be sent. The ConversationID found must have been returned as a result of a previous 'new' request. An error will occur if the ConversationId (see below) provided cannot be found.
End	An existing conversation is active with which the current SOAP message should be associated. When the SOAP request is processed, the conversation must be terminated. The ConversationID (see below) found must have been returned as a result of a previous 'new' request. An error will occur if the ConversationId provided cannot be found.

The ConversationId Element

The ConversationId uniquely identifies the conversation, and it returned only after a 'New' request is successfully processed. The user should never modify or create this ID. The ConversationId must be present on an 'Old' or 'End' request.

Transaction Processing

In Portus, the platform's Transaction Manager (TM) will be engaged to handle transactions. A default TM is provided as part of the ASG installation. It can be found in <asg install dir>/libraries/transactionManagerDummyDll.so. The environment variable TMSTUB is used to point to the transaction manager shared object. This shared object provides interfaces to handle the transaction.



Note: The framework for engaging transaction managers is subject to change. Currently the transactionManagerDummyDll.so does not provide any "real" transactionality. Rísaris are current looking for early adopters to work with us to fully implement and test this technology.

In the SOAP Header, the "TransactionState" and "TransactionId" are used for transactional processing; see the table below for more information.

The TransactionState Element

If a transaction involves modifications that will occur over multiple SOAP messages, the conversational processing logic must be used to associate the SOAP messages and thus the transactional information.

If a transaction involves modifications that will all be completed as part of the one SOAP message, the conversational processing logic is not required.

When the request uses an active conversation, and is add/update/delete, a transaction is implicitly started.

If a conversation is ended before the transaction is committed, a implicit back out will occur.

The Transaction State may be one of the following:

State	Description
Commit	<p>This will cause a commit to be issued when the current SOAP message has been processed.</p> <p>If no conversation existed previously for this SOAP request, an error will result.</p> <p>When returned in the SOAP response, it indicates that the transaction has been committed, and transaction ID is provided for reference.</p>
Backout	<p>This will cause a backout to be issued when the current SOAP message has been processed.</p> <p>If no conversation existed previously for this SOAP request, an error will result.</p> <p>When returned in the SOAP response, it indicates that the transaction has been backed out, and transaction ID is provided for reference.</p>

The Transaction Id Element

The Transaction ID is purely informational, and has no functional bearing on the transaction process. It is returned on any SOAP message and is intended to be used for tracking purposes.

It is not mandatory to provide the Transaction ID. As only 1 transaction can ever be active on a conversation, Portus will auto-reconnect to the current transaction internally.

Adabas specific headers

When working with Adabas services, there are a number of specific SOAP headers that will be available

These headers are listed in the Using Portus with Adabas section.

Relational database specific headers

When working with relational database services, i.e. MySQL, MS SQL Server, DB2, Oracle, etc, the specific SOAP Header are

SOAGateway_Internal_AutoCommit	Turn off the AutoCommit flag on the database
--------------------------------	--

Soap Operations for Server Configuration

Portus exposes a number of SOAP operations / methods to retrieve and alter the Portus Server Configuration from any SOAP enabled client.



Note: These interfaces are likely to change in the future, they are provided on a trial basis, be aware that you might have to change any "applications" built on top of these interfaces ! As soon as "stable" interfaces are available, this fact will be announced and documented.

adaptorList

This operation is used to load up a Portus driver library, and return the associated internal information.

This operation takes 1 input, libraryName should be set to the name of the library to load and query.

configList

This operation is used to list the current configuration in use by the Portus server. The current configuration is useful where you wish to make changes to existing configuration items, or just to make sure you are not trying to add, for example, a resource *URI* that is already in use.

This operation currently has no options.

The full configuration document, minus the XML header, is returned in the *soap response*.

configRemove

This operation is used to reset the value of an item in the configuration, or to remove an item from the configuration.

Only certain items or levels of items may be altered using this operation. The definition in the WSDL shows what elements may be used. Refer to it for further details on what type of items may be removed and what items may only have their value reset.

A configuration item which is reset will assume it's default value if it is an item that cannot be removed from the configuration.

Configuration items changed by this operation are effective as soon as the engine can make them so.

This operation currently has no options.

configReplace

This operation is used to replace the current configuration file, or to write a new configuration to file on the server.

If the element '*configFileName*' is omitted or left empty, then the configuration file currently in use will be overwritten - if it is in the configuration directory.

If the element '*configFileName*' is specified, then the configuration will be written to a file of that name in the configuration directory.

The newly written configuration will not come into effect until the Portus server is next started.

If the element '*configFileName*' was specified, and you wish to use this configuration, then before re-starting the Portus server you must alter the system environment variable *XMIDDLE_CONFIGURATION_FILE* so that it refers to your new configuration file.

This operation currently has no options.

configSet

This operation is used to set the value of an item in the configuration, or to add an item to the configuration.

Only certain items or levels of items may be altered using this operation. The definition in the WSDL shows what elements may be used. Refer to it for further details on what type of items may be added and what items may only have their value altered.

Configuration items changed by this operation are effective as soon as the engine can make them so.

This operation currently has no options.

3 REST

- Introduction 24

Introduction

Web services can also use other technologies, apart from SOAP, such as RESTful implementations on top of HTTP. Representational State Transfer (REST) is an approach based on the architectural style of the Web itself. The Portus also provides this URL based approach to access resources.

REST Overview

Portus allows users to access any web service via a REST-style URL request. In general, this is a more simplistic way of accessing services, useful in demo scenarios, and with clients that do not have support for SOAP, but do have support for retrieving URLs information (such as Microsoft Excel).

A REST request is similar to the WSDL request, but with extra arguments. Generally, it is recommended that the WSDL is retrieved first, as it gives the client the ability to see what fields have been set as keys. All operations that are possible using the WSDL are possible with REST, with some caveats.

<i>Operation</i>	<i>Notes</i>
get	MTOM is not supported. In the case where binary objects are returned on request, the XML will be escaped into HTML, and a link to the binary object will also be returned.
add/update	HTTP POST must be used.
delete	HTTP DELETE must be used.

Example

The following is an example of retrieving data with a REST request

```
http://host:port/myService?LIST&ID=4*&Name=J*
```

This will attempt to call the "list" operation, passing in a value of 4* to the ID field (which has been defined as a primary/secondary key) AND the Name field set to J*

Enhanced REST Operations

Portus provides several operations for each web service so it has enhanced its REST implementation to support them e.g. SELECT and INVOKE. Typically these may require complex parameters in order to be called.

```
http://localhost:56005/adabas_Employees_9?  
SELECT  
&condition[1].personnel_id>50012100  
&condition[1].personnel_id<=50012700  
&condition[2].personnel_id=50012900
```

The example above specifies 2 condition blocks. This will return data where the (personnel_id > 50012100 and personnel_id <= 50012700) or personnel_id = 50012900

Database WSDLs

A Portus database WSDL defines requests which reflect database access.

Supported Requests

1. LIST
2. GET
3. DELETE
4. ADD
5. UPDATE
6. SELECT
7. SELECTCOUNT

URI

As usual in the definitions element there will be a value for the targetNamespace uri:

```
<definitions targetNamespace="uri://46.46.46.46:56421/Customers"
name="CustomersRootCollection">
```

The uri gives us the starting portion of a REST request:

```
http://46.46.46.46:56421/Customers
```

Note that in Portus WSDLs a unique identifier (UNIQID) is prepended to various elements and also contained in the uri e.g. in this case Customers in the name CustomersRootCollection.

Messages

For each the above requests there will be a message entry in the WSDL with the following names:

getRequest, listRequest, deleteRequest, addRequest, updateRequest, selectRequest and selectCountRequest

e.g.

```
<message name="listRequest">
```

```
<part name="CustomersGroupListKey" element="asg:CustomersGroupListElement"/>
```

```
</message>
```

```
.  
.
<message name="getRequest">
<part name="CustomersGroupGetKey" element="asg:CustomersGroupGetElement"/>
</message>
.  
.
<message name="deleteRequest">
<part name="CustomersGroupDeleteKey" element="asg:CustomersGroupDeleteElement"/>
</message>
```

Note that there will be some others which are in the WSDL which are not supported in a REST request i.e. selectNext and selectEnd.

e.g.

```
<message <message name="selectNextRequest">
<part name="CustomersGroupSelectNextRequest" element="asg:CustomersGroupSelectNextElement"/>
</message>
```

Each message element has a part element which gives further details about the request structure via REST:

LIST

```
<part name="UNIQIDGroupListKey" element="asg:UNIQIDGroupListElement"/>
<xs:element name="UNIQIDGroupListElement" type="asg:UNIQIDGroupKeyType"/>
<xs:complexType name="UNIQIDGroupKeyType">
<xs:sequence>
```



```
<xs:element name="ID" nillable="true" type="xs:int"/>
<xs:element name="Account_ID" nillable="true" type="xs:int"/>
</xs:sequence>
</xs:complexType>
```

At this point we know our input parameter(s). A feature of the LIST request is that these parameters can be wild carded as shown below and/or omitted:

```
http://46.46.46.46:56421/Customers?LIST&ID=*
http://46.46.46.46:56421/Customers?LIST&ID=2
http://46.46.46.46:56421/Customers?LIST&ID=2*
http://46.46.46.46:56421/Customers?LIST&ID=* &Account_ID=2*
http://46.46.46.46:56421/Customers?LIST&Account_ID=2*
http://46.46.46.46:56421/Customers?LIST&Account_ID=*5
```

GET

```
<part name="UNIQIDGroupGetKey" element="asg:UNIQIDGroupGetElement"/>
<xs:complexType name="UNIQIDGroupPrimaryKeyType">
<xs:sequence>
<xs:element name="ID" nillable="true" type="xs:int"/>
</xs:sequence>
</xs:complexType>
```

At this point we know our input parameter(s). Note that a GET targets a specific row in the database and returns one record or none if not found:

```
http://46.46.46.46:56421/Customers?GET&ID=25
```

DELETE

```
<part name="UNIQIDGroupDeleteKey" element="asg:UNIQIDGroupDeleteElement"/>
<xs:element name="UNIQIDGroupDeleteElement" type="asg:UNIQIDGroupPrimaryKeyType"/>
<xs:complexType name="UNIQIDGroupPrimaryKeyType">
```

```
<xs:sequence>
<xs:element name="ID" nillable="true" type="xs:int"/>
</xs:sequence>
</xs:complexType>
```

At this point we know our input parameter(s). Note that a DELETE targets a specific row in the database. If successful it returns a 'delete successful' message or an error stating that it does not exist.

<http://46.46.46.46:56421/Customers?DELETE&ID=25>

ADD

```
<part name="UNIQIDRoot" element="asg:UNIQIDRootAddElement"/>
<xs:element name="UNIQIDRootAddElement" type="asg:UNIQIDRootType"/>
<xs:complexType name="UNIQIDRootType">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="UNIQIDGroup"
type="asg:UNIQIDGroupType"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="UNIQIDGroupType">
<xs:sequence>
<xs:element name="ID" nillable="true" type="xs:int"/>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="Surname" type="xs:string"/>
<xs:element name="Street" type="xs:string"/>
<xs:element name="City" type="xs:string"/>
<xs:element name="State" type="xs:string"/>
<xs:element name="Zip" type="xs:string"/>
<xs:element name="Phone" type="xs:string"/>
```

```

<xs:element name="SSN" nillable="true" type="xs:int"/>
<xs:element name="Account_ID" nillable="true" type="xs:int"/>
</xs:sequence>
</xs:complexType>

```

At this point we know our input parameter(s). All elements are at the same level i.e. not in a structure so can be sequentially added to the REST request.

```
http://46.46.46.46:56421/Customers?AD&ID=Value&FirstName=Value&Surname=Value&Street=Value&City=Value&State=Value&Zip&Phone=Value&SSN&Account_ID=Value
```

UPDATE

```

<part name="UNIQIDRootUpdate" element="asg:UNIQIDRootUpdateElement"/>
<xs:element name="UNIQIDRootUpdateElement" type="asg:UNIQIDRootType"/>
<xs:complexType name="UNIQIDRootType">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="UNIQIDGroup"
type="asg:UNIQIDGroupType"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="UNIQIDGroupType">
<xs:sequence>
<xs:element name="ID" nillable="true" type="xs:int"/>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="Surname" type="xs:string"/>
<xs:element name="Street" type="xs:string"/>
<xs:element name="City" type="xs:string"/>
<xs:element name="State" type="xs:string"/>
<xs:element name="Zip" type="xs:string"/>
<xs:element name="Phone" type="xs:string"/>

```

```
<xs:element name="SSN" nillable="true" type="xs:int"/>
<xs:element name="Account_ID" nillable="true" type="xs:int"/>
</xs:sequence>
</xs:complexType>
```

At this point we know our input parameter(s). All elements are at the same level i.e. not in a structure so can be sequentially added to the REST request. Note that as ID (see UNIQID-GroupPrimaryKeyType)is the primary key, the value passed in the request should exist in the database table.

```
http://46.46.46.46:56421/Customers?UPDATE&ID=existingKeyValue&FirstName=Value&Surname=Value&Street=Value&City=Value&State=Value&Zip&Phone=Value&SSN&Account_ID=Value
```

SELECT

```
<part name="UNIQIDGroupSelectKey" element="asg:UNIQIDGroupSelectElement"/>
<xs:element name="UNIQIDGroupSelectElement" type="asg:UNIQIDGroupSelectType"/>
<xs:complexType name="UNIQIDGroupSelectType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="1" name="condition">
      <xs:complexType>
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0" name="ID">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="Condition" type="asg:conditionType"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
```

```

<xs:element maxOccurs="unbounded" minOccurs="0" name="Account_ID">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Condition" type="asg:conditionType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

```

At this point we know our input parameter(s).

N.B.

Both select and selectCount have input the elements of which are contained in a structure. The parent element is condition which will contain elements which are primary and secondary keys (in the underlying database). The SOAP equivalent message portion would be:

```

<!--1 or more repetitions:-->
<condition>
  <!--Zero or more repetitions:-->
  <ID Condition="?"></ID>
  <!--Zero or more repetitions:-->
  <Account_ID Condition="?"></Account_ID>
</condition>

```

(As per the WSDL the Condition type can be EQ, NE, LT, LE, GT, GE, STARTS, CONTAINS and ENDS).

```
<condition>
```

```
<ID Condition="GT">4</ID>
```

```
<ID Condition="LE">10</ID>
```

```
</condition>
```

```
<condition>
```

```
<Account_ID Condition="EQ">23</Account_ID>
```

```
</condition>
```

The above element will select records where the ID is greater than 4 AND less than or equal to 10 OR where Account_ID is equal to 23.

There are 2 condition elements so use array notation for those (one being the base).

Use numeric notation for the condition type i.e. for GT use >

e.g.

```
http://46.46.46.46:56421/Customers?SELECTCOUNT&condition[1].ID>4&condition[1].ID<=10&condition[2].Account_ID=23
```

Program WSDLs

A Portus WSDL which is program based supports an INVOKE request.

Simple Example

Excerpt from typical program WSDL:

```
<xs:element name="invokeInputElement">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="SOABSP_CALCULATRoot">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```

<xs:element name="SOABSP_CALCULATGroup">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OPERATION" type="xs:string"/>
      <xs:element name="OPERAND_1" nillable="true" type="xs:int"/>
      <xs:element name="OPERAND_2" nillable="true" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

```

- In the definitions element there will be a value for the targetNamespace uri:

```
<definitions targetNamespace="uri://www.versatec.info:56421/SOABSP_CALCULAT" ...
```

The uri gives us the starting portion of a REST request:

```
http://www.versatec.info:56421/SOABSP_CALCULAT
```

Note also that in Portus WSDLs a unique identifier (UNIQID) is prepended to various elements and also contained in the uri e.g. in this case SOABSP_CALCULAT:

```
<xs:element name="SOABSP_CALCULATRoot">
```

- There will be an element with a name of invokeInputElement. This is reflected in the F=INVOKE portion of the REST request.

```
http://www.versatec.info:56421/SOABSP_CALCULAT?F=INVOKE
```

- invokeInputElement will contain an element with a name UNIQIDRoot
- UNIQIDRoot will contain an element with a name UNIQIDGroup

- UNIQIDGroup will contain the elements that are passed in the INVOKE for a REST request e.g.

```
<xs:element name="OPERATION" type="xs:string"/>
```

```
<xs:element name="OPERAND_1" nillable="true" type="xs:int"/>
```

```
<xs:element name="OPERAND_2" nillable="true" type="xs:int"/>
```

```
http://www.versatec.info:56421/SOABSP_CALCULAT?F=INVOKE&OPERATION=mul&OPER-  
AND_1=2345&OPERAND_2=6789
```

Complex Example

Excerpt from more complex program WSDL:

```
<xs:element name="invokeInputElement">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="QEESPN01Root">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="QEESPN01Group">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="QEESPS01">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="REDEFINE_001_IPF">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="TIPO_IPF" type="xs:string"/>
```

```
<xs:element name="REDEFINE_002_NUM_IPF">
```

```
<xs:complexType>
```



```
<xs:sequence>
<xs:element maxOccurs="10" name="NUMN_IPF" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="FECHA_DESDE" nillable="true" type="xs:decimal"/>
<xs:element name="REDEFINE_003_COD_IPF">
<xs:complexType>
<xs:sequence>
<xs:element name="ALFA2" type="xs:string"/>
<xs:element name="ALFA8" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="FECHA_NAC" type="xs:string"/>
<xs:element maxOccurs="10" name="DUP_F10" nillable="true" type="xs:decimal"/>
<xs:element maxOccurs="100" name="D_SALIDA">
<xs:complexType>
<xs:sequence>
<xs:element name="MOMMAP" type="xs:string"/>
<xs:element name="SALIDA_DATA" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

```
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- In the definitions element there will be a value for the targetNamespace uri:

```
<definitions targetNamespace="uri://meath-nua:56008/QEESPN01" name="QEESPN01RootCollection">...
```

The uri gives us the starting portion of a REST request:

```
http://meath-nua:56008/QEESPN01
```

In Portus WSDLs a unique identifier (UNIQID) is prepended to various elements and also contained in the uri e.g. in this case QEESPN01

- There will be an element with a name of invokeInputElement. This is reflected in the F=INVOKE portion of the REST request.

```
http://meath-nua:56008/QEESPN01?F=INVOKE
```

- invokeInputElement will contain an element with a name UNIQIDRoot
- UNIQIDRoot will contain an element with a name UNIQIDGroup
- UNIQIDGroup will contain the elements that are passed in the INVOKE for a REST request e.g.

```
<xs:element name="QEESPS01">
```

```
<xs:complexType>
```

```
<xs:sequence>
  <xs:element name="REDEFINE_001_IPF">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="TIPO_IPF" type="xs:string"/>
        <xs:element name="REDEFINE_002_NUM_IPF">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="10" name="NUMN_IPF" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="FECHA_DESDE" nillable="true" type="xs:decimal"/>
  <xs:element name="REDEFINE_003_COD_IPF">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ALFA2" type="xs:string"/>
        <xs:element name="ALFA8" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="FECHA_NAC" type="xs:string"/>
  <xs:element maxOccurs="10" name="DUP_F10" nillable="true" type="xs:decimal"/>

```

```
<xs:element maxOccurs="100" name="D_SALIDA">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MOMMAP" type="xs:string"/>
      <xs:element name="SALIDA_DATA" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- If the element names within UNIQIDGroup are in a structure then the REST request must reflect that:

```
QEEPSN01?INVOKE&QEEPS01.REDEFINE_001_IPF.TIPO_IPF=0&QEEPS01.RE-
DEFINE_001_IPF.REDEFINE_002_NUM_IPF.NUMN_IPF[0]=0&QEEPS01.RE-
DEFINE_001_IPF.REDEFINE_002_NUM_IPF.NUMN_IPF[1]=0&QEEPS01.RE-
DEFINE_001_IPF.RE-
DEFINE_002_NUM_IPF.NUMN_IPF[2]=0&QEEPS01.FECHA_DESDE=0&QEEPS01.RE-
DEFINE_003_COD_IPF.ALFA2=0&QEEPS01.RE-
INFOC_IPF&FECHA_INI&FECHA_FIN&FECHA_SALIDA_M&FECHA_SALIDA_A&FECHA_SALIDA_D&FECHA_SALIDA_H&FECHA_SALIDA_M&FECHA_SALIDA_S&FECHA_SALIDA_D&FECHA_SALIDA_H&FECHA_SALIDA_S
```

1. QEEPS01 is the top level element name. It has a child REDEFINE_001_IPF which in turn has a child element TIPO_IPF so the parameter should be specified as:

```
&QEEPS01.REDEFINE_001_IPF.TIPO_IPF=0
```

2. If an element can occur more than once (maxOccurs > 1) then use array notation:

```
&QEEPS01.REDEFINE_001_IPF.REDEFINE_002_NUM_IPF.NUMN_IPF[0]=0&QEEPS01.RE-
DEFINE_001_IPF.REDEFINE_002_NUM_IPF.NUMN_IPF[1]=0&QEEPS01.RE-
DEFINE_001_IPF.REDEFINE_002_NUM_IPF.NUMN_IPF[2]=0
&QEEPS01.MOMMAP&QEEPS01.SALIDA_DATA&QEEPS01.MOMMAP2&QEEPS01.SALIDA_DATA5
```

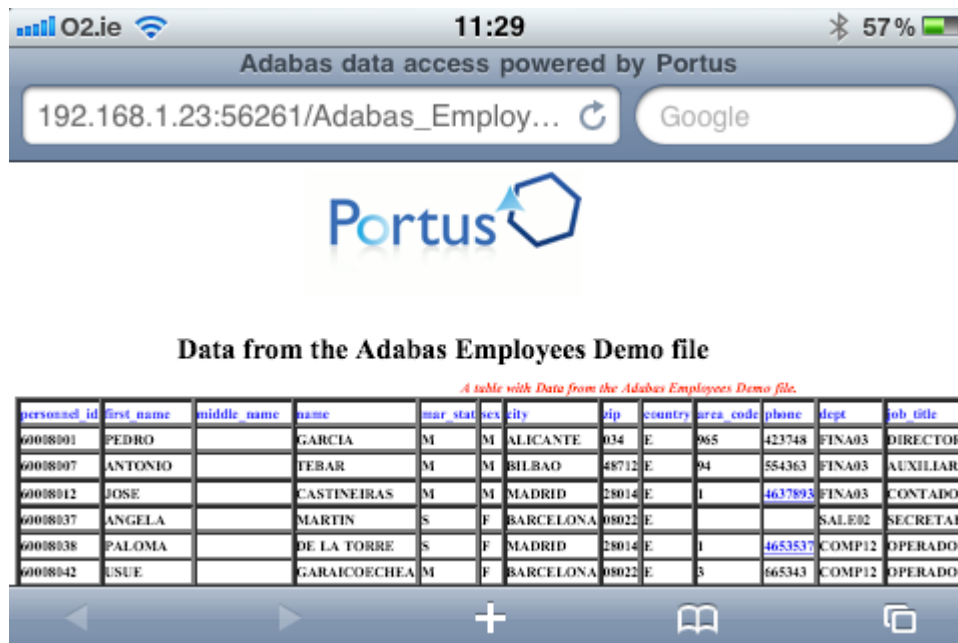
XSL Transformation

REST requests also support XSL transformation. The XSL file should be defined in the Control Centre, and have the same name as the XRD and XSD file. When a REST request is made, the URL for the XSL is added into the returned XML response, thus allowing the client to retrieve the XSL, and apply the transformation.

For clients that do not support client-side XSL transformation, such as some Android and Blackberry devices, it is possible to apply the transformation on the server side (e.g. Portus will apply the transformation, and return the transformed data). This is determined by the HTTP User-Agent header, and should normally be done transparently. It is possible to force client or server transformation with an option on the REST URL.

`http://host:port/myService?LIST&__xslTransform=server&ID=1*`

`http://host:port/myService?LIST&__xslTransform=client&ID=1*`



Data from the Adabas Employees Demo file

A table with Data from the Adabas Employees Demo file.

personnel id	first name	middle name	name	mar stat	sex	city	zip	country	area code	phone	dept	job title
6008901	PEDRO		GARCIA	M	M	ALICANTE	034	E	965	423748	FINA03	DIRECTOR I
6008907	ANTONIO		TEBAR	M	M	BILBAGO	48712	E	94	554363	FINA03	AUXILIAR II
6008912	JOSE		CASTINEIRAS	M	M	MADRID	28014	E	1	4637893	FINA03	CONTADOR
6008937	ANGELA		MARTIN	S	F	BARCELONA	08022	E			SALE02	SECRETARI
6008938	PALOMA		DE LA TORRE	S	F	MADRID	28014	E	1	4653537	COMP12	OPERADOR
6008942	ESUE		GARAICOECHA	M	F	BARCELONA	08022	E	3	665343	COMP12	OPERADOR

Using different encodings

See here for more information about the `__encoding` option on REST request.

4

Frequently Asked Questions

- How do I activate the Software AG sagenv file post Portus Installation? 42
- How do I modify the machine identifier in the JESMSGLG? 42

How do I active the Software AG sagenv file post Portus Installation?

This issue only applies to Portus running on Linux, AIX or Solaris. If your license does not have any Adabas or Natural drivers enabled, this issue does not apply.

If the sagenv file is not available during the Portus install, Adabas or Natural drivers may not work correctly until this file is made available. For example, you have installed Portus before installing Adabas.

Follow these steps to enable a new sagenv file in an existing Portus installation.

1. Edit the `[INSTALL_HOME]/xmiddleEnv.sh`
2. Append the following to the end of this file (assuming sagenv file is `/opt/softwareAg/sagenv.new`)
:

```
if test -e "/opt/softwareAg/sagenv.new"; then  
source /opt/softwareAg/sagenv.new &> /dev/null  
  
fi
```

3. Save and close this file.
4. Stop and then Start the Portus server. See here for more information.
5. Modify the `[INSTALL_HOME]/apache2/conf/adabas_soa_gw.conf` and change the User directive to use the sag user.
6. Change the file permissions of Portus files and directories

```
chown -R sag [INSTALL_HOME]
```

7. Review the `error_log` for errors/warnings.
8. Adabas or Natural drivers should now be added successfully. See here for more info.

How do I modify the machine identifier in the JESMSG LG?

During the FTP of Portus to z/OS, the machine identifier will be set to the hostnam or IP address of the FTP server. This text will be displayed on messages appearing in the JESMSG LG. To change this, modify the SYSPARM member of the CONF dataset and set this as required. It is recommended that this is set to the hostname or IP address of the z/OS machine.

5 Performance Hints

This section outlines some suggestions to improve the performance of Portus.

■ Remove XSD

This only applies for web services which use the invoke operation, e.g services built from the Natural, Cobol or DLL drivers.

Portus will validate the incoming XML against an XSD. This ensures that the contents and structure of the payload are correct, and will catch potential errors early on in processing. But XML validation is a relatively expensive operation, so it is possible to turn this off if required.

You may want to back up your existing XSD files before deletion. Use the "Import Service Definition" and ensure the XSD box is checked. See [here](#) for more information.

To delete the XSD, use the "Delete Service Definition", and ensure the XSD box is checked. The XSD has now been deleted from the server and validation of the payload will not take place.

■ Turning off Access Logging

Each time Portus handles a request, it writes some logging information to the access log via Apache. By default, this file is `access_log / access.log / DD:ACCESS` based on the platform, *nix, Windows, z/OS respectively.

To restrict this logging, see the following Apache directive [here](#)

To remove this logging, remove the CustomLog directive from your `httpd.conf / HTCONF`. This can be accomplished by adding a # in front of the directive.

■ Use PFS caching

This only applies on z/OS or z/VSE.

Edit your SYSPARM and ensure caching of the Portus filesystem has been turned on. The option is `CACHESIZE=N` option on the `CDI_DRIVER` directive

E.g

```
CDI_DRIVER=('pfs,PAANPFS,CONTAINER=CIO://DD:PFS,CHARSET=ASCII,LRECL=4096,CACHESIZE=4096')
```

■ Enable/disable Streaming

By default, when a user issues a list request, with key data of "*", i.e. listing all records in the database, Portus will send back records in a "streamed" fashion. For example, as soon as one record is retrieved, it is immediately sent back to the client using the HTTP chunking protocol. It has been found that this is the most effective way of handling large amounts of data, but there is a small performance offset in doing this. There are a number of directives that affect how streaming is applied. These directives must be part of the Apache configuration file.

SoaGatewayStreaming On : This is the default setting. Responses will be streamed back to the client using the HTTP Chunking protocol when a list is requested that will retrieve every record in the database.

SoaGatewayStreaming Off : No streaming will ever take place. Use this option if you are concerned about performance, and will be listing every record in the database.

SoaGatewayStreaming Force : Portus will always attempt to stream data back to the client. This is most effective if the SOA Gateway is running on a machine with low resources, and low memory usage is a priority.

■ Change MPM settings

The Portus uses the [Apache worker MPM](#) to handle requests. This can be modified to increase server threads, therefore allowing the server to serve more requests. See the Apache documentation for more information.



Important: Ensure that the ServerLimit of 1 is maintained at all times. Portus will not function correctly if more than server process is started.

6 Internationalization

- Setting codepages 46
- Which codepage do I use? 46
- SOAP versus REST differences 46
- Troubleshooting 47

Portus uses IBM's [International Components for Unicode](#) to support internationalization (i18n). This supports text data conversion between almost any codepage.

Setting codepages

The Single Byte Character Set (SBCS) or Multi Byte Character Set (MBCS) codepage can be set on the driver, for more information, see [here](#). The codepages can also be set on the on the web service itself, by simply left-clicking the web service and entering the codepage in the appropriate section of the web service Properties.



Important: The codepage set on the web service overrides the one set on the driver.

Which codepage do I use?

This depends on what sort of information your service is going to return. Generally the ASCII codepage is sufficient for the English language. The ISO-8859-1 (often called latin1) codepage should suffice for most languages of Western Europe. The windows-1251 codepage supports Cyrillic languages such as Russian and Bulgarian. The ISO-8859-8 codepage can be used for Hebrew script.

The ICU home page has provided a [useful web page](#) which displays the ICU internal name, and a list of the aliases that Portus will recognise. This page will also display the codepage map, which will allow you to choose the codepage best suited to your service.

SOAP versus REST differences

Generally when using WSDL and SOAP, once the correct codepage has been set, the payload should be recognised or returned correctly.

When using REST requests, things are slightly different. Non-ASCII characters entered on a URL bar of a browser will be escaped into their native hex value, of the form %XX. This native hex value differs depending on what codepage the browser recognises the character as. For example, a browser running in the latin1 codepage will recognise Á as %C1, but a browser running in the Cyrillic codepage will recognise Б as %C1.

For this reason Portus allows users to provide an extra field on the REST request. This field is called `__encoding`. Thus users can indicate what codepage their browser is running in.



Important: By default, Portus assumes the escaped values are in the ISO-8859-1. The `__encoding` field is not required in this case.

Example 1

The browser escapes the Russian Б into %C1. You need to tell Portus that this is the Cyrillic encoding.

The URL should be `http://host:port/Service?LIST&key=%C1&__encoding=windows-1251`.

Example 2

The browser escapes the Hebrew Shin (ש) into %F9. You need to tell Portus that this is the Hebrew encoding

The URL should be `http://host:port/Service?LIST&key=%F9&__encoding=iso-8859-8`.

Troubleshooting

When Portus cannot display a character in the requested codepage, it writes a message to the error log, and continues to attempt to process the rest of the payload. If you find your responses are missing some characters, check the `error_log/error.log/XMIDCARD` on *nix, Windows and z/OS respectively.

The error message to check should be something like this :

```
Unicode char 0xF1 is not representable in encoding ASCII.
```


7 Creating a Stylesheet for your Portus Data

- Create HTML page from City XML 50

XSL stands for EXtensible Stylesheet Language, and is a style sheet language for XML documents. XSLT stands for XSL Transformations.

XSLT can be applied to the response payload of Portus REST requests.

It is commonly used, but not restricted, to creating HTML pages based on the REST response.

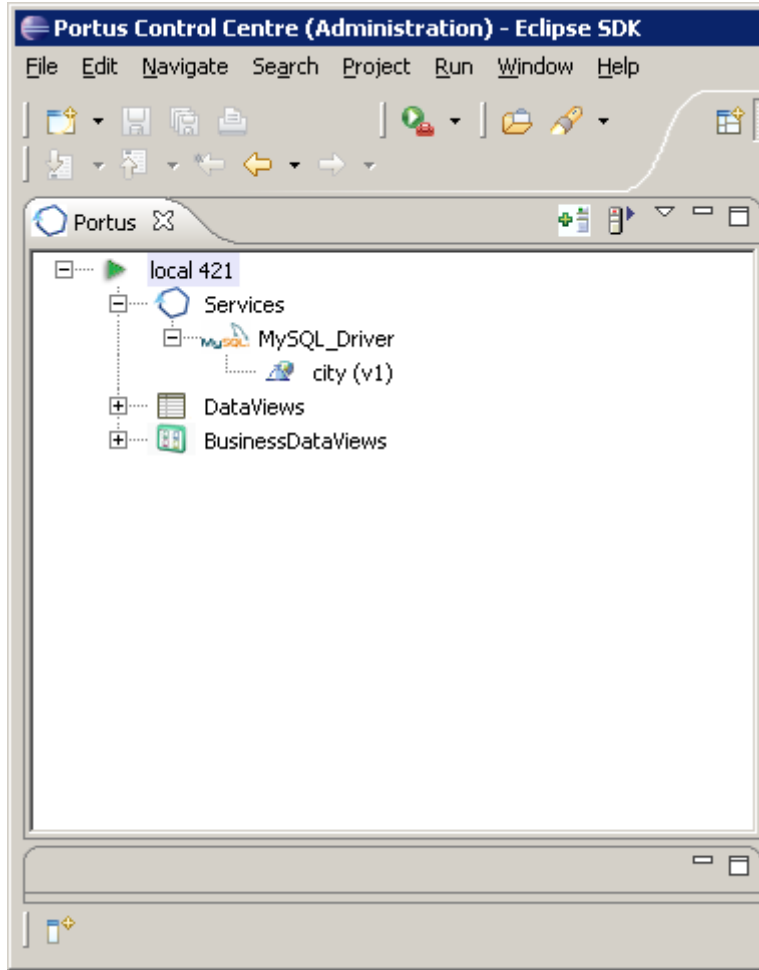
When configured to use XSLT, a Portus service will (by default) return an XML stylesheet processing instruction embedded in the XML. Therefore, the client consuming this data should be able to load the XSL link, and apply the XSLT to the payload. In some cases, the client may not have the ability to understand XML processing instructions (browsers on mobile devices are a good example). In this case, Portus has the ability to apply the XSLT on the server side, and send the transformed results. In this case, providing the `__xslTransform=server` option on the REST request will tell Portus to apply the XSLT before sending the payload, and no XML stylesheet instruction will be included.

E.g `http://host/Service?LIST&ID=*&__xslTransform=server`

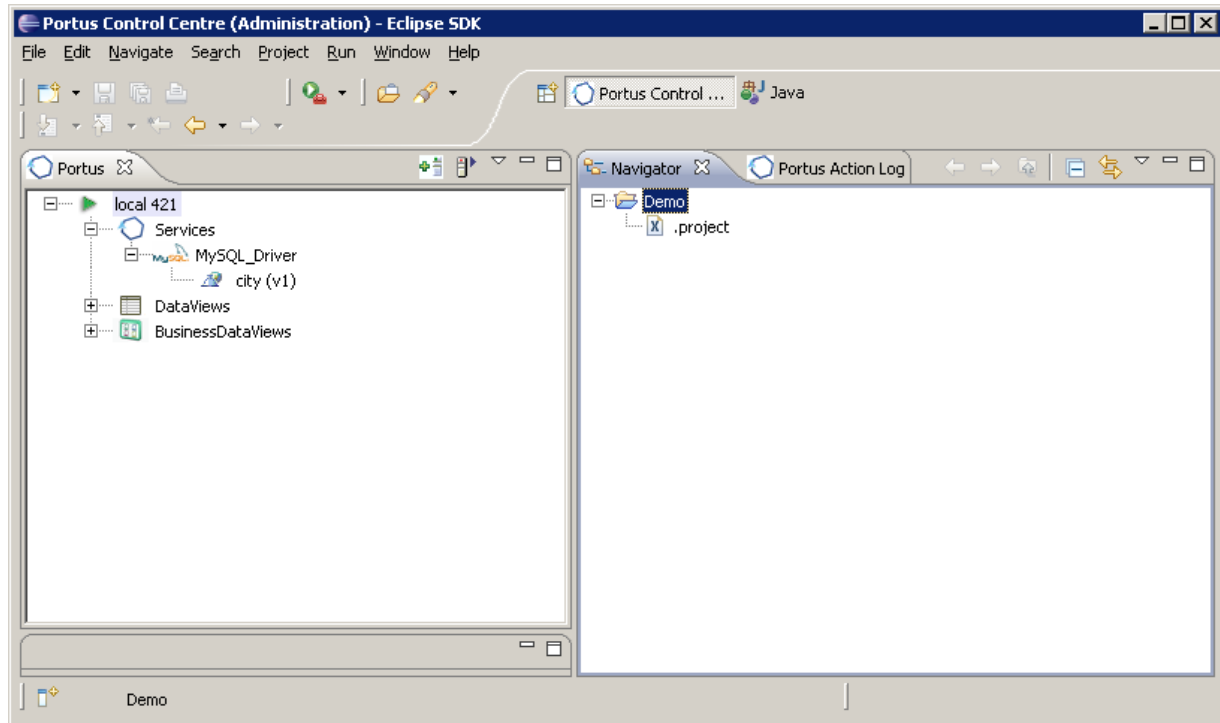
Create HTML page from City XML

- This tutorial will outline how to apply XSLT to a MySQL service. Without any transformation, the service will return raw XML, and we will apply an XSLT to display this in a HTML table. For the purpose of this tutorial we will be using the MySQL World database table, city. See [here](#) for the steps required to set this up. Please ensure that you select the city table when you get to the Discovery step.

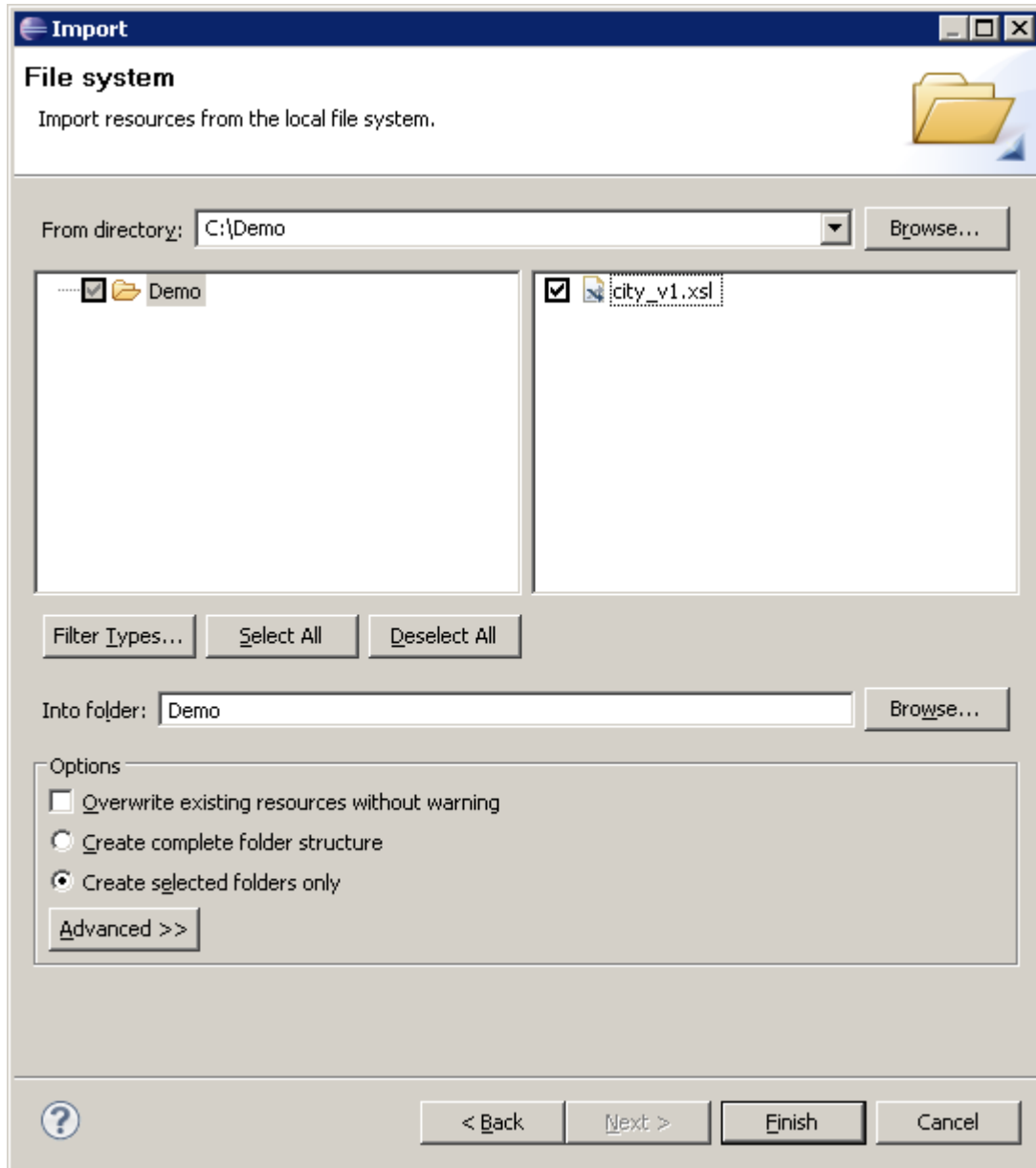
When this step is completed you should have a Service similar to that shown below e.g. a Service capable of accessing the city table.



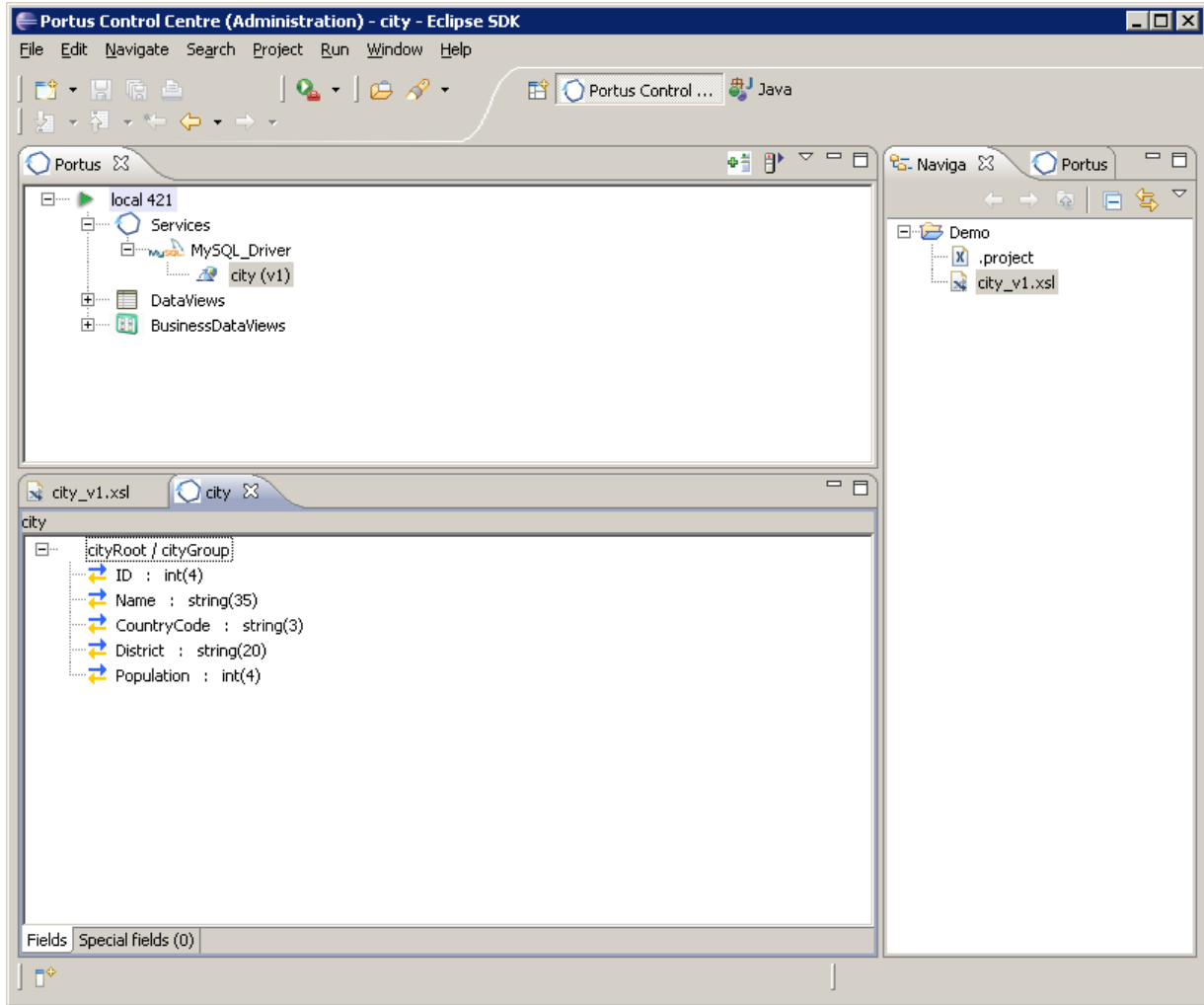
- A generic stylesheet is provided which can be modified to support your Service. Save the following XSL file to disk.
 - ⚠ **Important:** The name of the saved stylesheet has to be in the format 'ServiceName_version.xml'. e.g. we will save our file as city_v1.xml
- If you do not already have a project created see here on how to do so. In our case we have created a project named Demo. If this is not visible open Window -> Show View -> Navigator You should have a view similar to this:



- Right-click your project in the Navigator View and select **Import**
- Expand **General** and select **File System**. Click **Next**
- Click **Browse** and select the directory where you saved the above XSL. Check the XSL, and click **Finish** e.g.



- Double-click your XSL file to open it for editing in a default editor or right-click on the file and select Open With... to choose your own editor.
- Right-click on your Service in the Servers View and select Edit DataView.
- You should now have a view similar to this:



- The DataView provides us with 3 important items which are required when we come to edit the stylesheet:
 1. The ROOT name: e.g. cityRoot.
 2. The GROUP name e.g. cityGroup.
 3. The element names e.g. ID, Name, CountryCode, District and Population.
- Select the XSL tab to edit its contents.
 1. Find the entry `<xsl:template match="changeThisRoot">` and modify changeThisRoot to your root name.
 2. Find the entry `<xsl:template match="changeThisGroup">` and modify changeThisGroup to your group name.
 3. Find the entry ColumnHeader1. It should be wrapped in TR tags as follows:

`<TR>`

```
<Td><font color="#3333FF">ColumnHeader1</font></Td>
</TR>
```

Create as many Td tags entries within the TR tag as there are elements in your DataView e.g. in this case 5.

Change each ColumnHeader1 value to the element names. For headers these do not have to match the element names but for simplicity we will do so here e.g.

```
<TR>
<Td><font color="#3333FF">ID</font></Td>
<Td><font color="#3333FF">Name</font></Td>
<Td><font color="#3333FF">CountryCode</font></Td>
<Td><font color="#3333FF">District</font></Td>
<Td><font color="#3333FF">Population</font></Td>
</TR>
```

4. Find the entry XRDElementName1. It should be wrapped in tr tags as follows:

```
<tr>
<td><xsl:value-of select="XRDElementName1" /></td>
</tr>
```

Create as many td tags entries within the tr tag as there are elements in your DataView e.g. in this case 5.

Change each XRDElementName1 value to the element names.

 **Important:** These must match the element names exactly e.g.

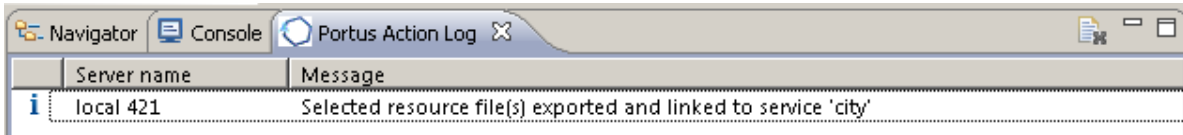
```
<tr>
<td><xsl:value-of select="ID" /></td>
<td><xsl:value-of select="Name" /></td>
<td><xsl:value-of select="CountryCode" /></td>
<td><xsl:value-of select="District" /></td>
<td><xsl:value-of select="Population" /></td>
```

</tr>

5. Save the XSL (Ctrl+S).
6. In the Navigator View left-click the XSL file and, holding down the left button, drag and drop the file onto the Service as shown:.



7. You should see a message similar to this in the Portus Action Log:.



8. Select the Service in the Servers View. If the Properties View is not open select Window -> Show View -> Properties.

Select the 'WSDL URL is ...' entry in the Properties View which should open a browser window.

In the browser window change ?WSDL to ?LIST&CountryCode=BO* e.g. http://localhost:56005/city?LIST&CountryCode=BO* and hit enter.

The results should be displayed as follows:



Data

ID	Name	CountryCode	District	Population
193	Santa Cruz de la Sierra	BOL	Santa Cruz	935361
194	La Paz	BOL	La Paz	758141
195	El Alto	BOL	La Paz	534466
196	Cochabamba	BOL	Cochabamba	482800
197	Oruro	BOL	Oruro	223553
198	Sucre	BOL	Chuquisaca	178426
199	Potosí	BOL	Potosí	140642
200	Tarija	BOL	Tarija	125255

Copyright [Ostia Solutions](#)



9. Congratulations! You have now created a stylesheet for your Service.

8 Language Structure Support

▪ The Portus Representation of Data	60
▪ 'Tuple' Based Databases	60
▪ Record Based Databases	61
▪ Calling Application Programs	61
▪ Representing Individual Fields	62
▪ Representing Structures	63
▪ Representing Arrays	65
▪ Representing Redefines	67
▪ Summary	70

Portus currently supports the calling of C functions, COBOL subroutines and Natural subprograms. In the future, it will also support PL1 and potentially assembler in a similar way. All of these languages have the following concepts

- Simple or base types to represent data
- Structures, which are chunks of memory split up into individual fields with simple/base types
- Arrays which are two or more instances of a structure or simple type.
- A field area in the language which is mapped over by a language structure. In COBOL or Natural terms, this is called a redefinition of the field while in C it is represented by a union or could potentially be done with pointer arithmetic.

This section of the documentation will describe how such constructs are treated by Portus.

The Portus Representation of Data

Portus externally knows everything as a single node within an XML document. XML documents can contain structures of XML nodes, arrays of XML nodes or arrays of structures so it can represent any data structure that is required. Different languages and databases have different ways of representing data and it is necessary within Portus to enable the mapping of these external fields into the type of structures that are required internally by the various drivers supported by Portus. In particular, if the Portus representation of the data is modified in a way that is incompatible with how an application program expects to be called, at best, incorrect results will be returned but more than likely the stability of Portus itself will be impacted due to program abends.

'Tuple' Based Databases

A 'tuple' based database is simply a database technology for which the data is presented in name/value pairs or 'tuples'. ADABAS is a tuple based database as when requests are being provided to ADABAS, or data is being returned by ADABAS, it is returned as a field name/Value pair which can easily be translated from and to the external XML structures.

Relational databases such as DB2 also work on the principle of a tuple. The data is provided using column names along with the data associated with the column. Again, this is easily mapped to the external XML field structure and does not need further explanation.

Record Based Databases

Databases such as VSAM and IMS/DB present their data in the form of records. A record is simply a single contiguous piece of storage that contains the data related to the request. While it is held contiguously internally in the database, it will almost always be made up of multiple individual fields. The individual fields within the record are then addressed by offset, length and type. While they could be addressed in this most simple way, most if not all organizations will have a language definition of the individual fields in the form of a structure. When referencing fields in the structure, the language implementation can then calculate the offset and length of each field referenced and then process it base on its type. Portus uses these language definitions to create definitions for record based databases.

Calling Application Programs

When calling an application program, generally there will be one or more parameters to the program. Note that while the application may treat these as input or output parameters, the application must receive all of the parameters during the call.

To be clear on this point, Portus has the concept of input parameters, output parameters, input-output parameters and parameters which are neither input nor output in its input and output messages for a service. This relates purely to the input/output messages that Portus will build for the service definition. The parameters `_Must_` be passed in their entirety to the application being called regardless of their direction definition to Portus.

Depending on the programming language, parameters can be provided as:

1. A list of individual fields (the most common way for C)
2. A single structure containing all of the input/output information (mostly used in CICS)
3. A list of fields or structures (used by COBOL and Natural most commonly)

Essentially it can be stated that an application program will be called with one or more parameters and each parameter may be a single field, a structure or an array. Within a structure we could have other structures or arrays while it is also possible to have arrays of structures.

While all this sounds quite complicated and involved, when it's broken down it is not as complicated as it may seem. Portus has the concept of a level 1 field name which may itself be a field or the name of a structure. There may be multiple levels below this level 1 field but the number level 1 fields will dictate how many parameters are passed to the applications.

Representing Individual Fields

The following are equivalent parameter definitions in various languages:

NATURAL

```
PARAMETER
 1 OPERATION (A3)
 1 OPERAND_1 (I4)
 1 OPERAND_2 (I4)
 1 RESULT   (I4) BY VALUE RESULT
END-DEFINE
```

COBOL

```
linkage section.
01      Operation      PIC X(3).
01      Operand1       PIC S9(9) COMP-4.
01      Operand2       PIC S9(9) COMP-4.
01      Result         PIC S9(9) COMP-4.
procedure division USING Operation Operand1 Operand2 Result.
```

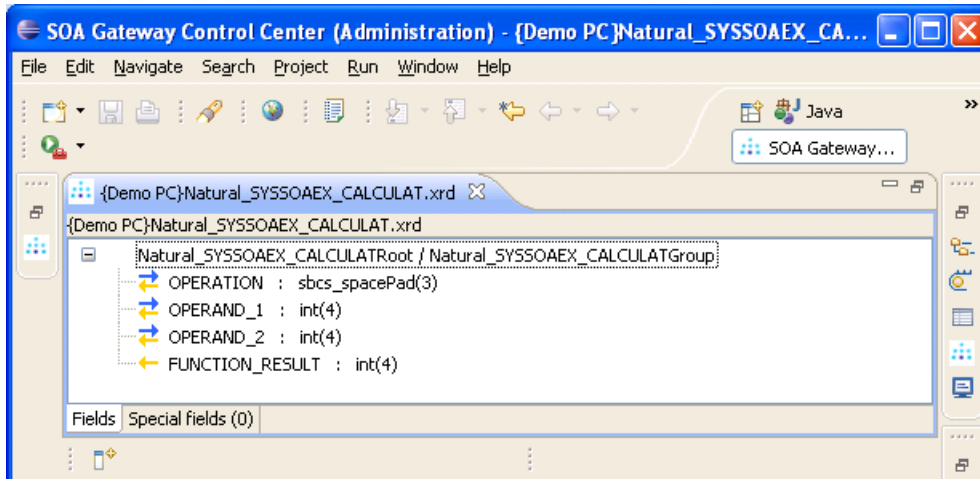
C

```
int calc( char operation[3], int *operand1, int *operand2, int *result )
```

Each of the above represents in their respective languages:

- A 3 character field that will contain an operation code.
- A 4 byte binary field that will contain the first operand for the calculation
- A 4 byte binary field that will contain the second operand for the calculation
- A 4 byte binary field that will contain the result of the calculation

The screenshot below illustrates the Portus representation created based on the Natural PDA area, however, this could also be used for any of the other languages:



Some points to note about this.

- - The external names have no significance in terms of the internal call to the application code.
- - You will note that OPERATION, OPERAND_1 and OPERAND_2 are input/output fields while FUNCTION_RESULT is output only. Again, this will have no significance internally as all fields must be passed to the application program as it expects this.
- - If you do not wish a field to be included in the input or output messages, it must be set with a direction of 'none'. It *must not* be deleted as if it is deleted, Portus will end up passing an incorrect set of parameters to the program being called.

The principle of not deleting anything in the Portus representations of this data is critical for the consistent and stable running of the SOA Gateway. This is because Portus representation must reflect what an application program expects to receive as parameters. Any removal or changing of the order here will cause problems because it will result in a different representation being provided to the application program.

Representing Structures

In most cases, far more data must be passed to or returned from an application than will fit in a single field. For this reason, structures are generally used to pass data backwards and forwards between applications.

The following show how individual fields along with a simple structure are represented in various languages:

NATURAL

```
PARAMETER
1 INITIAL (P7)
1 I_RATE (P2.2)
1 YEARS (I2)
1 RESULT
  2 YEAR (I2) BY VALUE RESULT
  2 SIMPLE (A17) BY VALUE RESULT
  2 COMPOUND (A17) BY VALUE RESULT
END-DEFINE
```

COBOL

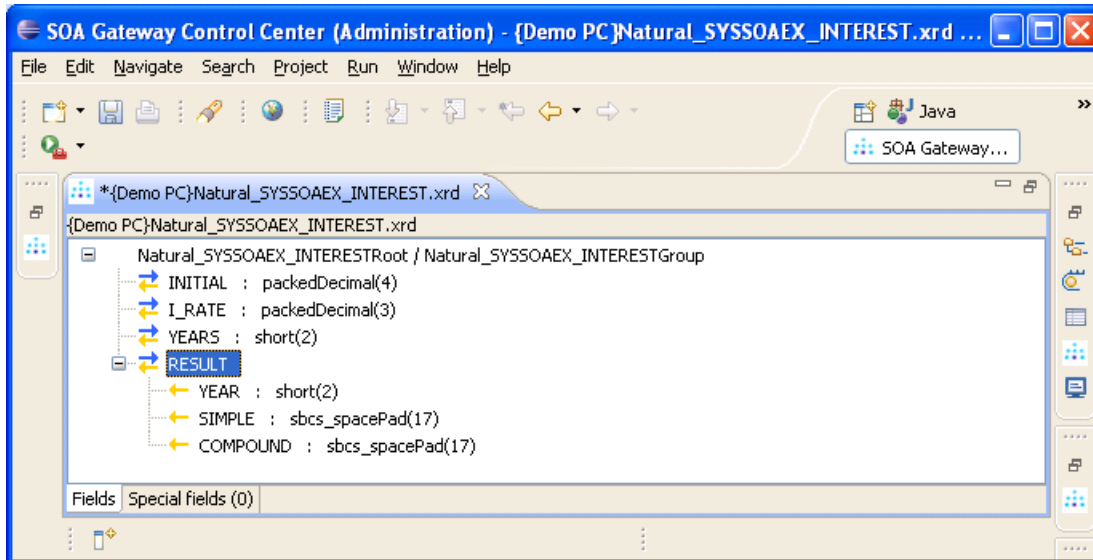
```
LINKAGE SECTION.
01 INITIAL-AMOUNT          PIC S9(7) COMP-3.
01 I-RATE                  PIC S99V99 COMP-3.
01 YEARS                   PIC S9(4) COMP.
01 RESULT.
  02 RESULT-TABLE.
    03 YEAR                 PIC S9(5) COMP .
    03 SIMPLE               PIC ZZ,ZZZ,ZZZ,ZZ9.99 DISPLAY .
    03 COMPOUND             PIC ZZ,ZZZ,ZZZ,ZZ9.99 DISPLAY .
*
PROCEDURE DIVISION USING INITIAL-AMOUNT I-RATE YEARS RESULT .
```

C

```
typedef struct {
  short year;
  char simple[17];
  char compound[17];
} result_h ;

int interest ( int *INITIAL, int *I_RATE, int *YEARS, struct result_h *result )
```

The equivalent representation of this parameter list in Portus is as follows:



It will be noted that the INITIAL, I_RATE, YEARS and RESULT fields are at level 1 while the elements of the RESULT structure YEAR, SIMPLE and COMPOUND are at level 2. This will result in 4 parameters being passed to the application code with the 4th parameter being a structure. The following should be noted:

- As with the previous example, if any of the fields in the structure are removed, it renders the structure invalid unless the application program is changed too. If an element or elements of a structure are not to appear in the output or input messages, they should be given a direction of 'none' so that they are still in the structure passed to the application but will not be seen in the service definition.
- The format of the fields also cannot be changed as this could impact on their length and thus pass what is not expected to the application program.

Representing Arrays

Arrays are a very common way of providing a lot of the same information or returning lists from application programs. The following example builds on the previous example to return an array of 50 structures. To return simply an array of values, the 'structure' would simply contain one element.

NATURAL

```
PARAMETER
1 INITIAL (P7)
1 I_RATE (P2.2)
1 YEARS (I2)
1 RESULT (1:50)
  2 YEAR (I2) BY VALUE RESULT
  2 SIMPLE (A17) BY VALUE RESULT
  2 COMPOUND (A17) BY VALUE RESULT
END-DEFINE
```

COBOL

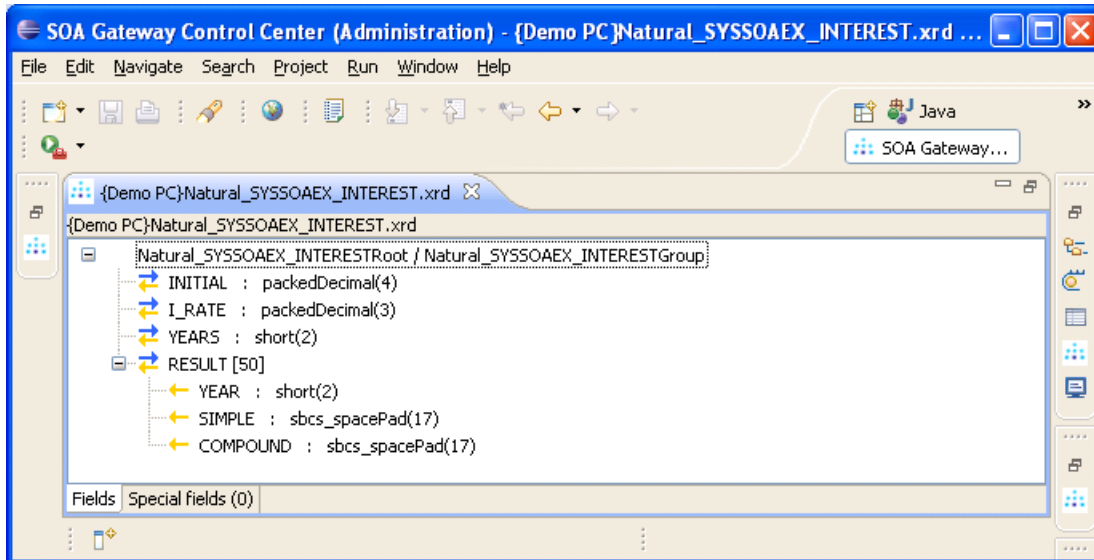
```
LINKAGE SECTION.
01 INITIAL-AMOUNT          PIC S9(7) COMP-3.
01 I-RATE                  PIC S99V99 COMP-3.
01 YEARS                   PIC S9(4) COMP.
01 RESULT.
  02 RESULT-TABLE          OCCURS 50.
    03 YEAR                PIC S9(5) COMP .
    03 SIMPLE              PIC ZZ,ZZZ,ZZZ,ZZ9.99 DISPLAY .
    03 COMPOUND            PIC ZZ,ZZZ,ZZZ,ZZ9.99 DISPLAY .
*
PROCEDURE DIVISION USING INITIAL-AMOUNT I-RATE YEARS RESULT .
```

C

```
typedef struct {
    short year;
    char simple[17];
    char compound[17];
} result_h ;

int interest ( int *INITIAL, int *I_RATE, int *YEARS, struct result_h *result[50] )
```

The above is represented in Portus in the following way:



Again, the application program is expecting a 4th parameter with 50 instances of the following array above. Removing a field from the structure or changing the number of array occurrences without changing the application will result in invalid results and potentially will destabilise the system. Once again if the field should not appear in the input or output messages for the service, simply set the field's direction to 'none' and leave the structure intact.

Representing Redefines

Redefines ultimately involve the mapping of a base field to one or more different layouts. The base field in this case is the original field upon which the redefine(s) are based. Redefines are used for a number of reasons:

- It may be that the format of an area is different depending on some value passed as a parameter or perhaps earlier in the area itself.
- Some programmers like to define a large base area and redefine it so that fields may be added at a later date without changing the memory profile of the application. It should be noted here that a redefinition of a base area may have a smaller length than the base area but may never be larger than the base area as otherwise, storage overwrites will be the results.

Ultimately it could be said that a redefine will generally involve mapping a structure on to a flat piece of storage. The reason this is relevant to Portus is that in most cases for a service, the input and output messages will be created from the redefinition (i.e. the discrete fields), however, Portus must create the base field based on the structure and pass the base field to the application program.

The following shows a simple redefine in multiple languages as before. Note in the following examples:

- The base field is 200 bytes long.
- The total of the redefined structure is 78 bytes long

NATURAL

```
PARAMETER
 1 BASE-FIELD (B200)
 1 REDEFINE BASE-FIELD
 2 FIELD1 (A20)
 2 FIELD2 (B4)
 2 FIELD3 (I4)
 2 FIELD4 (B50)
END-DEFINE
```

COBOL

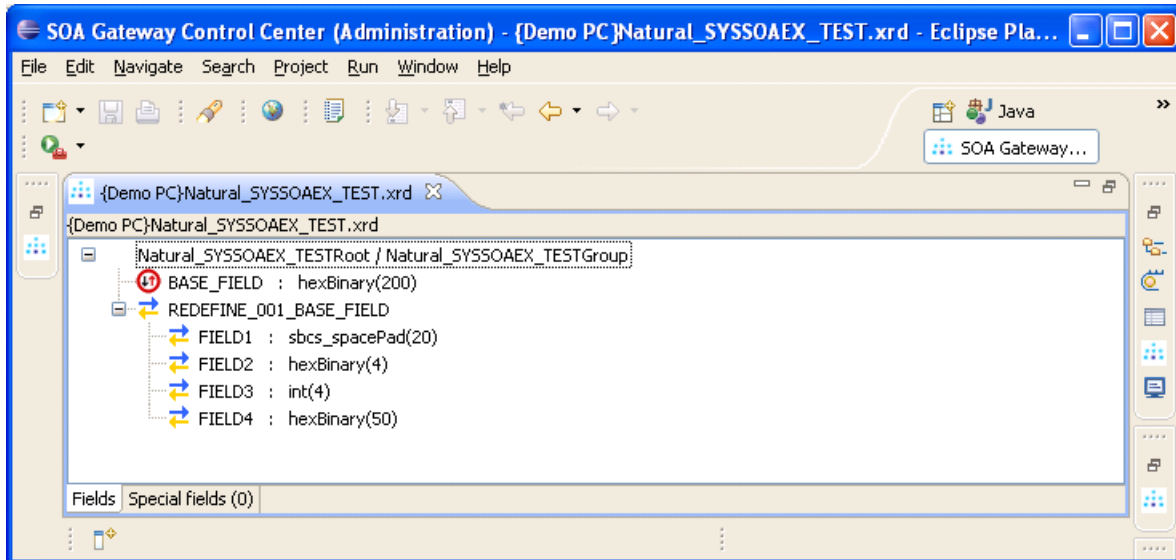
```
LINKAGE SECTION.
01 BASE-FIELD PIC X(200).
01 FILLER REDEFINES BASE-FIELD.
   02 FIELD1 PIC X(20).
   02 FIELD2 PIC 9(4) BINARY.
   02 FIELD3 PIC 9(8) COMPUTATIONAL.
   02 FIELD4 PIC 9(50) BINARY.
```

C

```
union {
    unsigned char base_field[200];

    struct {
        char          field1[20];
        unsigned char field2[4];
        int           field3;
        unsigned char field4[50];
    } redef_base;
} base;
```

This is represented in Portus as follows:



Some notes about this:

- You will notice that by default, the base field is disabled. In order for Portus to know which version of the field to expose, one or the other must be disabled. If you wish to expose the base field in the input/output messages, then enable the field 'BASE_FIELD' and the field 'REDEFINE_001_BASE_FIELD' will be automatically disabled.
- Note that then a structure is disabled, only the base of the structure is disabled in order to maintain the direction settings for the fields in the event that the structure is enabled again.
- Never remove/delete a field from a redefinition as the presence of fields illustrates to Portus how to build the base field expected by the application. Change its direction to 'none' if you do not wish to see it in an input or output message for the service.
- The base field, even though disabled, must never be removed. This is the actual field type the application is expecting with language constructs enabling it to process the data based on the redefinition.
- In essence, a redefine is similar to a structure and from an external point of view with input and output messages will appear the same way, however, Portus must know the base field to correctly call the application program.

Summary

When dealing with Portus views/XRDs created based on application program parameter lists or structures, follow these simple rules:

- Never delete fields from the generated Portus view. If you do not wish to make the fields available as input and/or output messages, set their direction as 'none'. You may also elect to set a fixed value for such fields to force specific behaviour in an application.
- Always ensure that with redefinitions, only the base *OR* the redefined fields are enabled; never both.
- If an application program changes, Ostia recommend that you recreate the view of the application using Portus Control Centre wizard rather than attempting to modify an existing XRD.

9 Data Masking

- Example 72

Data masking is the process of obscuring (masking) specific data elements within data stores. It ensures that sensitive data is replaced with realistic but not real data. The goal is that sensitive customer information is not available outside of the authorized environment. Data masking is typically done while provisioning non-production environments so that copies created to support test and development processes are not exposing sensitive information and thus avoiding risks of leaking.

Portus provides functionality which allows you to select fields whose content you do not wish to divulge. This masking takes place as the data flows through Portus, therefore is masked "in-flight". The data sitting on the back-end database is not changed in any way. Almost 40 different algorithms are provided which you can use to maintain data integrity. The algorithms include generate a random email address, create a hash value, generate random credit card numbers, random dates and times, and look up a list of seed values in an external database table.

This allows users to query your data secure in the knowledge that its real content remains unexposed.

Example

Employee Data without masking

```
<EMPLOYEE_ID>100</EMPLOYEE_ID>  
<FIRST_NAME>Steven</FIRST_NAME>  
<LAST_NAME>King</LAST_NAME>  
<EMAIL>SKING@SKING.COM</EMAIL>  
<HIRE_DATE>2003-06-17</HIRE_DATE>  
<SALARY>24000</SALARY>
```

Employee Data with masking

```
<EMPLOYEE_ID>100</EMPLOYEE_ID>  
<FIRST_NAME>UKfaUAKuPFFAuPFap</FIRST_NAME>  
<LAST_NAME>PPupUpaAfUfkpuFkpFkkFfFa</LAST_NAME>  
<EMAIL>VIJAY@SMITHS.COM</EMAIL>  
<HIRE_DATE>2002-08-16</HIRE_DATE>  
<SALARY>823999</SALARY>
```

Algorithms used are random text for FIRST_NAME and LAST_NAME, random email for EMAIL, random date for HIRE_DATE and random number between 0 and 999999 for SALARY.

To find out more about this feature please contact your sales area representative.

