



---

# Deployment Guide

Version 6.1, December 2003

IONA, IONA Technologies, the IONA logo, Orbix, Orbix/E, Orbacus, Artix, Orchestrator, Mobile Orchestrator, Enterprise Integrator, Adaptive Runtime Technology, Transparent Enterprise Deployment, and Total Business Integration are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

---

#### COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2003 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

M 3 1 3 2

Updated: 06-Jan-2004

# Contents

<b>List of Figures</b>	<b>v</b>
<b>Preface</b>	<b>vii</b>
<b>Chapter 1 Orbix Configuration and Deployment</b>	<b>1</b>
Introduction to Orbix Configuration and Deployment	2
Deploying Orbix Configuration from the Command Line	6
Generating a Deployment Descriptor	7
Deploying on a Machine without a GUI	11
<b>Chapter 2 Orbix Domain Deployment Descriptor</b>	<b>13</b>
Deployment Descriptor Structure	14
Domain Configuration Elements	17
Profile Configuration Elements	21
<b>Chapter 3 Advanced Configuration and Deployment</b>	<b>29</b>
Specifying Custom Locations for Domain Files	30
Specifying Custom Library Paths	38
Using Custom XML Files	39
Deploying on Multihomed Host Machines	43
Specifying Address Mode Policies	48
<b>Chapter 4 Migrating from Orbix 5.1 Deployments</b>	<b>53</b>
Migrating from Orbix 5.1 Driver Files	54
Converting to an Orbix 6.1 Descriptor	56
<b>Appendix A Orbix Deployment DTD</b>	<b>63</b>
Orbix Component Template Structure	64
<b>Index</b>	<b>69</b>

## CONTENTS

# List of Figures

Figure 1: Overview of Orbix Configuration and Deployment	2
Figure 2: Orbix Configuration GUI	7
Figure 3: Domain Settings Screen	8
Figure 4: Services Settings Screen	9
Figure 5: Summary Screen	10
Figure 6: Custom Domain Settings	33
Figure 7: Custom Domain Summary	34
Figure 8: Custom Configuration Locations	35
Figure 9: Partially Set Custom Locations	36
Figure 10: Custom XML Components	40
Figure 11: Select Custom Components	40
Figure 12: Dialog for Two Nodes	43
Figure 13: Dialog for More than Two Nodes	44
Figure 14: Multihomed Hostname	45
Figure 15: Services Selected on Multihomed Host	46
Figure 16: Multihomed Message	47
Figure 17: Selecting an Address Mode Policy	49
Figure 18: Specifying a Hostname	50
Figure 19: Node Daemon Settings Dialog	51

## LIST OF FIGURES

# Preface

Orbix enables you to develop and deploy enterprise-level applications across different platform and programming language environments. This guide examines the Orbix configuration and deployment process in detail.

**WARNING:** The scope of this guide is limited to the configuration and deployment features that are supported by IONA. Unsupported configuration and deployment features are not documented. These are proprietary features and are subject to change without notice.

---

## Audience

This guide is aimed at IONA customers and partners who wish to customize their configuration and deployment. It assumes prior knowledge of Orbix.

---

## Related documentation

The document set for Orbix includes the following related documentation:

- *Administrator's Guide*
- *Configuration Reference*
- *Management User's Guide*

The latest updates to the Orbix documentation can be found at:

<http://www.iona.com/docs>

---

## Additional resources

The [IONA knowledge base](http://www.iona.com/support/knowledge_base/index.xml) ([http://www.iona.com/support/knowledge\\_base/index.xml](http://www.iona.com/support/knowledge_base/index.xml)) contains helpful articles, written by IONA experts, about Orbix and other products. You can access the knowledge base at the following location:

The [IONA update center](http://www.iona.com/support/updates/index.xml) (<http://www.iona.com/support/updates/index.xml>) contains the latest releases and patches for IONA products:

If you need help with this or any other IONA products, contact IONA at [support@iona.com](mailto:support@iona.com). Comments on IONA documentation can be sent to [docs-support@iona.com](mailto:docs-support@iona.com).

## Typographical conventions

This guide uses the following typographical conventions:

**Constant width** Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class.

Constant width paragraphs represent code examples or information a system displays on the screen. For example:

```
#include <stdio.h>
```

**Italic** Italic words in normal text represent *emphasis* and *new terms*.

Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:

```
% cd /users/your_name
```

**Note:** Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

## Keying conventions

This guide may use the following keying conventions:

**No prompt** When a command's format is the same for multiple platforms, a prompt is not used.

**%** A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.

**#** A number sign represents the UNIX command shell prompt for a command that requires root privileges.

**>** The notation `>` represents the DOS or Windows command prompt.



...	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

## PREFACE

# Orbix Configuration and Deployment

*This chapter gives an overview of Orbix configuration and deployment, and explains how manually deploy a configuration domain.*

---

**In this chapter**

The following topics are discussed in this chapter:

<a href="#">Introduction to Orbix Configuration and Deployment</a>	<a href="#">page 2</a>
<a href="#">Deploying Orbix Configuration from the Command Line</a>	<a href="#">page 6</a>
<a href="#">Generating a Deployment Descriptor</a>	<a href="#">page 7</a>
<a href="#">Deploying on a Machine without a GUI</a>	<a href="#">page 11</a>

# Introduction to Orbix Configuration and Deployment

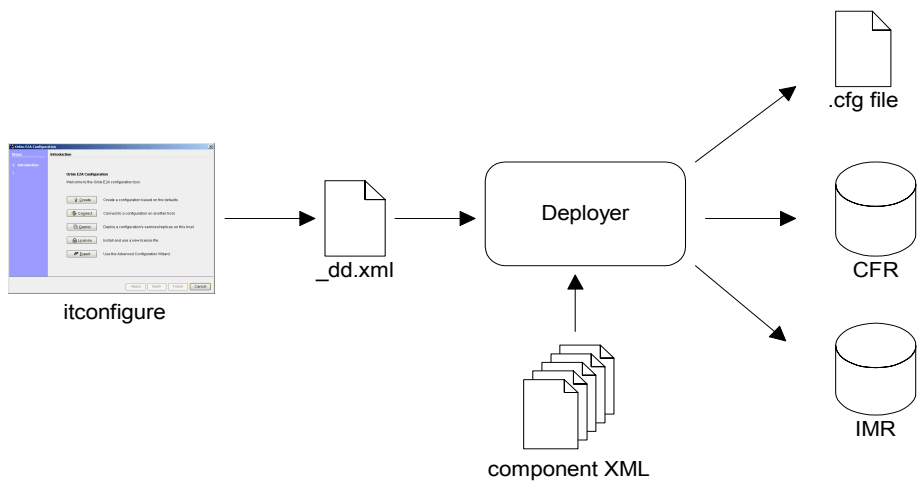
## Overview

This section introduces Orbix configuration and deployment. It includes the following topics:

- “Orbix configuration and deployment process”.
- “Orbix configuration tool (itconfigure)”.
- “Domain deployment descriptor”.
- “Orbix deployer and component XML files”.
- “Deployed configuration models”.
- “Implementation Repository”.

## Orbix configuration and deployment process

Figure 1 shows a general overview of the Orbix configuration and deployment process.



**Figure 1:** Overview of Orbix Configuration and Deployment

Figure 1 can be described as follows:

1. The Orbix configuration tool (`itconfigure` command) is used to generate the domain deployment descriptor (`domain-name_dd.xml`).
2. The deployer parses the deployment descriptor, taking input from the standard XML templates for the various Orbix components and services. You can also specify custom XML files.
3. The deployer deploys the configuration domain into a configuration domain file or the Configuration Repository (CFR), and also into the Implementation Repository (IMR).

The components in Figure 1 are described in more detail in the topics that follow.

### Orbix configuration tool (`itconfigure`)

The Orbix configuration tool (`itconfigure` command) guides you through configuring Orbix components in your environment. You can use it to perform tasks such as installing a license, creating a configuration domain, or linking to an existing configuration domain.

You can run the Orbix configuration tool in GUI and command-line modes. You should create a domain deployment descriptor by using the Orbix configuration tool in GUI mode (as shown in Figure 1). You can also create a configuration domain in non-GUI mode by passing a previously created deployment descriptor to the `itconfigure` command.

The GUI imposes constraints and performs validity checking, for example, on the combinations of Orbix services that are permitted. Finally, the GUI creates a domain deployment descriptor (`domain-name_dd.xml`). You can choose to create the configuration domain specified by this deployment descriptor straight from the GUI. Alternatively, you can save the descriptor and create your domain later, in GUI or non-GUI mode.

For a comprehensive description of how to use the Orbix configuration tool, see the *Orbix Administrator's Guide*.

### Domain deployment descriptor

The domain deployment descriptor file (`domain-name_dd.xml`) describes the contents of a configuration domain. For example, for a domain named `sample-domain`, a deployment descriptor named `sample-domain_dd.xml` specifies the services, components, features and hosts that are included in that domain. By default, the deployment descriptor file is stored in your `etc\domains` directory, for example:

```
<install-dir>\etc\domains\sample-domain\sample-domain_dd.xml
```

The Orbix configuration GUI generates the deployment descriptor, which it then uses to automatically deploy the specified configuration into your environment (as shown in [Figure 1](#)).

Alternatively, you can also save the deployment descriptor before it is deployed by the GUI, and then perform a command-line deployment at a later stage. This is particularly useful if you want to customize your configuration by editing your deployment descriptor, or use multiple deployments with the same configuration.

For full details of how to perform a command-line deployment, see [“Deploying Orbix Configuration from the Command Line” on page 6](#). For details on the contents of the deployment descriptor file, see [Chapter 2](#).

---

### Orbix deployer and component XML files

The deployer parses the deployment descriptor produced by the Orbix configuration GUI. It also takes input from the XML templates for the various Orbix components and services (for example, `appserver.xml` and `event_log.xml`). By default, these XML templates are stored in the following directory:

```
<install-dir>\asp\6.1\etc\conf
```

These template files all conform to a standard XML format as specified by the `ABDeploy.dtd` file. For details of this DTD file, see [Appendix A](#).

You can also specify custom XML files to the deployer. For details, see [“Using Custom XML Files” on page 39](#).

---

### Deployed configuration models

Depending on which option you chose in the configuration GUI, the deployer gathers your configuration information into either a configuration file or a Configuration Repository (CFR), and creates scripts to start and stop the domain services.

The Interoperable Object References (IORs) that the deployer obtains by preparing the domain services are essential part of this configuration domain data. If these are stored in a file, and clients need access to these IORs, you need to make sure that this file is accessible for all clients (using NFS or similar network services). If you are dealing with a larger number of clients, or expect to modify configuration data, using a Configuration Repository might be your preferred choice.

A Configuration Repository is a centralized database for all configuration information. This centralized configuration model is suitable for environments with a potentially large number of clients and servers, or when configuration is likely to change.

---

### **Implementation Repository**

The deployer also stores server process information in the Implementation Repository (IMR). This specifies whether the process can be started up on demand by a node daemon, and includes details such as POA names, and ORB names.

For more details on Orbix configuration models and the IMR, see the *Orbix Administrator's Guide*.

---

# Deploying Orbix Configuration from the Command Line

## Overview

You can use the Orbix configuration GUI to generate and deploy the domain deployment descriptor. However, for users who cannot deploy using a GUI application, Orbix also provides a command-line version of the configuration tool (`itconfigure -nogui`). This parses a pre-existing deployment descriptor and deploys the specified configuration domain. This section lists the requirements for a command line deployment. The two sections that follow explain the two main steps:

- [“Generating a Deployment Descriptor”](#).
- [“Deploying on a Machine without a GUI”](#).

---

## Requirements

The Orbix configuration tool requires that the following environment variables are set:

- `IT_PRODUCT_DIR` should point to your Orbix installation.
- `JAVA_HOME` should point to a JDK installation.
- `PATH` should include the following directory:

```
<install-dir>\asp\6.1\bin
```

Before performing a command-line deployment, you must ensure that a properly-formed deployment descriptor exists. The deployment descriptor is the XML source document from which `itconfigure` builds and deploys the configuration domain. It specifies what components are to be included in the configuration domain, and takes the form `domain-name_dd.xml`.



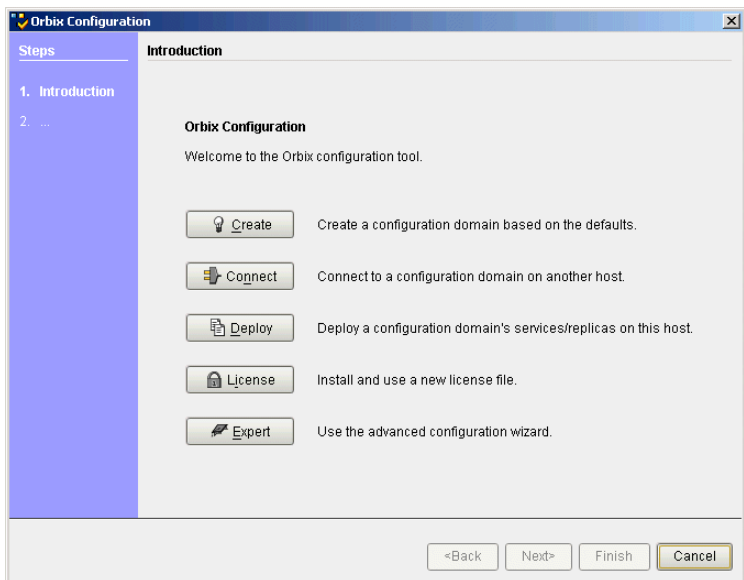
---

# Generating a Deployment Descriptor

---

## Overview

The recommended method of generating a deployment descriptor is to first run the Orbix configuration tool on a GUI-enabled machine, and then save the deployment descriptor for later use in command-line mode. This ensures that the generated XML document is valid.



**Figure 2:** *Orbix Configuration GUI*

## Generating the descriptor

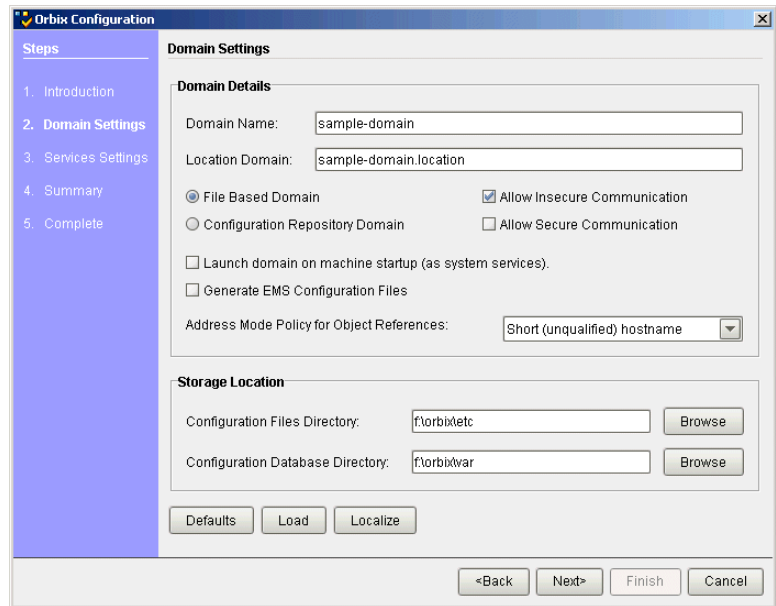
To generate the deployment descriptor in GUI mode, perform the following steps:

1. On a machine with GUI capabilities, enter the following command:

```
<install-dir>\asp\6.1\bin\itconfigure
```

This displays the **Orbix Configuration** GUI, shown in [Figure 2](#).

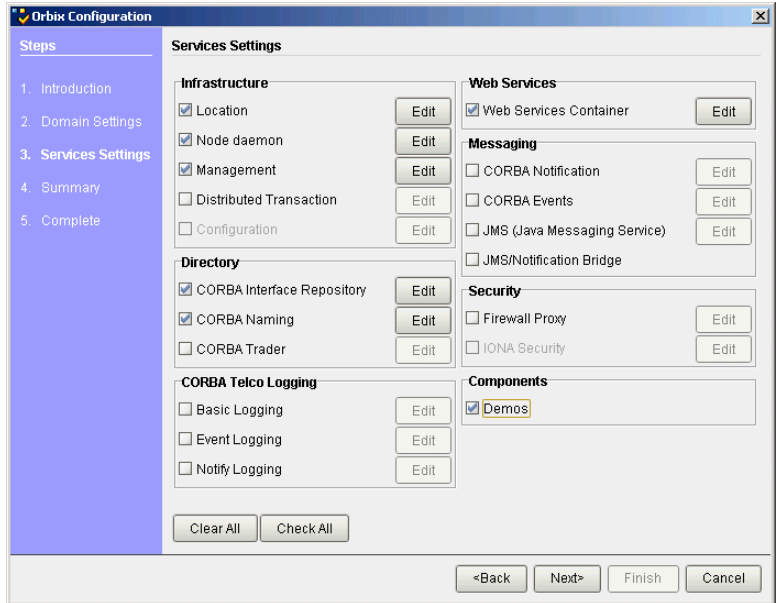
2. Click the **Expert** button, and enter your domain name, in the **Domain Details** panel, shown in [Figure 3](#). You can also specify other settings such as whether to use a file or CFR-based configuration domain.



The screenshot shows the 'Orbix Configuration' window with the 'Domain Settings' panel selected in the 'Steps' sidebar. The 'Domain Details' section includes text boxes for 'Domain Name' (sample-domain) and 'Location Domain' (sample-domain.location). It features radio buttons for 'File Based Domain' (selected) and 'Configuration Repository Domain'. There are checkboxes for 'Allow Insecure Communication' (checked) and 'Allow Secure Communication' (unchecked). Other options include 'Launch domain on machine startup (as system services)' and 'Generate EMS Configuration Files', both unchecked. A dropdown menu for 'Address Mode Policy for Object References' is set to 'Short (unqualified) hostname'. The 'Storage Location' section has text boxes for 'Configuration Files Directory' (f:\orbix\etc) and 'Configuration Database Directory' (f:\orbix\var), each with a 'Browse' button. At the bottom of the panel are 'Defaults', 'Load', and 'Localize' buttons. The main window footer contains '<Back', 'Next>', 'Finish', and 'Cancel' buttons.

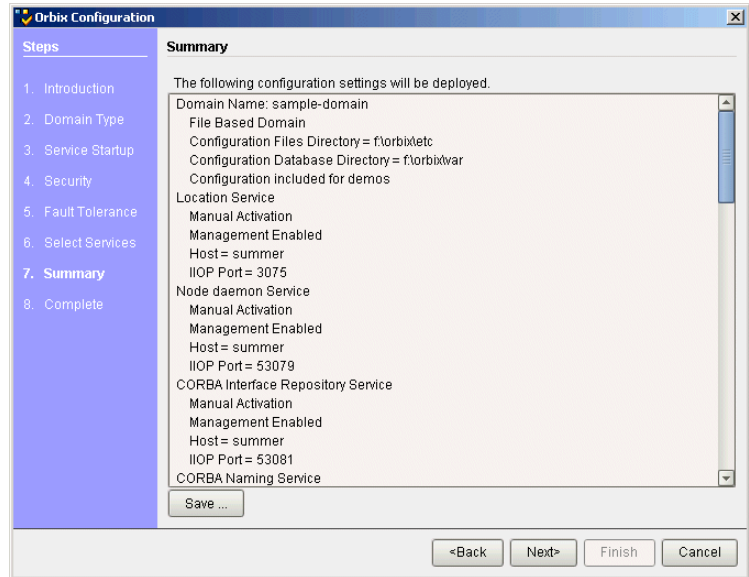
**Figure 3:** *Domain Settings Screen*

3. Click **Next** to display the **Services Settings** screen, and select the services that you require. [Figure 4](#) shows a simple example with a small number of services selected.



**Figure 4:** *Services Settings Screen*

4. Click **Next** to display the **Summary** screen, shown in [Figure 4](#).



**Figure 5:** Summary Screen

5. Click the **Save** button at the bottom left of the screen, and save the deployment descriptor to your chosen location; for example, the default location is:

```
install-dir\etc\domains\sample-domain\
```

6. Click the **Cancel** button to exit the configuration GUI.

---

# Deploying on a Machine without a GUI

---

## Overview

This section shows how to deploy a pre-generated deployment descriptor on a machine without GUI capabilities. It includes the following sections:

- [Deploying on the command line.](#)
  - [Localizing the domain.](#)
  - [Changing the domain name.](#)
- 

## Deploying on the command line

To deploy a deployment descriptor on the command line, perform the following steps:

1. Copy your deployment descriptor file to the machine without GUI capabilities.
2. At the command prompt, change directory to the location of your domain deployment descriptor, for example:

```
install-dir\etc\domains\sample-domain
```

3. Enter the following command:

```
itconfigure -load sample-domain_dd.xml -nogui
```

`itconfigure` reads the specified deployment descriptor, finds the profile matching the current host's IP address and deploys the services in this profile. If no such match is found, `itconfigure` prints an information message and exits.

---

## Localizing the domain

If the descriptor contains exactly one profile/node, and that node does not match the local host, use the following command:

```
itconfigure -load sample-domain_dd.xml -nogui -localize
```

This replaces the name and IP address of that node with name and IP address of the local host.

---

## Changing the domain name

If you wish to change the name of the configuration domain, use following command:

```
itconfigure -load sample-domain_dd.xml -nogui -name my-domain
```

The name specified on the command line overrides the name specified in the descriptor.

### Example output

The following example output is displayed for a host machine called ARAN:

```
COPY RESOURCES f:\orbix\asp\6.1\templates\etc\admin TO
  f:\orbixorbix\etc\domains\my-domain
COPY RESOURCES f:\orbix\asp\6.1\templates\etc\log4j TO
  f:\orbix\etc\domains\my-domain
STARTING TO PREPARE: iona_services.management
COMPLETED PREPARE: iona_services.management
STARTING TO PREPARE: iona_services.locator.ARAN
COMPLETED PREPARE: iona_services.locator.ARAN
STARTING TO RUN: iona_services.locator.ARAN
COMPLETED RUN: iona_services.locator.ARAN
STARTING TO PREPARE: iona_services.node_daemon.ARAN
COMPLETED PREPARE: iona_services.node_daemon.ARAN
STARTING TO RUN: iona_services.node_daemon.ARAN
COMPLETED RUN: iona_services.node_daemon.ARAN
START-UP MODE: on_demand
STARTING TO PREPARE: iona_services.naming.ARAN
COMPLETED PREPARE: iona_services.naming.ARAN
COPY RESOURCES f:\orbix\asp\6.1\templates\etc\log4j TO
  f:\orbix\etc\domains\my-domain
START-UP MODE: on_demand
STARTING TO SHUTDOWN: iona_services.node_daemon.ARAN
COMPLETED SHUTDOWN: iona_services.node_daemon.ARAN
STARTING TO SHUTDOWN: iona_services.locator.ARAN
COMPLETED SHUTDOWN: iona_services.locator.ARAN
Configuration completed successfully
You can view the log in 'f:\orbix\etc\log'.

To set your environment to use these configuration settings run:
f:\orbix\etc\bin\my-domain_env.bat
```

For full details of all the options to the `itconfigure` command, see the *Orbix Administrator's Guide*.

# Orbix Domain Deployment Descriptor

*This chapter explains the data structure and grammar of the deployment descriptor file.*

**In this chapter**

---

The following topics are discussed in this chapter:

<a href="#">“Deployment Descriptor Structure” on page 14</a>
<a href="#">“Domain Configuration Elements” on page 17</a>
<a href="#">“Profile Configuration Elements” on page 21</a>

# Deployment Descriptor Structure

## Overview

The domain deployment descriptor file (*domain-name\_dd.xml*) describes the contents of a configuration domain. This section outlines the overall structure of this file.

## Document structure

The *<domain-name>\_dd.xml* file must conform to the following document structure:

### Example 1: *Deployment Descriptor Structure*

```

1 <?xml version="1.0" encoding="UTF-8"?>
  <dd:descriptor xmlns:dd="http://ns.iona.com/aspdd">
    <!--This deployment descriptor version 1.0 has been generated
      by Orbix tools-->
2   <dd:configuration>
     <dd:domain>domain-name</dd:domain>
     ...
   </dd:configuration>

   <!--Concrete node information for this deployment-->
3   <dd:nodes>
     <dd:node name="hostname" ip="ip-address" profile="hostname"
       <dd:resource name="some-resource" value="some-value"/>
       ...
     <dd:policies>
       <dd:policy name="some-policy" value="some-value" />
     </dd:policies>
     ...
     </dd:node>
     ...
   </dd:nodes>

5   <dd:feature id="feature-name">
     <dd:resource type="directory" name="some-resource"/>
   </dd:feature>

   <!--The following profiles will be deployed-->
5   <dd:profile id="hostname">

```



**Example 1:** *Deployment Descriptor Structure*

```

6      <dd:service name="service-name" ... >
        ...
        </dd:service>
        ...
7      <dd:component/>
        ...
        </dd:profile>
      </dd:descriptor>

```

This deployment descriptor structure is described as follows:

1. The `<dd:descriptor>` element is the containing root element of the deployment descriptor XML vocabulary. It specifies an XML namespace named `dd`. This element will indicate what version of the deployment descriptor XML vocabulary is being used. In this case, the absence of a version attribute indicates that this is version 1.0.
2. The `<dd:configuration>` element specifies the general configuration information for the domain (for example, its name, type, and location domain).
3. The `<dd:nodes>` element specifies information about the host machines included in the domain. A `<dd:node>` element can include `<dd:resource>` and `<dd:policies>` elements.
4. The `<dd:feature>` element specifies information about domain-level features.
5. The `<dd:profile>` element specifies a logical group of services and components that maps to a particular node.
6. The `<dd:service>` element specifies the details for a particular service (for example, the naming service).
7. The `<dd:component>` element specifies the details for a particular component (for example, Orbix demos). The difference between a component and a service is that services maintain live database information as part of the domain state, whereas a component does not.

These elements are described in more detail with examples in the sections that follow.

---

**Recommended deployment descriptor generation**

The recommended method of generating a deployment descriptor is to run the Orbix configuration tool on a GUI-enabled machine, and, if necessary, save the deployment descriptor for later use in command-line. Generating the descriptor in GUI mode ensures that the generated XML document is valid, and checked for dependencies.

Certain combinations of services and features are not permitted. For example, a descriptor that contains an indirect persistent, on-demand naming service, but no node-daemon, is invalid. Using different transports for different services is also invalid. Finally, a descriptor with a node daemon that has secure endpoints only, and a locator with insecure endpoints only is not valid either because the locator would not be able to communicate with the node daemon.

---

**Validating manual changes to a deployment descriptor**

You can edit the domain deployment descriptor file to meet your requirements using any text editor. However, any changes you make need to be checked for validity and dependencies.

Running the Orbix configuration tool enforces consistency on a deployment descriptor that has inconsistent relationships between services, or has incorrect container descriptions. You can validate manual changes to a deployment descriptor by running the following command:

```
itconfigure -nogui -load descriptor.xml -save somefile.xml
```

If the descriptor is correct, `descriptor.xml` and `somefile.xml` will be identical in structure. Otherwise, the configuration tool reports an error message and exits without saving into the specified document.

---

# Domain Configuration Elements

---

## Overview

This section explains the domain-specific information contained in an example deployment descriptor file.

---

## Example descriptor

The following extract from a deployment descriptor file named `my-domain_dd.xml` shows some example domain-specific elements:

### Example 2: *Domain-Specific Configuration*

```
<?xml version="1.0" encoding="UTF-8"?>
<dd:descriptor xmlns:dd="http://ns.iona.com/aspdd">
  <!--This deployment descriptor version 1.0 has been generated
  by Orbix tools-->
  <dd:configuration>
    <dd:domain>my-domain</dd:domain>
    <dd:source>file</dd:source>
    <dd:location_domain>my-domain.location</dd:location_domain>
  </dd:configuration>
  <!--Concrete node information for this deployment-->
  <dd:nodes>
    <dd:node name="summer" ip="10.2.4.82" profile="summer" />
  </dd:nodes>
  <!--The following profiles will be deployed-->
  <dd:profile id="summer">
    ...
  </dd:profile>
</dd:descriptor>
```

## Domain elements

The following table explains all the domain-specific elements:

**Table 1:** *Domain-Specific Elements*

Element	Description
<code>&lt;dd:descriptor&gt;</code>	Specifies the XML namespace details for the deployment descriptor.

**Table 1:** *Domain-Specific Elements*

Element	Description
<code>&lt;dd:configuration&gt;</code>	Specifies the general configuration information for the domain (for example, its name, type, and location domain)
<code>&lt;dd:domain&gt;</code>	Specifies the configuration domain name (in this case, <code>my-domain</code> ).
<code>&lt;dd:source&gt;</code>	Specifies the configuration domain type. Can be either <code>file</code> , <code>cfr</code> , or <code>link</code> ( <code>.cfg</code> text file, Configuration Repository, or a link domain).
<code>&lt;dd:location_domain&gt;</code>	Specifies the location domain name. This takes the form <code>&lt;domain-name&gt;.location</code> (for example, <code>my-domain.location</code> ).  A location domain is a group of servers that are registered with the same locator daemon.
<code>&lt;dd:nodes&gt;</code>	This is a container for all host machines in a configuration domain that belong to the same <code>dns</code> domain. It has a single <code>dns</code> attribute (for example, <code>dns="dublin.emea.myco.com"</code> ).  There can be multiple <code>&lt;dd:nodes&gt;</code> in one deployment descriptor. For example:  <pre> &lt;dd:nodes dns="dublin.emea.myco.com"&gt;   &lt;dd:node name="summer" ip="10.2.4.82"     profile="summer.dublin.emea.myco.com"   /&gt;   &lt;dd:node name="onion" ip="10.2.1.101"     profile="onion.dublin.emea.myco.com"   /&gt; &lt;/dd:nodes&gt;  &lt;dd:nodes dns="boston.amer.mycorp.com"&gt;   &lt;dd:node name="jupiter" ip="10.5.3.18"     profile="jupiter.boston.amer.mycorp.c om" /&gt; &lt;/dd:nodes&gt; </pre>

**Table 1:** *Domain-Specific Elements*

Element	Description
<dd:node>	<p>Specifies the identity of a particular host machine in the domain. It has three attributes:</p> <ul style="list-style-type: none"> <li>• <code>name</code> specifies the hostname.</li> <li>• <code>ip</code> specifies the IP address.</li> <li>• <code>profile</code> specifies a logical group of services and components to deploy on the specified node.</li> </ul> <p>A &lt;dd:node&gt; element can also include optional &lt;dd:resource&gt; and &lt;dd:policies&gt; elements.</p>
<dd:profile>	<p>Specifies a logical group of services and components. Its <code>id</code> attribute corresponds to the &lt;dd:node <code>profile</code>&gt; attribute. In this version of Orbix, only one profile per node is supported.</p>
<dd:feature>	<p>Specifies information about optional domain-level features. These are implemented separately from the deployer and invoked at the end of the deployment process. The following example is for integration with IBM Tivoli management:</p> <pre data-bbox="803 1043 1247 1277"> &lt;dd:descriptor ...   &lt;dd:feature     xmlns:dd="http://ns.iona.com/aspdd"     id="tivoli-integration"&gt;       &lt;dd:resource type="directory"         name="configuration-files" /&gt;     &lt;/dd:feature&gt;   ... &lt;/dd:descriptor&gt; </pre>

**Table 1:** *Domain-Specific Elements*

Element	Description
<dd:resource>	<p>Specifies resources used by domain-level features. For example:</p> <pre data-bbox="771 404 1205 453">&lt;dd:resource type="directory"   name="configuration-files"/&gt;</pre> <p>This specifies a resource that is a file system directory named <code>configuration-files</code>.</p>
<dd:policies>	<p>As a child of the &lt;dd:node&gt; element, specifies policies that apply to all services on that node. Currently, there is only one available policy:</p> <p><code>address_mode</code></p> <p>For example:</p> <pre data-bbox="771 730 1237 939">&lt;dd:nodes&gt;   &lt;dd:node name="orion2" ip="10.2.1.101"&gt;     &lt;dd:policies&gt;       &lt;dd:policy name="address_mode"         value="ip"/&gt;     &lt;/dd:policies&gt;   &lt;/dd:node&gt; &lt;/dd:nodes&gt;</pre> <p>For more details on this example, see <a href="#">“Converting to an Orbix 6.1 Descriptor” on page 56</a>.</p> <p>Policies can also be specified on a per-service bases (see <a href="#">“Profile Configuration Elements” on page 21</a>). Service-specific policies override node-specific policies.</p>

---

# Profile Configuration Elements

---

## Overview

A profile specifies a group of configured services and components for a particular node. This section explains the profile-specific information contained in a example deployment descriptor file.

---

## Example descriptor

The following is a complete listing of a deployment descriptor file named `my-domain_dd.xml`. It shows an entire profile configured for a default domain:

### Example 3: *Profile Configuration*

```
<?xml version="1.0" encoding="UTF-8"?>
<dd:descriptor xmlns:dd="http://ns.iona.com/aspdd">
  <!--This deployment descriptor has been generated by ASP
  tools-->
  <dd:configuration>
    <dd:domain>my-domain</dd:domain>
    <dd:source>file</dd:source>
    <dd:location_domain>my-domain.location</dd:location_domain>
  </dd:configuration>

  <!--Concrete node information for this deployment-->
  <dd:nodes>
    <dd:node name="summer" ip="10.2.4.83" profile="summer" />
  </dd:nodes>
  <!--The following profiles will be deployed-->
  <dd:profile id="summer">
    <dd:service name="locator">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" proxified="false"
        managed="true" authenticated="false" />
      <dd:endpoint protocol="iiop" port="3075" />
    </dd:service>
  </dd:profile>
</dd:descriptor>
```

**Example 3:** *Profile Configuration*

```

<dd:service name="node_daemon">
  <dd:activation mode="manual" />
  <dd:run mode="direct_persistent" proxified="false"
    managed="true" authenticated="false" />
  <dd:endpoint protocol="iiop" port="53079" />
</dd:service>
<dd:service name="naming">
  <dd:activation mode="on_demand" />
  <dd:run mode="indirect_persistent" proxified="false"
    managed="true" authenticated="false" />
  <dd:endpoint protocol="iiop" port="0" />
</dd:service>
<dd:service name="management">
  <dd:activation mode="manual" />
  <dd:run mode="direct_persistent" proxified="false"
    managed="true" authenticated="false" />
  <dd:endpoint protocol="iiop" port="53085" />
  <dd:endpoint protocol="http" port="53185" />
</dd:service>
<dd:component name="demos" />
</dd:profile>
</dd:descriptor>

```

**Service elements**

The following table explains the profile-specific elements

**Table 2:** *Profile-Specific Elements*

Element	Description
<dd:service>	Specifies the identity of a service. It has two attributes: <ul style="list-style-type: none"> <li>name is the service name (for example, locator).</li> </ul>



**Table 2:** *Profile-Specific Elements*

Element	Description
<dd:activation>	<p>Specifies how a service is activated. Its single <code>mode</code> attribute has two possible values:</p> <ul style="list-style-type: none"><li>• <code>manual</code> specifies that it must be activated using a start command or a script.</li><li>• <code>on_demand</code> means that the node daemon starts the service when requested by a client.</li><li>• <code>system_service</code> specifies that the service will be started at boot time. On Windows, the service will be installed as an NT service. On Unix, appropriate run control scripts will be created. For more details, see Part IV in the <i>Administrator's Guide</i>.</li></ul>

**Table 2:** *Profile-Specific Elements*

Element	Description
<code>&lt;dd:run&gt;</code>	<p>Specifies how a service is run. It has the following attributes, all of which are optional:</p> <ul style="list-style-type: none"> <li>• <code>mode</code> specifies whether the service uses the locator to resolve persistent object references (indirect persistence), or its IOR contains a well-known address for the server process (direct persistence). Possible values are <code>indirect_persistent</code> or <code>direct_persistent</code>. Defaults to <code>indirect_persistent</code>.</li> <li>• <code>proxified</code> specifies whether service is registered with the Firewall Proxy Server. Possible values are <code>true</code> or <code>false</code>. This attribute is optional. Defaults to <code>false</code>.</li> <li>• <code>managed</code> specifies whether service is registered the management service. Possible values are <code>true</code> or <code>false</code>. Defaults to <code>false</code>.</li> <li>• <code>authenticated</code> specifies whether the service is registered with the security service. Possible values are <code>true</code> or <code>false</code>. Defaults to <code>false</code>.</li> <li>• <code>perflog</code> specifies whether the service is configured for performance logging. This is necessary for integration with Enterprise Management Systems (for example, IBM Tivoli). Possible values are <code>true</code> or <code>false</code>. Defaults to <code>false</code>.</li> </ul>

**Table 2:** *Profile-Specific Elements*

Element	Description
<dd:endpoint>	<p>Specifies details of a service communication endpoint. It has three attributes:</p> <ul style="list-style-type: none"> <li>• <code>protocol</code> specifies the protocol used by the service. Possible values are <code>iiop</code> and <code>http</code>, as well as <code>fps&lt;n&gt;</code>, where <code>&lt;n&gt;</code> is the number of the proxy group. This protocol is only used by the Firewall Proxy Service to indicate its proxy ports.</li> <li>• <code>port</code> specifies the port number used by the service (for example, <code>9000</code>).</li> <li>• <code>secure</code> specifies if the endpoint is secure. Values are <code>true</code> or <code>false</code>. A secure endpoint is one that includes TLS (Transport Layer Security). For example, if <code>secure="true"</code> is set on an endpoint where <code>protocol="http"</code>, a <code>https</code> endpoint is configured.</li> </ul>

**Table 2:** *Profile-Specific Elements*

Element	Description
<code>&lt;dd:configuration&gt;</code>	<p>Specifies configuration overrides for the service. This enables you to change a small number of configuration settings in your domains, at the scope of a service, without modifying the shared description.</p> <pre> &lt;dd:service name="..." ... &gt;   &lt;dd:configuration     name="variable-name"     value="value" action="set"     stage="preprepare" /&gt;   &lt;dd:configuration     name="variable-name"     action="unset" /&gt;   ... &lt;/dd:service&gt; </pre> <p>Available actions are <code>set</code> and <code>unset</code>. The default is <code>set</code>, so the <code>action</code> attribute can be omitted. Configuration overrides only change the value at the service instance scope.</p>

**Table 2:** *Profile-Specific Elements*

Element	Description
<code>&lt;dd:policies&gt;</code>	<p>Specifies information about any policy overrides for that service. Currently, there is only one available policy:</p> <p><code>address_mode</code></p> <p>Specified values must match those already specified in the <code>&lt;dd:node&gt;</code> element (see <a href="#">“Domain Configuration Elements” on page 17</a>).</p> <p>The following example shows policy overrides for address modes and ORB hostnames:</p> <pre data-bbox="781 652 1201 808"> &lt;dd:service ... &gt;   &lt;dd:policies&gt;     &lt;dd:policy name="address_mode"       value="ip"/&gt;   &lt;/dd:policies&gt; &lt;/dd:service&gt; </pre> <p>For more details on this example, see <a href="#">“Converting to an Orbix 6.1 Descriptor” on page 56</a>.</p>
<code>&lt;dd:component&gt;</code>	<p>Specifies a component for the profile. It has a single <code>name</code> attribute. An example value is <code>demoss</code>.</p>

### Service and component XML Files

`<dd:service>` and `<dd:component>` elements have corresponding XML source documents containing the data needed to deploy the configuration domain. Many of these XML source documents correspond to Orbix services. Other XML documents contain core information that is needed for all configurations.

**Note:** These XML source documents are proprietary IONA documents. These XML source documents and their XML schema are not fully documented and subject to change without notice.

However, to enable you to write and use your own custom XML source documents, a subset of the schema is documented and supported. Custom XML files that comply with this partial schema will continue to work with future versions of Orbix, even though the overall schema may change. For details of the partial schema, see [Appendix A](#).

# Advanced Configuration and Deployment

*This chapter explains advanced custom configuration and deployment features offered by Orbix.*

**In this chapter**

---

The following topics are discussed in this chapter:

<a href="#">Specifying Custom Locations for Domain Files</a>	<a href="#">page 30</a>
<a href="#">Specifying Custom Library Paths</a>	<a href="#">page 38</a>
<a href="#">Using Custom XML Files</a>	<a href="#">page 39</a>
<a href="#">Deploying on Multihomed Host Machines</a>	<a href="#">page 43</a>
<a href="#">Specifying Address Mode Policies</a>	<a href="#">page 48</a>

---

# Specifying Custom Locations for Domain Files

---

## Overview

This section explains how to specify custom locations for all your configuration domain's files by passing properties to the `itconfigure` tool. It includes the following topics:

- [“Configuration domain files”](#).
- [“Command-line properties for custom locations”](#).
- [“Setting all locations on the command line”](#).
- [“Partially setting custom locations”](#).
- [“Redeploying an existing domain”](#).

---

## Configuration domain files

Orbit configuration domain files include `start`, `stop`, and `_env` scripts, domain databases, domain log files, and configuration (`.cfg`) files.

Specifying custom locations for these domain files enables you to use a directory structure such as the following:

- `domains/bin/*_env|start*|stop*`
- `domains/config/*.cfg|cfr-*.cfg`
- `domains/dbs/<domain>/<service>/...`
- `domains/logs/<domain>/...`

---

## Command-line properties for custom locations

By default, domain `start/stop` and environment scripts are stored in the `bin` subdirectory of your `<config_dir>`. Domain configuration files are stored in the `domains` subdirectory in your `<config_dir>`. By default, database files are stored in the `<domain_name>/dbs` subdirectory of your `<var_dir>`. Service log files are stored in the `<domain_name>/logs` subdirectory of your `<var_dir>`.



The default locations for `<config_dir>` and `<var_dir>` are shown in [Table 3](#). These locations can be overwritten using the properties and command-line options to `itconfigure` displayed in [Table 3](#).

**Table 3:** *Properties and Options for Custom Directory Locations*

Location for	Property	Command line option	Default location
Configuration files and scripts for all domains ( <code>&lt;config_dir&gt;</code> )	<code>com.iona.deploy.config.dir</code>	<code>-etc</code>	<b>Windows:</b> %IT_PRODUCT_DIR%\etc <b>UNIX:</b> /opt/etc/iona, \$IT_PRODUCT_DIR/etc or \$HOME/etc
Database and log files for all domains ( <code>&lt;var_dir&gt;</code> )	<code>com.iona.deploy.data.dir</code>	<code>-var</code>	<b>Windows:</b> %IT_PRODUCT_DIR%\var <b>UNIX:</b> /opt/var/iona, \$IT_PRODUCT_DIR/var or \$HOME/var

For more fine-grained control of the location of your domain scripts and files, you can use the properties shown in [Table 4](#).

**Table 4:** *Properties for Custom File Locations*

Location for	Property	Default location
Domain start/stop and env scripts	<code>com.iona.deploy.config.bin.dir</code>	<code>&lt;config_dir&gt;/bin</code>
Domain configuration files	<code>com.iona.deploy.config.domains.dir</code>	<code>&lt;config_dir&gt;/domains</code>
Domain data files	<code>com.iona.deploy.domain.db.dir</code>	<code>&lt;var_dir&gt;/&lt;domain_name&gt;/dbs</code>
Domain log files	<code>com.iona.deploy.domain.log.dir</code>	<code>&lt;var_dir&gt;/&lt;domain_name&gt;/logs</code>

**Note:** If all four properties are specified, values for the `etc` and `var` directories do not need to be specified (their default values are not relevant). However, if either one of these values is not specified, it defaults to a subdirectory of the `var` or the `etc` directory.

### Setting all locations on the command line

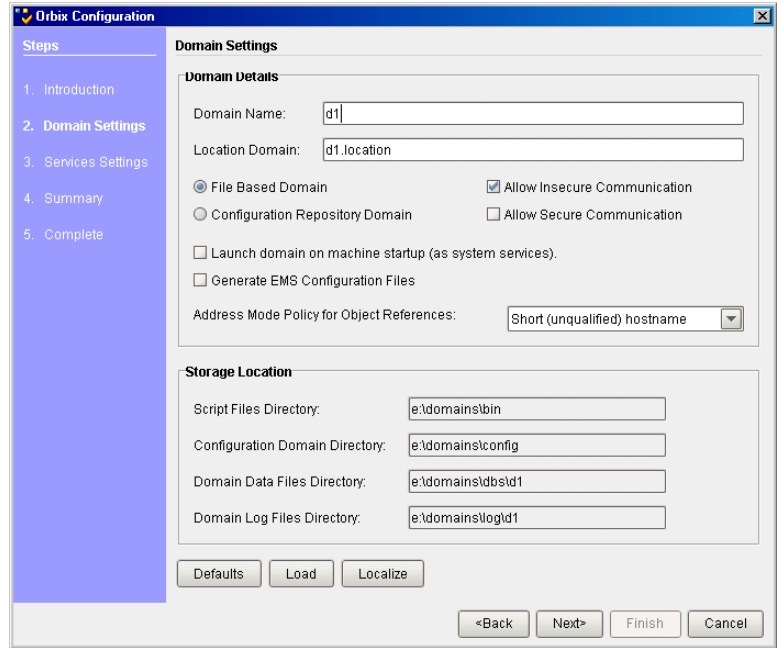
The `itconfigure` command enables you to specify the custom locations for the domain log, data, script and configuration files. The configuration GUI also provides feedback on locations that are passed to `itconfigure` as properties. If all four configuration file locations are set, the GUI does not prompt for the `config` and `var` directories. Instead, it displays the values for these four directories in non-editable text fields.

To deploy your custom locations and also view them in the configuration GUI, perform the following steps.

1. Specify your custom locations to `itconfigure` on the command line, for example:

```
E:\Program Files\IONA\asp\6.1\bin>itconfigure -name dl \
-Dcom.iona.deploy.config.domains.dir=e:\domains\config \
-Dcom.iona.deploy.config.bin.dir=e:\domains\bin \
-Dcom.iona.deploy.domain.db.dir=e:\domains\dbs\dl \
-Dcom.iona.deploy.domain.log.dir=e:\domains\log\dl
```

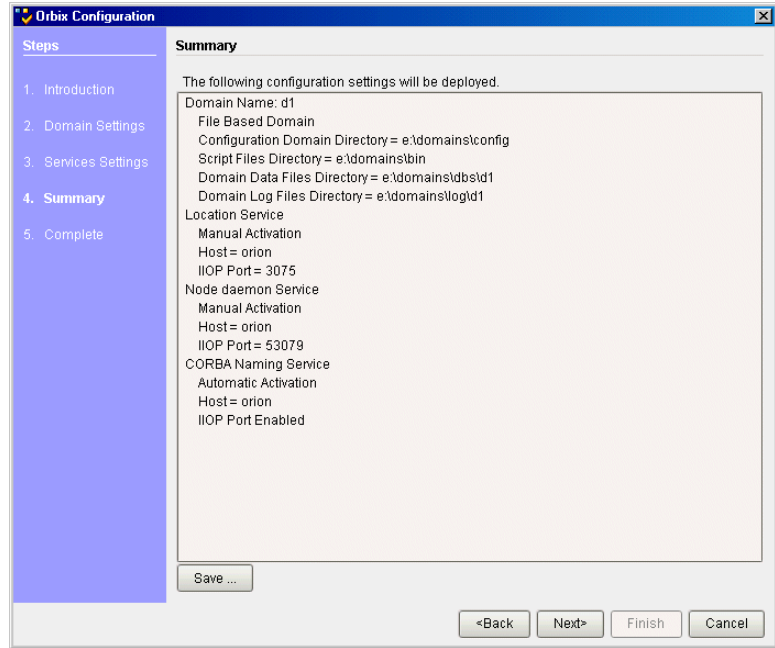
This launches the configuration GUI. You can proceed to deploy your domain as usual.



**Figure 6:** Custom Domain Settings

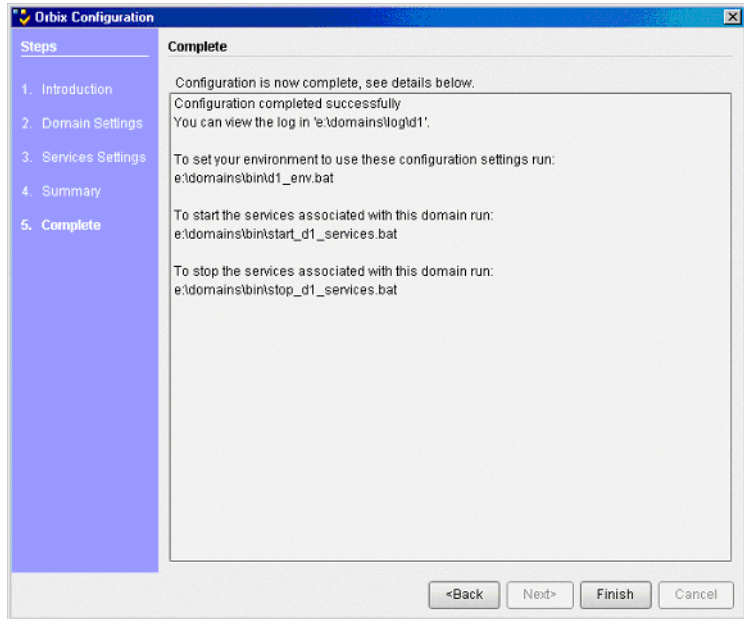
2. Click **Expert** to use the advanced configuration wizard. This displays your custom locations in the **Domain Settings** screen, shown in [Figure 6](#).
3. Click **Next** and select the domain services (for example, locator, node daemon and naming service).

4. Click **Next**. The **Summary** screen is shown in [Figure 7](#):



**Figure 7:** Custom Domain Summary

5. Click **Next**. After all services have been deployed, the **Complete** screen displays the custom locations for your environment scripts, shown in [Figure 8](#).



**Figure 8:** *Custom Configuration Locations*

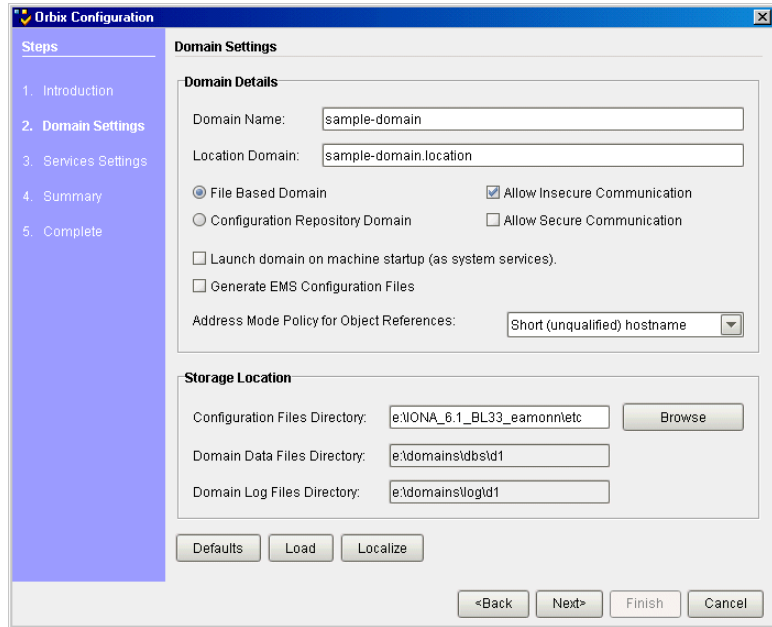
### Partially setting custom locations

If not all four custom locations have been set, a value for the configuration and/or data directories is required, so that the missing value can be replaced with a subdirectory of the configuration or data directory. The GUI displays the configuration and data directories in editable text fields, and displays the directories that have already been set in non-editable text fields.

For example, specify the following on the command line:

```
E:\Program Files\IONA\asp\6.1\bin>itconfigure \
-Dcom.iona.deploy.domain.db.dir=e:\domains\dbs\d1 \
-Dcom.iona.deploy.domain.log.dir=e:\domains\log\d1
```

This will be displayed in the **Domain Settings** screen, shown in [Figure 9](#). You can select the default configuration directory (`e:\Program Files\iona\etc` in this case), or overwrite this value with a custom location.



**Figure 9:** *Partially Set Custom Locations*

If you click **Next** and continue to select and deploy services, your domain files will be located as follows:

service and deployer log files	<code>e:\domains\log\d1</code>
databases	<code>e:\domains\dbs\d1</code>
scripts	<code>e:\Program Files\iona\etc\bin</code>
configuration files	<code>e:\Program Files\iona\etc\domains</code>

**Note:** If the `etc` directory does not exist and needs to be created, `itconfigure` requires your confirmation. However, it does not require confirmation to create the domain log and domain database directories.

**Redeploying an existing domain**

---

Before deploying, the `itconfigure` tool checks for existing scripts in the `bin` directory, configuration files (and sub-directories named `<domain_name>`) in the `domains` directory, databases in the `data` directory, and logs in the `log` directory.

If any such files exist, this indicates that a domain with the same name already exists. The `itconfigure` tool only continues and deletes the existing files after your confirmation. This has the same effect as in the default case. For example, domain log files and domain databases are located in `<var_directory>/<domain_name>/dbs` and `<var_directory>/<domain_name>/logs`. Only the sub-directories are deleted, leaving the `<var_directory>/<domain_name>` directory.

---

# Specifying Custom Library Paths

---

## Overview

This section explains how to specify a custom library path using a command-line option to the `itconfigure` command.

This feature enables you to put shared libraries in different directories and still deploy, without needing to change system defaults that may need root/administrator permissions.

---

## Using the `-libs` option

The `-libs` (shorthand `-L`) option to the `itconfigure` command has the following syntax:

```
-libs <library-path>
```

or

```
-L <library-path>
```

Specifying this option causes `itconfigure` to pass the supplied library path to the deployer. The deployer then prepends the path to the built-in path used when preparing and running Orbix services.

The library path argument is a list of directories to be searched for shared libraries when a service is run. The syntax of the list is the same as the platform-specific path syntax, as shown in the following examples.

### UNIX:

```
itconfigure -load sample_dd.xml -libs /usr/my_libs:/home/me/lib  
-nogui
```

### Windows:

```
itconfigure -load sample_dd.xml -libs c:\usr\my_libs;d:\me\lib  
-nogui
```



---

# Using Custom XML Files

---

## Overview

This section explains how to automate the process of deploying an Orbix configuration domain, and subsequently adding or modifying some of its configuration data (for example, adding a scope for a service developed at your site).

In previous versions of Orbix (for example 5.x), you could only do this by manually modifying the `ABDriver.dtd` and `<domain_name>_driver.xml` files generated by the `itconfigure` tool. Orbix 6.0.2 and higher enable you to do this by passing a system property to the `itconfigure` command. This section includes the following topics:

- [“Passing custom XML to itconfigure”](#).
- [“Deploying custom XML with the GUI”](#).
- [“Custom XML example”](#).
- [“Rules for writing XML files”](#).

---

## Passing custom XML to itconfigure

To use custom XML files, you must first supply the path to the directory containing your files to the `itconfigure` tool. You can do this by passing a system property to `itconfigure`, for example:

```
itconfigure -Dcom.ionadeploy.custom.xml.dir=e:\custom\conf
```

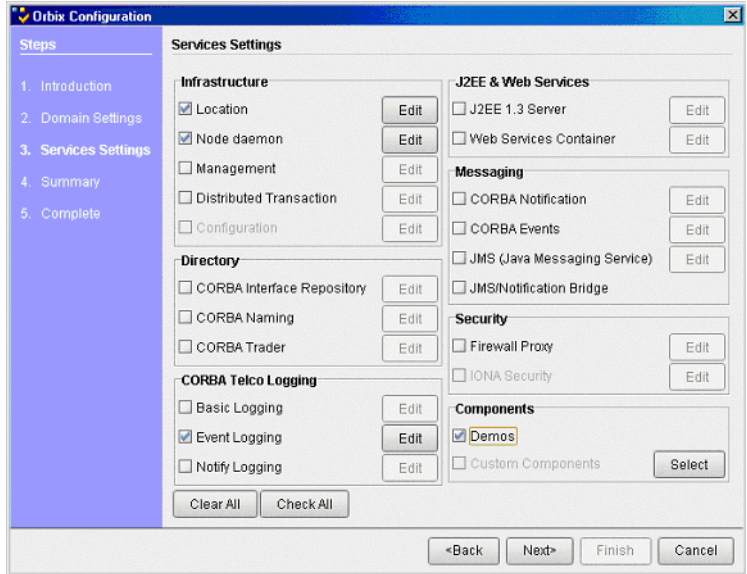
The specified directory should exist and contain at least one file with the `.xml` extension.

---

## Deploying custom XML with the GUI

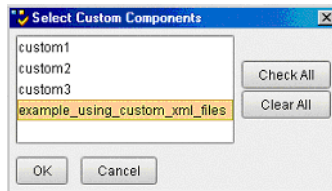
To deploy custom XML files, perform the following steps:

1. Run the configuration GUI and click **Expert**.
2. Click **Next** to display the **Services Settings** screen, shown in [Figure 10](#). The **Custom Components** checkbox at the bottom right of the screen is disabled. This is unchecked when no custom components are selected.



**Figure 10:** Custom XML Components

3. Click the **Select** button on the right of the **Custom Components** checkbox to display the **Select Custom Components** dialog, shown in [Figure 11](#). This enables you to select components from your specified directory.



**Figure 11:** Select Custom Components

4. Click **OK**. The **Custom Components** checkbox is then displayed as checked.

## Custom XML example

For example, if you select the custom XML file with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ABDeploy SYSTEM "ABDeploy.dtd">
<ABDeploy>
  <service>
    <service>
      <dataId>example_using_custom_xml_files</dataId>
    </service>
  </service>

  <process>
    <stage action="filePopulate">
      <source>
        <Dsection>main</Dsection>
      </source>
    </stage>
  </process>

  <section name="main">
    <configScope>
      <dataId>custom</dataId>
    </configScope>

    <configData scope="custom">
      <dataId>custom:example:var</dataId>
      <dataType>list</dataType>
      <dataValue>This</dataValue>
      <dataValue>is</dataValue>
      <dataValue>just</dataValue>
      <dataValue>an</dataValue>
      <dataValue>example!</dataValue>
    </configData>
  </section>
</ABDeploy>
```

Then the generated configuration will include the following fragment:

```
custom
{
  custom:example:var = ["This", "is", "just", "an",
"example!"];
};
```

**Note:** If you select more than one custom component in the GUI, the order in which they are deployed is non-deterministic. Do not make any assumptions about the order in which custom components are deployed, except that they will be deployed after all Orbix services and components (such as demos).

### Rules for writing XML files

If you must write your own XML files, you should obey the following rules:

- Only use a simple service element (i.e., one with just a `dataId` child).
- Use simple process elements and stages with one of the following actions only: `filePopulate`, `configPopulate`.
- Do not use constraints.
- Use `configData` elements with a `dataType` of `list`, `string`, or `long`.
- Do not use external entities.

**WARNING:** The schema for the Orbix deployer XML files is not fully documented. A subset of the complete DTD is supported and documented. Unsupported features are subject to change without notice. For details, see [Appendix A](#).

---

# Deploying on Multihomed Host Machines

---

## Overview

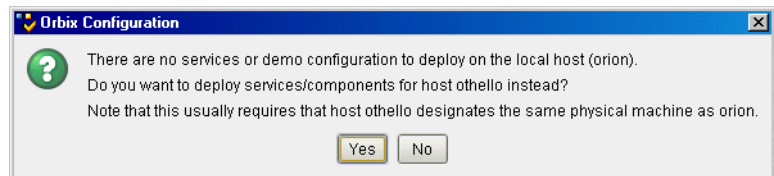
This section explains how to deploy Orbix services for a virtual or additional network interface on a localhost machine. It includes the following topics:

- [“Background to multihomed deployment”](#).
  - [“Deploying on the command line”](#).
  - [“Deploying with the GUI”](#).
  - [“Modifying hostnames without the -multihome option”](#).
- 

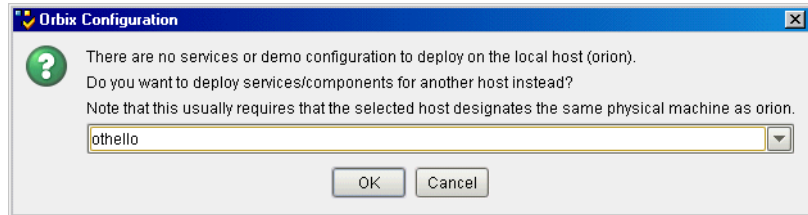
## Background to multihomed deployment

When you click the configuration tool **Finish** button, the behavior in GUI mode is identical with command line mode. The `itconfigure` tool has a descriptor loaded in memory, and must now decide which services to deploy. This selection process is as follows:

- `itconfigure` looks for the node that matches the localhost IP address—as obtained by `InetAddress.getLocalHost().getHostAddress()`.
- If a match is found, `itconfigure` deploys the services specified in this node’s profile.
- If no match is found, `itconfigure` displays either [Figure 12](#) or [Figure 13](#), depending on the total number of nodes specified in the descriptor:



**Figure 12:** *Dialog for Two Nodes*



**Figure 13:** *Dialog for More than Two Nodes*

You can choose to deploy the services for one of these nodes. However, be aware that this will only succeed if you are on a multihomed machine, and the selected node IP address/name maps to another non-default physical interface on this machine. The only exception to this is when there is no interaction between the services on the selected node.

Using the JDK 1.3.1 API, it is not possible to detect that two network interfaces belong to the same physical host. This means that `itconfigure` cannot resolve this conflict on its own and hence requires your confirmation. This confirmation tells `itconfigure` to behave as if it were running on the selected node, in which case it would find the matching profile and proceed to deploy this profile's services.

### Deploying on the command line

On the command line, you can avoid the confirmation step described in [“Background to multihomed deployment”](#) by specifying to `itconfigure` in advance which node to consider as its local node.

If the descriptor has a `<dd:node>` name attribute set to the name of a virtual or additional network adapter (for example: `charlie`), you can deploy the services for this node using the following command:

```
E:\Program Files\IONA\asp\6.1\bin>itconfigure -nogui -multihome
charlie -load <descriptor>
```

The `-multihome` option causes `itconfigure` to look for a node matching the one specified as the `-multihome` value instead of finding one that matches the localhost's default IP address.

### Deploying with the GUI

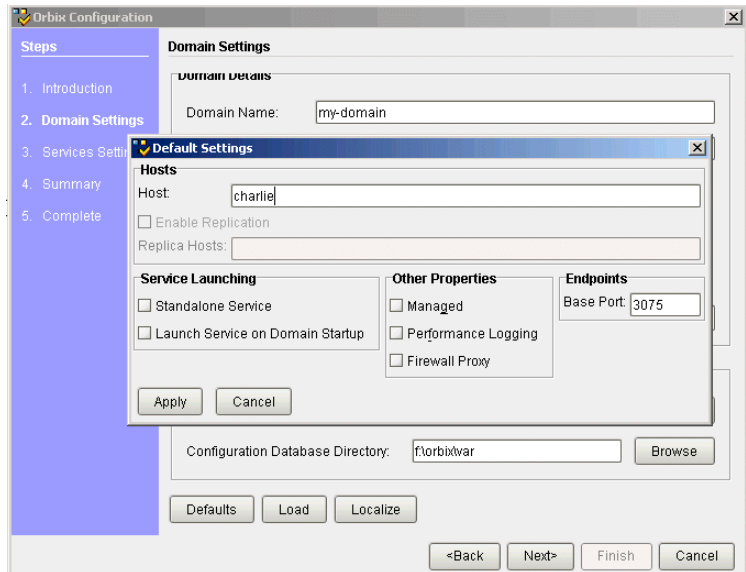
In GUI mode, you can also use the `-multihome` option to avoid the confirmation step described in [“Background to multihomed deployment”](#). In addition to looking for matches with the host specified with the `-multihome` option, when you click **Finish**, `itconfigure` also initializes the default host

with the one specified with the `-multihome` value. Unless you manually alter the host field(s), no question dialog appears when you click **Finish** because there is no conflict to resolve.

To deploy the services using the GUI, perform the following steps:

1. Enter the following command:

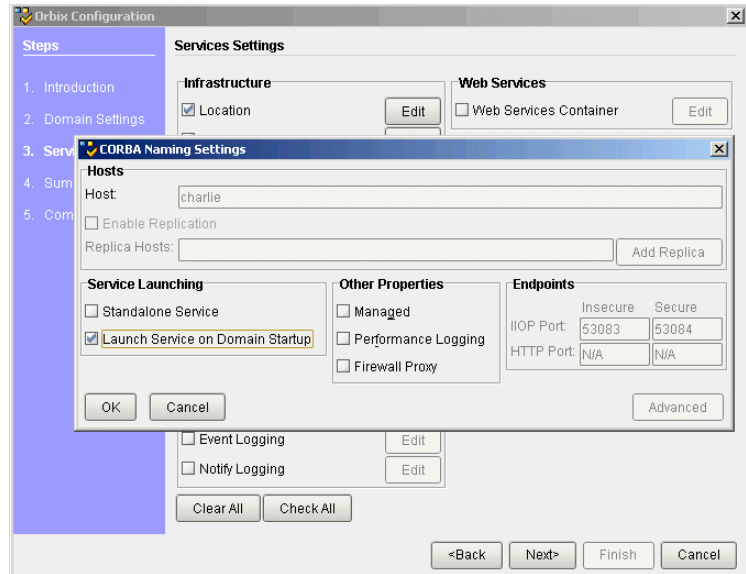
```
E:\Program Files\IONA\asp\6.1\bin>itconfigure -multihome charlie
```



**Figure 14:** *Multihomed Hostname*

2. Select **Expert** mode, and click **Defaults**. Because you selected the `-multihome` option, the default host is already set to `charlie`, shown in [Figure 14](#).

3. Click **Next**, select your domain services, and edit a service's details. The services are automatically selected for deployment on host `charlie`, shown in [Figure 15](#).



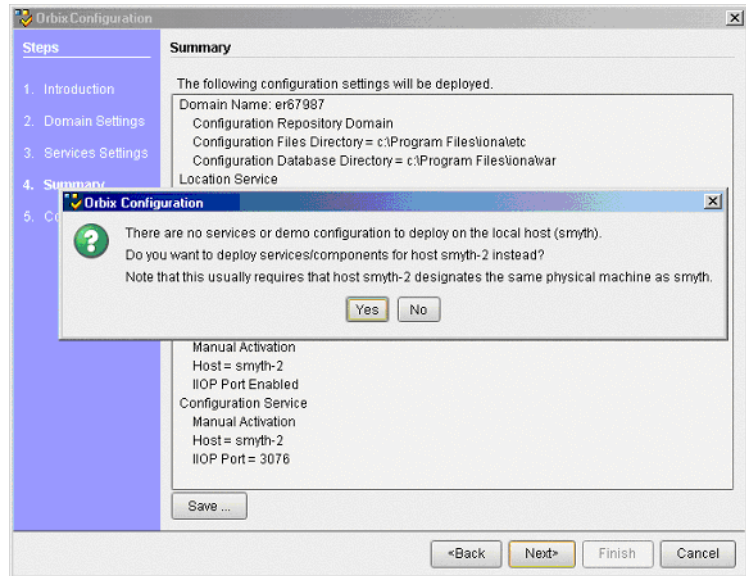
**Figure 15:** Services Selected on Multihomed Host

4. Close the dialog, click **Next**, and **Next** again. `itconfigure` deploys the service. Specifying the `-multihome` option tells `itconfigure` that it should act as if it were running on `charlie`.



### Modifying hostnames without the -multihome option

If you do not use the `-multihome` option, and you modify the content of the `hosts` field (for example, from `smyth` to `smyth-2`), `itconfigure` displays the question shown in Figure 16.



**Figure 16:** *Multihomed Message*

If you click **Yes**, the services are deployed. The last sentence in this message shows that this conflict can also arise when the host is a truly remote machine, and forcing local deployment would not make sense.

---

# Specifying Address Mode Policies

---

## Overview

This section explains how to use address mode policies to control the way in which host names and/or IP addresses are published in IORs. In previous versions of Orbix, you could do this by literally specifying the host DNS alias or IP address. Orbix 6.0 and later use policies. These are portable and enable you design your configuration domain on one host (run `itconfigure` in GUI mode and save the descriptor), and deploy it elsewhere, without the need to supply actual hostnames or IP addresses at that later stage.

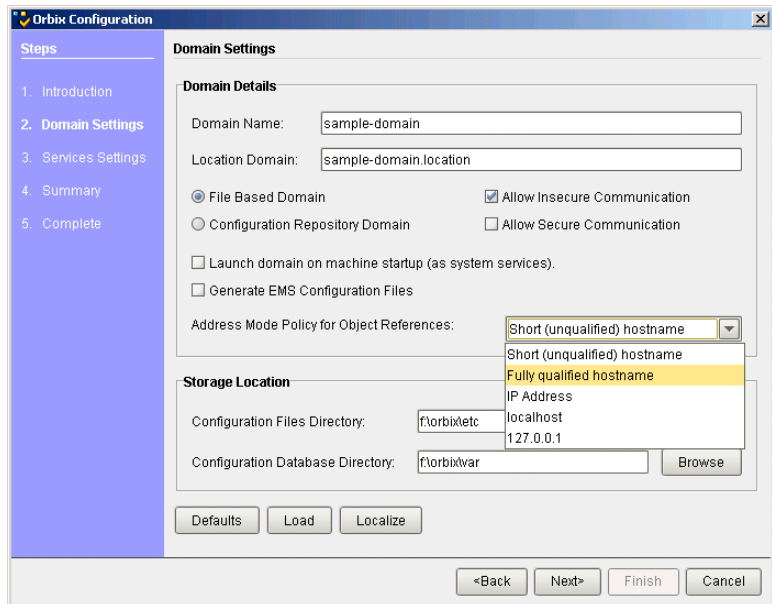
This section includes the following topics:

- [“Selecting an address mode”](#).
- [“Specifying a fully-qualified hostname”](#).
- [“Persistence of address mode policies”](#).
- [“Restrictions and special cases”](#).

## Selecting an address mode

To select an address mode, perform the following steps:

1. Run the configuration GUI and click **Expert**. This displays the **Domain Settings** screen.
2. Select your preferred policy using the **Address mode policy for Object References** drop-down box, shown in [Figure 17](#).



**Figure 17:** Selecting an Address Mode Policy

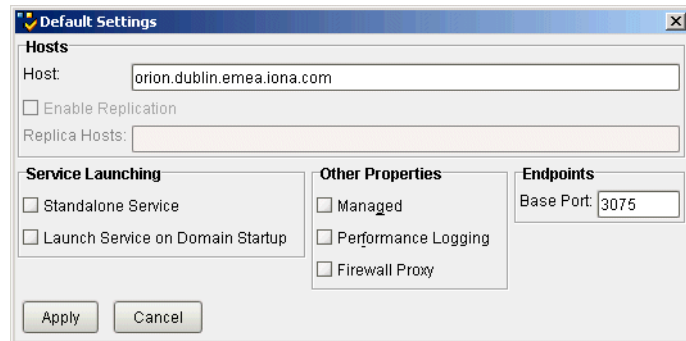
## Specifying a fully-qualified hostname

To use fully qualified hostnames in IORs, you must ensure that `itconfigure` knows the fully qualified host name. Depending on your network configuration, this cannot always be obtained with JDK 1.3 APIs.

However, you can do this by invoking the `itconfigure` command using the `-host` option, for example:

```
itconfigure -host orion.dublin.emea.iona.com
```

Alternatively, you can edit the host field in the **Default Settings** dialog. This opens when you click **Defaults** on the **Domain Settings** screen, shown in [Figure 18](#):



**Figure 18:** *Specifying a Hostname*

### Persistence of address mode policies

If you chose not to deploy now, and save the descriptor to deploy on other hosts, you can still use the selected address mode policy on the other hosts because the policy is persisted by the descriptor.

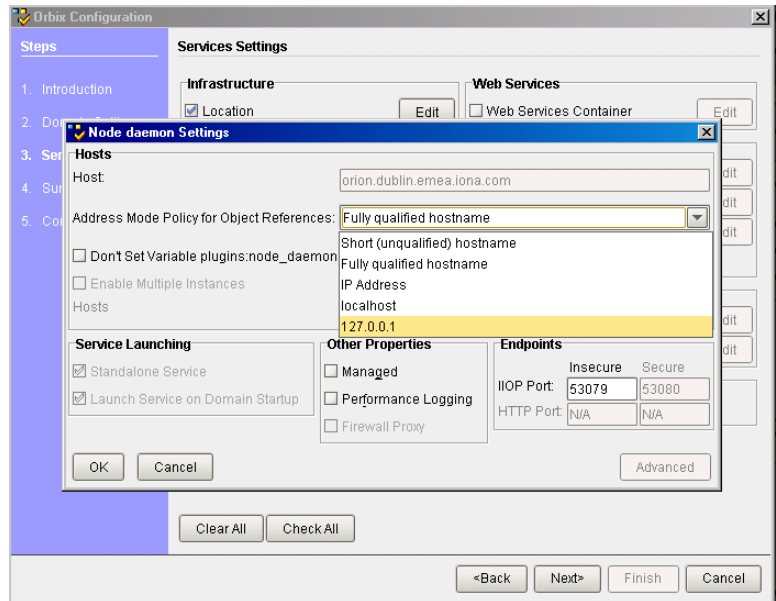
The descriptor stores addresses as policies (instead of literal string IP addresses or names). This enables you to apply the same policy on other hosts, using the `-localize` option to `itconfigure`. For more information, see [“Deploying on a Machine without a GUI” on page 11](#).

### Restrictions and special cases

While the deployment descriptor schema supports node-specific address mode policies, the `itconfigure` GUI only allows you to specify the address mode policy on a global level—for all nodes.

If you must use different policies on different nodes, please refer to [Chapter 2](#), and manually edit the descriptor. The same applies if you want one more level of granularity and specify address mode policies on a per-service basis. There is one case, however, where `itconfigure` allows you to specify service-specific address mode policies:

**Specific address mode policies for the node daemon:** The **Node Daemon Settings** dialog, shown in [Figure 19](#), enables you to specify the address mode policy for node daemons:



**Figure 19:** *Node Daemon Settings Dialog*

Therefore, if you want all services, except for the node daemon, to publish fully qualified host names, you must first change the global address mode policy to `fully qualified hostname`. For the node daemon, you can override this with the `localhost` IP policy (`127.0.0.7`).

**Note:** This policy will be used for all node daemon in the domains. `itconfigure` does not allow you to interactively specify the node daemon's address mode policy on a per instance basis.

It is also possible (although not recommended) to avoid giving the node daemon an explicit name. The default is `iona_services.node_daemon.<hostname>`, the ORB name. You can do this by checking the box labelled **Don't set variable plugins:node\_daemon:name**, displayed in [Figure 19](#).

If a node daemon does not have a name assigned to it in the configuration, on startup, it will register itself with the locator and identify itself to the locator as node daemon named *<host>*, where *<host>* is the value the node daemon obtains by a call to the `gethostname()` function. Obviously, this value will depend on the host on which the node daemon is started.

It is important to note that this may break the mapping between process and node daemon. A process that was registered to be monitored and started on demand by node daemon *<activating host>* can only be activated if a node daemon with the name *<activating host>* exists. In addition, generated start and stop scripts will not be able to stop such a node daemon.

# Migrating from Orbix 5.1 Deployments

*For users who have modified Orbix 5.1 driver files, this chapter explains how to migrate to Orbix 6.1.*

---

**In this chapter**

The following topics are discussed in this chapter:

<a href="#">Migrating from Orbix 5.1 Driver Files</a>	<a href="#">page 54</a>
<a href="#">Converting to an Orbix 6.1 Descriptor</a>	<a href="#">page 56</a>

---

# Migrating from Orbix 5.1 Driver Files

---

## Overview

This section explains how to migrate from Orbix 5.1 driver files to an Orbix 6.1 deployment descriptor. This applies to customers who have modified `ABDriver.dtd` and/or `<domain>_driver.xml` files. It includes the following topics:

- [“Approach to migration”](#).
  - [“Using the itconfigure command line”](#).
  - [“Using the itconfigure GUI”](#).
  - [“Migrating custom XML”](#).
- 

## Approach to migration

The approach used is to generate an Orbix 6.1 deployment descriptor on the fly by retrieving the domain topology (selected domain services) from the driver file, and the service details (for example, port numbers) from the `ABDriver.dtd` file. This descriptor is then passed to the `itconfigure` tool, as if it had been created by `itconfigure`.

The implementation is limited to driver files for domains without replicated services. Driver file entries with the component attribute `role=replica` result in an exception. The deployer also rejects driver files for link domains (links can always be re-created), and driver files for domains that include a J2EE application server.

---

## Using the itconfigure command line

For example, to generate an Orbix 6.1 deployment descriptor using the command line, enter the following:

```
E:\Program Files\IONA\asp\6.1\bin>itconfigure -nogui -compatible \  
-load e:\drivers\my-domain_driver.xml \  
-entities e:\drivers\ABDriver.dtd \  
-etc e:\etc -var e:\var
```

---

## Using the itconfigure GUI

For example, to generate an Orbix 6.1 deployment descriptor using the configuration GUI, enter the following:

```
E:\Program Files\IONA\asp\6.1\bin>itconfigure -compatible \  
-load e:\drivers\my-domain_driver.xml \  
-entities e:\drivers\ABDriver.dtd \  
-etc e:\etc -var e:\var
```



The services specified in the driver file are displayed as selected in the GUI, with their service details as specified in the `ABDriver.dtd` file. You can subsequently add more services, or change the details for the pre-selected services in the GUI, before proceeding to deploy the domain.

---

### Migrating custom XML

Migration can also be used in conjunction with custom component files (see [“Using Custom XML Files” on page 39](#)).

If your Orbix 5.1 driver files specify one or more components that are not recognized as Orbix components, and you pass the directory containing these XML files using the `-Dcom.iona.deploy.custom.xml.dir` property, the deployment will also include your custom components.

If you use `itconfigure` in GUI mode, and chose to save the descriptor, this descriptor also includes your custom components.

---

# Converting to an Orbix 6.1 Descriptor

---

## Overview

This section explains how an Orbix 6.1 deployment descriptor is constructed from an Orbix 5.1 `<domain_name>_driver.xml` and `ABDriver.dtd` file. This includes the following topics:

- “Step 1—Construction of an empty descriptor”.
- “Step 2—Parsing of driver files and construction of node profiles”.
- “Step 3—Obtaining the service details from `ABDriver.dtd`”.
- “Step 4—Obtaining the address mode policy”.
- “Example conversion”.
- “Conversion for virtual hosts”.
- “Adding new Orbix 6.1 features”.

---

## Step 1—Construction of an empty descriptor

An empty deployment descriptor is constructed with a domain name and location domain name as found in `ABDriver.dtd`.

If no definition for the `config.domain.name` entity is found, an exception is thrown. If no value for the `location_domain_name` entity is found, the Orbix 6.1 default is used (`<domain_name>.local`). Initially, the domain type is file-based.

---

## Step 2—Parsing of driver files and construction of node profiles

The `<domain_name>_driver.xml` files are parsed to enable construction of service entries for the deployment descriptor’s local node profile. Any constraints and the ordering of the driver file entries are ignored. Orbix 6.1 does not depend on the order of the entries in a deployment descriptor when deploying a domain—it automatically constructs it correctly. Driver component entries are processed as follows:

- A component named `config_rep.xml` causes the descriptor’s domain type to be changed to CFR based, and adds a service element into the descriptor’s local node profile.
- Components named `init.xml`, `init_svcs.xml`, `file_core.xml`, `file_svcs.xml`, `comet.xml`, `admin.xml`, `tool_corba.xml` are ignored.
- A component named `link.xml` results in an exception (no conversion of driver files for link domains).

- A component with the `role` attribute set to `replica` results in an exception (no conversion of driver files for domains with replicas).
- A component named `demons.xml` results in a `component` element being added to the descriptor's local node profile.
- All other components, provided they are known Orbix components, result in a `service` element being added to the descriptor's local node profile. If they are not known Orbix components (for example, `custom.xml`), a `component` element is added to the descriptors local node profile.

### Step 3—Obtaining the service details from `ABDriver.dtd`

For every driver component entry for which a corresponding service element has been added to the descriptor's local node profile, `ABDriver.dtd` is consulted to determine the service details:

**Direct/Indirect Persistence:** `cfr`, `management`, `locator` and `node_daemon` service elements are always set to be direct persistent—regardless the constraints in the driver component element and the content of `ABDriver.dtd`.

For all other services, if the `<service_name>.direct_persistence` entity is defined in `ABDriver.dtd`, and if its value is `true` or `yes`, the service is set to be direct persistent. The default for a service element is indirect persistent.

**Start Mode:** `cfr`, `management`, `locator` and `node_daemon` service elements are always set to be started manually—regardless the constraints of the driver component element and the content of `ABDriver.dtd`.

For all other services, if the `<service_name>.mode` entity is defined in `ABDriver.dtd`, and if its value is `manual` or `boot`, the service is set to be started manually (default for a service element is on demand).

Subsequently, if the `config.daemon.install` entity is defined in `ABDriver.dtd` and if its value is `true`, the startup mode of a service is promoted to system service, if it had been manual. On Windows it is installed as an NT service.

**Ports:** If the component's security attribute in the `<domain_name>_driver.xml` file is set to `iiopOnly` or `iiopTls`, and if the `<service_name>.port` entity is defined (is a number and not zero), an endpoint element is created in the corresponding service element in the descriptor.

If the component's security attribute in the `<domain_name>_driver.xml` file is set to `iioptls` or `tlsOnly`, and if the entity `<service_name>.tls.port` is defined (is a number and not zero), a secure endpoint element is created in the corresponding service element in the descriptor.

If no port entities can be found for a service (other than the management service) that by now is marked as direct persistent, an exception is thrown.

For the management service, the `<domain_name>_driver.xml` and `ABDriver.dtd` files may have specified this as an indirect persistent service, and therefore no non-zero IOP ports for the management service are defined in `ABDriver.dtd`. Instead of throwing an exception, default endpoints elements are created in the descriptor (IOP port 53086, IOP TLS port 53086, HTTP port 53185, HTTPS port 53186). This is necessary because the management service in Orbix 6.1 is always direct persistent.

Finally, if the `manage_services` entity is defined in `ABDriver.dtd` and if its value is `true`, or if the `<service_name>.managed` entity is defined and its value is `true`, the corresponding service element in the descriptor is set to be managed.

#### Step 4—Obtaining the address mode policy

The default behavior of the deployer towards address mode policies (whether hostnames or IP addresses used in IORs) is to use the unqualified host name, and to assume all services and components are to be deployed on the localhost. The name and IP address of the localhost are obtained by `InetAddress.getLocalHost()`.

If the `host.hostname_for_iors` entity is present in `ABDriver.dtd`, this default behavior is overwritten as follows:

- If the deployer fails to obtain the `InetAddress` of the host identified by the value of the `host.hostname_for_iors` entity (i.e. if `InetAddress.getByName()` throws an `UnknownHostException`), the conversion fails.
- Otherwise the converter creates a `dd:nodes` element in the descriptor, and sets its `dns` attribute set to the DNS domain name. This is obtained from the `InetAddress` object's hostname, after stripping off the first part of the name, so this may be an empty string.

For example, the following entry in `ABDriver.dtd`:

```
<!ENTITY host.hostname_for_iors = "orion.dublin.emea.iona.com">
results in: <dd:nodes dns="dublin.emea.iona.com">
```

If the entity value is an IP address, or an unqualified host name, it depends on your network configuration whether a DNS name is specified.

Next, a `dd:node` element is created as a child of the `dd:nodes` element. The value for the `name` attribute of `dd:node` is obtained as the `hostname` member of the above `InetAddress` object, the value for the `ip` attribute as the `host` address member of the `InetAddress` object. For example, the following entry in `ABDriver.dtd`:

```
<!ENTITY host.hostname_for_iors "10.2.1.101">
```

results in:

```
<dd:nodes>
  <dd:node name="orion" ip="10.2.1.101" profile="orion" />
</dd:nodes>
```

**Rules for inferring the address mode policy:** By comparing the value of the `dns` attribute (of `dd:nodes`), and the values of the `name` and `ip` attributes (of `dd:node`) with the original entity value, the address mode policy is inferred. If this is not `short`, it is stored as a `dd:policy` element under the `dd:node` element. The rules for this process are as follows:

- If the entity value is the literal `localhost`, the address mode policy is set to `localhost`.
- Otherwise, if the entity value is the literal `127.0.0.1`, the address mode policy is set to `localhost_ip`.
- Otherwise, if the entity value matches the value for `ip` attribute on the `dd:node` element, the address mode policy is set to `ip`.
- Otherwise, if the entity value matches the string obtained by concatenating the value of the `name` attribute on the `dd:node` element with (a dot and) the value of the `dns` attribute of the `dd:nodes` element, the address mode policy is set to `long`.
- Otherwise, the address mode policy is `short`.

If the entity value specifies the IP address of the localhost, the value of the `name` attribute on the `dd:node` element may not be identical with the default name of the localhost. This is the case for example, if on the network, IP address `10.2.1.101` is known to belong to host `orion`, but the DNS resolution on `orion` has a different virtual name for this host (for example, `orion-2`).

**Ensuring ORB name compatibility:** By default, the value `dd:node` element's `name` attribute is used to determine host-qualified service ORB names. This may result in different ORB names in the 6.1 domain than those in the 5.1 domain. To prevent this—and to allow for hostnames used in ORB names that are not the name of an existing host (5.1 accepted any string entered in the **What is the unqualified hostname?** text box)—the converter also checks if any of the following entities are defined:

```
cfr.orbname
locator.orbname
node_daemon.orbname
naming.orbname
```

To ensure ORB name compatibility between Orbix 5.1 and Orbix 6.1, the last part of the name in the value of the first entity found—if different from the `dd:node` element's `name` attribute—is also recorded as a policy under the `dd:node` element.

### Example conversion

Assume the following contents of `c:\winnt\system32\drivers\etc\hosts` on host `orion` (IP address 10.2.1.101):

```
127.0.0.1      localhost
orion2
```

and the following in the `ABDriver.dtd` file:

```
<!ENTITY host.hostname_for_iors "10.2.1.101">
<!ENTITY naming.orbname "iona_services.naming.orion">
```

In this case, `InetAddress.getByName("10.2.1.101").getHostName()` returns `orion2`.

And `InetAddress.getByName("10.2.1.101").getHostAddress()` returns `10.2.1.101`.

To ensure that in Orbix 6.1 the same address mode policy and ORB names are used as were in the Orbix 5.1 domain, the descriptor has the following entries:

```
<dd:nodes>
  <dd:node name="orion2" ip="10.2.1.101">
    <dd:policies>
      <dd:policy name="address_mode" value="ip"/>
      <dd:policy name="hostname_for_orbs" value="orion"/>
    </dd:policies>
  </dd:node>
</dd:nodes>
```

## Conversion for virtual hosts

Changes in the conversion process for hostnames and address mode policies ensure that you can migrate 5.1 driver and entity files that used virtual hostnames/IP addresses. See [“Deploying on Multihomed Host Machines” on page 43](#) for more details.

One important difference however is that—while the actual conversion of the driver and entities files from a remote host may succeed as it did in Orbix 6.0.2—subsequent deployment can fail because services may not be able to communicate with each other. For example, a locator is prepared and subsequently started on the localhost (for example, `orion`), but when the node daemon is started it fails to communicate with the locator, which listens on a network address on the remote host. In practice, you should avoid such conversions, because they will not yield the expected results.

**Note:** All other entities (apart from those needed to resolve references in `<domain_name>_driver.xml`) are ignored. In particular, all path related entities (`<service_name>.bin.dir`, and the associated parameter entity `%binDir`) are ignored. Also, address list entities are ignored because the deployer reconstructs that information when processing the generated descriptor.

## Adding new Orbix 6.1 features

Because address mode policies (and hostname policies for the ORB) are now persisted in the deployment descriptors, you can migrate 5.1 domains, and also add Orbix 6.1 features and services to your domains, without losing what has been extracted from the driver and entities files.

The follows steps show how to migrate and add new features at the same time:

1. Convert the driver and/or entities file to a descriptor, without deploying the services, as follows:

```
itconfigure -nogui -compatible -load <driver> -entities
<entities>
```

2. Process the descriptor using proprietary tools to add the new feature (for example, a security service).
3. Deploy the extended descriptor using the following command:

```
itconfigure -nogui -load <extended_descriptor> -etc <etc_dir>
-var <var_dir>
```





# Orbix Deployment DTD

*This appendix lists the supported DTD for the Orbix component XML templates. These XML template files are used to deploy Orbix components and services. The supported DTD is a subset of the complete DTD. Unsupported features are not documented.*

---

**In this appendix**

The following topics are discussed in this appendix:

[“Orbix Component Template Structure” on page 64](#)

# Orbix Component Template Structure

## Overview

The Orbix component XML template documents use a Document Type Definition (DTD) document to define the tags and values that make up the data and the structure the data takes. The DTD defining the configuration data is `ABDeploy.dtd`.

All XML documents used as source for an Orbix configuration must specify `ABDeploy.dtd` as its DTD, and conform to the structure it defines.

**WARNING:** The schema for the Orbix deployer XML files is not fully documented. Only a subset of the complete DTD is supported and documented. Unsupported features are not documented, and are subject to change without notice.

## ABDeploy.dtd

[Example 4](#) shows the subset the `ABDeploy.dtd` file that is supported by IONA. This file defines the structure of configuration XML component templates.

**Example 4:** *The DTD defining Orbix configuration source documents.*

```

<!-- Application Builder Data Deployment Definition -->
<!ENTITY % ABDriver SYSTEM "ABDriver.dtd">
<!ENTITY % DynamicDriver SYSTEM "dynamic_deploy.dtd">
1 <!ELEMENT ABDeploy (service?, process?, section*)>
  <!ELEMENT configData (dataType, (dataValue?))>
  <!ATTLIST configData
    scope CDATA #IMPLIED>
  <!ELEMENT dataId      (#PCDATA)>
  <!ELEMENT dataType    (#PCDATA)>
  <!ELEMENT dataValue   (#PCDATA)>
2 <!-- service -->
  <!ELEMENT service (data)>
3 <!-- process -->
  <!ELEMENT process (stage*)>

```

**Example 4:** *The DTD defining Orbix configuration source documents.*

```

4 <!ELEMENT stage (source*)>
  <!ATTLIST stage
5       action (filePopulate | configPopulate) #REQUIRED>
6 <!ELEMENT source (file?, Dsection*)>
  <!ELEMENT file (#PCDATA)>
  <!ELEMENT Dsection (#PCDATA)>
  <!ATTLIST Dsection
        os NMTOKENS #IMPLIED
        os_family ( unix | windows ) #IMPLIED
        security ( iiopOnly | iiopTls | tlsOnly | is2_iiop |
                  is2_semi | is2_tls) #IMPLIED

7 <!-- section -->
  <!ELEMENT section ((configScope | configData)*)>
  <!ELEMENT configScope (dataId)>

  <!-- -->
  %DynamicDriver;
  %ABDriver;

```

The numbered elements in [Example 4](#) are explained as follows:

1. `<ABDeploy>` is the root element of every Orbix configuration document. It must be the first tag and is required for the document to be valid. `<ABDeploy>` may contain one `<service>` element, `<process>` element, and any number of `<section>` elements.
2. `<service>` specifies information about a service that the deployer needs to deploy it. This element must be present in custom XML files to satisfy the more general syntax. Aside from using its `id` attribute to identify the custom component for your own documentation purposes, it is of no further relevance to custom XML files.
3. `<process>` specifies when and how certain `<section>` elements are processed. May contain any number of `<stage>` elements.

4. A `<stage>` element can reference one or multiple `sections` that may reside in one or more XML files. A `<stage>` element has one or more `<source>` elements.

The `action` attribute of the `stage` determines the target location of the configuration data processed in the `<stage>`. It decides where the configuration data specified in the `configData` elements in the stage's `sections` will be placed.

IONA XML files place configuration data into the configuration domain file or CFR (`action="configPopulate"`), in the handler files (`action="populateHandler"`), secure handler files (with iS2 only, `action="populateSecureHandler"`), or the CFR boot configuration file (`action="configBoot"`).

Most custom XML files will have the `action` attribute set to `"configPopulate"`. The configuration data will be placed in the domain configuration domain file or CFR. An exception is the use of `"populateHandler"` action to overwrite default values of substitution variables.

In the case of file-based domains, there is only one repository for configuration data—the configuration domain file. In that case, the effects of using `"populateHandler"` and `"configPopulate"` are identical. However, they are not identical in CFR based domains.

Finally, a custom XML file's `<stage>` element should rarely ever have more than one `<section>` child element.

5. `<source>` has an optional attribute `file`, the value of which, if specified, indicates in which XML file the sections referred to in the CDATA of the `<Dsection>` child elements can be found. Custom XML files most likely specify their sections locally, so this attribute is not needed.

A `<source>` element can have one or several `<Dsection>` child elements.

6. A `<Dsection>` element is a reference to set of `<configData>` elements which is itself contained in a `<section>`. The `Dsection's` CDATA effectively provides the mapping.

It is an error if a `<Dsection>` element references a `<section>` that cannot be found (in the local file, or in the file denoted by its parent source element).

While `Dsections` in IONA XML files can have `constraint` attributes (meaning the data in the references sections is processed only if the constraint is met), custom XML files should not use these constraint attributes.

7. A `<section>` element is a container for a set of `<configScope>` and/or `<configData>` elements. It has one mandatory `name` attribute, which is used to map to `<Dsection>` elements appearing in the as child elements of a `<process>` element. A `<section>` element must contain at least one `<configScope>` or `<configData>` element.

`<section>` elements are used support multiple installation and configuration scenarios.

## Summary

In practice the full complexity described in [Example 4](#) will rarely be needed. Most custom XML files will provide sufficient functionality if the following conditions are met:

- the `<process>` element contains one `<stage>` element (the `action` attribute of which is set to `configPopulate`).
- the `<stage>` element contains one `<source>` element, without the `file` attribute being set.
- the `<source>` element contains one `<Dsection>` element (without any attributes), and this `<Dsection>` element's CDATA of which is the same as the name of a `<section>` to be found further on in the document.
- the document contains one section element.
- the `<section>` element contains any number of `<configData>` elements.
- a `<configData>` element and its child elements hold the equivalent information to an `itadmin variable create` command—they specify variable scope, name, type and values(s).



# Index

## A

ABDeploy.dtd 4, 64  
ABDeploy element 65  
ABDriver.dtd 54  
authenticated attribute 24

## B

bin directory 30

## C

Complete screen 35  
config.daemon.install entity 57  
configData element 42  
config directory 30  
configPopulate action 42  
configuration  
  file 4  
  overrides 26  
  repository (CFR) 5  
Custom Components checkbox 39  
custom directories 30  
custom XML files 39

## D

dataId element 42  
dataType 42  
dbs directory 30  
dd:activation element 23  
dd:component element 15, 27  
dd:configuration element 15, 18, 26  
dd:descriptor element 15, 17  
dd:domain element 18  
dd:endpoint element 25  
dd:feature element 15, 19  
dd:location\_domain element 18  
dd:node element 15, 19, 59  
dd:nodes element 15, 18, 59  
dd:policies element 20, 27  
dd:policy element 59  
dd:profile element 15, 19  
dd:resource element 20  
dd:run element 24

dd:service element 15, 22  
dd:source element 18  
Defaults button 45  
deployer 4  
deployment descriptor  
  overview 3  
  structure 14  
dns attribute 18  
Domain Details panel 8  
Domain Settings screen 36, 49

## E

Expert button 8, 33

## F

filePopulate action 42  
fps 25  
fully qualified hostname 51

## G

getHostAddress() function 43  
gethostname() function 52  
getLocalHost() function 58

## H

host.hostname\_for\_iors entity 58  
http 25

## I

iiop 25  
implementation repository (IMR) 5  
InetAddress object 58  
interoperable object reference (IOR) 4  
ip attribute 19, 59  
itconfigure 3, 7  
  -compatible option 54  
  -entities option 54  
  -host option 49  
  -libs option 38  
  -L option 38  
  -multihome option 44

## INDEX

- nogui option 16, 44
- passing properties 31
- save option 16

IT\_PRODUCT\_DIR 6

## J

JAVA\_HOME 6

## L

- localhost IP policy 51
- location\_domain\_name entity 56
- logs directory 30

## M

- managed attribute 24
- manage\_services entity 58
- manual attribute 23
- mode attribute 24
- multihomed hosts 43

## N

- name attribute 19, 22, 27, 44

Node Daemon Settings dialog 51

## O

- on\_demand attribute 23

Orbix Configuration GUI 7

## P

- PATH 6
- perflog attribute 24
- port attribute 25
- process element 42, 65
- profile attribute 19
- protocol attribute 25
- proxified attribute 24

## R

- requirements 6

## S

- Save button 10
- section element 67
- secure attribute 25
- Select Custom Components dialog 40
- service element 65

- service-specific address mode 50
- Services Settings screen 8, 39
- source element 66
- stage element 42, 66
- start scripts 30
- stop scripts 30
- Summary screen 9, 34
- system\_service attribute 23





