

# SANKHYA™ Varadhi™

**Cross Platform Middleware**

**User Guide and Reference Manual**

**THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION OF SANKHYA TECHNOLOGIES PRIVATE LIMITED. Use, duplication, and disclosure are subject to license restrictions.**

**Copyright (C) 2001-2003 Sankhya Technologies Private Limited**

**All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means viz., electronic, mechanical, photo-copying, recording, or otherwise, without the prior consent of the publisher.**

**SANKHYA, Varadhi, The Digital Bridge, SANKHYA Software, Varadhi Services and SANKHYA TECHNOLOGIES are trademarks of Sankhya Technologies Private Limited. OMG marks and logos are trademarks or registered trademarks, service marks and/or certification marks of Object Management Group, Inc. registered in the United States. All other brands and names are the property of their respective owners.**

**Part no: 10030104003**

---

---

**SANKHYA™ Varadhi™**

**User Guide & Reference Manual**

**Sankhya Technologies Private Limited**

**Part no: 10030104003**

---

**Table 1: Revision History**

<b>Revision Number</b>	<b>Revision History</b>	<b>Date</b>
001	Updated for Varadhi1.1A	24 Jul 2002
002	Updated for Varadhi1.1B Included Event Service and ns customization.	07 Nov 2002
003	Updated for Varadhi 1.1C	16 Jan 2003

## Preface

**SANKHYA Varadhi** is an Object Request Broker compliant with OMG's CORBA 2.2 (Minimum CORBA) specification.

**Varadhi** Naming Service is compliant with OMG's CosNaming specification 1.0.

The document contains 2 parts:

**Part-1** 'User Guide' provides the user with the basic concepts of CORBA technology and how SANKHYA Varadhi can be used to build a distributed application. It also provides an overview of all the SANKHYA Varadhi utilities like '*vconf*', '*dumpior*', '*ns*' '*nsman*' '*es*' etc.,

**Part-2** 'Reference Manual' helps the user by providing detailed description on all the SANKHYA Varadhi utilities with suitable examples so that the extensive features of Varadhi can be fully utilized.

### Audience

This document intends to provide the user of SANKHYA Varadhi, an overview of Object Management Group's Common Object Request Broker Architecture (CORBA) technology and an in depth coverage on SANKHYA Varadhi Middleware. It explains the basic concepts and terms related to CORBA technology and how SANKHYA Varadhi utilities can be used to create a simple CORBA application.

SANKHYA Varadhi supports C++ binding for OMG's Interface Definition Language (IDL). Therefore the user is assumed to be familiar with programming in C++.

## Notational Conventions

The guide follows the following conventions

- % - The 'percentage' sign denotes a Unix shell prompt.
- > - The 'greater than' symbol represents a Windows command prompt.
- Italic* - The words given in italics represent a SANKHYA Varadhi utility or an option.
- Varadhi code - The guide differentiates the normal text from a program code through this color. Any code or part of a code/ command line statements in this document will therefore be represented using this color.
- ... - Indicates that some portion of the material has been removed to simplify the description.
- [ ] - Indicates an optional argument that can be used in the command line.

# Contents

## Preface

<b>Part 1 - User Guide .....</b>	<b>1</b>
<b>1.1 Introduction .....</b>	<b>1</b>
1.1.1 Overview .....	2
1.1.2 Language Support in Varadhi .....	4
1.1.3 Interoperability.....	5
<b>1.2 Basic Concepts .....</b>	<b>5</b>
1.2.1 Distributed Object Oriented Systems .....	5
1.2.2 Distributed Embedded Systems .....	5
1.2.3 Distributed Object Oriented Embedded Systems Development.....	6
1.2.4 CORBA Technology.....	6
<b>1.3 Developing A Simple CORBA Application .....</b>	<b>8</b>
1.3.1 Separate Interface from Implementation .....	8
1.3.2 Create An IDL File .....	9
1.3.3 Compile IDL code to the language of Client and Server.....	9
1.3.4 Develop The Client Application .....	10
1.3.5 Develop The Server Application .....	11
1.3.6 Compile Client And Server Application.....	14
<b>1.4 Using SANKHYA Varadhi .....</b>	<b>15</b>
1.4.1 Overview Of Development Process.....	15
1.4.2 Setting Up Varadhi Host Development Environment .....	15
1.4.3 Setting up Varadhi Target Platform.....	16
1.4.4 Compiling The IDL Source File .....	17
1.4.5 Develop the Client and Server applications.....	18

1.4.6	Compiling And Linking The Application.....	18
1.4.7	Using Demo Makefiles .....	19
1.4.8	Additional Compiler Flags During Build .....	21
1.4.9	Running Client and Server Application.....	22
1.4.10	ORB run-time command line options.....	22
<b>1.5</b>	<b>Development Tools And Utilities .....</b>	<b>23</b>
1.5.1	IDL Compiler For C++ Mapping.....	23
1.5.2	VCONF Configuration Tool.....	23
1.5.3	DUMPIOR Utility.....	24
<b>1.6</b>	<b>SANKHYA Varadhi - Common Object Service .....</b>	<b>25</b>
1.6.1	Naming Service.....	25
1.6.2	ns - Varadhi Naming Server .....	26
1.6.3	nsman - Varadhi Naming Service Manager.....	27
1.6.4	Event Service .....	27
1.6.5	es - Varadhi Event Service.....	31
<b>1.7</b>	<b>Customizing SANKHYA Varadhi .....</b>	<b>32</b>
1.7.1	Introduction.....	32
1.7.2	Creating a new Varadhi Platform .....	33
1.7.3	Creating a Varadhi Platform for Porting.....	35
1.7.4	Using Varadhi Platform .....	37
	<b>Part 2 - Reference Manual .....</b>	<b>38</b>
<b>2.1</b>	<b>Varadhi IDL Compiler - IDLC .....</b>	<b>38</b>
2.1.1	Standards Conformance.....	38
2.1.2	Synopsis .....	38
2.1.3	Description.....	38
2.1.4	Options.....	39
2.1.5	Usage Examples.....	39

<b>2.2</b>	<b>VCONF Configuration Tool .....</b>	<b>41</b>
2.2.1	Synopsis .....	41
2.2.2	Description.....	41
2.2.3	Options.....	41
2.2.4	Usage Examples.....	42
2.2.5	VCONF Configuration File Format.....	44
2.2.6	VCONF - Interactive Mode of Operation.....	46
2.2.7	Exit Status and Error Messages .....	48
<b>2.3</b>	<b>DUMPIOR Utility .....</b>	<b>51</b>
2.3.1	Introduction.....	51
2.3.2	Synopsis .....	51
2.3.3	Description.....	51
2.3.4	Usage Examples.....	51
<b>2.4</b>	<b>ns - Varadhi Naming Server .....</b>	<b>53</b>
2.4.1	Introduction.....	53
2.4.2	Synopsis .....	53
2.4.3	Description.....	53
2.4.4	Options.....	53
2.4.5	Usage Examples.....	54
2.4.6	Maximum Configured limits .....	55
<b>2.5</b>	<b>Customizing Varadhi Naming Server .....</b>	<b>56</b>
2.5.1	Building customized 'ns' .....	57
2.5.2	Using Naming Service library APIs .....	63
2.5.3	API to shutdown Naming Service .....	65
<b>2.6</b>	<b>nsman - Varadhi Naming Service Manager .....</b>	<b>66</b>
2.6.1	Introduction.....	66
2.6.2	Synopsis .....	66
2.6.3	Description.....	67
2.6.4	Options.....	67

2.6.5	CosNaming::NamingContext operations .....	68
2.6.6	Usage Examples.....	69
<b>2.7</b>	<b>es - Varadhi Event Service .....</b>	<b>81</b>
2.7.1	Synopsis .....	81
2.7.2	Description.....	81
2.7.3	Options.....	81
2.7.4	Usage Examples.....	82
2.7.5	Maximum Configurable limits.....	84
<b>2.8</b>	<b>Varadhi ORB run-time options .....</b>	<b>86</b>
2.8.1	Introduction.....	86
2.8.2	Synopsis .....	86
2.8.3	Description.....	86
2.8.4	Options.....	87
2.8.5	Usage Examples.....	88
<b>2.9</b>	<b>Varadhi Configurations .....</b>	<b>90</b>
2.9.1	Parameter Configurations .....	91
	2.9.1.1 Memory Management Parameters.....	93
2.9.2	Feature Configuration Parameters .....	95
2.9.3	Service Configuration Parameters .....	96
2.9.4	Varadhi Library Configuration Parameters .....	96
<b>2.10</b>	<b>IDLC generated Stub and Skeleton code .....</b>	<b>98</b>
2.10.1	Overview.....	98
2.10.2	Generated Stub and Skeleton Files .....	98
2.10.3	Client Side Interface mapping .....	100
	2.10.3.1 Stub class for an IDL interface .....	100
	2.10.3.2 _var type for Interfaces .....	101
2.10.4	POA Based Server Side Interface Mapping .....	101
	2.10.4.1 Inheritance Based Mapping .....	102
	2.10.4.2 Tie Based Mapping .....	103

2.10.5	Exceptions.....	103
2.10.5.1	CORBA System Exceptions .....	103
2.10.5.1.1	CORBA System Exception Vendor minor codes	104
2.10.5.2	User Exceptions .....	107
<b>2.11</b>	<b>Using corbaloc Object URL in SANKHYA Varadhi .....</b>	<b>108</b>
2.11.1	Registering a Service/CORBA Object with SANKHYA Varadhi ....	108
2.11.2	Exit Status and Error Messages .....	112

**Index**

---

# USER GUIDE

**Sankhya Technologies Private Limited**

---

---

# *Part 1 - User Guide*

---

## **1.1 Introduction**

With increasing CPU power and bandwidth, distributed computing systems combined with the power of embedded systems have become increasingly relevant in the present day scenario. For integrating systems with different attributes, there is a need for a solution that can allow the developer to concentrate on implementing his application logic without worrying about the complexities (such as platform, transport, hardware requirements etc..) involved.

**SANKHYA Varadhi** is Sankhya's object middleware solution for cross platform distributed systems development.

Varadhi provides software developers, the tools and components required to develop distributed software applications that can run either within an organization's Intranet or across the Internet.

Varadhi enables distribution of software across Windows, Linux and Solaris hosts and embedded systems like mobile phones and PDAs.

Varadhi manages the diversity in programming languages, location and host computers thereby enabling application developers to concentrate on implementing the application logic.

### 1.1.1 Overview

SANKHYA Varadhi for C++ includes the following developer tools:

- `idlc` - The IDL Compiler (generates C++ stubs and skeletons)
- `vconf` - Interactive Configuration Tool
- `dumpior` - A tool to view stringified IOR
- `ns` - Varadhi Naming Service
- `nsman` - Varadhi Naming Service Manager
- `es` - Varadhi Event Service

**idlc** generates C++ server skeletons and client stubs from the IDL specification provided by the user. It can also be used to check syntax of IDL files.

**vconf** can be used to interactively configure and generate custom configurations of the ORB and POA.

**SANKHYA Varadhi** includes the following run-time components:

- Object Request Broker (Varadhi ORB)
- Portable Object Adapter (POA)
- Client Stubs (Generated by Varadhi IDL Compiler from IDL)
- Server Skeletons (Generated by Varadhi IDL Compiler from IDL)

**SANKHYA Varadhi SE** (Standard Edition) - Hosts Supported:

- Windows NT and Windows 2000
- RedHat Linux 7.2
- Solaris 2.7

## SANKHYA Varadhi XE (Cross Platform - Embedded Edition)

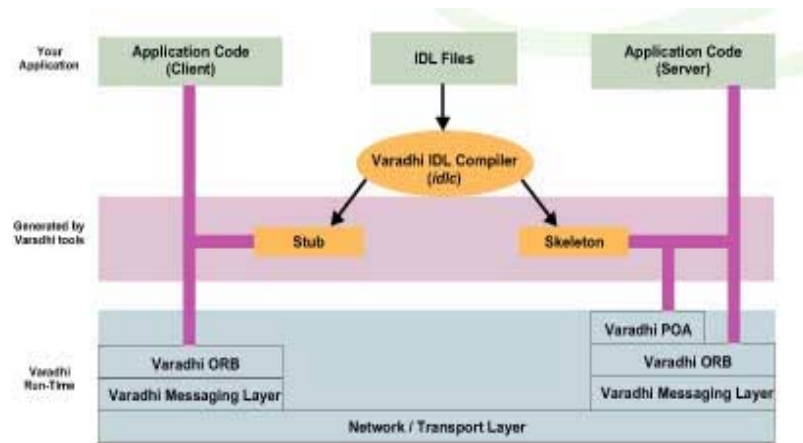
### - Host and Target Platform Support:

- Host: Windows NT; Target: Windows CE (Handheld PC); CPU: x86
- Host: Windows NT or Solaris; Target: OSE; CPU: PowerPC
- Host: RH Linux 7.2; Target: MontaVista Linux; CPU: PowerPC
- Host: Solaris; Target: QNX; CPU: x86

### SANKHYA Varadhi XE - CPU Platform Support:

- Motorola PowerPC
- Intel x86
- MIPS

MIPS Libraries can run without an underlying operating system on a MIPS Malta Board.



## SANKHYA Varadhi - Portable, Extensible and Configurable !

Varadhi has a modular and layered architecture. This provides easy portability, extensibility and configurability of the ORB. Extensibility and

Portability are achieved by abstraction of the Transport and OS layer respectively.

The abstraction of the OS layer allows easy portability to any embedded operating environment or real-time OS. The protocol layer is also abstracted so that Environment Specific Inter-Operable Protocols can be supported.

Varadhi can be configured to use static memory allocation for various ORB objects. This enables the generation of an ORB image that has a statically determined memory foot-print, essential for embedded systems.

SANKHYA Varadhi run-time is provided as a library that can be linked with the application. Varadhi provides support for customization of the run-time for specific requirements. Varadhi provides *vconf*, a configuration utility to configure the Varadhi run-time library.

Varadhi comes with a Naming Service utility and a Naming Service Manager tool. Varadhi Naming Service is compliant with OMG's CosNaming specification 1.0.

Varadhi includes an Event Service utility that is compliant with OMG's Event Service 1.1 specification.

Varadhi also includes '*dumpior*' - a utility to dump the contents of CORBA stringified Interoperable Object Reference.

### **1.1.2 Language Support in Varadhi**

SANKHYA Varadhi supports C++ binding for OMG's Interface Definition Language. Varadhi IDL compiler (*idlc*) generates C++ stub and skeleton code for C++ based Distributed Application development.

### 1.1.3 Interoperability

Varadhi uses OMG's **GIOP** (General Inter-ORB Protocol) as its default messaging protocol over TCP/IP (OMG's **IOP** (Internet Inter-ORB protocol) specification). Any application developed with Varadhi will naturally run with any other CORBA compliant Object Middleware software, that supports **IOP**. With built-in IOP support, Varadhi is interoperable with other ORBs in Distributed Embedded Environments. In fact, Varadhi, with its small memory foot print and support for static allocation of memory is ideally suited for those nodes of the distributed system that are embedded.

To summarize, a Varadhi client can interoperate with any CORBA IOP compliant server implementation. A Varadhi server will support requests from any CORBA IOP compliant client.

## 1.2 Basic Concepts

### 1.2.1 Distributed Object Oriented Systems

In a distributed object oriented system, all services are packaged in the form of an object. Typically, a client obtains a handle or pointer to an object and then invokes various operations through the object handle as shown in the following example:

```
a_router *rp = get_router_by_name("core_router");  
rp->add_route(...);
```

### 1.2.2 Distributed Embedded Systems

A simple application may contain both the server (Object Implementation) and client (Object User) implemented in a single language and within a single process or task. More complex systems, may use different languages

to implement the client and server. In addition, the client and server may run on different computers, which may in turn use different hardware architectures and operating systems.

With embedded systems, there are often additional complexities due to the large number of transports that may be used like serial interface, USB, Ethernet, PCI and shared memory, the large number of diverse CPUs and hardware architectures used, and additional constraints on available memory (DRAM as well as ROM).

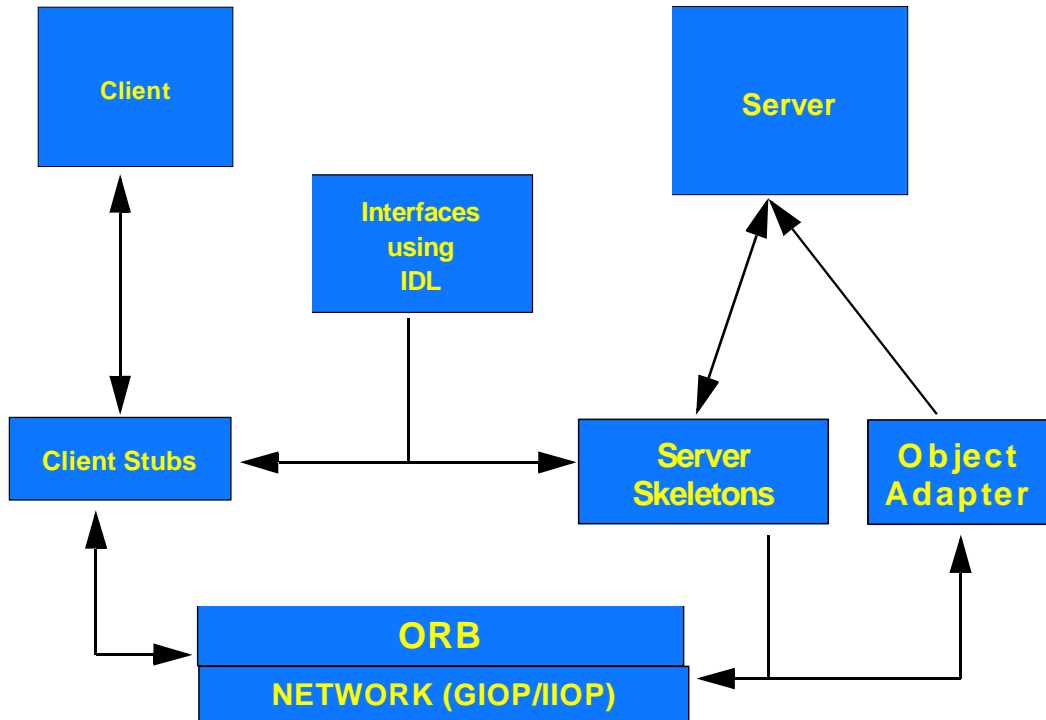
### **1.2.3 Distributed Object Oriented Embedded Systems Development**

SANKHYA Varadhi provides the tools and components required for implementing distributed object oriented embedded systems, which allow the developer to concentrate on implementing the application logic in the client and server, almost completely leaving the complexities of managing language, location, transport and hardware architectural dependencies to Varadhi.

### **1.2.4 CORBA Technology**

The Object Management Group, an association of several hundred companies world wide has standardized distributed object technology in software development through CORBA specifications. The CORBA specification is based on OMG's Object Management Architecture (**OMA**) for object technology. CORBA specification is the standardized solution for reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments.

Using CORBA it is possible to develop applications in heterogeneous environment across all major hardware platforms and operating systems.



### CORBA Architecture

In CORBA, the ORB provides the mechanisms by which objects transparently make requests and receive responses, providing interoperability between applications on different machines in heterogeneous distributed environments.

For embedded applications, CORBA is too large to meet size and performance requirements. Such scenarios require a cut-down version of CORBA. For systems with limited resources, OMG specifies a subset (or profile) of CORBA called “**Minimum CORBA**”.

As the above diagram (CORBA Architecture) shows, the IDL compiler creates the stubs and skeletons for the client and server side respectively. The stubs marshal the parameters from the client side to the ORB.

On the server side, the server decodes the object information sent by the client through the Object Request Broker (ORB), and gets the object reference. The Object Adapter then activates the appropriate servant object using this object reference. The result from the server is sent to the client through the skeleton.

For more information on CORBA architecture, Refer to OMG specifications or visit <http://www.omg.org>

## 1.3 Developing A Simple CORBA Application

### 1.3.1 Separate Interface from Implementation

The first step in developing a Varadhi based application is to clearly separate the interface of an object (For example the set of member function declarations in a C++ class or java class) from its implementation (the actual member function definitions in C++ or Java).

The Object Management Group has standardized a language for describing type interfaces in a programming language independent manner which is referred to as **CORBA IDL** (Common Object Request Broker Architecture - Interface Definition Language).

SANKHYA Varadhi supports OMG's IDL, and provides an IDL compiler (*idlc*) with C++ binding.

### 1.3.2 Create An IDL File

Using IDL Language, a developer can specify an object interface (or more correctly the type interface as more than one object may support a single type interface) in a language independent manner. Here is an example IDL specification:

```
// "Adder" interface
interface Adder
{
    long add(in long x, in long y);
};
```

The above sample interface describes a type 'Adder', containing a single method '*add*' which takes two input parameters '*x*' and '*y*' of long type, and returns a long value. In IDL, long stands for a 32-bit integer. IDL supports the following types of parameter.

- '*in*' - parameters for passing values from client to server.
- '*out*' - parameters for receiving values from server.
- '*inout*' - parameters for passing values to server and also to receive values from server.

### 1.3.3 Compile IDL code to the language of Client and Server

Once the interface is specified in IDL, it can be compiled to the chosen language of client and server application development. Here is an example of how Varadhi's IDL compiler ('*idlc*') with C++ binding can be used to compile the above IDL specification.

```
% idlc add.idl
```

*idl* generates the necessary C++ client stub code and C++ server skeleton code in header and source files in the following way on Unix hosts.

- 1) add\_st.h - Client side stub header file
- 2) add\_st.cc - Client side stub source file
- 3) add\_sk.h - Server side skeleton header file
- 4) add\_sk.cc - Server side skeleton source file

On Windows the files ‘add\_st.cpp’ and ‘add\_sk.cpp’ will be created instead of ‘add\_st.cc’ and ‘add\_sk.cc’ respectively.

### 1.3.4 Develop The Client Application

Once the interface is specified, the client can be implemented. The following shows a sample CORBA compliant client implementation, for the adder example mentioned above.

```
// Include "Adder" stub (client) header file.
#include "add_st.h"

int main( int argc, char *argv[] )
{
    // Initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // Get "Adder" object, Object reference is in file adder.ior
    Adder_var client = get_objref("adder.ior");

    // Invoke operation on the remote "Adder" object
    long l_add = client->add( 10, 25 );
```

```

cout << "Result of Adding 10, 25 = " << l_add << endl;

l_add = client->add( -268435455, 268435453 );
cout << "Result of Adding -268435455, 268435453 = "
    << l_add << endl;
CORBA::release( client ); // Free Adder object.
return 0;
}

```

As the above example code shows, the client implementation is simplified to obtaining an object reference to the server object.

Varadhi locates the server implementation, packages the method invocation in an interoperable CORBA GIOP message, sends the request to the server, receives the response and returns the return value from the server program to the client.

### 1.3.5 Develop The Server Application

The server application can be implemented as follows:

```

// Include "Adder" skeleton (server) header file
#include "add_sk.h"

// The Actual Implementation of Adder in Server
class Adder_impl : virtual public POA_Adder
{
public:
    Adder_impl()
    {

```

```
};

CORBA::Long add(CORBA::Long x, CORBA::Long y)
{
    return (x + y); // Here is the logic for adder::add !
};
};

// Rest of Server Program, to initialize Varadhi and act as a Server
// Daemon ...
int main( int argc, char *argv[] )
{
    // Initialize the ORB

    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // Create, register and activate server object
    Adder_impl* server = new Adder_impl;
    PortableServer::ObjectId_var oid = poa->activate_object(server);
    register_with_poa_and_activate_object(orb, server);
    // Save object reference to a file, so clients can access it.
    // Example only - alternatively object references can be
    // registered with and obtained from name servers as well.
    save_object_reference(server, "adder.ior");

    // Give control to the ORB
    orb->run();
}
```

The server implementation requires the developer to complete the implementation of the application object (*Adder\_impl*) and register it with Varadhi. Once the object is registered, the server gives control to Varadhi, which will automatically handle client requests using the implementation class *Adder\_impl* and send appropriate responses to the client.

The developer needs to tell Varadhi, the existence of each implemented CORBA object. OMG's **Portable Object Adapter**, defines a standard way of interfacing Object Request Brokers (like Varadhi) to Server Implementations (like *Adder\_impl*).

The following shows a sample code in C++ for the above function.

```
// Get a reference to the Root POA.
CORBA::Object_var obj =
    orb->resolve_initial_references("RootPOA");
PortableServer::POA_var poa =
    PortableServer::POA::_narrow(obj);

// Create an "Adder" servant, and
// activate the server Object.
Adder_impl* server = new Adder_impl;
PortableServer::POA::ObjectId_var oid =
    poa->activate_object(server);
```

POA specification allows the Object adapters to be arranged in a tree structure, and the *RootPOA* denotes the root of this POA tree. The above code, registers the *Adder\_impl* object to the *RootPOA*.

### 1.3.6 Compile Client And Server Application

Once the client and server code are implemented, they should be built with IDL compiler generated code. The client application should be linked with the stub code and the server application should be linked with the skeleton code. In addition, the ORB run-time should be linked along with the application in the case of library based ORB.

The following is an example of how Varadhi client and server applications can be compiled using g++ on Solaris or Linux platform.

```
% g++ -o client -I$VARADHI/include add_st.cc client.cc
    $VARADHI_PLATFORM/lib/libvaradhi.a -lnsl -lsocket
```

In the above compilation, *add\_st.cc* is the stub code generated by Varadhi IDL compiler and *libvaradhi.a* is the Varadhi run-time library for the particular Platform.

Similarly, server application can be compiled using *add\_sk.cc* and other server source files. *add\_sk.cc* is the skeleton code generated by Varadhi IDL compiler.

On Windows, compilation using VC++ can be done as below.

```
> cl /Feclient.exe /I%VARADHI%\include
    /DENABLE_CPP_EXCEPTIONS /DWIN32 /TP add_st.cpp
    client.cc /link %VARADHI_PLATFORM%\lib\varadhi.lib
    ws2_32.lib netapi.lib
```

Refer to \$VARADHI/samples/adder directory for working with the adder demo on Unix hosts. On Windows host, refer to

```
%VARADHI%\samples\adder .
```

## 1.4 Using SANKHYA Varadhi

### 1.4.1 Overview Of Development Process

Here is the complete overview of the steps to be followed for developing an application using SANKHYA Varadhi:

1. Setting up Varadhi host development environment
2. Setting up Varadhi target platform
3. Compiling the IDL source file
4. Developing Client and Server applications
5. Compiling and linking the application
6. Running Client and Server applications.

The following sections explain each step in detail.

### 1.4.2 Setting Up Varadhi Host Development Environment

To work with Varadhi, the following environment variables need to be set. Also, the makefiles provided with the samples require these environment variables to be set.

VARADHI	-	Points to root of installation
VARADHI_HOST	-	Host platform name like sol2, linux, win32.

In order to set these variables and PATH variable on Unix, source the Unix shell script *varadhi.csh* in the SANKHYA Varadhi installation root directory.

```
% source <varadhi_root>/varadhi.csh
```

where <varadhi\_root> is the directory where SANKHYA Varadhi is installed.

On Windows host, run `varadhi.cmd` instead. In a Windows Command Prompt, type

```
> <varadhi_root>\varadhi.cmd
```

### 1.4.3 Setting up Varadhi Target Platform

Varadhi Platform is a directory which contains Varadhi run-time libraries and other platform specific files. Some Varadhi platforms may additionally contain sources for porting the Varadhi OS layer and Transport layer to custom target environments.

A Varadhi Platform can be created using "*vconf*", the interactive Varadhi configuration utility. Varadhi is shipped with pre-built Target platform binaries for native host development on Solaris, Linux and Windows NT/2000.

VARADHI\_PLATFORM is the environment variable that points to the root of a Varadhi platform directory. In order to set the above variable on Unix, source the Unix shell script *platform.csh* in the Varadhi platform root directory.

```
% source <varadhi_platform_root>/platform.csh
```

On Windows host, run *platform.bat* present in the Varadhi platform root directory.

In a Windows Command Prompt, type

```
> <varadhi_platform_root>\platform.bat
```

For more information refer to section 1.7 “Customizing SANKHYA Varadhi”.

#### 1.4.4 Compiling The IDL Source File

Use Varadhi's IDL compiler ‘*idlc*’ to compile the IDL file and generate stub and skeleton code.

Invoke the *idlc* IDL compiler for C++ as below.

```
% idlc add.idl
```

where “add.idl” is the IDL source file. For example, add.idl can contain the following IDL code.

```
// "Adder" interface
interface Adder
{
    long add(in long x, in long y);
};
```

For the above, *idlc* generates 4 sets of C++ files that specify the adder interface on Unix hosts (with ‘.cc’ files being replaced by ‘.cpp’ files on MS Windows)

- 1) add\_st.h - Client side stub header file
- 2) add\_st.cc - Client side stub source file
- 3) add\_sk.h - Server side skeleton header file
- 4) add\_sk.cc - Server side skeleton source file

### 1.4.5 Develop the Client and Server applications

For developing the client and server applications refer to sections 1.3.4 and 1.3.5

### 1.4.6 Compiling And Linking The Application

Building an application using Varadhi involves compiling the IDL generated stub and skeleton code and the application code.

The client application should be linked with the stub code and the server application should be linked with the skeleton code. In addition, the ORB run-time, in the Varadhi Platform directory, should be included along with the application.

The following an example of how Varadhi client and server applications can be compiled using g++ on Unix.

```
% g++ -o client -I$VARADHI/include add_st.cc client.cc  
$VARADHI_PLATFORM/lib/libvaradhi.a -lnsl -lsocket
```

In the above compilation, *add\_st.cc* is the stub code generated by Varadhi IDL compiler and *libvaradhi.a* is the Varadhi run-time library for the Varadhi platform.

On Windows, compilation using VC++ can be done as below.

```
> cl /Feclient.exe /I%VARADHI%\include  
/DENABLE_CPP_EXCEPTIONS /DWIN32 /TP add_st.cpp  
client.cc /link %VARADHI_PLATFORM%\lib\varadhi.lib  
ws2_32.lib netapi.lib
```

Similarly, server application can be compiled using *add\_sk.cc* instead of *add\_st.cc*. *add\_sk.cc* is the skeleton code generated by Varadhi IDL compiler.

To build applications that act as both client and server, the application needs to be linked with both stub code (*add\_st.cc*) and skeleton code (*add\_sk.cc*).

Varadhi includes makefiles for building the demo programs present in the ‘samples’ directory in the Varadhi installation. See below on how to modify demo makefiles for building your applications.

### 1.4.7 Using Demo Makefiles

The makefiles present in the demo directory (`$VARADHI/Samples` on Unix hosts) can be modified for building user applications. For example, a part of makefile to build the Adder application on Unix is given below.

```
# Platform
include $(VARADHI_PLATFORM)/src/host.inc
include $(VARADHI_PLATFORM)/src/platform.inc

# Use Additional Compiler Flags Based on Varadhi Configuration
# Selected using vconf
# VCONF_OPTIONS is defined to be $(VCONF_RTTI_YES) or
# $(VCONF_RTTI_NO)
# in VCONF generated makefile (normally _config.inc)
#
VMAKEFILE = _config.inc
include $(VARADHI_PLATFORM)/src/$(VMAKEFILE)
```

```
VCONF_OPTIONS = $(VCONF_FLAGS)
CC_EXTRA_FLAGS =

DEFINES    = $(STD_DEFINES)
INCLUDES   = $(STD_INCLUDES)
OPTIONS    = $(STD_OPTIONS)

CFLAGS = $(INCLUDES) $(OPTIONS) $(VCONF_OPTIONS)\
        $(DEFINES) $(CC_EXTRA_FLAGS)

# Name of IDL File
IDL = add

# Client and Server File Base Names
CLIENT = client
SERVER = server

# Client and Server Objects
CLIENT_OBJECTS = $(IDL)_st.$(OBJ) $(CLIENT).$(OBJ)
SERVER_OBJECTS = $(IDL)_sk.$(OBJ) $(SERVER).$(OBJ)

# Default Target
all: $(CLIENT)$(EXE) $(SERVER)$(EXE)
# Build Rules
...
.....
```

The make macro “IDL” refers to the base name of the IDL file. Just change

this macro of the demo makefile to the respective IDL file's basename and run make.

The demo makefile takes client application source as client.cc and server application source as server.cc. The client application generated will be 'client' and the server application generated will be 'server'. This can be changed by modifying the "CLIENT" and "SERVER" make macros.

For Windows, makefile.nt is provided for use with 'NMAKE' utility and can be modified in the same manner.

#### **1.4.8 Additional Compiler Flags During Build**

Any additional compiler flags needed for application build can be included by giving suitable values against CC\_EXTRA\_FLAGS make macro in the makefile.

For example, to build the debug version of client and server applications the makefile can be modified as given below.

```
CC_EXTRA_FLAGS = -g
```

Invoking make as shown below on Unix

```
% make
```

would build the debug version of client and server applications.

Instead of changing the makefile, the value for CC\_EXTRA\_FLAGS can also be passed in the command line as shown below.

```
% make CC_EXTRA_FLAGS =-g
```

On Windows

```
> nmake /f makefile.nt CC_EXTRA_FLAGS=/Zi
```

### 1.4.9 Running Client and Server Application

Client and Server applications can be run from any host. By default, the server listens on port number 2000. To change this default port number, use the `--VaradhiPORT` option.

Example: `% server --VaradhiPORT 6789`

For example, the adder demo can be run by invoking 'server' and 'client' as given below.

On Unix csh Prompt,

```
% server &  
% client
```

On Windows Command prompt

```
> start server  
> client
```

### 1.4.10 ORB run-time command line options

ORB options can be passed to a client or server application via command line. OMG defines a set of ORB options such as “*-ORBid*” and “*-ORBInitRef*”. These options start with “*-ORB*” prefix.

In addition, Varadhi defines a set of ORB run-time command line options which start with “*--Varadhi*” prefix such as “*--VaradhiPORT*” and “*--VaradhiVERSION*”. The option “*--VaradhiPORT*” specifies the port at which the server ORB will listen while the option “*--VaradhiVERSION*” lists the version information and copyright notice of SANKHYA Varadhi ORB. The option *--VaradhiTIMEOUT* sets the timeout value for blocking transport system calls.

These options are parsed by the CORBA ORB API *CORBA::ORB\_init()*. *ORB\_init()* consumes all argv options and passes only non-ORB options to the application. Refer to “Part II - Reference Guide” for more information.

## 1.5 Development Tools And Utilities

### 1.5.1 IDL Compiler For C++ Mapping

SANKHYA Varadhi's IDL compiler, *idlc*, provides C++ mapping for IDL source files. It creates stubs and skeletons for C++ application from the source IDL file. For more information on *idlc*, refer to “Part II - Reference Manual”.

### 1.5.2 VCONF Configuration Tool

*vconf* is Varadhi's configuration utility. It is used to create Varadhi platforms using the configuration parameters specified in the configuration files. This allows customized version of Varadhi. A Varadhi Platform is needed before developing any application.

For more information on *vconf* and Configuration File Format, refer to “Part II - Reference Manual”

### **1.5.3 DUMPIOR Utility**

SANKHYA Varadhi's *dumpior* is an utility to dump the contents of CORBA stringified Interoperable Object Reference. It takes an IOR file as input and outputs the content of the IIOP profile in it.

For more information on '*dumpior*', refer to "Part II - Reference Manual".

---

## 1.6 SANKHYA Varadhi - Common Object Service

### 1.6.1 Naming Service

In a distributed system, it is often useful to associate names with objects and maintain a mapping of name to object reference. Clients can then find server objects by looking up the name in the map. In most systems, the map itself is dynamically created, with server objects registering themselves and binding their references to names, and clients resolving names to objects using the map.

OMG's Common Object Services Specification describes a standard Naming service which can be used by CORBA clients and servers to bind and resolve names in a standard way.

**Varadhi** Naming Service is compliant with OMG's CosNaming specification 1.0.

Varadhi Naming Service implements the **CosNaming** interface specified in OMG's CORBA 2.2 Common Object Services Specification.

The CORBA Naming Service is composed of two basic interfaces, *NamingContext* and *BindingIterator*. The *NamingContext* and the *BindingIterator* are CORBA Objects and are accessed via object references. The CORBA::ORB operation **resolve\_initial\_references()** is used to get the object references for the Root Naming Context in the Naming Service.

The *NamingContext* contains “named” object references or other “named” NamingContexts. The CORBA NamingService can be constructed as a set of interconnected NamingContext objects and NamingService processes.

The *BindingIterator* is used to iterate across object references contained

within a particular NamingContext. It basically provides a mechanism to get the number of named object references within a NamingContext. It provides the operations *next\_one()* and *next\_n()* to retrieve individual named object references or contexts.

## 1.6.2 ns - Varadhi Naming Server

SANKHYA Varadhi includes a Naming Server, 'ns', with basic support. 'ns' is a regular Varadhi Server application and takes all ORB options. One useful option is the "*--VaradhiPORT <portno>*" to specify the port in which the Naming Server will run. By default, 'ns' will listen on port 2809.

Varadhi Naming Server can be invoked as shown below.

```
% ns --VaradhiPORT 2809
```

This starts the Naming Server which listens for client request at port number 2809.

On Windows host, 'ns' can be started from the Start Menu or as shown below.

```
> start ns --VaradhiPORT 2809
```

This will start Varadhi Naming Server which listens on port 2809. For more information on 'ns', refer to "Part II - Reference Manual".

### 1.6.3 nsman - Varadhi Naming Service Manager

*nsman* is a client utility of Varadhi Naming Server for administration of the Naming Service from command line. It supports the following Naming Server operations:

1. bind
2. rebind
3. unbind
4. list
5. resolve
6. save
7. restore
8. new\_context
9. bind\_context
10. rebind\_context
11. bind\_new\_context
12. list\_context
13. ncdestroy
14. save\_context
15. restore\_context

For more information on '*nsman*', refer to "Part II - Reference Manual".

### 1.6.4 Event Service

In a distributed system, an event occurring in one part of the system may have to trigger an action in a different part of the system. When an event is generated by the event supplier, the event will have to be delivered to the event consumer.

In general, a consumer may poll the supplier for an event and pull events from the supplier. Alternatively, a supplier may push events to the consumer as and when they occur.

There can be multiple consumers for a single event. Also, a single supplier may generate different kinds of events, each with a different set of consumers.

One way to manage this complexity is by creating an Event Channel. A supplier would then supply events (either in push or pull mode) to the Event Channel. A consumer interested in receiving events can then register (add) itself to the appropriate event channel. The channel would then handle the delivery of events to all the registered consumers.

OMG's Common Object Services Specification, specifies a standard way of implementing Event Channels as a standard CORBA service. The following are the key features of Varadhi Event Service.

- Conforms to OMG's Event Service Specification 1.1
- User configurable limits
- Static or Dynamic Memory Allocation
- Supports standard Varadhi command line options
- C++ Exceptions or Varadhi Exceptions
- Generic(untyped) push, pull, mixed communication model

A brief description on each of the terms used in Event Service is given below.

## **1. Event Channel**

CosEvents Service provides the Event Channel Object that decouples the communication between the consumer and the supplier. It acts as the

supplier for the consumer application (objects interested in knowing about the changes) and as a consumer for the supplier application (Objects being changed). As a result the consumers and suppliers registered in an event channel can communicate without knowing each other's identities.

## **2. Push Consumer**

A consumer that does not actively poll for or pull events from the Event Channel is a Push Consumer. When there are events in the channel, the event channel will push the event to the consumer using a previously registered call back.

## **3. Push Supplier**

Push supplier is an active object that pushes events in to the event channel whenever changes need to be notified to the interested consumers.

## **4. Pull Consumer**

A consumer that actively polls for or pulls events from the Event Channel is a Pull Consumer. When there are events in the channel, the event channel conveys the event to the consumer while polling. The pull operation of the pull consumer blocks until the the event data is available. The try\_pull operation does not block.

## **5. Pull Supplier**

Pull Supplier is a passive object that doesn't initiate the events notification(data transfer). The event channel has to actively poll or pull event from the pull supplier.

## **6. ConsumerAdmin**

Push/Pull consumers use ConsumerAdmin to register in the Event channel which returns the respective supplier proxies depending on the consumer model.

## **7. SupplierAdmin**

Push/Pull suppliers use SupplierAdmin to register into the Event channel which returns the respective consumer proxies depending on the supplier model.

## **8. Supplier Proxies**

The supplier proxies of the Event Channel act as the supplier for the consumer application. The supplier proxies can be either pull or push supplier proxies. The Consumer application can use the relevant supplier proxy of the EventChannel. ie,

ProxyPull supplier - Pull consumer

ProxyPush Supplier - Push consumer

Each Consumer application will be connected to a unique supplier proxy of the Event Channel.

## **9. Consumer Proxies**

The consumer proxies of the Event Channel acts as the consumer for the supplier application. The consumer proxies can be either pull or push consumer proxies. The Supplier application can use the relevant consumer proxy of the EventChannel. ie,

ProxyPull Consumer - Pull Supplier  
ProxyPush Consumer - Push Supplier

Each Supplier application will be connected to a unique consumer proxy of the Event Channel. For more information on Varadhi Event Service refer to part 2 - Reference manual.

### 1.6.5 es - Varadhi Event Service

The Varadhi Event server provides support for CosEventComm module and CosEventChannelAdmin module specified in OMG's CORBA services - Common Object Services Specification. It can take all ORB options. One useful option is the “*--VaradhiPORT <portno>*” to specify the port in which the Event Server will run. By default, ‘es’ will listen on port 2809

Varadhi Event Server can be invoked as shown below on Unix hosts.

```
% es --VaradhiPORT 2050
```

This starts the event server that listens at port 2050.

On Windows hosts,

```
> start es --VaradhiPORT 2050
```

For more information on Varadhi Event Server refer to part II - Reference manual

## 1.7 Customizing SANKHYA Varadhi

### 1.7.1 Introduction

CORBA based applications, especially servers, require a number of parameters for their operations which are provided through command-line arguments. In an embedded environment, passing options via command line is not possible and so the parameters should be configured into the ORB before deployment.

SANKHYA Varadhi provides the utility '*vconf*' to configure Varadhi and Varadhi applications. Using *vconf*, a developer creates a Varadhi Platform with user-configured parameters. Varadhi Platform is a directory that contains the configured Varadhi Run-Time library.

Varadhi Platform configuration allows different customized versions of Varadhi run-time library, suitable for a particular development need. Possible among them are listed below.

- Varadhi with support for static or dynamic allocation of memory
- Varadhi with profile support
- Varadhi with debug trace
- Varadhi with C++ Exception support for CORBA Exceptions
- Varadhi without C++ Exception support.

Some of the configurations are mutually exclusive like "static" and "dynamic" allocation of memory.

*vconf* creates Varadhi Platform using the configuration parameters specified in the configuration file.

For more information about Varadhi configuration and configuration file format, refer to “Part II - Reference Manual”.

On Unix, the configuration files shipped with the product is in **\$VARADHI/etc/config** directory. On Windows, the configuration files are present in **%VARADHI%\etc\config** directory.

### 1.7.2 Creating a new Varadhi Platform

Varadhi can be customized for a particular configuration using *vconf*. *vconf* takes configuration files as input and creates a Varadhi Platform specified in one of the configuration files.

For example, to create a default platform for your host (native target) that supports C++ Exception and with no support for static allocation of Heap memory, use the following.

a) Change directory to your workspace (WORK\_DIR).

On Unix csh Prompt,

```
% cd $WORK_DIR
```

On Windows Command Prompt,

```
> cd %WORK_DIR%
```

b) Invoke *vconf* with two config files, one providing configuration options for Varadhi Run-time library and another one that specifies the target platform. Varadhi Host Environment should be set before using *vconf*.

On Unix csh Prompt,

```
% vconf defaults sol2-gcc
```

In Windows Command Prompt,

```
%WORK_DIR%\> vconf defaults win32
```

For linux, use linux-gcc instead of sol2-gcc.

c) This will create a directory named “varadhi” with the following contents.

platform.csh	- Script to set the target platform environment
src	- Source directory created for the platform
src/*.{cc,h}	- Sources for the particular configuration.
src/makefile	- Makefile to build the configuration.
src/host.inc	- Make include file that needs to be included in all the application builds using this Platform.
src/platform.inc	- Make include file that needs to be included in all the application builds using this Platform.
src/_config.inc	- Make include file generated by vconf according to the specified configuration parameters.
lib	- Directory for placing the configured libraries and object files.

On Windows, plattform.bat file will be created instead of platform.csh.

### 1.7.3 Creating a Varadhi Platform for Porting

Varadhi can be ported to supported CPU targets by creating a Varadhi Platform using *vconf*.

These Varadhi platforms additionally contain sources for porting the Varadhi OS layer and Transport layer to custom target environments. *vconf* is used to create Varadhi Platforms for ported sources by specifying target specific configuration files from \$VARADHI/etc/config (On windows, %VARADHI%\etc\config) directory.

For example, to create a platform for PPC target that has no C++ Exception support and with support for static allocation of Heap memory, use the following.

a) Change directory to your workspace.

On Unix csh Prompt,

```
% cd $WORK_DIR
```

On Windows Command Prompt,

```
> cd %WORK_DIR%
```

b) Invoke *vconf* with two config files, one providing configuration options for the Varadhi run-time library and another one that specifies the CPU (for e.g PPC) platform.

On Unix csh Prompt,

```
% vconf static ppc-gcc
```

On Windows Command Prompt,

```
%WORK_DIR% \> vconf static ppc-gcc
```

c) This will create a directory named “varadhi” with the following contents.

- platform.csh - Script to set the target platform environment.  
(On Windows platform.bat will be created instead of platform.csh)
- src - Source directory created for the platform.
- src/\*.{cc,h} - Sources for the particular configuration and also sources that needs to be ported for a specific OS and Transport.
- src/makefile - Makefile to build the configuration.
- src/host.inc - Make include file that needs to be included in all the application builds using this Platform.
- src/platform.inc - Make include file that needs to be included in all the application builds using this Platform.
- src/\_config.inc - Make include file generated by vconf according to the specified configuration parameters.
- lib - Directory for placing the configured libraries and object files.

#### **1.7.4 Using Varadhi Platform**

The newly created Varadhi Platform can be used just like the pre-configured platform by sourcing the script file *platform.csh* in the newly created platform directory. On Windows, run the *platform.bat* file instead.

---

# REFERENCE MANUAL

**Sankhya Technologies Private Limited**

---

---

## Part 2 - Reference Manual

---

### 2.1 Varadhi IDL Compiler - IDLC

#### 2.1.1 Standards Conformance

SANKHYA Varadhi's IDL Compiler, '*idlc*' provides C++ mapping for IDL source files. It creates stubs and skeletons for C++ application, from the source IDL file.

#### 2.1.2 Synopsis

```
idlc [ -I<include path> ] [ -dir<output directory> ] [-s] idlfile
```

#### 2.1.3 Description

*idlc* takes the IDL source file <idlfile> as input and creates Stub and Skeleton code for C++ applications. The generated files have the following extensions for Unix hosts.

```
<idlfile_basename>_st.h- Stub header file  
<idlfile_basename>_st.cc- Stub source file  
<idlfile_basename>_sk.h- Skeleton header file  
<idlfile_basename>_sk.cc- Skeleton source file
```

On Windows hosts the files <idlfile\_basename>\_st.cc and

`<idlname_basename>_sk.cc` will be replaced with `<idlname_basename>_st.cpp` and `<idlname_basename>_sk.cpp` respectively.

where `<idlname_basename>` is the IDL file name without the `‘.idl’` extension.

### 2.1.4 Options

`-I <include_path>` This option specifies the directory to search for IDL files specified in the `#include` directive in an IDL file. The space between `-I` and the path is optional. This option can occur more than once in a command line.

Example:

```
% idlc -I../include -I ../test/include my.idl
```

`-dir <output dirname>` This option specifies the output directory (to use) where the generated client stub files and server skeleton files should be placed. The space between `-dir` and the path is optional. If more than one occurrence of this option is encountered, `idlc` ignores them expect for the first `-dir` option value.

Example:

```
% idlc -dir../output my.idl
```

`-s` This option performs IDL syntax check only.

### 2.1.5 Usage Examples

The following example shows how to generate C++ client stub and server

skeleton code for the IDL file test.idl, and place them in ./out directory.

On Unix csh prompt,

```
% idlc -dir ./out test.idl
```

On Windows command prompt,

```
> idlc -dir .\out test.idl
```

On Unix, this generates the following files in the output directory "./out".

```
test_st.h  
test_st.cc  
test_sk.h  
test_sk.cc
```

On Windows NT/2000, this generates the following files in the output directory ".\out".

```
test_st.h  
test_st.cpp  
test_sk.h  
test_sk.cpp
```

## 2.2 VCONF Configuration Tool

*vconf* is Varadhi's configuration utility. It creates Varadhi Platforms using the configuration parameters specified in the config file(s). This allows customized version of Varadhi for embedded applications. The Varadhi Platform environment should be created before building any application.

### 2.2.1 Synopsis

```
vconf [-V] [-h] [-i] [ [-q] [-v] ] [ -d dirname ] <cfg_file(s)>
```

### 2.2.2 Description

*vconf* is used to create a Varadhi Platform for a specific environment. Configuration files are provided to create specific Varadhi Platform. One of the configuration files given as input to *vconf* should specify the platform information using *'-type platform'* option of the *'declare'* command.

### 2.2.3 Options

- V            Displays tool version and usage.
- h            Displays tool help.
- i            Starts *vconf* in interactive mode
- q            Specifies quiet mode. This option disables message display like version number and copyright.
- v            This option sets verbose mode. It prints the tool version and also the configuration parameters used for creating the Platform.
- d dirname   This option specifies the directory name where Varadhi platform should be created. By default, *vconf* creates Platforms under “*varadhi*” directory in the location where it

is invoked

cfg\_file(s) Specifies the configuration file(s) to load. vconf searches for the file in the following order:

- ./file\_name
- ./file\_name.cfg
- \$VARADHI/etc/config/file\_name.cfg

#### 2.2.4 Usage Examples

The following example shows how to create a Varadhi Platform for Solaris using the default configuration file defaults.cfg at \$VARADHI/etc/config directory.

On Unix csh prompt,

```
% vconf defaults sol2-gcc
```

```
vconf 1.8; Varadhi ORB Configurator
Copyright (C) 2000-2002 Sankhya Technologies Private Limited.
All Rights Reserved.
```

```
Creating the directories ...
```

```
Copying files ...
```

```
Completed Copying of files ...
```

```
Creating the configuration ...
```

```
g++ -c -I/opt/sankhya/varadhi/include -I/src -Wall -ansi -pedantic
-Wno-long-long -O2 -DENABLE_CPP_EXCEPTIONS -frtti -
DVARADHI_GENERATED_HEADER_FILE=\"_config
.h\" -DVARADHI_GENERATED_MAKE_FILE=\"_config.inc\"
-o ../lib/./lib/defaults_dyn.o defaults.cc
```

Varadhi Platform created in ./varadhi directory  
Creating the scripts ...

Source the generated platform.csh file in the platform directory  
to setup Varadhi development environment.

Configuration Completed.

**On Windows command prompt,**

```
> vconf defaults win32
```

```
vconf 1.8; Varadhi ORB Configurator  
Copyright (C) 2000-2002 Sankhya Technologies Private Limited.  
All Rights Reserved.
```

```
"Creating the directories ..."
```

```
1 file(s) copied.
```

```
1 file(s) copied.
```

```
1 file(s) copied.
```

```
"Copying files ..."
```

```
1 file(s) copied.
```

```
"Completed Copying of files ..."
```

```
cl /nologo /c /Foadd_st.obj /IC:\SANKHYA\Varadhi\include  
/IC:\SANKHYA\Varadhi\platforms\default-win32\src /DWIN32  
/DENABLE_CPP_EXCEPTION/  
DVARADHI_GENERATED_HEADER_FILE=\"_config.h\"  
/DVARADHI_GENERATED_MAKE_FILE=\"_config.inc\"  
/Fo..\lib\defaults_dyn.obj /Tp defaults.cc
```

```
1 file(s) copied.  
Creating the configuration ...  
Varadhi Platform created in .\varadhi directory  
Creating the scripts ...  
Varadhi configuration created in .\varadhi directory  
Creating the scripts ...
```

Run the generated platform.bat file in the platform directory to set up Varadhi development environment.

Configuration Completed.

### 2.2.5 VCONF Configuration File Format

The configuration file consists of configuration commands with parameters for configuring Varadhi's ORB. Each configuration command occupies one line. Comments can be inserted by using the character '#'. Any text after this '#' character is ignored.

Varadhi customization is achieved by using the following configuration commands.

#### **declare**

This command is used to configure any Varadhi parameter.

#### **Syntax:**

```
declare <param> -type <type_id> -default <value> \  
[flag <flag_name>] -action <action_id>
```

where

- <param> - The parameter to configure
- <type\_id> - Type of the parameter. It can be numeric, string, qstring, bool, list, platform or service.
- <value> - Value can be anything of type <type\_id>
- <flag\_name> - Flag to be defined for use in Make. The generated flag name will be the symbol name in uppercase with the symbol value appended to it in uppercase with a leading underscore.
- <action\_id> - Action specifies the action to perform on the parameter. The valid values are:  
 cpp\_define, create, make\_target, make\_macro, make and register.

Example:

```
declare varadhi::orb::msg_param_size -type numeric -default 448
    -action cpp_define
declare varadhi::poa::max_child_list -type numeric -default 4
    -action cpp_define
declare varadhi::generated_header_file -type qstring -default
    _config.h -action make_macro
declare varadhi -type string -default "make" -action make
declare linux-gcc -type platform -default
    "make -f $VARADHI/etc/config/vconf.mk -action create
```

## Set

This command is used for setting a value to a Varadhi parameter.

**Syntax:**

```
set <param> -value <value>
```

where

<value> can be any valid value for the <param> as specified in the declare command.

Example:

```
set varadhi::enable_cpp_exceptions -value yes
set varadhi::orb::max_sock_fd -value 32
```

**include**

This command can be used to include a valid configuration file.

**Syntax:**

```
include "cfg file"
include <cfg file>
```

where "cfg file" is a valid vconf configuration file.

Example:

```
include "myconf.cfg"
```

**2.2.6 VCONF - Interactive Mode of Operation**

*vconf* provides an interactive mode of operation for manipulation and generation of configuration files, and to create Varadhi Platform. "-i" command line option takes VCONF into interactive mode.

In addition to the above commands mentioned in the previous section, the following commands are also supported in interactive mode.

### \* help

This provides the help for vconf commands.

Syntax:

```
help [command name]
```

where <command name> is a valid vconf command.

### \* list

This command lists the value(s) of the configuration parameter(s)

Syntax:

```
list [-l] <symbol>
```

where

<symbol> - is a parameter for which the values should be listed.

-l - Option to display long list of the configuration parameter.

Example:

```
list -l varadhi::poa::max_number_of_poa
```

The execution of the above command gives the maximum number of POAs that can be created by Varadhi with static memory configuration.

#### \* **dump**

This command creates the configuration file with the `vconf` commands used during the interactive mode.

Syntax:

```
dump <file_name>
```

where `<file_name>` is the name of the new configuration file to be created.

#### \* **generate**

Syntax:

```
generate
```

This command executes the actions in the configuration file or in buffer. If a “platform” is specified through any ‘`declare`’ command, then ‘`generate`’ will create the Varadhi Platform.

### **2.2.7 Exit Status and Error Messages**

`vconf` will exit with status -1 on the following error conditions.

**Syntax error at line <n> in config file.**

- This error message is displayed when the config file is not in the proper syntax.

**Error at line <n> in config file**

- This error occurs when the configuration parameter specified in the configuration file is invalid.

**Redeclaration of <symbol> at line <n> in config file**

- This error message is displayed when a symbol is redeclared using another "declare" command.

**Symbol not found at line <n> in config file**

- This error message is displayed when "set" command is used to change the value of a symbol that was not declared before.

**Missing Options**

- This error message is displayed when options are missing in a "declare" command.

**Missing Option**

- This error message is displayed when the '-value' option is missing in a "set" command.

**Syntax error. Missing '>'**

- This error message is displayed when the closing angle bracket (>) is missing in an "include" command.

**Missing Filename**

- This error message is displayed when the name of the file to include

is not specified in the "include" command.\

**Can't open config file <filename>**

- This error message is displayed when the configuration file specified in the "include" command cannot be opened.

**Invalid non-interactive command**

- This error message is displayed when the config file contains commands not supported for non-interactive mode of operation.

**Syntax error. -id is allowed only for "service" type**

- This error message is displayed when a "declare" command contains an '-id' option and the type of the symbol is not "service".

## 2.3 DUMPIOR Utility

### 2.3.1 Introduction

'*dumpior*' is a SANKHYA Varadhi utility to dump the contents of CORBA stringified Interoperable Object Reference.

### 2.3.2 Synopsis

```
dumpior <iorfile>
```

### 2.3.3 Description

*dumpior* takes the <ior file> as input and dumps the contents in human readable format. Currently it supports only IIOP profiles. *dumpior* takes no options.

### 2.3.4 Usage Examples

The following example shows how to dump the stringified IOR generated by an Adder application.

On Unix csh prompt,

```
% dumpior adder.ior
```

On Windows command prompt,

```
> dumpior adder.ior
```

For the above command, '*dumpior*' gives the following output.

```
struct IOR
{
  type_id ==> IDL:Adder:1.0
  struct IOProfile
  {
    tag ==> 0
    struct IOProfileBody
    {
      struct Version
      {
        major ==> 1
        minor ==> 1
      }
      hostname ==> 200.200.200.200
      port ==> 2005
      object_key ==> /0:0:372052
    }
  }
}
```

**Note:** The IP address 200.200.200.200 is just an example

## 2.4 ns - Varadhi Naming Server

### 2.4.1 Introduction

'ns' is Varadhi's Naming Service for binding and resolving objects using simple names.

### 2.4.2 Synopsis

```
ns [ORB options] [ -h ] [ -d ] [ -r [ IOR_FILE ] ]
```

### 2.4.3 Description

SANKHYA Varadhi's Naming Server provides CosNaming interface specified in OMG's CORBA 2.2 Common Object Services Specification. Objects can be named and registered with the Varadhi Naming Server. Any CORBA application that requires the Object Reference of Objects registered with 'ns' can send request to it to resolve the Object using the name.

### 2.4.4 Options

'ns' accepts all run-time ORB options supported by varadhi.

"*--VaradhiPORT*" is one useful option to specify a different port on which to listen. By default, 'ns' listens at port number 2809.

- |                   |  |
|-------------------|--|
| -h                | Display the ns usage and supported options.  |
| -d                | Run the Naming Server as a daemon program on Unix host.  |
| -r [ <ior_file> ] | Generate the stringified IOR for 'ns'. The default IOR file generated will be "ns.ior". The option -r takes an |

optional argument <ior\_file>. If <ior\_file> is specified, 'ns' generates the stringified IOR in <ior\_file> file.

### 2.4.5 Usage Examples

The following example shows how to run Varadhi Naming Server on Port number 2069.

On Unix csh prompt,

```
% ns --VaradhiPORT 2069 &
```

On Windows command prompt,

```
> start ns --VaradhiPORT 2069
```

**Note:** If more than one VaradhiPORT address is given as argument to the Naming server application as in the following, only the last argument will be taken as the port address and Naming server will listen at that port only.

```
% ns --VaradhiPORT 2069 --VaradhiPORT 2050
```

When the above is executed, the Naming Server will listen at port 2050 only and not in 2069.

Naming Server clients can then specify this address using the “**-ORBInitRef**” option as follows.

On Unix csh prompt,

```
% myappl -ORBInitRef NameService=a.b.c.d:2069
```

On Windows command prompt,

```
> myappl -ORBInitRef NameService=a.b.c.d:2069
```

Where a.b.c.d is the IP address.

Please refer to the demo application in \$VARADHI/samples/names (%VARADHI%\samples\names on Windows) for usage of Varadhi Naming Service.

#### **2.4.6 Maximum Configured limits**

Varadhi Naming Service is configured to allow a maximum of 300 Namingcontexts. It is also configured to allow a maximum of 120 bindings per Namingcontext.

The various parameters that are configured for customizing Varadhi Naming Server are explained in the next section. (Sec 2.5)

## 2.5 Customizing Varadhi Naming Server

Varadhi Naming Server can be customized and built according to the requirements of the user. The user can create his own target platform (configuration file) for the naming service that can then be used with Varadhi configuration utility *vconf* to build a customized Naming server. It is also possible to link an application Varadhi Naming Service to create a customised 'ns'.

The default values for the name service configuration parameters are given in the file 'ns.cfg'. The values of these parameters can be changed at build time and are user configurable. A brief description for each of these parameters is given below.

### 1. **varadhi::ns::max\_bindings\_per\_context**

Specifies the maximum number of bindings allowed per Naming Contexts. The default configured limit for this parameter is 120

### 2. **varadhi::ns::max\_nc\_kind\_size**

This specifies the size limit for the NameComponent kind. The default configured limit of `varadhi::ns::max_nc_kind_size` is 8 bytes.

### 3. **varadhi::ns::max\_nc\_id\_size**

This specifies the size limit of the NameComponent id. The configured limit of `varadhi::ns::max_nc_id_size` is 32 bytes.

By default, the above parameters take the values specified in the configuration file "ns.cfg"

## 2.5.1 Building customized 'ns'

### Prerequisite

1. The Naming Service library and the Naming Service porting source are provided for each platform(Solaris, Windows and Linux)
2. The Naming Service library, and the Naming Service porting source should be installed in the standard release area (<Varadhi\_installation\_dir>) of the respective hosts from the separate packages provided for Naming Service.
3. To customize Varadhi Naming Service, the Varadhi installation should contain
  - <varadhi\_installation\_dir>/lib/<host>/<NamingService library>
  - <varadhi\_installation\_dir>/src/port/ns\_main.cc
  - <varadhi\_installation\_dir>/src/port/ns\_cfg.cc

The following are the steps to create a customized Varadhi Naming server and running it.

### Step 1: Set up the Varadhi Host environment

Varadhi tools require the following environment variables to be set.

- |                     |  |
|---------------------|--|
| <b>VARADHI</b>      | - Directory path pointing to the root of installation. |
| <b>VARADHI_HOST</b> | - Host platform name like sol2, linux and win32.       |

In order to set these variables and the PATH variable, source varadhi.csh file present in the SANKHYA Varadhi installation root directory. On Windows host, run varadhi.cmd instead.

On Unix csh prompt

```
% source <varadhi-install-dir>/varadhi.csh
```

On Windows command prompt

```
> <varadhi-install-dir>\varadhi.cmd
```

where <varadhi-install-dir> is the directory where Varadhi is installed. By default Varadhi is installed under /opt in Unix and C:\sankhya\varadhi in Windows NT/2000.

## **Step 2. Create Varadhi Target Platform to customize Varadhi Naming Service**

Varadhi Naming Server can be customized for a particular target configuration using vconf.

Varadhi configuration files to create various Varadhi Target Platforms are provided under

On Unix hosts

```
$VARADHI/etc/config directory
```

On Windows hosts

%VARADHI%\etc\config directory

The porting source to customize Varadhi Naming Service for specific target environment is present under

On Unix hosts

\$VARADHI/src/port/ns\_main.cc

\$VARADHI/src/port/ns\_cfg.cc

On Windows hosts

%VARADHI%\src\port\ns\_main.cc

%VARADHI%\src\port\ns\_cfg.cc

### **Step 2.1 : Change directory to your workspace.**

On Unix csh prompt

```
% cd <Work-directory>
```

```
% set WORK_DIR = 'pwd'
```

On Windows command prompt

```
> cd <Work_directory>
```

```
<Work_directory>> set WORK_DIR= <full path of Work_directory>
```

**Step 2.1.1:** To create a NamingService platform for your host (native

target) that supports C++ Exception and **with no support for static heap allocation**,

**1.** Invoke vconf with the config files defaults.cfg (configuration options for the ORB), and ns.cfg (configuration options for ns), the config file to specify target platform as follows

For Solaris hosts, (On Unix csh prompt)

```
% vconf defaults ns sol2-gcc
```

For Linux hosts, (On Unix csh prompt)

```
% vconf defaults ns linux-gcc
```

For Windows hosts, (On Windows command prompt)

```
> vconf defaults ns win32
```

The vconf parameter "varadhi::lib" in the ns.cfg configuration file can be overridden and redefined for debug version of ns as given below.

```
set varadhi::lib -value debug_ns
```

**Step 2.1.2:** To create a Naming Service platform for your host (native target) that supports no C++ exception and **with support for static heap allocation**, the following steps need to be carried out.

**1.** Override the vconf parameter "varadhi::lib" in ns.cfg configuration file (for static version) of ns as follows.

```
set varadhi::lib -value static_ns
```

2. Invoke vconf as follows.

On Solaris hosts (On Unix csh prompt)

```
% vconf static ns sol2-gcc
```

On Linux hosts (On Unix csh prompt)

```
% vconf static ns linux-gcc
```

On Windows hosts

```
> vconf static ns win32
```

The customized Varadhi NamingServer ‘ns’ will be built and placed at `$VARADHI_PLATFORM/lib/ns`

`$VARADHI_PLATFORM/src/ns_main.cc` is the NamingService porting source where,

`$(VARADHI_PLATFORM)` is the root directory where Varadhi Platform is created.

Another way of creating a target platform for Varadhi Naming Server is to link an application with the Varadhi naming service library.

Step 3. Create Varadhi Target Platform that can be linked with an

application with Varadhi Naming Service library

Applications can link with Varadhi Naming Service library using the makefile macro `$(VARADHI_NS_LIB)`. This can be achieved using either of the following steps (3.1 or 3.2)

Step 3.1: Varadhi Naming Service platform created as in Step 2.

( or )

Step 3.2: Varadhi Naming Service platform created by overriding the vconf parameter "varadhi::lib" specified in "ns.cfg" configuration file as shown below.

Example:

```
declare varadhi::lib -type list -default dynamic_nslib -values
static_ns:dynamic_ns:static_nslib:dynamic_nslib -action make_target
```

Invoke vconf as mentioned in step 1 to create Varadhi target platform

Varadhi Naming Server platform created in step (3.2) will not build customized ns. It creates the platform with customized naming service library and not a Naming Server.

#### **Step 4: Setup the Varadhi Target Platform Environment**

The above configuration places a script file in the `$VARADHI_PLATFORM` directory to setup the Varadhi Target Environment. Here, `VARADHI_PLATFORM` refers to the root directory where Varadhi Platform is created.

On Unix csh prompt

```
% cd $WORK_DIR/varadhi
% source platform.csh
```

On Windows command prompt

```
% WORK_DIR%\> cd %WORK_DIR%\varadhi
% WORK_DIR%\varadhi> platform.bat
```

This will set the variable,

**VARADHI\_PLATFORM** - Directory path pointing to the root of the created Target platform.

## 2.5.2 Using Naming Service library APIs

Several APIs are available for the purpose of customizing Varadhi Naming service.

The following APIs are used to initialize Varadhi Naming service.

### 1. void ns\_init( int argc, char\* argv[], CORBA::ORB\_ptr orb )

This API supports all ns command line options. The default port of ns is taken as 2809. The POA activation in this case will be done in the ns library. The user application has to call ORB\_init and orb->run before invoking this API.

**Note:** Naming Server specific command line options can alone be passed to this API through "argv[]" for ns\_init()

### 2. void ns\_init( CORBA::ORB\_ptr orb, PortableServer::POA\_ptr

**poa, char\* ior\_file = NULL )**

This API ignores ns command line application. The default port used in this case is 2000. Before using this API the user application has to call ORB\_init and orb->run. The generation of ns.ior file can be restricted by suitably configuring the parameter '**varadhi::ns::generate\_ior\_file**' present in the file ns.cfg and passing a valid string to the ior file.

In this case, the user has to perform the POA activation.

For Varadhi Naming Server that listens at ports other than 2809, Naming Server clients can reach ns through either of the following.

- using **-ORBInitRef NameService** option  
(or)
- Building client applications using Varadhi NS platform built with a config file, specifying the NameService object registered in the required port

Example:

```
declare NameService -type service -id "IDL:omg.org/
CosNaming/NamingContext:1.0" -default "corbaloc::127.0.0.1:2000/
NameService" -action register
```

Applications linked with Naming Service library can use the API,

**void ns\_init( CORBA::ORB\_ptr orb, PortableServer::POA\_ptr poa,  
char\* ior\_file = NULL )**

to initialize the NamingContext factory.

### **2.5.3 API to shutdown Naming Service**

#### **1. void ns\_shutdown()**

This API performs the shutdown operation of the Naming Service.

For Building and running a sample application, refer to

`$VARADHI/samples/vnames/README`

`%VARADHI%\samples\vnames\README.TXT` ( On Windows hosts)

## 2.6 nsman - Varadhi Naming Service Manager

### 2.6.1 Introduction

'*nsman*' is Varadhi's Naming Service Manager. It is used to administer Varadhi's '*ns*' and also other Naming Services compliant with OMG's CosNaming specification.

### 2.6.2 Synopsis

General Syntax

```
nsman [ORB options] [ -h ] operation <options for operation>
```

Operation Syntax

```
nsman [ORB options] [ -h ] {bind | rebind} NAME ior_file
```

```
nsman [ORB options] [ -h ] {bind_context | rebind_context}
    NAME ior_file
```

```
nsman [ORB options] [ -h ] {resolve | unbind} NAME
```

```
nsman [ORB options] [ -h ] new_context NAME
```

```
nsman [ORB options] [ -h ] {bind_new_context | ncdestroy}
    NAME
```

```
nsman [ORB options] [ -h ] list no_of_bindings
```

```
nsman [ORB options] [ -h ] list_context no_of_bindings NAME
```

```
nsman [ORB options] [ -h ] {save | restore } <log_file >
```

```
nsman [ORB options] [ -h ] {save_context | restore_context}
    <log_file>
```

### 2.6.3 Description

'*nsman*' is a command line utility for managing Varadhi Naming Server, '*ns*'. All the operations of "*CosNaming::NamingContext*" interface of CosNaming can be invoked using '*nsman*' from the command line. '*nsman*' can also be used to invoke the *CosNaming::NamingContext* operations on any other OMG's CosNaming compliant Naming Server.

### 2.6.4 Options

'*nsman*' accepts all Varadhi Run-Time ORB options.

"**-ORBInitRef NameService=<addr>**" is an useful option to specify the location of the Naming Server. By default, '*nsman*' will look for the server at port 2809 on the host on which '*nsman*' is invoked.

-h        - Display nsman usage and supported options.

NAME    - NAME is a string of the form

id> [ . <kind> ] [ / [ <id> [ .<kind> ] ] ] \*

The <id> and <kind> strings corresponds to the '*id*' and '*kind*' members of the IDL struct data type "*CosNaming::NameComponent*".

NAME is the full path of the NamingContext or Object in the NamingContext tree.

Examples:

a.b

a.b/c.d

a.c/c.d/e.f

bin

usr/bin

`ior_file` - Stringified Interoperable Object Reference file.

`no_of_bindings`- Maximum number for bindings to return.

`<operation>`- The “*CosNaming::NamingContext*” IDL operations supported by ‘*nsman*’ from command line.

### 2.6.5 CosNaming::NamingContext operations

- bind** - To bind a name with the object reference in the NamingContext tree of the Naming Server.
- rebind** - To rebind a name that is already bound with an object reference, to a new object reference.
- resolve** - To get the object reference for the already bound name.
- unbind** - To remove the binding between a name and its object reference. This operation can also be used for unbinding NamingContext.
- new\_context** - Creates a new “NamingContext” and stores its reference in `new_context.ior` file
- bind\_context** - To bind a name with the NamingContext reference in the NamingContext tree.
- rebind\_context** - To rebind a name that is already bound to a “NamingContext” reference, to a new “NamingContext” reference in the NamingContext tree.
- bind\_new\_context** - To create a new NamingContext and bind a name with that reference
- ncdestroy** - To destroy a name context, if empty
- list** - To list the name and kind that are bound within the root context.
- list\_context** - To list the name and kind that are bound within the root context.
- save** - To save the current bindings in the SANKHYA

- Varadhi naming server 'ns' to a file recursively
- save\_context** - To save the current bindings in the SANKHYA Varadhi naming server 'ns' to a file non-recursively
- restore** - To restore the saved bindings in the SANKHYA Varadhi naming server 'ns' to the NamingContext recursively
- restore\_context** - To restore the saved bindings in the SANKHYA Varadhi naming server 'ns' to the NamingContext non-recursively

## 2.6.6 Usage Examples

### (a) bind

To bind the Object Name 'Adder' to the Object Reference in the stringified IOR file adder.ior, the following is used. Here, 'Adder' is bound under the context 'a.b'

On Unix csh prompt,

```
% nsman bind a.b/Adder adder.ior
```

On Windows command prompt,

```
> nsman bind a.b/Adder adder.ior
```

### (b) rebind

To rebind the Object Name 'Adder' to the Object Reference in the stringified IOR file adder2.ior, the following is used. Here, 'Adder' is bound under the context 'a.b'.

On Unix csh prompt,

```
% nsman rebind a.b/Adder adder2.ior
```

On Windows command prompt,

```
> nsman rebind a.b/Adder adder2.ior
```

### **(c) resolve**

To resolve the Object Name ‘Adder’ bound under the context ‘a.b’ and to obtain the Object Reference in the stringified IOR file resolve.ior, the following is used.

On Unix csh prompt,

```
% nsman resolve a.b/Adder
```

On Windows command prompt,

```
> nsman resolve a.b/Adder
```

### **(d) unbind**

To unbind the Object Name ‘Adder’, bound under the context ‘a.b’, the following is used.

On Unix csh prompt,

```
% nsman unbind a.b/Adder
```

On Windows command prompt,

```
> nsman unbind a.b/Adder
```

### **(e) new\_context**

To create a new Context, the following can be used. This will create a stringified IOR file ‘new\_context.ior’.

On Unix csh prompt,

```
% nsman new_context
```

On Windows command prompt,

```
> nsman new_context
```

### **(f) bind\_context**

To bind the context ‘a.b’ under the Root Context, the following is used. The file ‘context.ior’ is the stringified IOR file obtained using “new\_context” nsman operation, or any valid NamingContext reference.

On Unix csh prompt,

```
% nsman bind_context a.b context.ior
```

On Windows Command Prompt,

```
% nsman bind_context a.b context.ior
```

**(g) rebind\_context**

To rebind the context ‘a.b’ under the Root Context, the following is used. The file ‘context.ior’ is the stringified IOR file obtained using “new\_context” or “bind\_new\_context” nsman operations, or any valid NamingContext reference.

On Unix csh prompt,

```
% nsman rebind_context a.b context.ior
```

On Windows command prompt,

```
> nsman rebind_context a.b context.ior
```

**(h) bind\_new\_context**

To create a new Context ‘a.b’, and bind under the Root Context, the following is used. This will create a stringified IOR file ‘bind\_new\_context.ior’.

On Unix csh prompt,

```
% nsman bind_new_context a.b
```

On Windows command prompt,

```
> nsman bind_new_context a.b
```

**(i) ncdestroy**

To destroy an empty Naming Context ‘a.b’, the following is used.

On Unix csh prompt,

```
% nsman ncdestroy a.b
```

On Windows command prompt,

```
> nsman ncdestroy a.b
```

**(j) list**

To list the bindings under the Root NamingContext, the following is used.

On Unix csh prompt,

```
% nsman list 2
```

On Windows command prompt,

```
> nsman list 2
```

If the total bindings in the root context exceeds the maximum number of bindings requested (2 in this case), then all the bounded names will be listed.

**(k) list\_context**

To list the bindings under the Context 'a.b', the following is used.

On Unix csh prompt,

```
% nsman list_context 2 a.b
```

On Windows command prompt,

```
> nsman list_context 2 a.b
```

If the total bindings in the root context exceeds the maximum number of bindings requested (2 in this case), then all the bounded names will be listed.

**(l) save**

1. To save the current bindings in the SANKHYA Varadhi naming server 'ns' to the default log file, 'nslog.dat' recursively, the following is used.

On Unix csh prompt

```
% nsman save
```

On Windows command prompt

```
> nsman save
```

2. To save the current bindings in the SANKHYA Varadhi naming server 'ns' to the log file 'slog.dat' recursively, the following is used.

On Unix csh Prompt

```
% nsman save slog.dat
```

On Windows command prompt

```
> nsman save slog.dat
```

**(m) save\_context**

1. To save the contents of Root NamingContext to the default log file 'nslog.dat' non-recursively, the following is used.

On Unix csh prompt

```
% nsman save_context root
```

On Windows command prompt

```
> nsman save_context root
```

2. To save the contents of Root NamingContext of the Naming Service to the log file 'slog.dat' non-recursively, the following is used.

On Unix csh prompt

```
% nsman save_context root sclog.dat
```

On Windows command prompt

```
> nsman save_context root sclog.dat
```

**3.** To save the contents of NamingContext <context\_name> to the default log file 'nslog.dat' non-recursively, the following is used.

On Unix csh prompt

```
% nsman save_context <context_name>
```

On Windows command prompt

```
> nsman save_context <context_name>
```

**4.** To save the contents of the NamingContext <context\_name> to the log file: 'xxx.dat' non-recursively the following is used.

On Unix csh prompt

```
% nsman save_context <context_name> xxx.dat
```

On Windows command prompt

```
> nsman save_context <context_name> xxx.dat
```

**Note:** If the context doesn't exist "NotFound" Exception (Missing Node)

will be received.

**(n) restore**

1. To restore the contents of the default file ‘nslog.dat’ to the Naming Service recursively the following is used.

On Unix csh prompt

```
% nsman restore
```

On Windows command prompt

```
> nsman restore
```

Note: The default log file ‘nslog.dat’ should have been already saved using ‘**nsman save**’

2. To restore the contents of the log file ‘rlog.dat’ to the Naming Service recursively the following is used.

On Unix csh prompt

```
% nsman restore rlog.dat
```

On Windows command prompt

```
> nsman restore rlog.dat
```

**Note:** The log file ('rlog.dat' in the above case) should have been already saved using '**nsman save rlog.dat**'

**(o) restore\_context**

**1.** To restore the non recursive bindings of the root context as in the default log file 'nslog.dat' to the Naming Service, the following is used.

On Unix csh prompt

```
% nsman restore_context root
```

On Windows command prompt

```
> nsman restore_context root
```

**Note:** The default log file should have been already created using '**nsman save\_context root**'

**2.** To restore the non recursive bindings of the root context as in the log file 'log.dat' to the Naming Service, the following is used.

On Unix csh prompt

```
% nsman restore_context root log.dat
```

On Windows command prompt

```
> nsman restore_context root log.dat
```

**Note:** The log file (log.dat) should have already been saved using  
**'nsman save\_context root log.dat'**

**3.** To restore the non recursive bindings of the specified context as in the default log file 'nslog.dat' to the Naming Service, the following is used.

On Unix csh prompt

```
% nsman restore_context <context_name>
```

On Windows command prompt

```
> nsman restore_context <context_name>
```

**Note:** The default log file (nslog.dat) should have been already saved using  
**'nsman save\_context <context\_name> <log\_file>'**

**4.** To restore the non recursive bindings of the specified context as in the log file 'log.dat' to the Naming Service.

On Unix csh prompt

```
% nsman restore_context <context_name> log.dat
```

On Windows command prompt

```
> nsman restore_context <context_name> log.dat
```

**Note1:** The log file (log.dat) should have been already created using  
**'nsman save\_context <context\_name> log.dat'**

**Note2:** If the NamingContext doesn't exist while using "restore\_context", it needs to be created using nsman interfaces like bind\_context and bind\_new\_context, before the restore\_context usage. If the bindings already exist, **'AlreadyBound'** exception will be received.

## 2.7 es - Varadhi Event Service

'es' is Varadhi's Event Service designed for lightweight enterprise and embedded systems. Varadhi Events conforms to OMG's Event Service specification (Untyped Events).

### 2.7.1 Synopsis

```
es [ORB options] [ -h ] [ -r [ IOR_FILE ] ]
```

### 2.7.2 Description

SANKHYA Varadhi's Event Server supports '**CosEventCommn**' and '**CosEventChannelAdmin**' interfaces specified in OMG's CORBA 2.2 Common Object Services Specification.

### 2.7.3 Options

#### ORB options

'es' accepts all run-time ORB options supported by varadhi.

"**--VaradhiPORT**" is one useful option to specify a different port on which to listen. By default, 'es' listens at port number 2809.

#### Command Line Options

- h - Displays the usage and options supported by 'es'
- r [IOR\_File] - Generates the stringified IOR for es. This option also takes an optional argument IOR\_FILE. The argument is actually a name of the IOR file in whose name the file will be created.

## 2.7.4 Usage Examples

### Command Line Options

1. For displaying the usage and options supported by 'es', the event server can be invoked as shown below.

On Unix csh prompt

```
% es -h
```

On Windows command prompt

```
> es -h
```

2. To generate a stringified IOR file of es, the following is done.

On Unix csh prompt

```
% es -r
```

The above command creates the IOR file 'es.ior' by default.

(or)

```
> es -r ves.ior
```

The above command creates the IOR file in the same name as that of the argument to the -r option.

On Windows command prompt

```
> es -r  
(or)  
> es -r ves.ior
```

### **ORB options**

The following example shows how to run Varadhi Event Server on Port number 2069 in host with IP address a.b.c.d using one of the ORB options ‘--VaradhiPORT’.

On Unix csh prompt,

```
% es --VaradhiPORT 2069 &
```

On Windows command prompt,

```
> es --VaradhiPORT 2069
```

**Note:** If more than one VaradhiPORT address is given as argument to the Event server application as in the following, only the last argument will be taken as the port address and Event server will listen at that port only.

```
% es --VaradhiPORT 2069 --VaradhiPORT 2050
```

When the above is executed, the Event Server will listen at port 2050 only and not at 2069.

EventChannel reference of Varadhi event server can be obtained using either of the following ways.

- 'es.ior' of 'es'
- resolve\_initial\_references("EventService")

If 'es' is invoked without any option as shown below the event server runs on the default port 2809

On Unix csh prompt

```
% es
```

On Windows command prompt

```
> es
```

### **2.7.5 Maximum Configurable limits**

Like Naming Service, Varadhi Events Service is also built with many configurable parameters that are included in the file 'es.cfg' present in \$VARADHI/etc/config (%VARADHI%\etc\config on Windows hosts).

The explanation and maximum configured limit for each of these parameters are as given below.

#### **1. varadhi::es::max\_queue\_length**

This parameter specifies the Event Channel Queue Length. The configured limit is 32

#### **2. varadhi::es::max\_pull\_consumers**

This parameter specifies the maximum number of pull consumers. The

configured limit for this parameter is 1

### **3. `varadhi::es::max_pull_suppliers`**

This parameter specifies the maximum number of pull suppliers in an event channel. The configured limit for this 32

### **4. `varadhi::es::max_push_consumers`**

This parameter specifies the maximum number of push consumers. The configured limit is 32

### **5. `varadhi::es::max_push_suppliers`**

This specifies the maximum number of push suppliers. The configured limit for this parameter is 32

### **6. `varadhi::es::max_event_channels`**

This specifies the maximum number of event channels. The configured limit for this parameter is 1.

---

## 2.8 Varadhi ORB run-time options

### 2.8.1 Introduction

Options can be passed to either a Varadhi Server application or a Varadhi Client application. If an option is specific to Varadhi Server, it is ignored by the Varadhi Client and vice-versa. The file 'options.txt' present in \$VARADHI/docs/help (%VARADHI%\docs\help on Windows hosts.) describes the command line options supported by Varadhi.

### 2.8.2 Synopsis

```
% {<client>/<server>} [OptionName [OptionValue] ] ...
```

### 2.8.3 Description

Varadhi supports two types of options. They are OMG defined options and Varadhi specific options.

All Varadhi options are preceded by double hyphen ("--") followed by the string "*Varadhi*". In addition, OMG specified standard ORB options are also recognized. These options are preceded by single hyphen followed by the three letter "*ORB*".

Varadhi's run-time options (such as --VaradhiPORT) should appear in the command line before any application specific command line arguments. All command line arguments after Varadhi's run-time options are passed to the application.

The following section describes the list of options supported by Varadhi.

## 2.8.4 Options

### 1. --VaradhiPORT <portno>

where <portno> is a positive integer.

This option sets the port number that Varadhi server should use for listening in an IIOP based connection. By default, a Varadhi server listens on port 2000.

### 2. --VaradhiTIMEOUT <value>

where timeout is a positive integer (time in seconds).

This option sets the timeout value for blocking transport system calls.

### 3. --VaradhiVERSION

This option displays the version number and Copyright notice of SANKHYA Varadhi ORB.

### 4. -ORBid <orb\_identifier>

where <orb\_identifier> is an ORB identifier. The only valid value is “Varadhi”.

### 5. -ORBInitRef <serviceid>=<addr>

where

<service\_id> - Standard Service ID defined by OMG like  
NameService, Event Service

<addr> - URL to reach the particular Service’s server

application.

The currently supported URLs are:

1. IOR:..... - OMG IOR Stringified Object Reference
2. corbaloc:<' | "iiop:">[<version>'@'] [<host>[":"<port>]]["/"<objkey>]  
(A detailed explanation for using corbaloc object has been given under sec 2.9)
3. host\_ip\_addr:port\_no - Host (ip addr) and Port Number for TCP/IP IIOP connection.

#### 6. --VaradhiRedirect <old\_destn\_addr>=<new\_destn\_addr>

where

<old\_destn\_addr> - Original Address (in IOR) of the target object in the form IP\_Address:port

<new\_destn\_addr> - New address of the target object in the form IP\_Address:port

This option allows dynamically redirecting requests for a specific address to a new address.

### 2.8.5 Usage Examples

The following example shows how to make Varadhi server listen to port number 2001 instead of the default 2000 in an IIOP connection.

```
% server --VaradhiPORT 2001
```

The following example shows how to change the timeout parameter for blocking transport system calls.

```
% server --VaradhiTIMEOUT 20
```

The following example displays the version number and Copyright notice of SANKHYA Varadhi ORB.

```
% server --VaradhiVERSION
```

The following example shows how to redirect requests for an address 200.200.200.200:2000 to a new address 199.199.199.199:9999.

```
% client --VaradhiRedirect 200.200.200.200:2000=199.199.199.199:9999
```

## 2.9 Varadhi Configurations

The Varadhi Configuration parameters are used to create and configure Varadhi Platforms using *vconf* utility. Configuration parameters can be specified in a configuration file, or directly in '*vconf*' command line, when '*vconf*' is used in interactive mode.

Varadhi includes configuration files with pre-defined parameters and values. These files are placed in \$VARADHI/etc/config directory on Unix and in %VARADHI%\etc\config directory on Windows.

The following files are used to specify pre-defined Varadhi Run-Time configuration parameters.

- defaults.cfg** - Configuration file containing all pre-defined Varadhi Run-Time configuration parameters.
- static.cfg** - Configuration file that configures Varadhi Run-Time to use static memory allocation for heap memory. This configuration file includes defaults.cfg and overrides the values for static memory allocation.
- se.cfg** - Configuration file that configures Varadhi Run-Time with increased limits for Message Buffer, Number of Objects and Number of POAs. This requires defaults.cfg or static.cfg to be included before.

The following files are used to specify the Varadhi Target platform to create. These are the pre-defined platforms supported by Varadhi.

- sol2-gcc.cfg** - Configuration file for creating Solaris Varadhi Platform using GNU g++.

- linux-gcc.cfg** - Configuration file for creating Linux Varadhi Platform using GNU g++.
- win32.cfg** - Configuration file for creating Win32 Varadhi Platform using Microsoft VC++ 6.0.
- x86em-wince.cfg**- Configuration file for creating WinCE Varadhi Platform using Microsoft Embedded VC++ 3.0.

The following section explains each pre-defined Varadhi Configuration Parameters in detail.

### 2.9.1 Parameter Configurations

These parameters have the default maximum size of the marshalled message body.

#### 1. Varadhi::orb::msg\_param\_size

This parameter specifies the maximum size of the marshalled message body containing the parameters for an IDL operation. The default value specified in defaults.cfg is 448. This can be increased to a higher value for Varadhi run-time that needs to handle huge messages.

se.cfg specifies a maximum limit of 131072 bytes.

Example:

```
declare varadhi::orb::msg_param_size -type numeric -default 448  
-action cpp_define
```

#### 2. Varadhi::poa::max\_child\_list

This specifies the maximum number of child POAs that can be created in a

parent POA. The default value specified in defaults.cfg is 4. This can be increased to a higher value for Varadhi run-time that needs to create many POAs.

se.cfg specifies a maximum limit of 32.

Example:

```
declare varadhi::poa::max_child_list -type numeric -default 4 -  
action cpp_define
```

### 3. Varadhi::poa::max\_number\_of\_objects

This specifies the maximum number of objects that can be active in a POA at a time. The maximum value specified in defaults.cfg is 10. This can be increased for creating Varadhi Platforms for building Varadhi Servers that need to register more CORBA Objects in a POA.

se.cfg specifies a maximum limit of 10000 objects.

Example:

```
declare varadhi::poa::max_number_of_objects -type numeric -  
default 10 -action cpp_define
```

### 4. Varadhi::orb::max\_sock\_fd

This specifies the maximum number of file descriptors that can be used. If 0, it gets the maximum from the system.

Example:

```
declare varadhi::orb::max_sock_fd -type numeric -default 0 -  
action cpp_define
```

## 5. **Varadhi::orb::max\_number\_of\_location\_retry**

This specifies the maximum retries of the client request using LOCATION\_FORWARD replies in GIOP messages.

Example:

```
declare varadhi::orb::max_number_of_location_retry -type
numeric -default 10 -action cpp_define
```

### 2.9.1.1 Memory Management Parameters

#### 1. **Varadhi::poa::max\_number\_of\_poa**

This specifies the maximum number of POAs that can be created by Varadhi run-time with ‘static’ memory configuration. This is ignored for other configurations.

Example:

```
declare varadhi::poa::max_number_of_poa -type numeric -
default 4 -action cpp_define
```

The value specified in this parameter, multiplied by the value specified in ‘*varadhi::poa::max\_child\_list*’ parameter gives the maximum number of POAs that can be created.

#### 2. **Varadhi::orb::max\_number\_of\_ior**

This specifies the maximum number of InterOperable Object references that can be created by Varadhi run-time with ‘static’ memory configuration at a time. This is ignored for other configurations. The default value

specified in static.cfg is 31.

Example:

```
declare varadhi::orb::max_number_of_ior -type numeric -default
31 -action cpp_define
```

### **3. Varadhi::orb::max\_number\_of\_string\_32**

This specifies the maximum number of strings of 32 byte size that can be allocated by Varadhi run-time with ‘static’ memory configuration, at a time. This is ignored for other configurations. The default value specified in static.cfg is 31.

Example:

```
declare varadhi::orb::max_number_of_string_32 -type numeric -
default 31 -action cpp_define
```

### **4. Varadhi::orb::max\_number\_of\_string\_64**

This specifies the maximum number of strings of 64 byte size that can be allocated by Varadhi run-time with ‘static’ memory configuration, at a time. This is ignored for other configurations. The default value specified in static.cfg is 15.

Example:

```
declare varadhi::orb::max_number_of_string_64 -type numeric -
default 15 -action cpp_define
```

### **5. Varadhi::orb::max\_number\_of\_string\_256**

This specifies the maximum number of strings of 256 byte size that can be allocated by Varadhi run-time with ‘static’ memory configuration, at a

time. This is ignored for other configurations. The default value specified in `static.cfg` is 7.

Example:

```
declare varadhi::orb::max_number_of_string_256 -type numeric
        -default 7 -action cpp_define
```

## 6. **Varadhi::orb::max\_number\_of\_string\_1024**

This specifies the maximum number of strings of 1024 byte size that can be allocated by Varadhi run-time with ‘static’ memory configuration, at a time. This is ignored for other configurations. The default value specified in `static.cfg` is 3.

Example:

```
declare varadhi::orb::max_number_of_string_1024 -type numeric
        -default 3 -action cpp_define
```

## 7. **Varadhi::mem::use\_bss**

This parameter can be used to specify whether to allocate heap memory statically using BSS or to allocate heap memory initially in `CORBA::ORB_init()`.

### 2.9.2 **Feature Configuration Parameters**

The feature configuration parameters can be used to enable or disable particular features in Varadhi. These are boolean type parameters.

Example:

The following `vconf` command enables C++ exception handling and C++

RTTI feature.

```
declare enable_cpp_exceptions -type bool -default yes -flag
vconf_rtti -action cpp_define
```

### 2.9.3 Service Configuration Parameters

Named Object Services are registered with Varadhi run-time by using Service configuration parameters. The simple ObjectID to be registered is the configuration parameter names.

For example, “*declare NameService ...*”, registers an Object Service whose ObjectID is “NameService”.

#### Example:

```
declare NameService -type service -id "IDL:omg.org/
CosNaming/NamingContext:1.0"-default
"127.0.0.1:2069" -action register
```

The argument ‘-type’ specifies that this parameter is a Service configuration parameter. The argument ‘-id’ specifies the IDL Repository TypeID for the Object to be returned while the service is resolved using “*CORBA::ORB::resolve\_initial\_reference()*” call.

The argument ‘-default’ specifies the default ObjectURL to be used for reaching the Service. This can be a stringified IOR or corbaloc Object URL.

### 2.9.4 Varadhi Library Configuration Parameters

This configuration parameter specify the type of Varadhi run-time to be generated for a Varadhi Platform. The types are “dynamic”, “static”, “debug” and “profile”.

```
declare varadhi::lib -type list -default dynamic -values  
static:dynamic:profile:debug -action make_target
```

where

**dynamic** - version of Varadhi run-time that uses the OS memory manager  
for heap memory.

**static** - version of Varadhi run-time that uses Varadhi memory manager  
for heap memory.

**debug** - version of Varadhi run-time that prints debug messages for  
debugging purposes.

**profile** - version of Varadhi run-time that gives profile information

## 2.10 IDLC generated Stub and Skeleton code

### 2.10.1 Overview

From an IDL file, the stub and skeleton files are generated using ‘*idlc*’, the IDL to C++ compiler. The stubs and skeletons represent the client and server side of the IDL interface respectively.

The stub is essentially a proxy of the remote server object. It enables the user to invoke methods on server objects using normal C++ method call syntax on this proxy object. This essentially hides the user from the implementation and he access details of the remote object.

Skeleton is actually a proxy for the server side. The ORB dispatches the request to the Skeleton through the Portable Object Adapter (POA) in which the Server Object has registered. The Skeleton in turn dispatches this request to the appropriate implementation object.

### 2.10.2 Generated Stub and Skeleton Files

The following files are generated by the IDL compiler for a given IDL file on Unix hosts.

<idl\_basename>\_st.h- Stub header file

<idl\_basename>\_st.cc - Stub source file

<idl\_basename>\_sk.h - Skeleton header file

<idl\_basename>\_sk.cc- Skeleton source file

The stub header file (<idl\_basename>\_st.h) contains the mapping for the IDL types in the IDL source file. It contains the definition for the proxy object.

The stub source file (<idl\_basename>\_st.cc) has the proxy implementation of the interface and is responsible for sending the request to the remote ORB. It marshals and sends the request to the ORB and finally returns the result to the client as in the following code.

```

_idlclv_param.allocate(v_orb_idx);
_idlclv_param.marshall(&x, Param::write_long, &y, Param::write_long, 0);
v_orb()->orbPtr->send_request(this->__get_reference(), "add",
_idlclv_param, 1);

.... // Exception Handling Code

_idlclv_param.marshall(&_idlclv_res, Param::read_long, 0);
_idlclv_param.deallocate(v_orb_idx);
return _idlclv_res;

```

The skeleton header file (<idl\_basename>\_sk.h), contains the server side mapping of the IDL interfaces, if any.

The skeleton source file (<idl\_basename>\_sk.cc) contains the server side dispatch routines to dispatch the request from the ORB to the appropriate C++ implementation object.

```

if (v_os->strcmp(_idlclv_op, "add") == 0) {
    CORBA::Long x;
    CORBA::Long y;
    CORBA::Long _idlclv_res; _idlclv_param.marshall(&x,
        Param::read_long, &y, Param::read_long, 0);

    ....//Exception Handling Code

```

```

        _idlclv_res = add(x, y);

        ...//Exception Handling Code

        _idlclv_param.allocate(v_orb()->orbPtr->v_orb_idx);
        _idlclv_param.marshall(&_idlclv_res, Param::write_long, 0);
    }

```

The dispatch routine invokes the actual implementation and the result from the implementation is returned to the client through the ORB.

### 2.10.3 Client Side Interface mapping

#### 2.10.3.1 Stub class for an IDL interface

For an IDL interface, ‘*idlc*’ generates the stub classes in the <idl\_basename>\_st.h file. The stub class name is same as the IDL interface name. Objects of these types should not be constructed by the client application. Varadhi prevents construction of stub objects. For a CORBA server object, the client gets the pointer to the stub object from the ORB using ORB APIs like CORBA::ORB::string\_to\_object().

```

CORBA::Object_ptr obj = orb->string_to_object(stringified_ref);
Adder_ptr adder_obj = Adder::_narrow(obj);
....
CORBA::release(adder_obj);

```

The returned object should be freed explicitly using CORBA::release() after its usage.

### 2.10.3.2 `_var` type for Interfaces

The `_var` type is used for user defined IDL types for managing dynamically allocated memory. A '`_var`' is also generated by the `idlc` compiler for the interface types. The compiler generates a `<T>_var` class for each IDL interface type. This can be used instead of the `<T>_ptr` type in the above example. When a dynamically allocated object is assigned to the `<T>_var` object, `<T>_var` takes care of freeing the object. When the `<T>_var` is destroyed or goes out of scope, the memory allocated is automatically freed thereby relieving the programmer of memory management.

```
CORBA::Object_var obj = orb->string_to_object(stringified_ref);
Adder_var adder_var = Adder::_narrow(obj);
....
```

`CORBA::release()` need not be called since the `Adder_var` type will take care of releasing the proxy object when it goes out of scope.

### 2.10.4 POA Based Server Side Interface Mapping

The POA performs the job of creating object references, activating and deactivating servants, decoding and dispatching the client requests to the appropriate servant and invoking the appropriate operations of the servant.

POA based servant implementation is of two type. They are:

- **Inheritance Based Mapping**
- **Tie based Mapping**

### 2.10.4.1 Inheritance Based Mapping

In Inheritance based mapping, the implementation classes are derived from a base class based on the OMG IDL interface definition. The generated base classes are known as ‘Skeleton Classes’ and the derived classes are known as ‘Implementation Classes’ or “Servant Class”. For each operation specified in the interface a virtual function is declared in the skeleton class. The skeleton invokes the methods in the Servant through calls to these virtual functions.

The name of the skeleton class is formed by prefixing ‘POA\_’ with the actual name of the interface given in the IDL file. If the interface is within a module in the IDL file, the name of the skeleton class will be of the form POA\_<module name>::<Interface name>

In the “Adder” example given above, the generated skeleton class is ‘POA\_Adder’ which is derived from the servant base class ‘PortableServer::ServantBase’.

This is as shown below.

```
class POA_Adder : public virtual Adder, public virtual
    PortableServer::ServantBase
{
public:
    POA_Adder() {};
    ~POA_Adder() {};
    inline Adder_ptr _this();
    virtual CORBA::Object_ptr __get_object() { return this; };
    virtual void dispatch(char *op, Param& param);
};
```

### 2.10.4.2 Tie Based Mapping

The tie based mapping is based on delegation and C++ templates. A delegation class called 'tie' is generated by the IDL compiler in addition to the Inheritance based skeleton class. The actual implementation class need not derive from any of the IDL generated classes. It is just required to implement all the IDL operations for the Interface.

In the 'tie' based implementation, an object of the actual implementation class is tied to an object of the template 'tie' class. This is done by passing the reference or pointer to the actual implementation object to the appropriate constructor of the tie class. The template 'tie' class is registered with the POA. Operations on the actual implementation class is then invoked via delegation from the 'tie' class.

This technique is quite useful when there is a need to use the existing C++ class to implement the CORBA object, without inheriting from the generated skeleton class. This allows invocation of legacy systems from a wrapper class which is then tied to the 'Tie' based Servant class.

The name of the tie class is of the form 'POA\_<interface\_name>\_tie'. For the adder example, the tie class generated would be 'POA\_Adder\_tie'.

### 2.10.5 Exceptions

CORBA::Exceptions is an abstract class that is the base of all the CORBA exceptions.

#### 2.10.5.1 CORBA System Exceptions

CORBA System Exception is a class derived from the CORBA::Exception class. Varadhi supports all CORBA System Exceptions. Some of the less

used Systems Exceptions are not enabled by default. This can be enabled using the `vconf` parameter “`enable_addl_corba_system_exceptions`”.

To print the CORBA System Exception, Varadhi provides an API

```
V_ORB::print_exception(const CORBA::Exception &ex)
```

This API will print the exceptions along with the respective minor codes as given in the following sections.

#### **2.10.5.1.1 CORBA System Exception Vendor minor codes**

Each CORBA system exception includes a minor code to designate the subcategory of the exception. Minor exception codes are of type unsigned long and consist of a 20-bit "Vendor Minor Codeset ID" (VMCID), which occupies the high order 20 bits, and the minor code which occupies the low order 12 bits.

The OMG assigned Vendor Minor Code ID value for SANKHYA Varadhi is 0x53410000. This occupies the high order 20 bits of the exception minor code. The lower order byte indicates the exact cause of the exception.

The tables in the following pages specify the minor codes for the various CORBA System Exceptions.

The following table specifies SANKHYA Varadhi minor codes for the Transport/Network type CORBA System exceptions.

**TABLE 1. Transport/Network Message Codes**

Minor code	Explanation
0xC0	Client Connection Not Accepted
0xC1	Adding Client Failed
0xC2	Select Timed-out
0xC3	Receiving Message Failed
0xC4	Sending Message Failed
0xC5	Transport Initialization Failed
0xC6	Transport Bind Failed
0xC7	Connection To Server Failed
0xC8	Error in sending message
0xC9	Data Not Sent Properly
0xCA	Error In Reading Message
0xCB	Data Not Read Properly
0xCC	Listen Failed
0xCD	Invalid Operation
0xCE	Unknown Error

The following table specifies SANKHYA Varadhi minor codes for General code Type CORBA System exceptions.

**TABLE 2. General Message Codes**

Minor Code	Explanation
0x81	Memory Allocation Failed
0x82	Marshal Buffer Overflow
0x83	License Check Fail

The following table specifies SANKHYA Varadhi Minor Codes for Argument Parser related CORBA exceptions.

**TABLE 3. Argument Parser Message Codes**

Minor Code	Explanation
0x90	Argument Missing
0x91	Unknown argument
0x92	Flag

The following table specifies the SANKHYA Varadhi minor codes for ORB related CORBA System exceptions

**TABLE 4. ORB Message Codes**

Minor Code	Explanation
0xA0	Wrong Previous Reply
0xA1	Wrong Message Type
0xA2	Unknown Profile
0xA3	Invalid IOR String
0xA4	Object Type Not Available
0xA5	Servant Not Found

1. Refer the 'Exception' sample demo application present in \$VARADHI/samples (%VARADHI%\samples on Windows) to see how minor codes are used in CORBA Applications.

For example:

```
try {  
    ...  
  
} catch ( const CORBA::BAD_PARAM &ex ) {  
    cout << "Received BAD_PARAM Exception with"  
    cout << "  minor code    --> " << ex.minor() << endl;  
    cout << "  completed status --> " << ex.completed() << "\n\n";  
}
```

### 2.10.5.2 User Exceptions

A CORBA IDL exception is mapped to a C++ class that is derived from the standard **CORBA::UserException** class. The ‘User Exception’ class is derived from a base **CORBA::Exception** class defined in the CORBA module.

Varadhi supports CORBA UserExceptions in systems without C++ Exception support also. An example is provided along with SANKHYA Varadhi that illustrates use of CORBA UserException. Refer to the “*exceptions*” demo at \$VARADHI/samples directory (%VARADHI%\samples on Windows).

## 2.11 Using corbaloc Object URL in SANKHYA Varadhi

SANKHYA Varadhi has basic support for Object URLs of corbaloc format to be converted into object references. Currently, it supports objects that can be contacted by IIOP.

It supports the following format of corbaloc URLs

```
corbaloc:<':' | "iiop:">[<version>'@'] [<host>[ ":" <port>]] [ "/" <objkey>]
```

If host or port is not provided, the default values "localhost" and "2809" will be used respectively.

corbaloc format IORs can be used in the following cases.

1. -ORBInitRef Varadhi ORB option to bootstrap using 'CORBA::ORB::resolve\_initial\_references()' API,
2. In CORBA::ORB::string\_to\_object() API to convert the Object URL in to an object reference.

Pre-requisites

Tools : SANKHYA Varadhi vconf tool

### 2.11.1 Registering a Service/CORBA Object with SANKHYA Varadhi

Any service or a CORBA object whose object reference needs to be accessed via CORBA::ORB::resolve\_initial\_references(), or whose corbaloc format object URL needs to be converted to an Object Reference

using `CORBA::ORB::string_to_object()`, needs to register with SANKHYA Varadhi configuration utility ‘vconf’. ‘vconf’ provides a ‘register’ action to register a CORBA Object/Service with Varadhi.

To register a service, the following vconf configuration syntax is used.

```
% declare <service> -type service -id <IDL type> -default <host:port> -  
action register
```

## 1. Create a configuration file that registers the Service/CORBA Object

Open your favorite text editor and create `myobject.cfg` containing the following lines.

```
myobject.cfg
```

```
include <defaults.cfg>  
include <linux-gcc>
```

```
declare Adder -type service -id "IDL:Adder:1.0" -default "127.0.0.1:2809"  
-action register
```

## 2. Create a Varadhi Target Platform

Create a Varadhi Target Platform as below using the `myobject.cfg` file created above.

```
% cd my_work_dir  
% vconf myobject.cfg
```

### 3. Set up the Varadhi Target Platform Environment

The above configuration places a *platform.csh* file in the \$VARADHI\_PLATFORM directory to setup Varadhi Target Environment. Here, VARADHI\_PLATFORM refers to the root directory where Varadhi Platform is created. Source the file *platform.csh* to setup the Varadhi Target platform environment.

```
% cd varadhi
% source platform.csh
```

### 4. Using corbaloc URL on the client side

On the client side, corbaloc URL can be used to

a) Resolve an Object's reference using `resolve_initial_references()` API.

-ORBInitRef ORB option can be used to override the Object Location specified in the configuration file.

```
% -ORBInitRef NameService=corbaloc:....
```

The application uses `resolve_initial_references()` as below.

```
// Resolve initial Naming Context.
CORBA::Object_var nc_obj =
    orb->resolve_initial_references("NameService");
```

b) Convert corbaloc URL of CORBA Objects/Services to Object reference. The Objects/Services should have been registered with SANKHYA

Varadhi as explained in ‘Registering a Service/CORBA Object with SANKHYA Varadhi’

```
// Get "Adder" server's object reference using Object ID.  
CORBA::Object_ptr objref =  
    orb->string_to_object( "corbaloc:iiop:localhost:2809/Adder" );
```

Using corbaloc URL to access objects in SANKHYA Varadhi server application.

To use corbaloc format Object URL, the server should understand the simple object key provided by the client side. The server side, objects/ services registered with SANKHYA Varadhi will work only if the Object is activated with the "RootPOA" using

‘**PortableServer::POA::activate\_object\_with\_id()**’ API as follows.

```
// Create an "Adder" servant, and activate the server Object.  
Adder_impl* server = new Adder_impl;  
  
PortableServer::ObjectId_ptr oidp =  
    PortableServer::string_to_ObjectId("Adder");  
poa->activate_object_with_id(*oidp, server);
```

Limitations:

1. Currently only IIOP type protocol format is allowed.
2. On the server side for an Object/Service that is registered with SANKHYA Varadhi to be addressed using corbaloc URL by the clients, the server object should be registered in the Root POA using the API `PortableServer::POA::activate_object_with_id()`

3. If more than one address is specified in the corbaloc URL, only the first address is used.

### **2.11.2 Exit Status and Error Messages**

Varadhi will exit with status 1 on the following error conditions.

**Unknown argument** - This error message is displayed when the option name is unknown or blank

**Argument <option>** - This error message is displayed when the option requires a value (e.g port number) and the value is missing in the command line.

# Index

## **B**

BindingIterator 25  
Building customized 'ns' 57

## **C**

CC\_EXTRA\_FLAGS 21  
Client 9  
Client Side Interface mapping 100  
CORBA 22, 10  
CORBA IDL 8  
CORBA Technology 8  
corbaloc Object URL 108  
CosNaming 25  
Creating a new Varadhi Platform 33, 35

## **D**

Develop The Client Application 10  
Distributed Embedded Systems 5  
Distributed Object Oriented Embedded Systems Development 6  
DUMPIOR 24, 51

## **E**

es - Varadhi Event Service 31  
Event Service 27

## **F**

Feature Configuration 95

## **G**

GIOP 5

## **I**

IDL 9

idlc 10, 23, 38  
IIOP 5  
Inheritance Based Mapping 102  
Introduction 1

## **L**

Language Support in Varadhi 4

## **M**

make 21  
Minimum CORBA 7

## **N**

Naming Service 25  
NamingContext 25, 68  
ns 26, 53  
nsman 27, 66

## **O**

Object Management Architecture (OMA) 6  
Object Oriented Systems 5  
OMG 7  
ORB Configuration 96  
ORB run-time 22  
ORB run-time command line options 22  
ORB\_init() 23

## **P**

POA Based Server Side Interface Mapping 101  
Portable Object Adapter 13

## **R**

Registering a Service 108  
RootPOA 13

**S**

SANKHYA Varadhi 22  
Server 9, 11, 22  
Service Configuration 96  
Setting Up Varadhi Host Development Environment 15  
Skeleton 98  
Stub 98

**T**

Tie Based Mapping 103

**U**

User Exceptions 103, 107  
Using SANKHYA Varadh 15

**V**

Varadhi 22

orb

max\_number\_of\_ior 93  
max\_number\_of\_string\_1024 95  
max\_number\_of\_string\_256 94  
max\_number\_of\_string\_32 94  
max\_number\_of\_string\_64 94  
max\_sock\_fd 92  
msg\_param\_size 91

poa

max\_number\_of\_objects 92

Varadhi Names 26  
Varadhi Names Manager 27  
Varadhi Naming Service Manager 66  
Varadhi ORB run-time options 86  
Varadhi Platform 35, 37

VARADHI\_HOST 15  
VARADHI\_PLATFORM 16  
--VaradhiPORT 23  
--VaradhiTIMEOUT 23  
vco 23  
VCONF 23, 41  
vconf 32  
VCONF - Interactive Mode of Operation 46  
void ns\_init( CORBA

ORB\_ptr orb, PortableServer

POA\_ptr poa, char\* ior\_file = NULL ) 63  
void ns\_shutdown() 65

### **Symbols**

\_Var Type for Interface 100, 101

---

## For More Information-

---

Varadhi Evaluation	<a href="http://www.sankhya.com/info/varadhi.html">http://www.sankhya.com/info/varadhi.html</a>
Varadhi Download	<a href="http://www.sankhya.com/info/products/varadhi/download.html">http://www.sankhya.com/info/products/varadhi/download.html</a>
Varadhi Roadmap	<a href="http://www.sankhya.com/info/products/varadhi/roadmap.html">http://www.sankhya.com/info/products/varadhi/roadmap.html</a>
Varadhi Documentation	<a href="http://www.sankhya.com/info/products/varadhi/docs.html">http://www.sankhya.com/info/products/varadhi/docs.html</a>
Varadhi Sales & Support	<a href="mailto:sales@sankhya.com">sales@sankhya.com</a> , <a href="mailto:varadhi-support@sankhya.com">varadhi-support@sankhya.com</a>

---

## **SANKHYA™**

**Sankhya Technologies Private Limited**  
**“Jayashree”, Third floor, #13/2 Jayalakshmipuram,**  
**First street, Nungambakkam,**  
**Chennai 600 034 INDIA**  
**Tel: +91 44 2822 7358**  
**Fax: +91 44 2822 7357**

**Sankhya Technologies Private Limited**  
**#30-15-58, Third Floor, “Silver Willow”**  
**DabaGardens**  
**Visakhapatnam 530 020, INDIA**  
**Tel: +91 891 5542 666**  
**Fax: +91 891 5542 665**  
**<http://www.sankhya.com>**

---

SANKHYA, Varadhi, The Digital Bridge, SANKHYA Software and SANKHYA TECHNOLOGIES are trademarks of Sankhya Technologies Private Limited. OMG marks and logos are trademarks or registered trademarks, service marks and/or certification marks of Object Management Group, Inc. registered in the United States. All other brands and names are the property of their respective owners.

---