

## CORBA<sup>®</sup> With SANKHYA Varadhi<sup>™</sup> A Kick Start Guide

### Introduction

The document intends to introduce the various CORBA features and Varadhi functionalities through sample applications that are shipped with SANKHYA Varadhi. It also provides useful information on each of these samples so that the user can get an overview of the powerful features provided by Varadhi.

### Varadhi Samples

Varadhi provides several ready to use samples that demonstrate the use of standard CORBA features and Varadhi extensions. The samples contain one or more programs which together provide client/server functionality of a distributed application using Varadhi. Varadhi samples can be located as follows

On Unix csh

\$VARADHI/samples

On Windows

%VARADHI%\samples

Where \$VARADHI and %VARADHI% refer to the installation location on Unix (Solaris and Linux) and Windows respectively.

The table below lists a CORBA feature or Varadhi functionality illustrated by each Varadhi sample.

| CORBA Feature or Varadhi Functionality  | See this Varadhi sample |
|---|-------------------------|
| A simple Varadhi application  | <i>Adder</i>            |
| A single program acting as both client and server   | <i>Multiplier</i>       |
| Collocation of objects  | <i>Coloc</i>            |
| Usage of ORB::work_pending() and ORB::perform_work() to perform client and server tasks in a single main thread | <i>Pingpong</i>         |
| Usage of sequence of strings  | <i>Stringseq</i>        |
| IDL Union type  | <i>Union</i>            |
| Sending and receiving CORBA objects between clients and servers   | <i>Hostserver</i>       |

| <b>CORBA Feature or Varadhi Functionality</b>  | <b>See this Varadhi sample</b> |
|--|--------------------------------|
| POA's persistent lifespan policy for creating persistent object                        | <i>Persist</i>                 |
| CORBA Exception Handling and Varadhi Extension   | <i>Exceptions</i>              |
| Usage of Varadhi Naming Service, ns, to locate and invoke operation on a server object | <i>Names</i>                   |
| Usage of Varadhi Event service, es that is used by consumer and supplier applications. | <i>Events</i>                  |
| Usage of corbaloc format object url  | <i>corbaloc</i>                |
| Usage of ORB::work_pending() and ORB::perform_work()                                   | <i>chat</i>                    |
| Usage of event service   | <i>event_chat</i>              |
| Usage of CORBA local interfaces  | <i>local_interface</i>         |
| Usage of USE_DEFAULT_SERVANT POA Policy  | <i>defservant</i>              |
| Transferring a file from client location to server                                     | <i>vcopy</i>                   |
| Adding Varadhi Naming Service functionality to an application                          | <i>vnames</i>                  |
| Applying Fault tolerance in Varadhi applications                                       | <i>ftcounter</i>               |
| Usage of DII and DSI   | <i>dii_dsi</i>                 |
| Usage of 'Any' data type in IDL  | <i>any</i>                     |
| Multithreaded request handling in adder sample   | <i>multithread/adder</i>       |
| Invoking operations from two separate threads simultaneously.                          | <i>multithread/add_strseq</i>  |
| Collocation of objects in a multithreaded environment                                  | <i>multithread/colloc</i>      |
| Usage of multithreaded exceptions  | <i>multithread/exceptions</i>  |
| Varadhi for Web services   | <i>soap_wsdl</i>               |
| Usage of Request Interceptors  | <i>pi_scontext</i>             |

To download and evaluate SANKHYA Varadhi, go to:  
<http://www.sankhya.com/info/products/varadhi/download.html>

After installing Varadhi, refer to \$VARADHI/samples/README for information on building and running the demo programs.

Here is more information on the sample programs:

**1. Adder:**

This demo program illustrates handling of client request by the server for simple operations (here, addition of two numbers) and return the result (value) to the client.

**2. Multiplier:**

This is similar to adder demo. Here a single application acts as both client and server. By making use of the service of an adder server, the multiplier server multiplies two numbers using addition.

**3. Colloc:**

Collocation is a demo program to illustrate collocation of objects. The demo application multiplies two numbers using a collocated adder object.

**4. Pingpong:**

This demo program illustrates how the main application thread can use the polling mechanism to multiplex between ORB requests and other application specific activities. This demo program uses the following Varadhi functionalities to perform client and server tasks in a single main thread.

1. ORB::work\_pending()
2. ORB::perform\_work()

**5. StringSeq:**

StringSeq is a demo program to illustrate String Sequence usage. The samples directory contains a client/server implementation of String Sequence demo. The server stores a string sequence and client sets and gets values from it.

**6. Union:**

Union is a demo program to illustrate IDL Union data type usage. This is a client/server implementation of union demo which adds two unions just like the adder demo.

**7. Hostserver:**

Hostserver is a demo program to illustrate passing of objects between clients and servers. It behaves like a Naming server. The 'hostserver' program stores the Object Reference of Objects (servers) of "IDL:Host" type, running in different host machines. Clients use this Object reference to invoke operation on the individual 'host' servers.

**8. Persist:**

Persist is a demo program to illustrate PERSISTENT CORBA Object. The demo program adds two numbers using a Persistent CORBA Object.

**9. Exceptions:**

Exceptions is a demo program to illustrate exception handling in CORBA. The application adds two numbers using a client/server model and throws/ catches CORBA exceptions for wrong arguments. The demo also illustrates the usage of Varadhi Extension to deal with CORBA exceptions on systems with no support for CORBA exception handling.

**10. Names:**

Names is a demo program that uses Varadhi Naming Server to locate and invoke operation on a server object. This is a client/server implementation of adder demo that uses Varadhi Naming Service instead of stringified IOR files.

**11. Events:**

This demo demonstrates the push model of the event service. In this sample, the consumer and supplier applications use Varadhi event service.

**12. corbaloc:**

Corbaloc is a demo program to demonstrate the use of corbaloc format object url. The server creates "Adder" object and registers it. The client uses the corbaloc format Object URL to get the object reference of the "Adder" object using the Object ID "Adder".

**13. chat:**

Chat is a demo program that uses `ORB::work_pending()` and `ORB::perform_work()` operations. The demo program acts as both client and server. This is a normal interactive chat program. (like pingpong) that illustrates how the main application thread can use the polling mechanism to multiplex between ORB requests and other application specific activities.

**14. event\_chat:**

EventChat is a demo program to send/receive messages using Varadhi Event Service.

**15. any:**

This demo illustrates the usage of 'Any' data type using a client/server model.

**16. defservant:**

This demo program illustrates the use of `USE_DEFAULT_SERVANT` POA Policy. It also illustrates using a Single Servant for multiple Objects.

The server program sets a default servant with the POA and creates two object references. The client uses these object references to issue requests. The server uses the default servant to handle the requests from these two objects.

The server is a "whoami" server that returns the Object ID of the object invoking

the 'whoami()' request.

**Note:** This demo will work only with the full CORBA version of Varadhi.

#### 17. dii\_dsi:

This demo program performs dynamic invocation of request to objects by means of Dynamic Invocation interfaces (DII) and handles object invocation by means of Dynamic Skeleton interfaces. This demo program also uses Varadhi Interface Repository Server to determine the operation parameters and for type checking.

#### 18. vcopy:

vcopy is a demo program to transfer a file from client location to server.

#### 19. vnames:

vnames illustrates how the functionality of Varadhi Naming Service can be added to an application.

In this sample, the server, which is linked with Varadhi Naming Service library, acts as a Naming Service. The server and client use the in-built Naming Service in the server application for binding and resolving objects.

#### 20. local\_interface:

This demo program adds two numbers using CORBA local interfaces. The reference to local interface in varadhi is obtained using a call to create\_object().

#### 21. FTCounter:

FTCounter is a demo program that demonstrates the following:

- 1) A simple CORBA application (see 2) with redundant server objects.
- 2) Counter and fthost objects and a process to implement ft\_counter processing (in this demo these are simple counting operations)
- 3) Counter objects that implement Checkpointable interface.
- 4) A FTHost Object that implements PullMonitorable interface & processcontrol interface.
- 5) An 'ft\_counter' process that acts as Host Object Server, Fault Detector and Clock Generator for Counter Object. It also performs logging and recovery from fault.
- 6) At the end of each clock tick (clock()) to active counter object (ft\_counter process), the active counter object state is stored in the logging service from where the passive counter object (ft\_counter process) will restore the state in case of failure.
- 7) An 'ftsim' Process acts as a Fault Simulator for 'ft\_counter' process using 'processcontrol' interface.

The 'ft\_counter' process performs counting by issuing a request to the Counter Object. Two different instances of the 'ft\_counter' application should be run (in different terminal on the same or different hosts with VARADHI\_FT\_COUNTER Environment Variable properly set) to gain a simple Fault Tolerance FTCounter System.

FTCounter uses the following functionalities:

- `ft::is_alive()`
- `ft::get_state()`
- `ft::set_state()`
- `ORB::work_pending()`
- `ORB::perform_work()`

## 22. multithread:

The samples under Multithread (`$VARADHI/samples/multithread`) illustrate how multithreaded requests can be handled in Varadhi applications effectively. The following is a brief description on the samples present under this directory.

### *a) adder:*

Multithreaded Adder demo illustrates multithreaded request handling in the server side. The "Adder" servant implementation introduces a delay of 2 seconds in the call to "add()" operation. The client invokes the "add()" operation in multiple threads.

If multithreaded request handling is enabled, each client operation takes around 2 seconds. If multithreaded request handling is disabled, then the requests are serialized and the client operations take incremental time to complete.

### *b) add\_strseq:*

This demo program invokes 'Adder' and 'StringSequence' operations from two separate threads simultaneously. The "Adder" server adds two numbers and the "StringSequence" server adds two string sequences.

### *c) colloc:*

Collocation demo illustrates collocation of objects in a multithreaded environment. In two separate threads, the application multiplies two numbers using a collocated adder object.

The multiplier object, which is also collocated, uses the service of the adder object to multiply two numbers using addition.

### *d) exceptions:*

This demo is the multithreaded version of the exception demo found in the "samples" directory. In two separate threads, the application invokes "add()" operation

## 23) soap\_wsdl :

This demo program converts celsius to fahrenheit temperature and vice-versa using a service described in WSDL and SOAP protocol over HTTP in a client/server model.

## 24) pi\_scontext:

This is a demo program that performs dynamic invocation of request to objects by means of Dynamic Invocation interfaces and handles object invocation by means of static Skeleton interfaces. This demo program uses Request interceptors of Varadhi run time environment to transfer service context information. Client,

server implementation of pi\_scontext demo uses Varadhi Portable interceptors to update/retrieve service context information of GIOP request and reply messages during Conveyor::ping() operation invocation. Operation Conveyor::ping() of this demo doesn't make use of any explicit arguments. pi\_scontext demo also illustrates the flow of request interceptors.

**Conclusion:**

In this guide we have seen how Varadhi samples illustrate the key features available in CORBA.