Portrait
Foundation

WCF Service Support - Developers Guide

Edition 1.5

11 January 2013

Pitney Bowes
Software

Portrait Foundation
WCF Service Support - Developers Guide

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc..

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Software Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

This document describes the interface and use of the Portrait Foundation Service Generation Wizard (ServiceMaker), a mechanism for exposing a Portrait system's functionality as a WCF Service.

WCF services are easier to consume and provide more standards based functionality.

Intended audience

Users who need to understand how to expose Portrait functionality as a WCF service.

Users who need to understand how to consume calls to Portrait Foundation generated WCF Services.

Related documents

Request Token Tracing Overview

Software release

Portrait Foundation 4.4 Update 4 or later.

Contents

1 Overview

Portrait Service Support allows you to call Portrait Foundation Process Models through a service interface; typically as a web service. The term 'service' has been used in this document rather than 'web service' because of the possibility of delivering services without traditional web technologies.

This support consists of:

Configuration Suite: The user has the ability to create services and to define their interfaces. The interface to a service comprises any number of methods; each with their own set of inputs and outputs. Each service method is associated with a Foundation Process Model. The user must map each method's inputs and outputs on to the inputs and outputs of its Portrait Foundation Process Model.

Service Maker Wizard: This tool processes the deployed service definitions created above and creates the resources necessary for clients to call the service defined in the Configuration Suite.

Service and Configured Types assemblies: These are generated by the ServiceMaker wizard. They contain the callable service and all of the configured entities in the workspace.

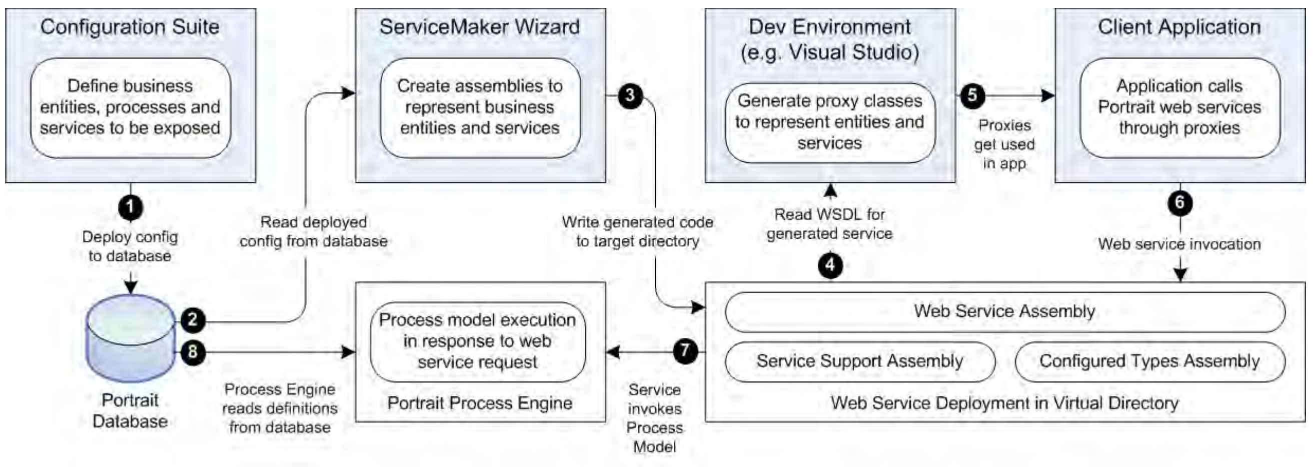
Service Support assembly: The resources created by the ServiceMaker Wizard use a core set of Foundation objects to drive Foundation's Process Engine to execute the service methods. These core services are available to developers to drive the Process Engine from services that have been written from scratch or are extensions to a service created by the generation tool. The assembly is called ServiceCore.

Runtime: Foundation's Process engine supports execution and resumption of Portrait Foundation Process Models driven through a service interface.

Components and process

The following diagram shows the components involved in the full development lifecycle from the Configuration Suite configuration through to execution of the service.

Figure 1 Components and process


Configuration Suite
Define business entities, processes and services to be exposed
ServiceMaker Wizard
Create assemblies to represent business entities and services
Dev Environment (e.g. Visual Studio)
Generate proxy classes to represent entities and services
Client Application
Application calls Portrait web services through proxies
1
Deploy config to database
Read deployed config from database
3
Write generated code to target directory
Read WSDL for generated service
4
5
Proxies get used in app
6
Web service invocation
Portrait Database
2
8
Process Engine reads definitions from database
Process model execution in response to web service request
Portrait Process Engine
7
Service invokes Process Model
Web Service Assembly
Service Support Assembly
Configured Types Assembly
Web Service Deployment in Virtual Directory

1.1 New Features

This version supersedes the Web service support that was originally introduced in version 3.1. This version has the following new features:

- It is based on Microsoft's Windows Communication Foundation (WCF) technology and introduces the ability to leverage support, functionality and configurability of that platform notably its support for the WS-* standards.
- The types generated from the imported WSDL present a significantly more familiar programming model for consumption of the services.
- The generation tool has a Wizard which much simplifies the capture of the dozens of parameters and options necessary for service generation.
- The generation tool optionally loads and saves inputs from a configuration file which simplifies the generation process.
- The generation tool creates an assembly containing all configured types (Data Objects, Parties, and Reference Data etc) from a workspace. This assembly can be used independently from web services, for example in a .Net DAT.
- Polymorphic Portrait Data Objects as parameters are supported. This allows for such things as for interfaces to specify "AnyKindOfParty" as an input and an "Agent" object to be passed in. This overcomes shortcomings in previous version which could not support such behaviour
- The definitions created provide a high degree of type safety both on method calls and on Portrait Data Objects.
- The definitions that client code will use is designed to be 'intellisense-friendly' .
- All parameter types are passed as objects; there is no need to use raw XML representations.
- To speed up the development cycle, the generation tool can generate an instantly usable virtual directory representing the service.

1.2 Conversation support

While some process models always run to completion once invoked, many process models are designed to operate in a 'conversational' way, where the model suspends its operation and requests input from its client. When used as part of an end-user application, such as the Contact Centre, the user is prompted to enter data through a Custom Interaction (CI) or a Generated Interaction (GI). Once the data is available, the client resumes the model and continues to process it. Some models may suspend and resume many times before they complete.

It is possible to call models that suspend and to resume them using Portrait Services Support.

Models that complete provide outputs that are mapped at configuration time onto the invoked method's return values. Models that suspend require the client to have some prior knowledge of what will be returned and what data is necessary for resumption.

1.3 Session management

Portrait Service Support uses Activity Tokens for session management, like other Portrait Foundation channels.

Portrait Foundation maintains state information keyed by the Activity Token. Any calls to services that have the same Activity Token have access to the same state data. These calls are effectively a 'session' bound together by the Activity Token.

Clients of a service may either use:

- a single Activity Token across multiple interactions - all requests are considered part of the same session
- one Activity Token per interaction - each request is considered to be a 'single shot'.

The same Activity Token must be used each time the model is resumed to drive the conversation.

If the method call does not contain an Activity Token, one is created and returned.

1.4 Request tokens

Every request that arrives at the Portrait Foundation web tier is allocated a Request Token that is passed through the system and may be logged along with performance data at various levels within the system.

This Request Token takes the form of a GUID.

Request Tokens are distinct from Activity Tokens.

Normally this default behaviour is what you require. However, if you have a specific Request Token that you would like to associate with a request instead of allowing Portrait Foundation to generate one internally, you may pass this in the RequestToken member of the State object. The State object is the first parameter on all Portrait Service Support method calls.

You might for example wish to do this if the call you are processing originated within Portrait Foundation itself, and you wished to associate it with the same Request Token as the original call.

When the service request completes, the State object contains the value of the Request Token that was actually used for this request. So, if you don't pass one in it will contain the Request Token that was generated in the web tier.

Similarly, the State object has a member ClientTag which may be used to pass in a Client Tag to associate with this request. Client Tags are pieces of text that identify what the client was doing in order to initiate this request, and are logged along with Request Token performance data. If you do not pass a Client Tag in, a simple one is generated internally and returned in the State Object.

2 ServiceMaker tool

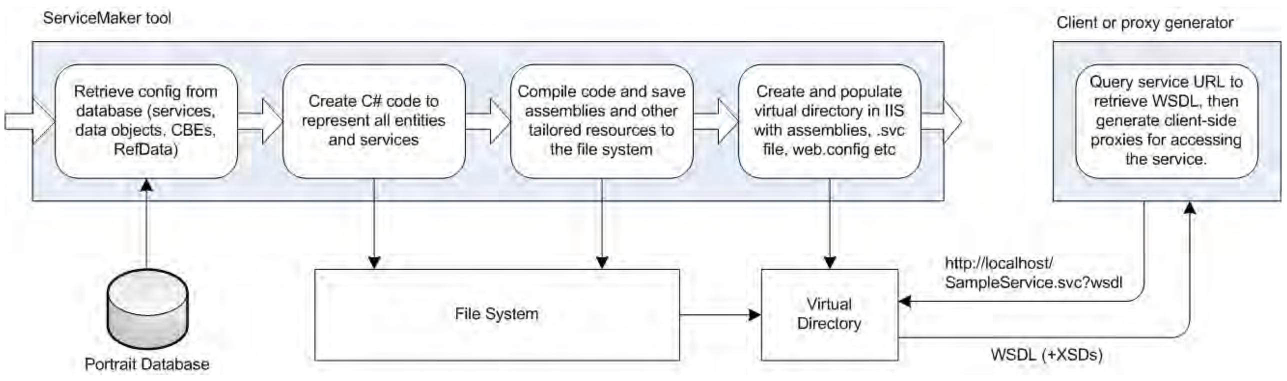
ServiceMaker is the tool used to process a deployed service definition into the resources necessary to execute it.

The tool can be run as a command line utility or via its Wizard style interface. The command line and Wizard interfaces capture the same parameters and produce the same outputs.

Generation Process

After creating and deploying the service definition and resources to the Portrait Foundation database; the process of turning this into a form capable of being programmed against is depicted below.

Figure 2 - Service generation process


ServiceMaker tool
Retrieve config from database (services, data objects, CBEs, RefData)
Create C# code to represent all entities and services
Compile code and save assemblies and other tailored resources to the file system
Create and populate virtual directory in IIS with assemblies, .svc file, web.config etc
Client or proxy generator
Query service URL to retrieve WSDL, then generate client-side proxies for accessing the service.
Portrait Database
File System
Virtual Directory
http://localhost/SampleService.svc?wsdl
WSDL (+XSDs)

2.1 Inputs

In general terms the inputs are:

- the details necessary to read the deployed service and configured types from a Portrait Foundation database.
- where the outputs are to be written
- whether to create a virtual directory
- whether to create Visual Studio 2008 Projects and source that can be used to alter and rebuild the generated services.
- whether to save or load settings from a file

2.2 Outputs

In general terms the outputs are:

- the contents of a virtual directory tailored for the service
- optionally, an IIS virtual directory defined to use the above contents
- source file and projects that represent the generated binaries.
- an assembly representing the configured service
- an assembly representing all of the types configured in the workspace

2.3 Invocation

The invocation methods are described below:

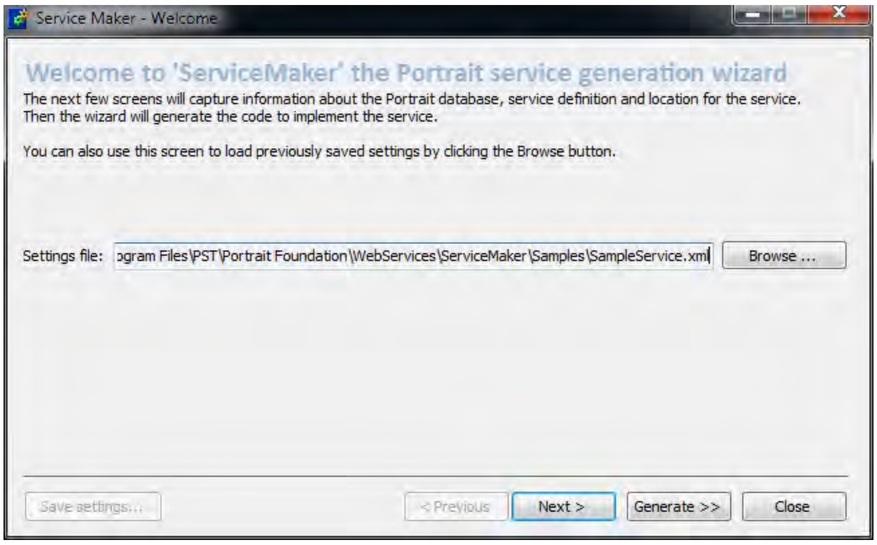
Wizard

The wizard screens appear if the **/UI** flag is not set or it is set to true. The following screens represent the provided SampleService configuration file being used by the invocation:

```
ServiceMaker /System Services /ConfigFilename “SampleService.xml”
```

Each page is listed below along with the ServiceMaker options they capture. These options are expanded in section 2.4 below.

Welcome page



Welcome to 'ServiceMaker' the Portrait service generation wizard

The next few screens will capture information about the Portrait database, service definition and location for the service. Then the wizard will generate the code to implement the service.

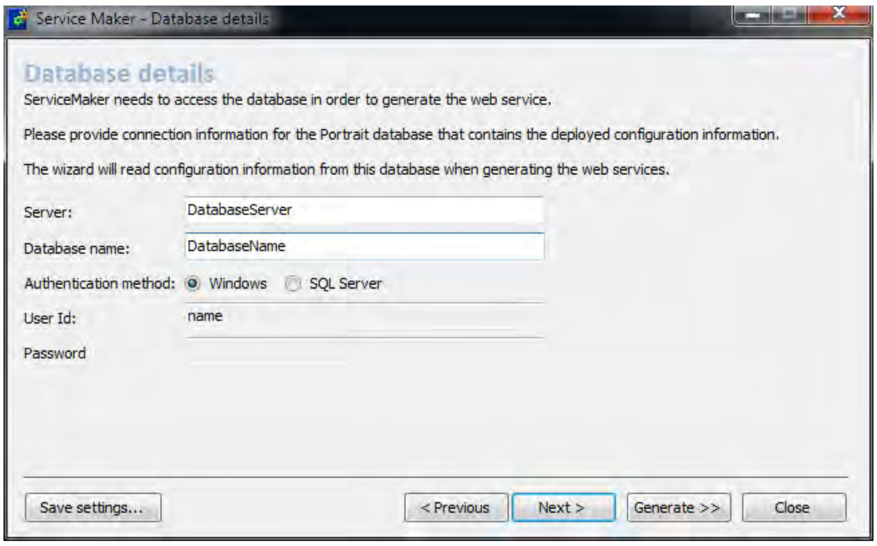
You can also use this screen to load previously saved settings by clicking the Browse button.

Settings file: ogram Files\PST\Portrait Foundation\WebServices\ServiceMaker\Samples\SampleService.xml

This page captures the name of ServiceMaker configuration file to read a starting set of values from.

This screen captures the equivalent of the command line option /ConfigFilename

Database details page



Database details

ServiceMaker needs to access the database in order to generate the web service.

Please provide connection information for the Portrait database that contains the deployed configuration information.

The wizard will read configuration information from this database when generating the web services.

Server: DatabaseServer

Database name: DatabaseName

Authentication method: [✓ Windows] SQL Server

User Id: name

Password

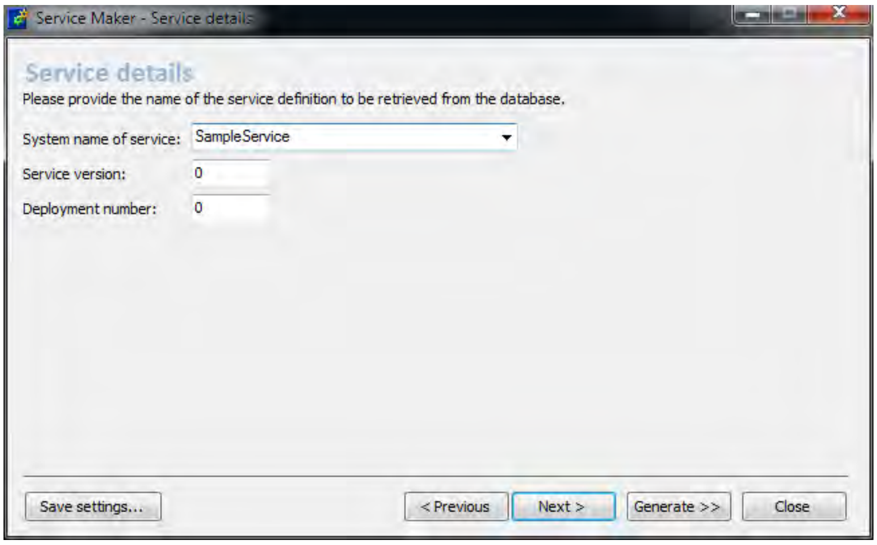
This page captures the details necessary to read the database.

This screen captures the equivalent of the command line options:

- /DatabaseServer
- /DatabaseName
- /DatabaseAuthentication
- /DatabaseSqlServerUserid

- /DatabaseSqlServerPassword

Service details page



Service Maker - Service details

Service details

Please provide the name of the service definition to be retrieved from the database.

System name of service: SampleService

Service version: 0

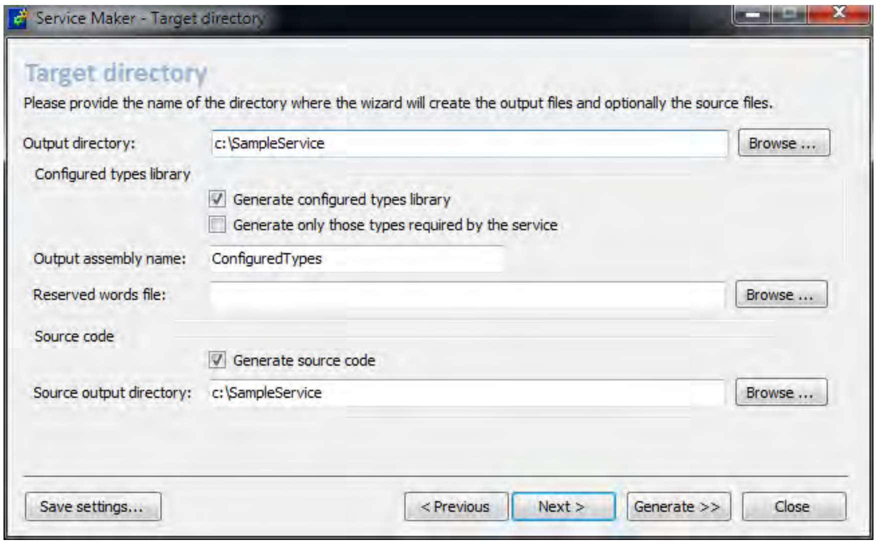
Deployment number: 0

This page captures the details of the service to generate.

This screen captures the equivalent of the command line options:

- /ServiceName
- /ServiceVersion
- /DeploymentNumber

Output details page



Service Maker - Target directory

Target directory

Please provide the name of the directory where the wizard will create the output files and optionally the source files.

Output directory: c:\SampleService

Configured types library

[✓] Generate configured types library

[ ] Generate only those types required by the service

Output assembly name: ConfiguredTypes

Reserved words file:

Source code

[✓] Generate source code

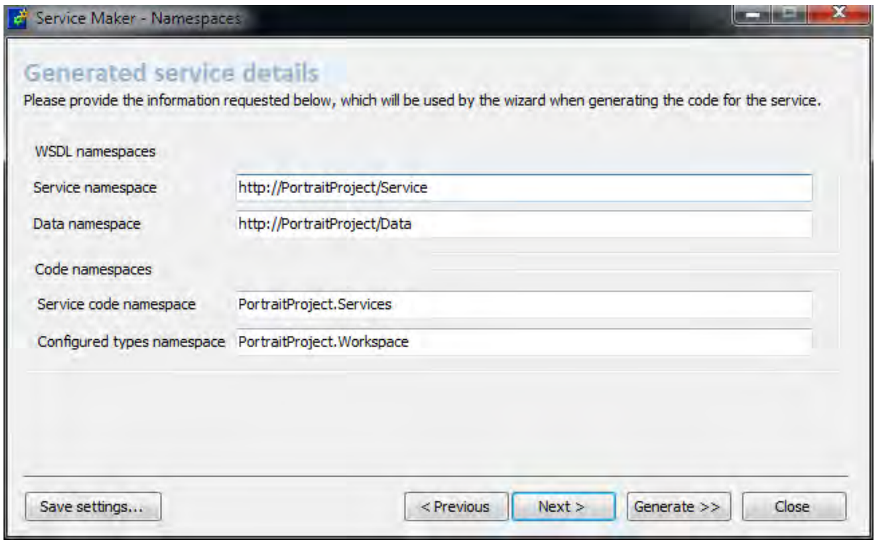
Source output directory: c:\SampleService

This page captures the details of resources to create.

This screen captures the equivalent of the command line options:

- /OutputDirectory
- /GenerateConfiguredTypes
- /GenerateReferencedTypesOnly
- /ConfiguredTypes
- /ReservedWordsFile
- /SourceOutputDirectory

Generated service details page



Please provide the information requested below, which will be used by the wizard when generating the code for the service.

WSDL namespaces

Service namespace	http://PortraitProject/Service
Data namespace	http://PortraitProject/Data

Code namespaces

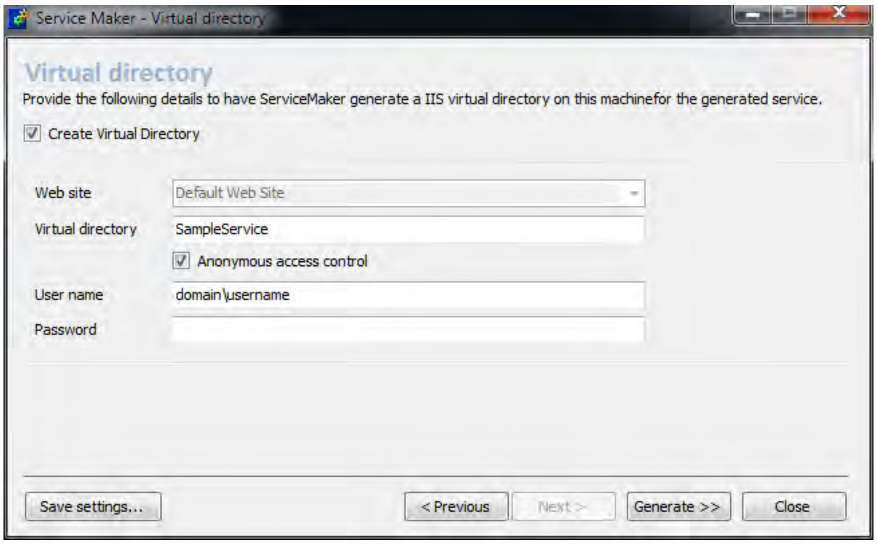
Service code namespace	PortraitProject.Services
Configured types namespace	PortraitProject.Workspace

This page captures the details of namespaces to use.

This screen captures the equivalent of the command line options:

- /ServiceContractNamespace
- /DataContractNamespace
- /ServiceCodeNamespace
- /TypesCodeNamespace

Hosting details page



Provide the following details to have ServiceMaker generate a IIS virtual directory on this machinefor the generated service.

☑ Create Virtual Directory

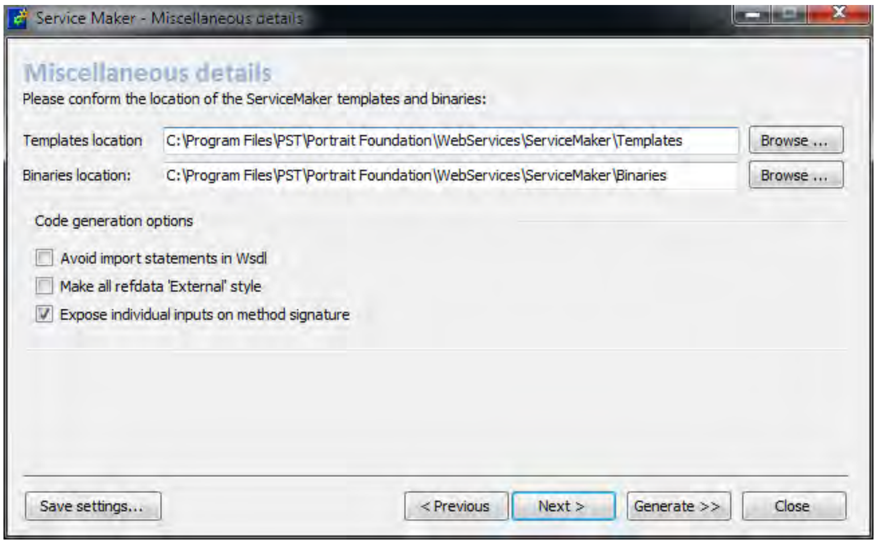
Web site	Default Web Site
Virtual directory	SampleService
	☑ Anonymous access control
User name	domain\username
Password	

This page captures the details of the hosting of the service.

This screen captures the equivalent of the command line options:

- /VdirCreateVdir
- /VdirName
- /VdirAnonymousAccess
- /VdirUser
- /VdirPassword

Miscellaneous details page


Service Maker - Miscellaneous details

Miscellaneous details

Please conform the location of the ServiceMaker templates and binaries:

Templates location: C:\Program Files\PST\Portrait Foundation\WebServices\ServiceMaker\Templates

Binaries location: C:\Program Files\PST\Portrait Foundation\WebServices\ServiceMaker\Binaries

Code generation options

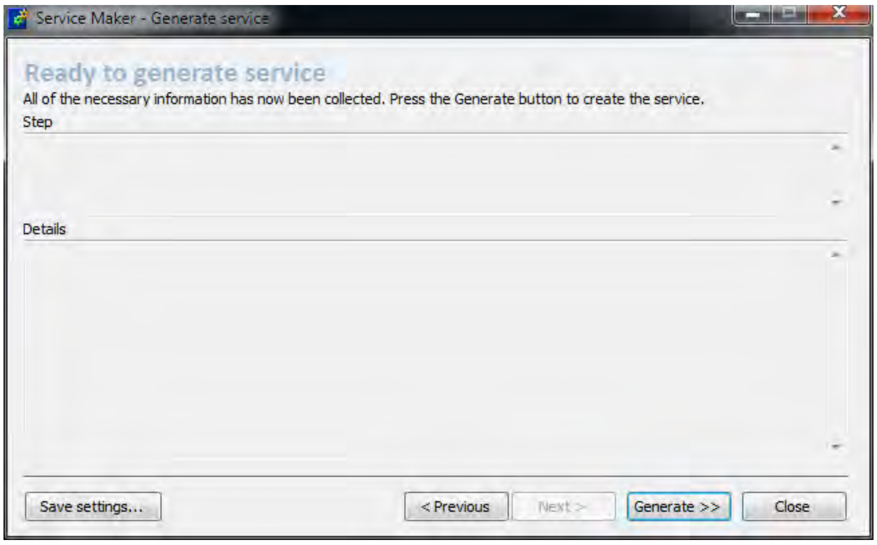
- [ ] Avoid import statements in Wsdl
- [ ] Make all refdata 'External' style
- [✓] Expose individual inputs on method signature

This page captures miscellaneous details.

This screen captures the equivalent of the command line options:

- /TemplateDirectory
- /BinariesDirectory
- /FlatWsdl
- /AllRefdataIsDynamic
- /InputsOnMethodSignature

Service Generation page


Service Maker - Generate service

Ready to generate service

All of the necessary information has now been collected. Press the Generate button to create the service.

Step

Details

This page initiates the generation process. Details of the progress are displayed in the Step and Detail controls. Once the generation process has been initiated it can be cancelled.

Note that validation of parameters is not performed until the Generate button is selected.

The Close button will provide the option of saving the currently selected options as a configuration file.

Command line invocation

The following example shows the use of the command line syntax. The invocation includes use of a configuration file. This is not mandatory; all of the values expressed in the configuration file could have been expressed on the command line.

```
ServiceMaker /System Services /UI- /ConfigFilename "SampleService.xml"
```

Which is equivalent to:

```
ServiceMaker /sy Services /UI- /cf "SampleService.xml"
```

2.4 Options

The options used for the generation of the service are captured from 3 sources:

- The Wizard captured value
- The command line
- The values in the named configuration file

Where values are specified in more than one place; the sources have precedence in the order shown above e.g. a command line switch will override anything found in a configuration file. A value entered into the Wizard will override a value found on the command line or a value found in the configuration file.

The following is a list of the command line options for ServiceMaker, for each option there is a short alias that can be used for brevity:

Option	Parameter	Description	Alias
/ConfigFilename	file	Names the file to read parameters from	cf
/System	name	Provides the name of the System that we are running in	sy
/DatabaseServer	name	Name of the SQL server instance to read the deployed service from.	ds
/DatabaseName	name	Name of the database in the SQL server instance that contains the deployed service	dn
/DatabaseAuthentication	SQL/Windows	The type of authentication for the database server	da
/DatabaseSqlServerUserid	user	The SQL server userid to use for non-windows authentication	dsu
/DatabaseSqlServerPassword	password	The SQL server password to use for non-windows authentication	dsp
/ServiceName	name	The name of the deployed service to find in the database	sen
/ServiceVersion	number	The version of the deployed service to find in the database	sv
/DeploymentNumber	number	The deployment number of the deployed service to find in the database	dn
/OutputDirectory	directory	The directory where the service resources will be placed. This directory is typically the directory that will be used as the file system location for an IIS virtual directory	od
/SourceOutputDirectory	directory	The directory where any source files and projects will be placed. Two projects are created, one for the service, one for the ConfiguredTypes assembly. If not specified, no source or projects will be created	sod
/Hosting	IIS/WAS/EXE	The type of hosting - only IIS is supported currently	ho

Option	Parameter	Description	Alias
/VdirCreateVdir	true/false	Indicates whether to create a virtual directory	vcv
/WebsiteId	string	The ID or Name of the Web Site (optional). If not supplied the Web Site with ID “1” will be used.	wid
/VdirName	name	The name of the virtual directory to create in IIS	vn
/VdirAnonymousAccess	true/false	Whether the virtual directory supports anonymous access	vaa
/VdirUser	user	The userid to use for anonymous access	vu
/VdirPassword	password	The password to use for anonymous access	vp
/ServiceContractNamespace	name	The namespace for the service and its methods that will appear in the WSDL. This value is used as the ServiceContract attribute’s namespace .	scn
/DataContractNamespace	name	The namespace that will be used for the data types that appear in the WSDL. This value is used as the DataContract attribute’s namespace	dcn
/TypesCodeNamespace	name	The namespace for the types defined in the ConfiguredTypes assembly	tcn
/ServiceCodeNamespace	name	The namespace for the service code in the generated service assembly	sna
/InputsOnMethodSignature	true/false	Causes methods to have the inputs as separate parameters. Without this option all inputs are provided via an Input object that is specialised for each method call	ios
/FlatWsdl	true/false	Causes WSDL generated for the service to not use the 'import' statement	fw
/AllRefdataIsDynamic	true/false	Makes all reference data types look like External reference data. This option has been provided to allow projects to coerce all reference data to have the same structure. Ordinarily Internal reference data is implemented through enumerations, and external reference date is represented via a union of the enumeration (for the configured values) and a string property for the dynamic or runtime values	ard
/TemplateDirectory	directory	The directory where the templates used by ServiceMaker are located	td
/SaveConfigFilename	file	The place to save the parameters used for this invocation	scf
/UI	true/false	Dictates whether forms are used or whether this program runs as a command line application	ui
/ConfiguredTypes	file	The name of the ConfigureTypes dll. Do not specify ‘.dll’ at the end of the name provided. Ordinarily the default ‘ConfiguredTypes’ is the value to use. If a service shares a virtual directory with another service and they are generated from different configurations, deployments or system, it may be necessary to avoid a name clash by altering the name of the Configured Types assembly	ct
/GenerateConfiguredTypes	true/false	Saves time by skipping generatiion of the Configured Types assembly. It can take a significant amount of time relative to the generation of the service itself to generate the types. It is not necessary to generate the types unless something other than the service or its	gct

Option	Parameter	Description	Alias
		interface has changed	
/GenerateReferencedTypesOnly	true/false	Reduces the size of the ConfiguredTypes assembly by only including the types that are referenced by the service methods	grt
/BinariesDirectory	directory	The location that contains the assemblies that must be copied into the **OutputDirectory**. ServiceMaker will copy all the .dll and .pdb files found in this directory to the **OutputDirectory**	bd
/ReservedWordsFile	file	The location of a file that contains a line-seperated list of reserved words that can't be included in the generated service	rwf
/ReleaseCode	true/false	Causes the generated assemblies to be release or debug assemblies	rc
/Help	none	Provides help	h

Note:

- Case is not significant for any of the switch names.
- Case is significant for any of the '*SQL/Windows*' type switch parameters.
- The "**-**" and "**/**" forms of options (e.g. "**-AllRefdataIsDynamic** " and "**/UI** ") may be intermingled.
- File or directory names should be enclosed in double quotes.
- Boolean values can be specified in any of the following forms:
    - **/AllRefdataIsDynamic** *true*
    - **/AllRefdataIsDynamic** *false*
    - **/AllRefdataIsDynamic** "true"
    - **/AllRefdataIsDynamic** "false"
    - **/AllRefdataIsDynamic** 1
    - **/AllRefdataIsDynamic** 0
    - **/AllRefdataIsDynamic**+
    - **/AllRefdataIsDynamic-**
- To override any value specified in the configuration file to the value it would default to use the syntax "**/AllRefdataIsDynamic** . "

Configuration File

The following is the supplied configuration file used to generate the SampleService example.

Note that no passwords are ever saved into the configuration files; they are, however, loaded if present.

The locations to load and save the configuration filenames are not saved or loaded.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceMaker>
    <System>Services</System>
    <DatabaseServer>DatabaseServer</DatabaseServer>
    <DatabaseName>DatabaseName</DatabaseName>
    <DatabaseAuthentication>Windows</DatabaseAuthentication>
    <DatabaseSqlServerUserid>name</DatabaseSqlServerUserid>
    <ServiceName>SampleService</ServiceName>
    <ServiceVersion>0</ServiceVersion>
    <DeploymentNumber>0</DeploymentNumber>
    <OutputDirectory>c:\SampleService</OutputDirectory>
    <Hosting>IIS</Hosting>
    <VdirCreateVdir>True</VdirCreateVdir>
    <VdirName>SampleService</VdirName>
    <VdirAnonymousAccess>True</VdirAnonymousAccess>
    <VdirUser>uk\pbishop</VdirUser>
    <TypesCodeNamespace>PortraitProject.Workspace</TypesCodeNamespace>
    <ServiceCodeNamespace>PortraitProject.Services</ServiceCodeNamespace>
  <ServiceContractNamespace>http://PortraitProject/Service</ServiceContractNamespace>
    <DataContractNamespace>http://PortraitProject/Data</DataContractNamespace>
    <InputsOnMethodSignature>True</InputsOnMethodSignature>
    <FlatWsdl>False</FlatWsdl>
    <AllRefdataIsDynamic>False</AllRefdataIsDynamic>
    <SourceOutputDirectory>c:\SampleService</SourceOutputDirectory>
    <ConfiguredTypes>ConfiguredTypes</ConfiguredTypes>
    <GenerateConfiguredTypes>True</GenerateConfiguredTypes>
</ServiceMaker>
```

2.5 Samples Provided

The following are provided as resources that match the SampleService used throughout this document.

The SampleService example consists of: the Configuration Suite configuration, ServiceMaker configuration file and a client project that can be used to exercise Service support. The examples in this text are taken from the resources shipped with the product. It should be possible to deploy the existing configuration unmodified, run ServiceMaker with the supplied configuration file and then use the client project to exercise the service.

The shipped ServiceMaker configuration file requires implementation specific values for database, user ids and IIS settings. The client project needs the reference to the ServiceMaker generated service to be added (as described in the code) and then will then compile and run exercising a cross section of features.

Sample command line invocation

This can be found at:

<PortraitInstallDirectory>\WebServices\ServiceMaker\Samples\SampleService.bat

Sample Configuration file

This can be found at:

<PortraitInstallDirectory>\WebServices\ServiceMaker\Samples\SampleService.xml

Sample Service configuration

This is contained in the "All Application Workspace" in the Showcase repository as SampleService.

Sample Client code

This project can be found in:

<PortraitInstallDirectory>\WebServices \ServiceMaker\SampleClient

IO and Decision Functions configuration files

ServiceMaker configuration files are provided for the IO web service and for the Decision Functions web service.

These can be found at:

<PortraitInstallDirectory>\WebServices \ServiceMaker\Samples

3 Client

This section contains details of how services are consumed.

Typically clients will use a proxy generator specific to the language, platform, or environment that they are in.

Considerations for a range of client environments are listed later in this section.

3.1 Invocations

The following samples are in C# but the creation, initialisation, and invocation should be easily translatable to other languages.

The various styles of invocation are listed below.

3.1.1 Request/Response style

Methods are generally invoked using the pattern:

```
MethodSpecificResponseObject = ExecuteMethod (StateObject, MethodSpecificRequestObject)
```

The examples are for a configured method called GetParty. GetParty has been configured in the configuration suite with two inputs; a PartyID and a PartyType.

ServiceMaker (via the imported WSDL) has defined

- the type `GetPartyRequest` containing `GetPartyInput` as the vehicle for the passing the inputs into the method.
- the type `GetPartyResponse` containing `GetPartyOutput` as the vehicle for the returning the outputs, outcome and completion state of the method
- `ExecuteGetParty()` as the method to call to invoke the model defined for the method GetParty
- `State` as a structure which contains the input and output data for session management and request tracing
  - The ActivityToken for session management – see 1.3 above
  - The RequestToken and ClientTag for request tracing – see 1.4 above

The following is taken from the provided SampleClient application that consumes the SampleService service.

```
// create the necessary state object
ServiceProxy.State State = new ServiceProxy.State();

#region GetParty - "object" signature

// retrieve the party by the "object" approach
SR.GetPartyRequest GetPartyRequest = new SR.GetPartyRequest();
GetPartyRequest.Input = new SR.GetPartyInput();
GetPartyRequest.Input.PartyID = "4EKaDX3ch5nNJyrNIngT8r3AAAAAB6pltb";
GetPartyRequest.Input.PartyType = "Agent";

GetPartyResponse = SampleServiceClient.ExecuteGetParty(ref State,
                                                       GetPartyRequest);
if (GetPartyResponse.Completion != SR.Completion.Completed ||
    GetPartyResponse.Outcome != "OK")
```

```
{
    throw new Exception("Unexpected response");
}

Agent = (SR.Party_Agent)GetPartyResponse.Output.Parties[0];

System.Console.WriteLine("GetParty - \"object\" signature succeeded. Marital is
'{0}'",
                          Agent.MaritalStatus);

#endregion
```

3.1.2 Inputs on method signature style

If the option "**/InputsOnMethodSignature** true" is specified via the command line, configuration file, or user interface then the invocation becomes as follows (the necessity for a Request object has been dropped):

```
#region GetParty - "long" signature

// retrieve the party by the "inputs on method signature" approach
SR.GetPartyResponse GetPartyResponse;

SR.Party_Agent Agent;
GetPartyResponse = SampleServiceClient.GetParty(ref State,
                                     "4EKaDX3ch5nNJyrNIngT8r3AAAAAB6pltb",
                                     "Agent");
if (GetPartyResponse.Completion != SR.Completion.Completed ||
    GetPartyResponse.Outcome != "OK")
{
    throw new Exception("Unexpected response");
}
Agent = (SR.Party_Agent)GetPartyResponse.Output.Parties[0];

System.Console.WriteLine("GetParty - \"long\" signature succeeded. Marital is '{0}'",
                          Agent.MaritalStatus);

#endregion
```

3.1.1 Generic style

ServiceMaker also emits an interface that is of the form:

```
ServiceSpecificResponseObject = ExecuteRequest (StateObject, ServiceSpecificRequestObject)
```

This interface is similar do the Request/Response form above but the Request and Response objects are defined as being *Service* not *Method* specific. Obviously the request object passed in will be a method specific object which is a specialisation of `ServiceSpecificRequestObject e.g. GetPartyRequest.`

This form is designed for use in circumstances where a common "Execute" call is desired by clients.

3.2 Reference Data

Reference data is divided into two categories:

Internal reference data

This has a set of values known at deployment time. This is represented by ServiceMaker as an enumeration – see MaritalStatus in 3.1.1 above.

External reference data

This is represented by an object containing an enumeration with the values known at deployment time and a string value. The string value is used for values that have been added to the list of valid values after deployment and are not therefore within the enumeration.

The string value takes precedence over the enumeration value (i.e. if the string value is not null it is used rather than the value of the enumeration). The string value is not populated when the objects are created unless the string value is outside of the values defined in the enumeration.

If the option "**/AllRefdataIsDynamic** true" is specified then all reference data is structured as per external reference data. This accommodates environments where internal reference data is altered after deployment or where a common approach to reference data fields is desirable.

3.3 Data Objects

Inheritance and Polymorphism

All Portrait Data Objects have **Categories** and **Types**. It is not unusual to define an interface as having inputs or outputs that are Data Objects of a **Category** where the **Type** is not specified. This style of parameter states that the input or output Data Object is of "Any Kind Of" Data Object **Type** within this Data Object **Category**. To support this, all Data Objects are derived from an ancestor called `AnyKindOf`***`Category`***. (e.g. `AnyKindofParty`). This allows anything derived from `AnyKindofParty` (e.g. `Agent`) to be passed in to a method input that specified `AnyKindofParty`. Conversely an output may be defined as a type such as `AnyKindofTask` and this can be cast back to its actual type.

CBE Data Objects (Party, Contract, Product, Task)

Portrait CBEs can be configured in an arbitrarily deep hierarchy; for example Agent is derived from Individual. Individual is derived from BaseParty. BaseParty is derived from `AnyKindOfParty`. ServiceMaker will represent any hierarchy defined in the configuration suite in the CBE Data Objects it generates.

Non-CBE Data Objects

Every Data Object **Category** and **Type** combination is generated as a separate type. All Data Objects within a **Category** are derived from the `AnyKindOf`***`Category`*** object e.g. `Email` is derived from `AnyKindofEnvelope`.

Any non-CBE Data Object within a **Category** is only derived from `AnyKindOf`***`Category`***. There is no hierarchy of objects for non-CBE types.

Nillability

Properties on Data Objects are typically 'nillable' – this allows the value of enumerations, and simple types as well as strings to be set to null. The data transmission mechanisms preserve null values and therefore allow fields to be expressed as 'not set'.

4 Deployment

The recommended approach to deploying Foundation WCF services is to install them alongside any other Foundation implementation applications. The implementation install should only be responsible for installing the services and not generating them on the fly.

ServiceMaker can be run in a development environment to generate the service. It is then down to the implementation team whether the generated source code is included in the project build process or the generated binaries are kept in source control.

During the implementation install, the name of the Foundation System can then be updated in the appSettings section of Web.config.

```
<appSettings>
    <!-- If you change the name of the system, you will also need to restart IIS
    because the Foundation components used by the service will have system specific
    in-memory lookups that won't get automatically rebuilt -->
    <add key="SystemName" value="MyPortrait"/>
</appSettings>
```

5 Debugging and diagnostics

Portrait Logging

Both the ServiceMaker tool and the runtime use Portrait Logging to record trace data and errors in the eventlog and/or via the Portrait Log Viewer. It is advisable to review entries in these logs when validating successful operation and when diagnosing problems.

WCF Tools

WCF provides comprehensive tracing facilities and the following viewer provides access to trace and error data.

"C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\SvcTraceViewer.exe"

The tracing is enabled via specifications in web.config - the following tool can be used to alter a service's web.config. Note that any alterations to web.config will be overwritten in the service is re-generated.

"C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\SvcConfigEditor.exe"

Import statements in WSDL

Some proxy generators do not handle the use of 'import' in the WSDL. Use the **"/FlatWsdl** true" option to remove use of import; it causes the WSDL to be generated as one file rather than many.

Basic Profile

By default the web.config file defines use of wsHttpBinding which supports WCF's implementation of the WS-* protocols. This is defined by the following statement in web.config.

```
<endpoint address="" binding="wsHttpBinding"
contract="ServiceCodeNamespace.IExampleService">
```

This can be set to use `binding="basicHttpBinding"` to support the WS-BasicProfile 1.1. This may be necessary for some clients that do not support the WS-* protocols. (Note: .Net 3.0 clients can use WCF, which means that they can make use of the WS-* protocols).

It is also useful as a diagnostic aid to use the basic profile if problems are experienced with `wsHttpBinding.` If regression to `basicHttpBinding` helps prove a level of interoperability, this can be helpful in ascertaining how to address issues that are specific to use of `wsHttpBinding.`

Assembly

Each of the generated assemblies has the details of the ServiceMaker parameters embedded within it to help diagnosis of problems. This feature can be used to verify compatibility between the service assembly and the Configure Types assembly.

6 Known Issues

Dynamically altered Data Objects

There is a general issue that if a Portrait Foundation Process Models adds extra properties to a data object (i.e. properties that are not in its formal definition in the Configuration Suite) then those fields are not available to the users of Services.

In these circumstances the following logging is created. Note that these are NOT logged as errors but only warnings and will not therefore appear in the event log of a live system. These errors should be been picked up in testing where analysis of logging should be more thorough.

```
INFO: Couldn't find property INVOLVEMENT_COLLECTION
Property INVOLVEMENT_COLLECTION is not defined on this DataObject - returned 80004005
Adding property INVOLVEMENT_COLLECTION to locked object [cat=Engagement, type=Action]
```

7 Extensibility

Extensibility is achieved through a number of approaches described below:

Templates

ServiceMaker is extensible because the resources it generates are created by processing templates. These templates are processed by the ServiceMaker code and simple textual substitution is done on values marked up accordingly. Typically the values are marked up in the form.

```
$servicename$
```

Projects are at liberty to modify these templates to achieve the desired results in the outputs. E.g. Adding a definition to "Web.config.template" will result in an altered "Web.config"in the output directory.

Changes here will be present in every subsequent service generation that use these templates.

Projects

If the option **/SourceOutputDirectory** is selected, two projects will be generated by ServiceMaker.

- The service project – containing the callable interface of the service.
- The configured types project – containing all deployed definitions

These can be used for debugging purposes and to make alterations to the behaviour or definitions of the code.

Obviously changes here can possibly be overwritten at each service generation.