DISTRIBUTED LOGGER FOR CORBA BASED SCADA APPLICATION

PROJECT WORK DONE AT
Electronic Research & Development Center of India

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
MASTER OF COMPUTER APPLICATIONS
OF BHARATHIAR UNIVERSITY, COIMBATORE.

SUBMITTED BY
SHOLY MOHAN
Reg.No.9938M0642

GUIDED BY

EXTERNAL GUIDE
Mr. B. Ramani,
Assistant Director,
Electronics Research & Development of India,
Thiruvananthapuram.

INTERNAL GUIDE
Mr. S. Ganesh Babu MCA.,
Lecturer,
Department of Computer Science & Engg.,
Kumaraguru College of Technology,
Coimbatore.

Department of Computer Science & Engineering
KUMARAGURU COLLEGE OF TECHNOLOGY
Coimbatore-641006
MAY 2002

R&DCI

भारतीय इलेक्ट्रॉनिकी अनुसंधान एवं विकास केन्द्र
(भारत सरकार सूचना प्रौद्योगिकी मंत्रालय
का स्वायत्त वैज्ञानिक संस्थान)
तिरुवनन्तपुरम यूनिट
डा.पे.सं. 6520, वेल्लयंबलम
तिरुवनन्तपुरम 695 033, भारत

ELECTRONICS RESEARCH AND DEVELOPMENT CENTRE OF INDIA
(An Autonomous Scientific Society of Ministry of Information Technology, Government of India)
Thiruvananthapuram Unit
P.B. No. 6520, Vellayambalam
Thiruvananthapuram 695 033, India

Phone: +91-471-320116 *Fax:* +91-471-331654, 332230 *email:* erdc@erdcitvm.org *Web:* www.erdcitvm.org

CERTIFICATE

This is to certify that the project work entitled DISTRIBUTED LOGGER FOR CORBA BASED SCADA APPLICATION is a bonafide record of the work done during December 2001 - April 2002 by Sholy Mohan, student of Department of Computer Science, Kumaraguru College Of Technology, Coimbatore in partial fulfillment for the award of the Degree of Master Of Computer Application (MCA) from Bharathiyar University.

External Guide

Ramani.B

(B.RAMANI)
JOINT DIRECTOR

hiruvananthapuram
h April 2002

ELECTRONICS RESEARCH & DEVELOPMENT CENTRE ★ A SCIENTIFIC SOCIETY UNDER MINISTRY OF INFORMATION TECHNOLOGY ★ GOVT. OF INDIA VELLAYAMBALAM TRIVANDRUM

Department of Computer Science & Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to the Bharathiar University)

Coimbatore-641006

CERTIFICATE

This is to certify that the project work entitled

DISTRIBUTED LOGGER FOR CORBA BASED SCADA APPLICATION

Done by

Sholy Mohan

9938M0642

Submitted in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications of Bharathiar University.

Professor and head
Department of Computer Science & Engineering

Internal Guide

Submitted to university examination held on 09-05-2002

Internal Examiner

External Examiner

DECLARATION

I here by declare that the project entitled **'DISTRIBUTED LOGGER FOR CORBA BASED SCADA APPLICATION '** submitted to **Bharathiar University** as the project work of Master of Computer Applications Degree, is a record of original work done by me under the supervision and guidance of **Mr. B.Ramani, Assistant Director ER&DCI, Thiruvananthapuram** and **Mr. S Ganesh Babu MCA, Lecturer, Kumaraguru College of Technology, Coimbatore** and this project work has not found the basis for the award of any Degree /Diploma/ Associateship/ Fellowship or similar title to any candidate of any university.

Place: THIRUVANANTHAPURAM

Sholy Mohan

Date: 16-04-2002

Internal guide

Ramani B

External Guide

ACKNOWLEDGEMENT

I am incredibly fortunate to have so many people helping me on this project. First I wish to thank ***Mr. B Ramani Assistant Director ER&DCI, Thiruvananthapuram*** *for entrusting this project to me and providing all facilities that made the experience a pleasant one. In particular my deepest thanks goes to* ***Mr. Shahin Research Associate****, for recognizing and approving this project and for his immense support, timely suggestions, and technical guidance throughout the project work.*

I am extremely thankful To ***Dr.S.Thangasamy, Professor and Head of the Department*** *for his kind suggestion he has given in every step throughout our studies and in this project work. I am grateful to my internal guide* ***Mr. S Ganesh Babu MCA, Lecturer, Kumaraguru College Of Technology*** *who offered his guidance and always supported me with keen interest , constant encouragement and suggestions in every step through out the project.*

I am extremely thankful to ***Ms Lancy Thomas*** *and* ***Ms Santha*** *for providing facility to carry out my work. My deepest gratitude and highest admiration goes to* ***Mr. Vinu Kumar, Mr. Benoy*** *and* ***Mr. Vinod*** *for their valuable suggestion and help. I deeply appreciate the efficient and committed work of Ms Deepa. Her attention to detail and deadlines made this project going smoothly. I would also like to thank the other staff especially* ***Mr. Sunder****.*

I wish to thank my parents for all their encouragement support and love. Without their encouragement I would never have gotten where I am.

Last but not the least I would like to thanks my Classmates and friends for their love and moral support. Finally thanking You God for your immense support and love

SYNOPSIS

"**DISTRIBUTED LOGGER FOR CORBA BASED SCADA APPLICATION**" is a CORBA based distributed Application developed for automating the Energy Control System. This is developed for ER&DCI Thiruvananthapuram. This is mainly developed for getting the online data from the devices and stores these values of the separate devices in appropriate files for future use. This contains three main modules Device Configuration Client, Logger Client and, Logger Server. Device Configuration Client configures new tags or modify/delete configuration from the existing configuration setting. Logger Client is responsible for storing all the online tag details from the device. The logger Server is responsible for the retrieving the tag details from the file and shows reports. Both server as well as the client part of this Distributed Logger for SCADA system is written in VC++

CONTENTS

1. Introduction

1.1 Project overview.

This document tries to give a brief idea about the **LOGGER** - an object oriented module based on CORBA 2 developed to work with the **CORSCADA package** which is going to implement by the Control & Instrumentation Department **Electronic Research & Development Center (ER&DC)** for **Baba Atomic Research Center (BARC).**

CORSCADA package is developed for automating the ENERGY CONTROL SYSTEM of the BARC. The project aims at the development of a platform independent, distributed LOGGER system making use of CORBA technology

Logger is one of the mandatory modules of the CORSCADA package developed using CORBA technology. The logger is designed to work with the CORSCADA package, which is a distributed **General Purpose Control System** (GPCS), developed by ERDC. All the Action points of the LOGGER are implemented as methods or attributes of CORBA network objects (servants). Actions are executed by invoking methods on objects provided by the LOGGER. Both the Server as well as the Client part of the LOGGER is written in VC++. The main functionality of this CORBA based LOGGER is to store the real time information about the tags those exposed by the plant's active devices. LOGGER gets information from the Database Server As well as from Device Server and the real time/online values for the tag details from the devices are got through the Device Server. And this CORSCADA package uses TRANSPUTERS as the sensor for getting the real time data from the devices.

The different modules that will make the whole system are as follows

- **Database Server** - It is the backbone of CORSCADA package. Database Server acts as the repository of the whole system. All CORBA server objects in the system register their existence and keep their data in the Database server. Database server provides security to the whole system.
- **Logger** - Logger could be capable of logging the online data from Devices. It should get tag values of all the configured servers and store them. Logger provides historic data to clients.
- **HMI** (Human Machine interface) – Human Machine Interface enables the communication between man and machine. The tag information is presented to users (like operator, technician etc.) in a user-friendly manner. The HMI client shall collect On-line data from Devices and display them according to user configuration on the HMI client. The HMI configuration server shall be responsible for the generation & maintenance of HMI configurations.
- **Device** - is a mandatory module in CORSCADA system. It shall acquire data from H/W devices and provide On-Line Data to all clients as and when required and shall provide functionality to set Tag vales on the hardware. It shall provide change in tag values periodically to the Logger. The "Device configuration client" shall provide the facility to configure the H/W devices, tag properties etc.

1.2 Organization Profile.

ER&DCI is a national center of excellence in Electronics Research and Development is an autonomous scientific society under the department of electronics, Govt of India, doing application oriented Research and Development in high test areas in Electronics and Information technology. The center at Thiruvananthapuram was started in 1974n under Department of Science and Technology and was subsequently taken under the department of electronics via gazette notification dated May 6th, 1988.

Ever since its inception Electronics R&D center Thiruvananthapuram has focused its efforts on the development of products and system using state-of-the-art-technologies. The fruits of the R&D efforts are effectively contributing in diverse areas such as power generation, distribution and energy management power optimization, communication and energy management process optimization, communication, transportation and rural upliftment

The center aim at the development of cost effective ATM based hard\ware and Software solution for Internet based application. Another mile stone project developed by the power electronics group is the wind electric generator controller using DSP technology; the controller provides performance characteristics superior to its imported equivalent.

Objectives of ER&DC

- To undertake application oriented Research, Design and Development in Electronics So as to generate state-of -the art producible, marketable, and field maintainable products and systems.
- To initiate R&D programs in selected areas in a time-bound and mission-oriented manner in line with the other national technology development initiatives by the Govt. of India.
- To promote accelerated growth of electronic industries in the region by working in close co-operation with the State Electronics Development Corporations and other public and private sectors industries including small and medium sectors.

The main five branches of ER&DC located at kolkatta, Lucknow, Mohali, Pune and Thiruvananthapuram in our countries have its own specific research design.

Control & Instrumentation

Application-oriented research and development relating to control and instrumentation has been the trust area of the group. Plant wide networking has also been added to this thrust area. Network concept center has now been set up at the group mainly to acquire expertise in ATM based networking, develop ATM interfaces for RTUs and process control besides offering state of the art training. The group has expertise in system study, system engineering and expertise in system engineering and implementation of turnkey projects.

Some of the project completed/ ongoing are

- Energy Management System for Bhilai Steel Plant.
- Data Acquisition system for Kolaghat Thermal Power Station
- Sintering Plant Automation for Bhilai Steel Plant
- Electricity distribution Automation for Thiruvananthapuram city.

2.System study & Analysis

2.1 Existing system.

2.1.1 Current SCADA

SCADA stands for Supervisory Control And Data Acquisition. As the name indicates, it is not a full control system, but rather focuses on the supervisory level. As such, it is a purely software package that is positioned on top of hardware to which it is interfaced, in general via Programmable Logic Controllers (PLCs), or other commercial hardware modules.

Widely used in industry for Supervisory Control and Data Acquisition of industrial processes, SCADA systems have made substantial progress over the recent years in terms of functionality, scalability, performance and openness such that they are an alternative to in house development even for very demanding and complex control systems as those of physics experiments. All SCADA systems provide real time monitoring of the utility system status. Timely data on electrical faults and other system abnormalities increases system reliability, decreases service time, and increases safety. The data provided by a SCADA system is critical in the management of a utility.

An application program of the SCADA system communicate with devices in the field system using the PCU and the I/O cards and provides facilities for Man-Machine interaction.

A SCADA archives real time data for future use and analysis. These data are later used for off-line system analysis of losses, load flows, or faults. Accurate archived data may be used for billing purposes and load profiling. Reports of varying data provide important information on outage time for use in the analysis of crew responses. SCADA is the center of a utility management system.

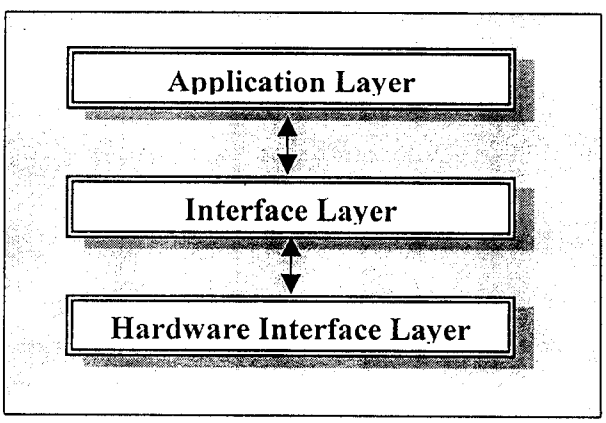
SCADA system originated in the utility industry, before the advent of mini and microcomputers. The common SCADA functions are Access Control, MMI, Trending, Alarm Handling, logging/Archiving, Report Generation and Automation.

2.1.2 Current System (THICS)

THICS console software runs on WINDOWS 95/98 machines. It has support for connecting a variety of devices like MODBUS based devices, ER&DC's earlier hardware based on the transputer (APACS) and also the entire set of intelligent RS 485 based I/O modules also designed by ER&DC. There is support for connecting a number of such PC based consoles if the transputer based APACS hardware is used.

THICS uses a three-layer approach for achieving its functionality. This is depicted in the figure shown below. The lowest layer that directly interacts with the hardware (RTU or PLC) is known as the **Hardware Interface Layer**. This is usually implemented as a device driver under THICS. This approach allows adding new device drivers without having to change any of the other THICS applications. The topmost layer is the **application layer**. This consists of a set of applications. Most important among them are the Human Machine Interface package (HMI), the Alarm module, the Report generation module and the Database builder application. These are all independent applications that allow the online and historical plant information to be viewed and manipulated in several ways.

The interface between the Hardware Interface Layer and the Application layer is known as the **Interface layer**. This is implemented currently as a DLL (Dynamic Link Library). This DLL termed THICSV2.DLL offers several functions for the exchange of information between the Application Layer and the Hardware Interface Layer. Most important among these functions are those used for reading from and writing to the THICS Real time database. This middleware layer also provides functions for querying the THICS database and obtaining configuration information of all the tags configured in THICS



2.1.3 Limitations of the current system

- The common SCADA functions like Access Control, HMI, Alarm Handling, logging/Archiving; Report Generation and Automation are functioning in the same machine.
- The current SCADA system (THICS developed by ERR&DC) is not a distributed type or platform independent

2.2 Proposed system

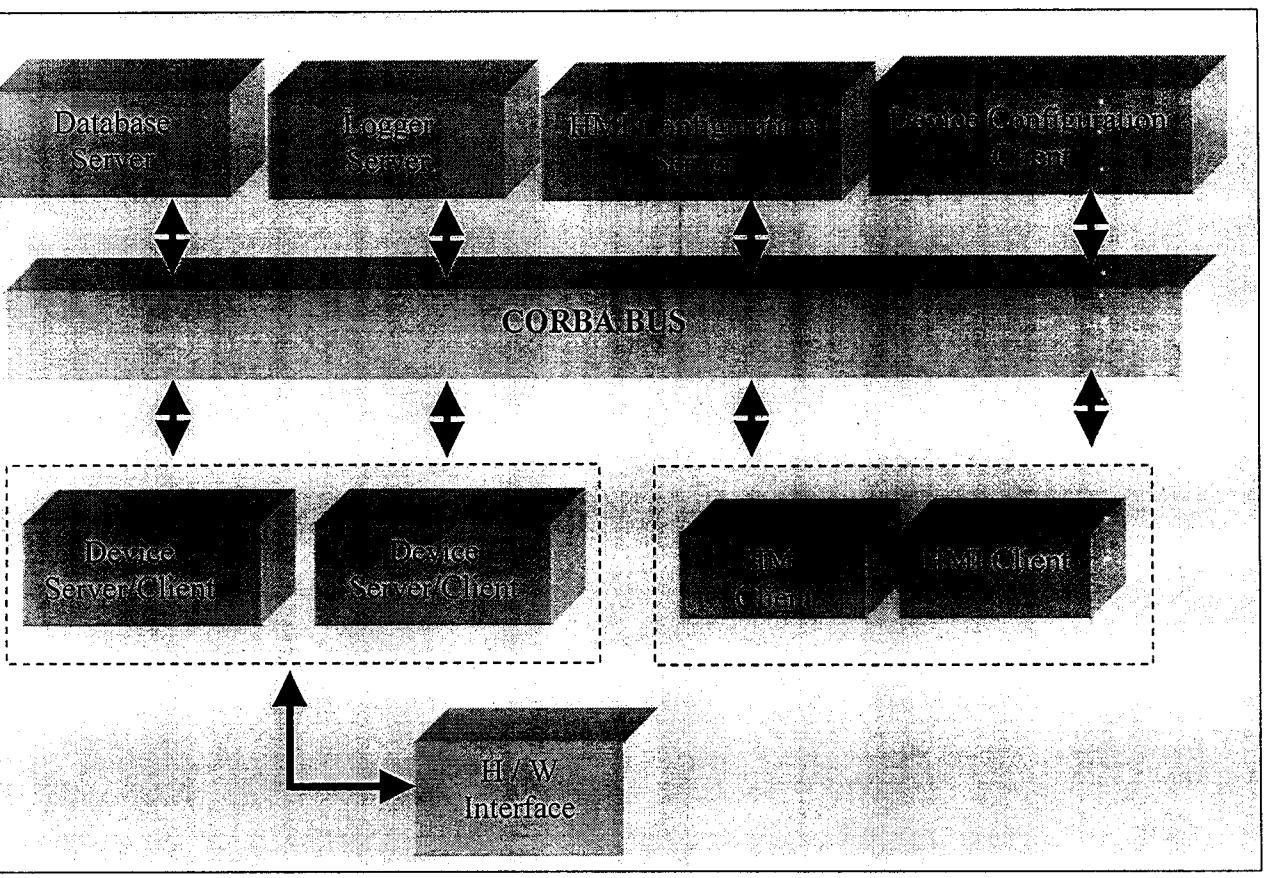
System Architecture for CORSCADA for Windows

The proposed CORSCADA system aims at the development of a platform independent, distributed system-making use of CORBA technology. This system contains SCADA objects like

- ⇒ **Database Server**
- ⇒ **HMI**
- ⇒ **Logger**
- ⇒ **Devices.**

These objects are integrated in a software bus according to the CORBA standard.

CORSCADA System Architecture



- A Database Server which will act as a repository to the whole system
- An HMI (Human Machine interface)
- A Device which will acquire data from H/W and provide Online data
- A Logger, which is capable of logging the online data from Devices.

DATABASE SERVER

The main functionality of the Database Server is that it acts as a repository for the whole system. When any device server comes up it will register with the database server. The database server monitors the status of all the registered servers in a particular interval and updates the status. The complete data for the whole system is concentrated in the machine where the database server is running. The Database server provides all active server lists, tag list of all active servers, available device configuration list, available device, Logger Tag list, Tag details.

Main functionalities are

- User Authorization and Security is provided
- Store device specific information of all device servers
- Expose device specific information to required clients

DEVICE MODULE

The Device module contains a **Device Configuration Client** used for Configuration of Device and a **Device Server** for the actual device activities.

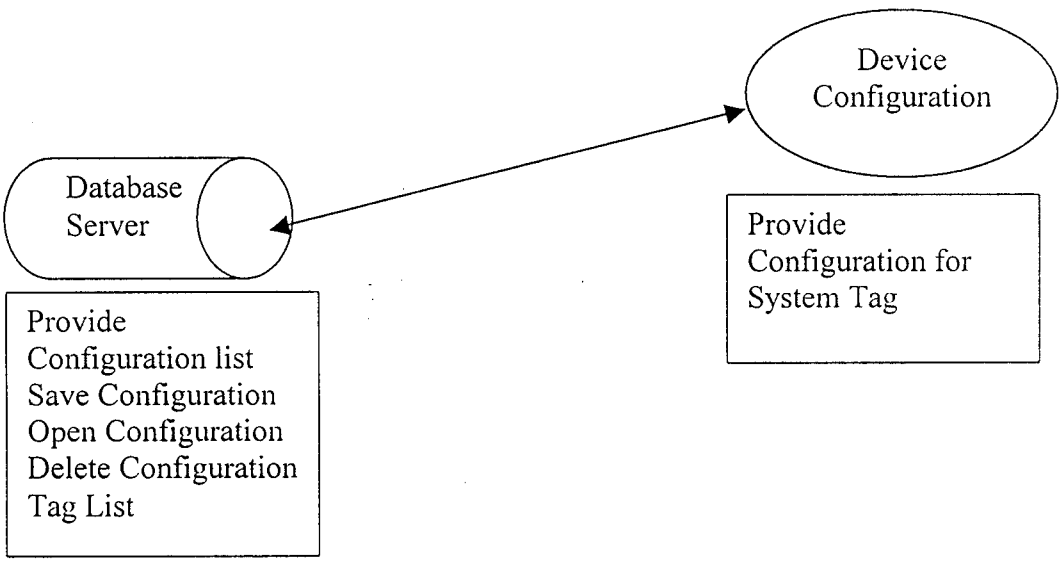
Device Configuration Client

Device Configuration Client Functionalities allows the user to Create, Open, Modify, and Delete and save Configuration through Database Server. **Device Configuration Client.** This GUI based program used for creating, modifying and deleting configurations of different type of device servers. Configurations are varying according to the type of device servers.

The entire configuration corresponding to a specific driver will keep its files in the directory \ **DATABASESERVER\DeviceName*** where the database server is running.

Configuration details

The configuration module creates different files for this driver. Which contains system details, tag details etc. All these files are keeping in the machine where the database server is running. Saving and open the files in database server achieved by the octet sequence method. In this case we have to keep these files locally for the operation of device server.



There are two types of tag types present in this CORSCADA package. The tag types are Analog, Digital. Each type of tags is keeping in separate files. And this file is also stored where the Database Server is running.

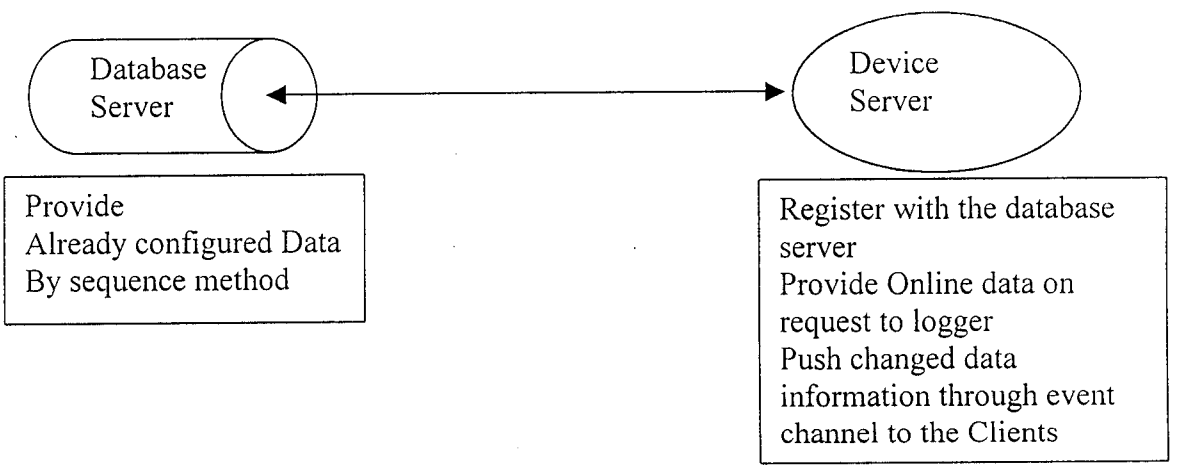
User interface has provided along with Device server for creating tags. We can separately configure the different type of tags. Modification and deletion of configurations are also provided. Select the already configured configuration from the list and download the information to the Device Server.

Device Server

The function of the **Device Server** is to acquire Data from H/W devices and provide Online Data to all clients. The main functionality of the **Device Server** is that it monitors for all the plant's active devices and it registers with the database server. **Device Server** - It should acquire Data from H/W devices like 80188 Micro Controller based RTUs, transputers and provide Online Data to all clients.

Main functionalities are

- Register with the database server as soon as it comes up
- Acquire Data from HW devices
- Provide Online data on request to logger.
- Push changed data information through event channel to the Clients



LOGGER MODULE

LOGGER should be capable of logging the On-line data from devices. It should be capable of supplying this logged data to requesting clients.

Logger functionalities includes

- Online Data logging from Devices
- Provide logged historic Data to applications like report, alarm, HMI etc.

When the Logger starts it *register with the* **Database Server** as well as the **Device Server** .The **Database server** monitors the status of all the registered servers, here the servers represents the active devices. The **Device Server** opens an Event Channel and push Tag Values on to the Channel. The logger binds to the channel and will retrieve values.

The LOGGER module has two major divisions.

- **LOGGER SERVER**
- **LOGGER CLIENT**

The LOGGER Server...

- Should provide reports to HMI Client the details about the logging information.

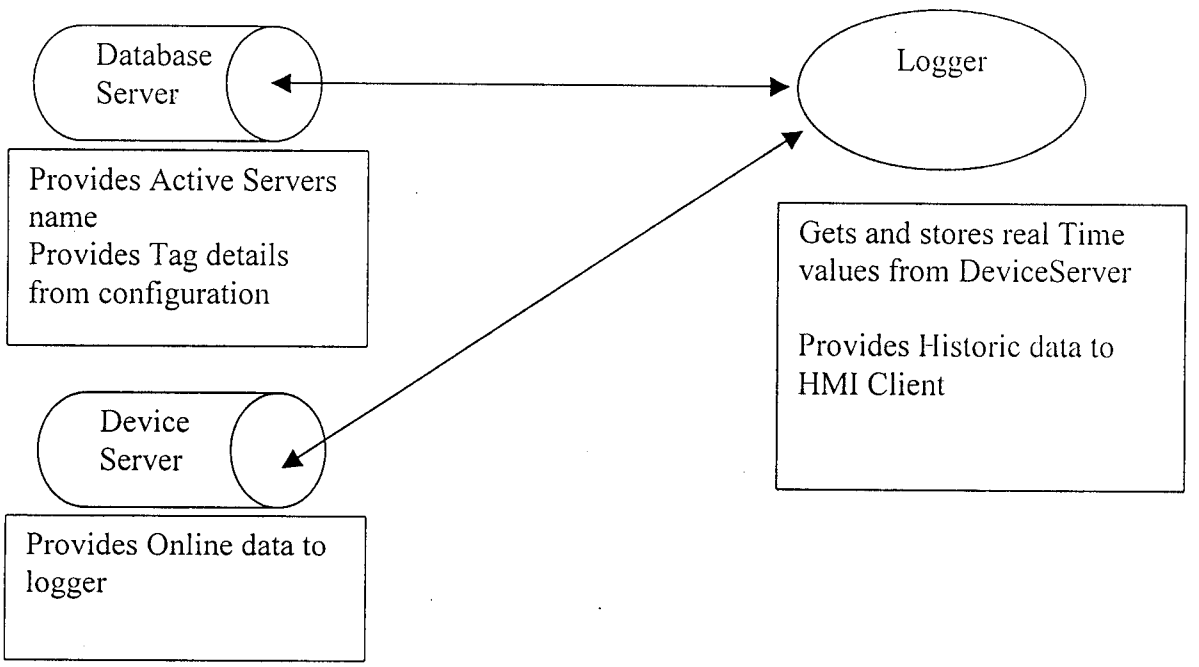
The reports include

- Providing tag details with in a specific time interval.
- Providing tag details for a specific date.
- Providing Log On Change records

*The **LOGGER CLIENT** functions includes*

- Getting information about the active devices and its configuration.
- Getting tag details of the configured tags from the Database Server.
- Extracting the Log Information and storing it for logging purpose.
- Acquisition of online data from the Device Server using multiple threads and store this into files.

Logging is typically performed on a cyclic basis i.e. once a certain time period is reached the logged data is monitored and processed according to the specified algorithm. Logging of data can be performed at configured time interval, or when a specific predefined event or change occurs. Logged data is transferred to the logged file once time is elapsed.



The logged information of each Tag from the specific devices is stored in appropriate files.

The file name convention used here is

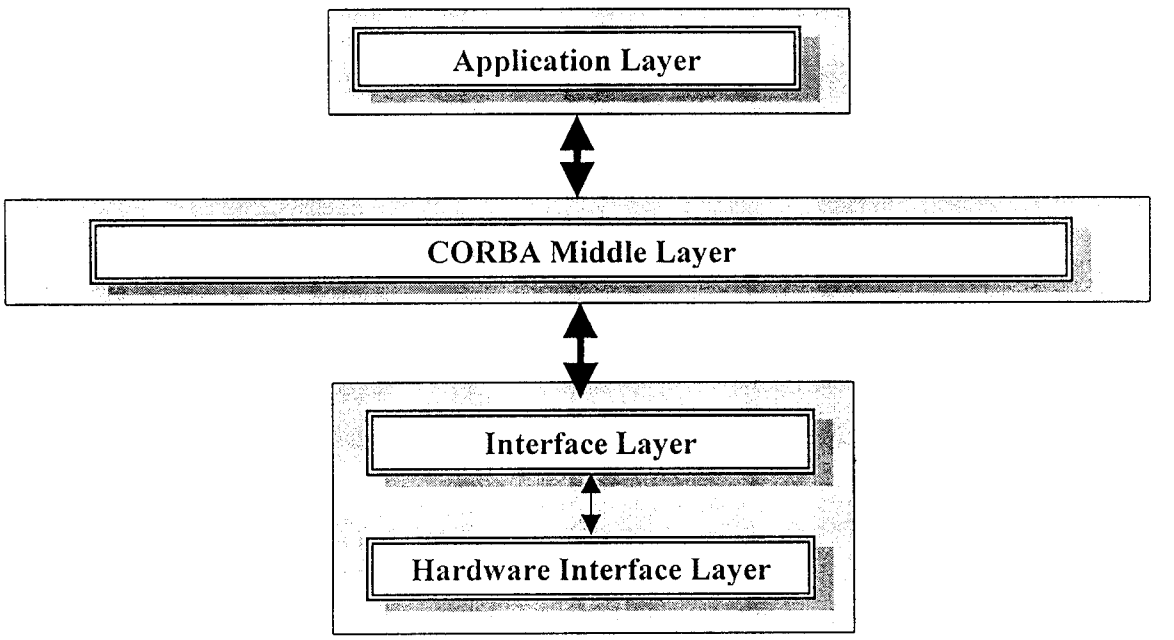
⇒ *ServerName_ConfigurationNameddmmyyy.chtd*

E.g. Thics_Config24022002.chtd

2.3 Major features of the system

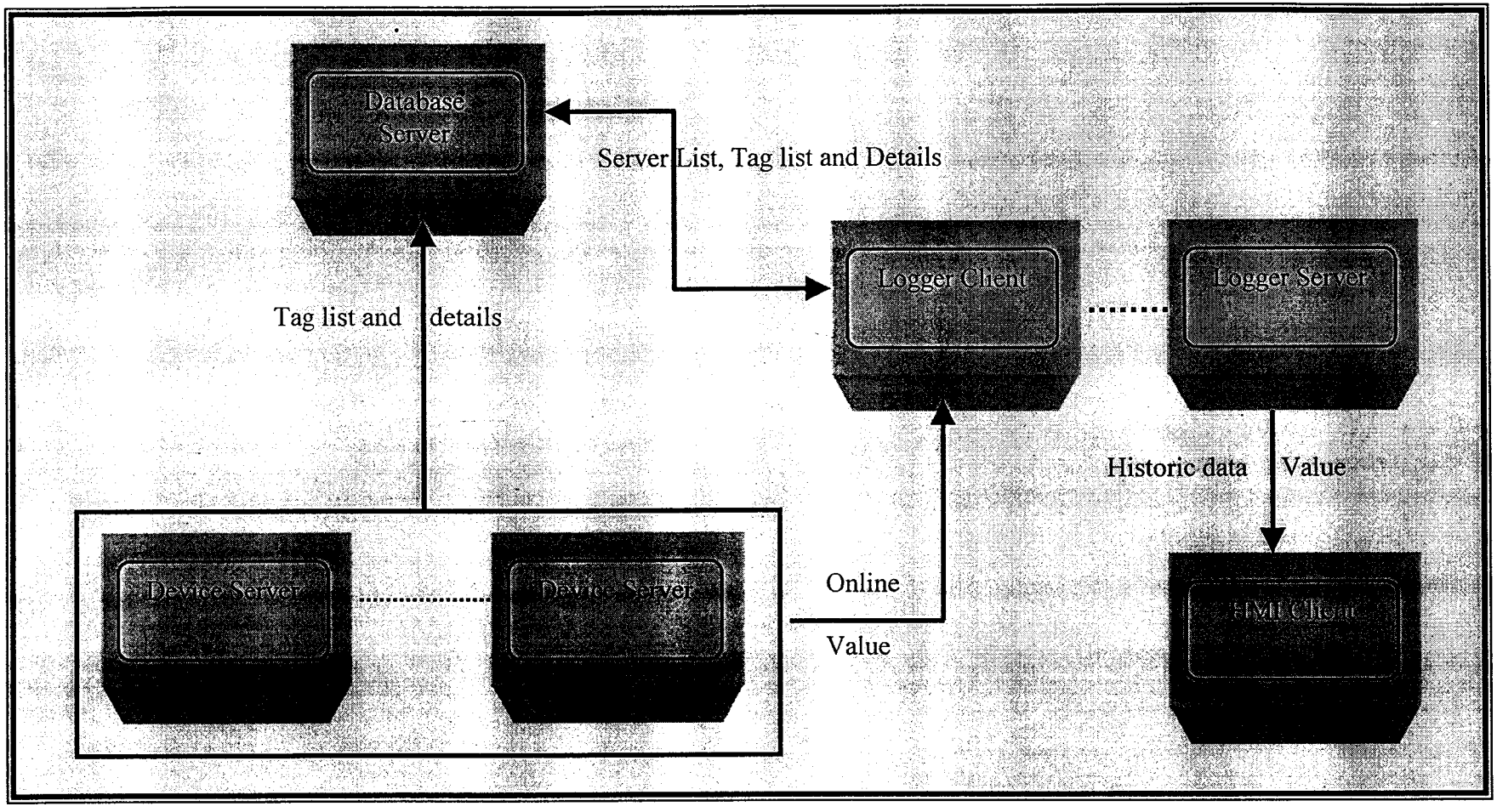
The main features of this distributed Logger for SCADA system are as follows

- This System completely follows the three-tier architecture by using CORBA as the middle layer.
- LOGGER is completely developed in a distributed heterogeneous environment.
- Since CORBA is used as the middle ware this provides faster Client access and thus reduce network load.
- Since the system is developed as distributed there is no need that every supporting servers should work in a single system.



In the case of distributed SCADA system the interface layers as well the hardware layer will reside in one system and application layer will be in another system and thus make it distributed.

Logger Architecture



3. Programming Environment

3.1 Hardware Configuration.

Networking

- Ethernet LAN should be available for the distributed operation of CORSCADA

Hardware Requirements

- Pentium 3 PC with minimum 1 GHz processor
- 20 GB
- 512 MB RAM
- Transputers

3.2 Description of software & tools used

Operating System - Windows 98

CORBA ORB - Visibroker 4.1 for C++

Application development tools Compiler - Microsoft Visual C++ 6.0

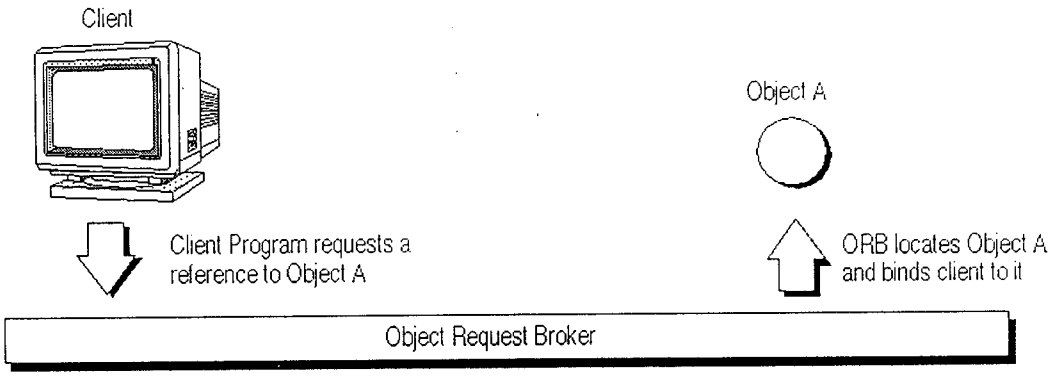
CORBA

The Common Object Request Broker Architecture (CORBA) allows distributed applications to interoperate (application-to-application communication), regardless of what language they are written in or where these applications reside.

The CORBA specification was adopted by the Object Management Group to address the complexity and high cost of developing distributed object applications. CORBA uses an object-oriented approach for creating software components that can be reused and shared between applications. Each object encapsulates the details of its inner workings and presents a well-defined interface, which reduces application complexity. The cost of developing applications is reduced, because once an object is implemented and tested, it can be used over and over again.

The Object Request Broker (ORB) in **figure 1** connects a client application with the objects it wants to use. The client program does not need to know whether the object implementation it is in communication with resides on the same computer or is located on a remote computer somewhere on the network. The client program only needs to know the object's name and understand how to use the object's interface. The ORB takes care of the details of locating the object, routing the request, and returning the result.

Figure 1 Client program acting on an object



VisiBroker

VisiBroker for C++ provides a complete CORBA 2.3 ORB runtime and supporting development environment for building, deploying, and managing distributed C++ applications that are open, flexible, and inter-operable. Objects built with VisiBroker for C++ are easily accessed by Web-based applications that communicate using OMG's Internet Inter-ORB Protocol (IIOP) standard for communication between distributed objects through the Internet or through local intranets. VisiBroker has a built-in implementation of IIOP that ensures high-performance and inter-operability.

IDL compilers

VisiBroker for C++ comes with the IDL compilers that make object development easier,

- `idl2cpp`: The `idl2cpp` compiler takes IDL files as input and produces the necessary client stubs and server skeletons (in C++).

Developing an application with VisiBroker

When you develop distributed applications with VisiBroker, you must first identify the objects required by the application. You will then usually follow these steps:

1. Write a specification for each object using the Interface Definition Language(IDL).

   IDL is the language that an implementer uses to specify the operations that an object will provide and how they should be invoked.

2. Use the IDL compiler to generate the client stub code and server POA servant Code.

   Using the `idl2cpp` compiler, we'll produce client-side stubs (which provide the interface to the objects' methods) and server-side classes (which provides classes for the implementation of the remote objects).

3. Write the client program code.

   To complete the implementation of the client program, initialize the ORB bind to the objects, invoke the methods on this object, and execute the methods.

4. Write the server object code.

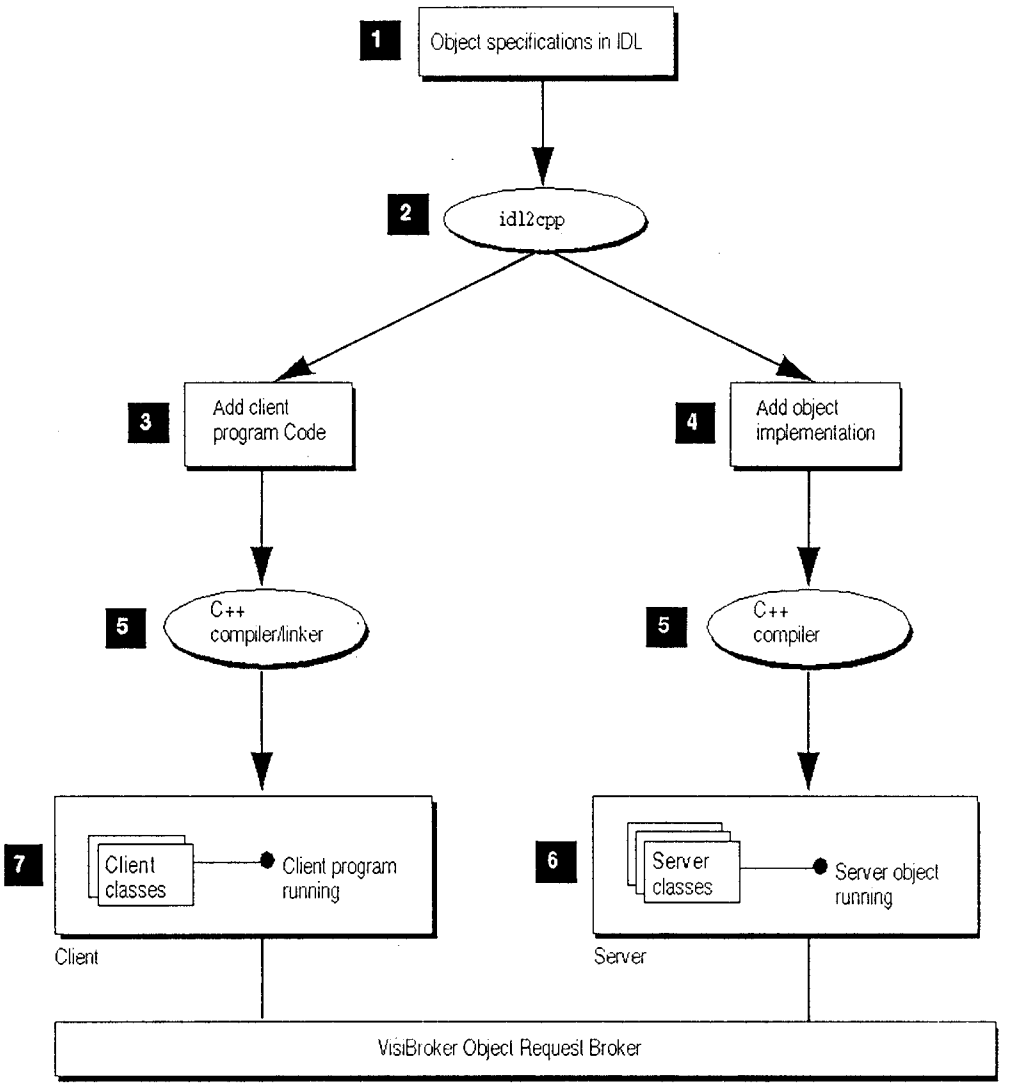
5. Compile the client and server code.

   To create the client program, compile and link the client program code with the client stub. To create the server, compile and link the server object code with the server skeleton.

6. Start the server.

7. Run the client program.

The Design Architecture of an application Using IDL



Generating client stubs and server servants

VisiBroker's idl2cpp compiler to generate C++ stub routines for the client program, and skeleton code for the object implementation uses the interface specification created in IDL. The client program for all member function invocations uses the stub routines. The skeleton code is used, along with code you write, to create the server that implements the objects. The code for the client program and server object, once completed, is used as input to your C++ compiler and linker to produce the client and server.

The `idl` file is compiled with the following command.

E.g. Prompt> idl2cpp ****.idl

Files produced by the idl compiler

The `idl2cpp` compiler generates four files from the `****.idl` file,

- `****_c.hh`--Contains the definitions for the classes.
- `****_c.cpp`--Contains internal stub routines used by the client.
- `****_s.hh`--Contains the definitions for the `POA` and `POA` servant classes.
- `****_s.cpp`--Contains the internal routines used by the server.

The `****_c.hh` and `****_c.cpp` files is used to build the client application. The `****_s.hh` and `****_s.cpp` files are for building the server object.

Implementing the client

The client program performs these steps:

1. Initializes the ORB.
2. Binds to an `interface` object.
3. Obtains the result from the methods by using the object reference returned by `bind()`.

Binding to the object

Before the client program can invoke the member function, it must first use the `bind()` member function to establish a connection to the server that implements the object. The implementation of the `bind()` member function is generated automatically by the `idl2cpp` compiler. The `bind()` member function requests the ORB to locate and establish a connection to the server. If the server is successfully located and a connection is established, a proxy object is created to represent the server's object. A pointer to the object is returned to the client program.

Implementing the server

The server program does the following:

1. Initialize the ORB
2. Create and setup the POA.
3. Activate the POA Manager
4. Activate objects
5. Wait for client requests

Initializing the ORB

The ORB provides a communication link between client requests and object implementations. Each application must initialize the ORB before communicating with it.

```
// Initialize the ORB.
CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);
```

Creating the POA

In basic terms, the POA (and its components) determine which *servant* should be invoked when a client request is received, and then invokes that servant. A servant is a programming object that provides the implementation of an *abstract object*. A servant is not a CORBA object. One POA (called the *root POA*) is supplied by each ORB. You can create additional POAs and configure them with different behaviors. You can also define the characteristics of the objects the POA controls.

The steps to setting up a POA with a servant include:

- Obtaining a reference to the root POA
- Defining the POA policies
- Creating a POA as a child of the root POA
- Creating a servant and activating it
- Activating the POA through its manager

Obtaining a reference to the root POA

All server applications must obtain a reference to the root POA to manage objects or to create new POAs.

Obtaining a reference to the root POA

```
// get a reference to the root POA
CORBA::Object_var obj = orb->resolve_initial_references ("RootPOA");
// narrow the object reference to a POA reference
PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);
```

Creating the child POA

The root POA has a predefined set of *policies* that cannot be changed. A policy is an object that controls the behavior of a POA and the objects the POA manages. If you need a different behavior, such as different lifespan policy, you'll need to create a new POA. POAs are created as children of existing POAs using `create_POA`. You can create as many POAs as you think are required. Child POAs do not inherit the policies of their parent POAs.

A child POA is created from the root POA and has a persistent lifespan policy. The POA Manager for the root POA is used to control the state of this child POA.

Creating the policies and the child POA

```
CORBA::PolicyList policies;
Policies.length(1);
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);
// Create myPOA with the right policies
PortableServer::POAManager_var rootManager = rootPOA->the_POAManager();
PortableServer::POA_var myPOA = rootPOA->create_POA( "bank_agent_poa",
rootManager,policies);
```

About Smart Agent

VisiBroker's Smart Agent (`osagent`) is a dynamic, distributed directory service that provides facilities used by both client programs and object implementations. A Smart Agent must be started on at least one host within your local network. When your client program invokes `bind()` on an object, the Smart Agent is automatically consulted. The Smart Agent locates the specified implementation so that a connection can be established between the client and the implementation. The communication with the Smart Agent is completely transparent to the client program.

If the PERSISTENT policy is set on the POA, and `activate_object_with_id` is used, the Smart Agent registers the object or implementation so that client programs can use it. When an object or implementation is deactivated, the Smart Agent removes it from the list of available objects. As with client programs, the communication with the Smart Agent is completely transparent to the object implementation.

Location Service

The Location Service is an extension to the CORBA specification that provides general-purpose facilities for locating object instances. The Location Service communicates directly with one Smart Agent, which maintains a *catalog,* which contains the list of the instances it knows about. When queried by the Location Service, a Smart Agent forwards the query to the other Smart Agents, and aggregates their replies in the result it returns to the Location Service.

The Location Service knows about all object instances that are registered on a POA with the `BY_INSTANCE` Policy. The server containing these objects may be started manually or automatically by the OAD.

Event Service

The Event Service package provides a facility that de-couples the communication between objects. It provides a *supplier-consumer* communication model that allows multiple *supplier objects* to send data asynchronously to multiple *consumer objects* through an event channel. The supplier-consumer communication model allows an object to communicate an important

change in state, such as a disk running out of free space, to any other objects that might be interested in such an event.

Supplier-Consumer communication model

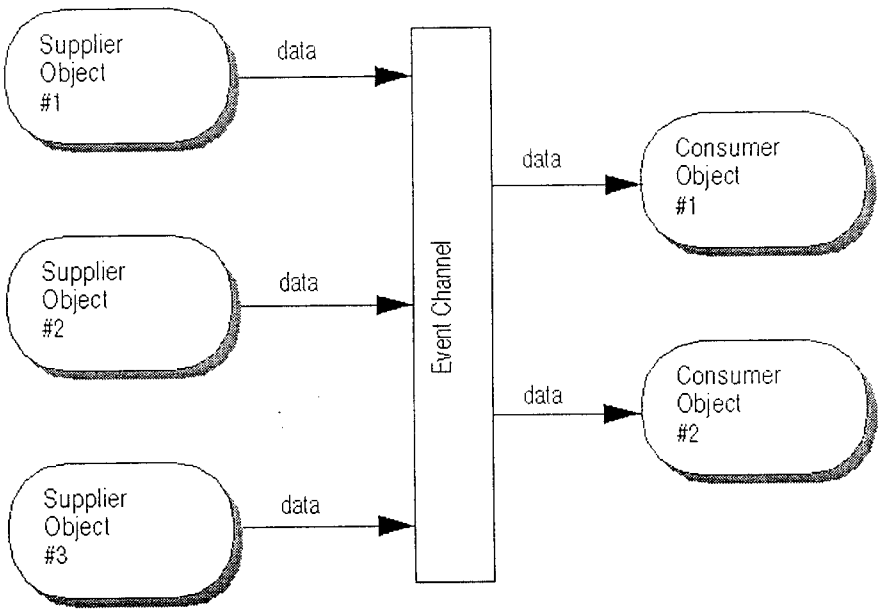
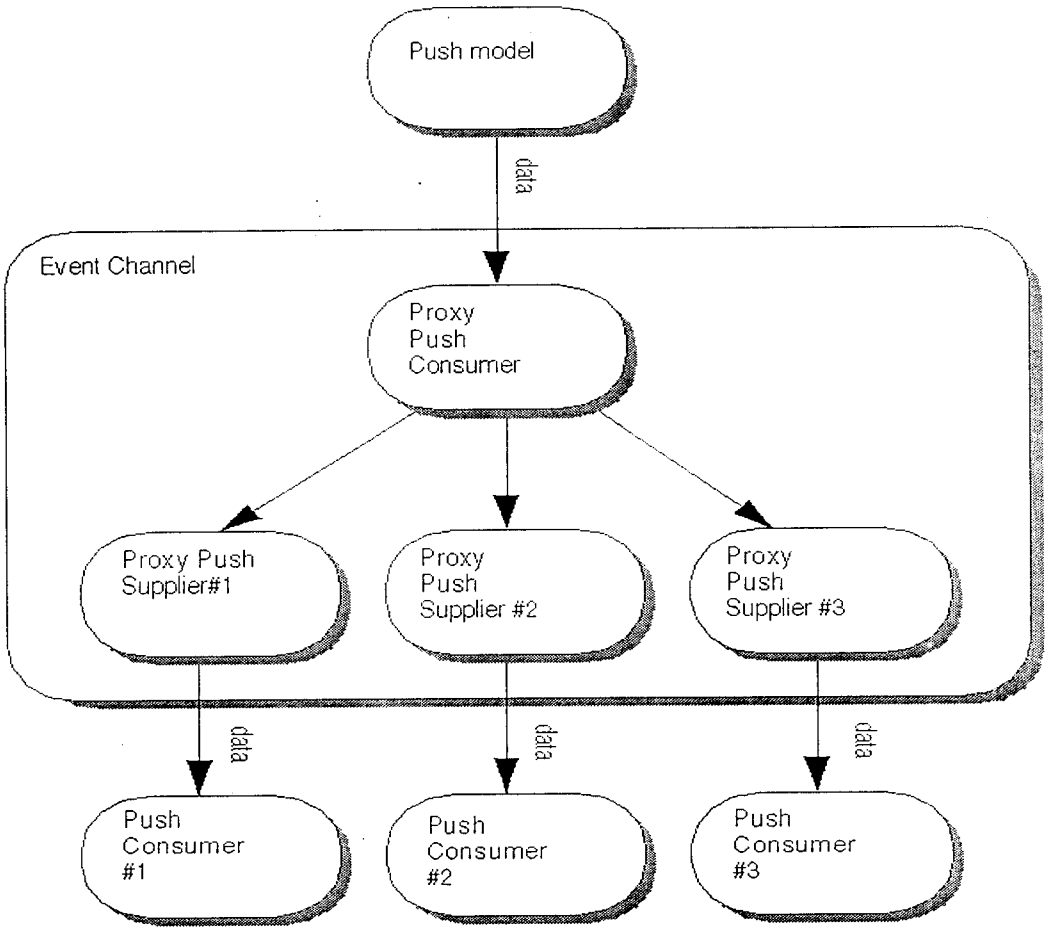


Figure shows three supplier objects communicating through an event channel with two consumer objects. The supplier objects handle the flow of data into the event channel, while the consumer objects handle the flow of data out of the event channel. If each of the three suppliers shown in figure sends one message every second, then each consumer will receive three messages every second and the event channel will forward a total of six messages per second. The event channel is both a consumer of events and a supplier of events. The data communicated between suppliers and consumers is represented by the `Any` class, allowing any CORBA type to be passed in a type safe manner. Supplier and consumer objects communicate through the event channel using standard CORBA requests.

Push model

The *push model* is the more common of the two communication models. An example use of the push model is a supplier that monitors available free space on a disk and notifies interested consumers when the disk is filling up. The push supplier sends data to its `ProxyPushConsumer` in response to events that it is monitoring. The push consumer spends most of its time in an event loop, waiting for data to arrive from the `ProxyPushSupplier`. The `EventChannel` facilitates the transfer of data from the `ProxyPushSupplier` to the `ProxyPushConsumer`.



The figure above shows a push supplier and its corresponding `ProxyPushConsumer` object. It also shows three push consumers and their respective `ProxyPushSupplier` objects.

Introduction to VC++

Visual c++ 6 provides an excellent development system for building great applications. The Visual C++ Developer Studio is the core of VC++ product. It is an integrated application that provided a complete set of programming tools. The developer studio includes a project manager for keeping track of your program source title and a set of resource such as menu, dialog box and icons. It also provides programming wizards (application wizard and class wizard), which helps us to generate the basic code for the program, defines c++ class handle window messages and perform other task. You can build and execute your program from within the developer studio, which automatically runs the optimizing compiler, the incremental linker and any other requested to build tools. You can also debug programs using integrated debugger and you can view and manage program symbols and c++ classes using the class view windows.

Some features that make visual C++ an excellent choice for developing solutions are:

- Code reusability
- Portability and cross-platform support

Win 32 API

In Windows application you need to package your application with GUI **elements** and **windows.** If are also serious about making money of your application, you need to be aware of the **guidelines** provided by Microsoft so that your application will produce similar effects when using standard elements or windows terminology. This is to provide consistent UI response for all windows application to avoid need for user to relearn about the same activities.). API is a collection of libraries (that included many classes) that made creating windows and user interface elements easy for the developer of the application. Imagine otherwise if you had to write hard code for creating windows and UI elements each time you used it in your program, leave alone the code necessary for the processing involved in the application

Windows SDK Library enabled us to make a program under windows environment. The system defined functions provided by the windows operating system is called the

windows API. API is a large collection of functions written in c. This act as an interface between an application and the windows environment. The Win32 API (Application Programming Interface) is the common programming interface for the Microsoft Windows operating system

Some advantages of Win32 API and a 32-bit compiler such as Visual C++ are

- True Multithreaded Applications
- 32 bit linear memory. Applications no longer have limits imposed by segmented memory. All memory pointes are based on the applications virtual address and are represented as a 32-bit integer.
- No memory model. This mean that there is no need for near and far pointers as all pointers can be thought as far.
- A well-designed Win32 application is generally faster.

The Win32 application-programming interface (API) defines the 32-bit members of the Windows operating system family from the programmer's point of view. Some members of the Windows family use the entire Win32 API, while others use subsets. With Visual C++, you can program for Windows using either C or C++ and the Win32 API, or using C++ and MFC.

Threads

A thread is basically a path of execution through a program. It is also the smallest unit of execution that Win32 schedules. A thread consists of a stack, the state of the CPU registers, and an entry in the execution list of the system scheduler. Each thread shares all of the process's resources.

A process consists of one or more threads and the code, data, and other resources of a program in memory. Typical program resources are open files, semaphores, and dynamically allocated memory. A program executes when the system scheduler gives one of its threads execution control. The scheduler determines which threads should run and when they should run. Threads of lower priority may have to wait while higher priority threads complete their

tasks. On multiprocessor machines, the scheduler can move individual threads to different processors to "balance" the CPU load.

Each thread in a process operates independently. Unless you make them visible to each other, the threads execute individually and are unaware of the other threads in a process. Threads sharing common resources, however, must coordinate their work by using semaphores or another method of interprocess communication.

The thread execution begins at the function specified by the *lpStartAddress* parameter. If this function returns, the **DWORD** return value is used to terminate the thread in an implicit call to the **ExitThread** function.

The **CreateThread** function may succeed even if *lpStartAddress* points to data, code, or is not accessible. If the start address is invalid when the thread runs, an exception occurs, and the thread terminates. Thread termination due to a invalid start address is handled as an error exit for the thread's process.

The thread object remains in the system until the thread has terminated and all handles to it have been closed through a call to **CloseHandle**.

Events

An "event" - a synchronization object that allows one thread to notify another that an event has occurred An Event allows one thread to notify another that an event has occurred. Events are useful when a thread needs to know when to perform its task

Events have two types: manual and automatic. A manual **Event** stays in the state set by SetEvent or ResetEvent until the other function is called. An automatic **Event** automatically returns to a nonsignaled (unavailable) state after at least one thread is released.

The handle returned by **CreateEvent** has EVENT_ALL_ACCESS access to the new event object and can be used in any function that requires a handle to an event object.
Any thread of the calling process can specify the event-object handle in a call to one of the wait functions. The single-object wait functions return when the state of the specified object is signaled. The multiple-object wait functions can be instructed to return either when any one or when all of the specified objects are signaled. When a wait function returns, the waiting thread is released to continue its execution.

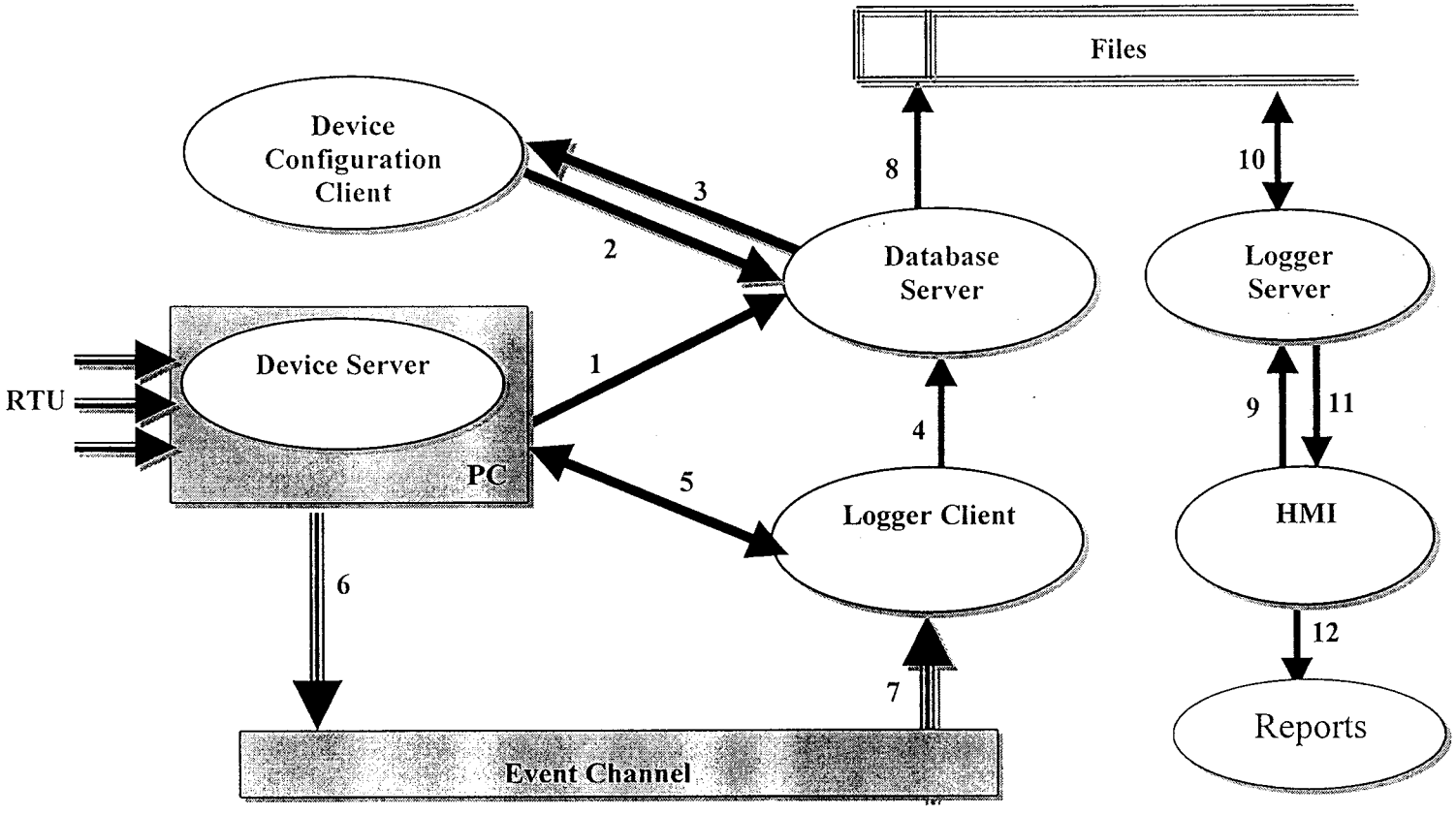
. Use the **SetEvent** function to set the state of an event object to signaled. Use the **ResetEvent** function to reset the state of an event object to nonsignaled.

When the state of a manual-reset event object is signaled, it remains signaled until it is explicitly reset to nonsignaled by the **ResetEvent** function. Any number of waiting threads, or threads that subsequently begin wait operations for the specified event object, can be released while the object's state is signaled.

When the state of an auto-reset event object is signaled, it remains signaled until a single waiting thread is released; the system then automatically resets the state to nonsignaled. If no threads are waiting, the event object's state remains signaled.

CloseHandle function to close the handle. The system closes the handle automatically when the process terminates. The event object is destroyed when its last handle has been closed.

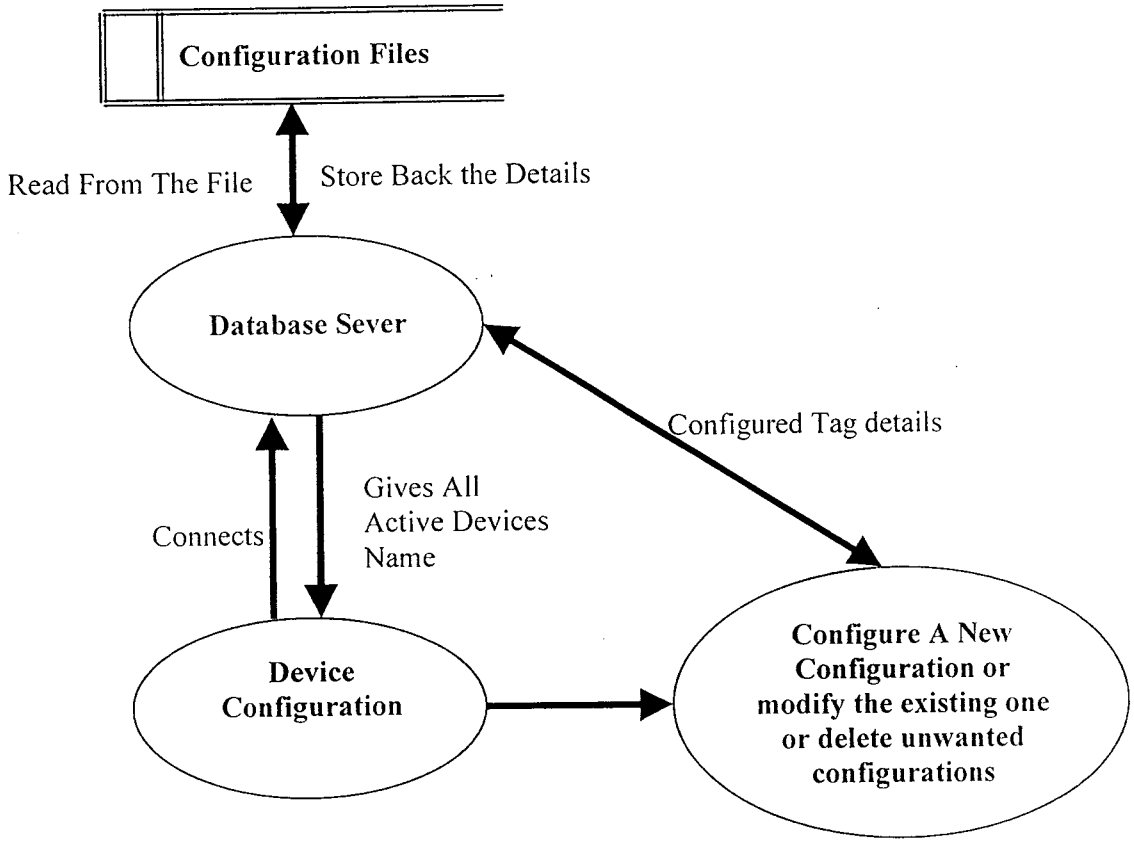
Over All Data Flow Diagram



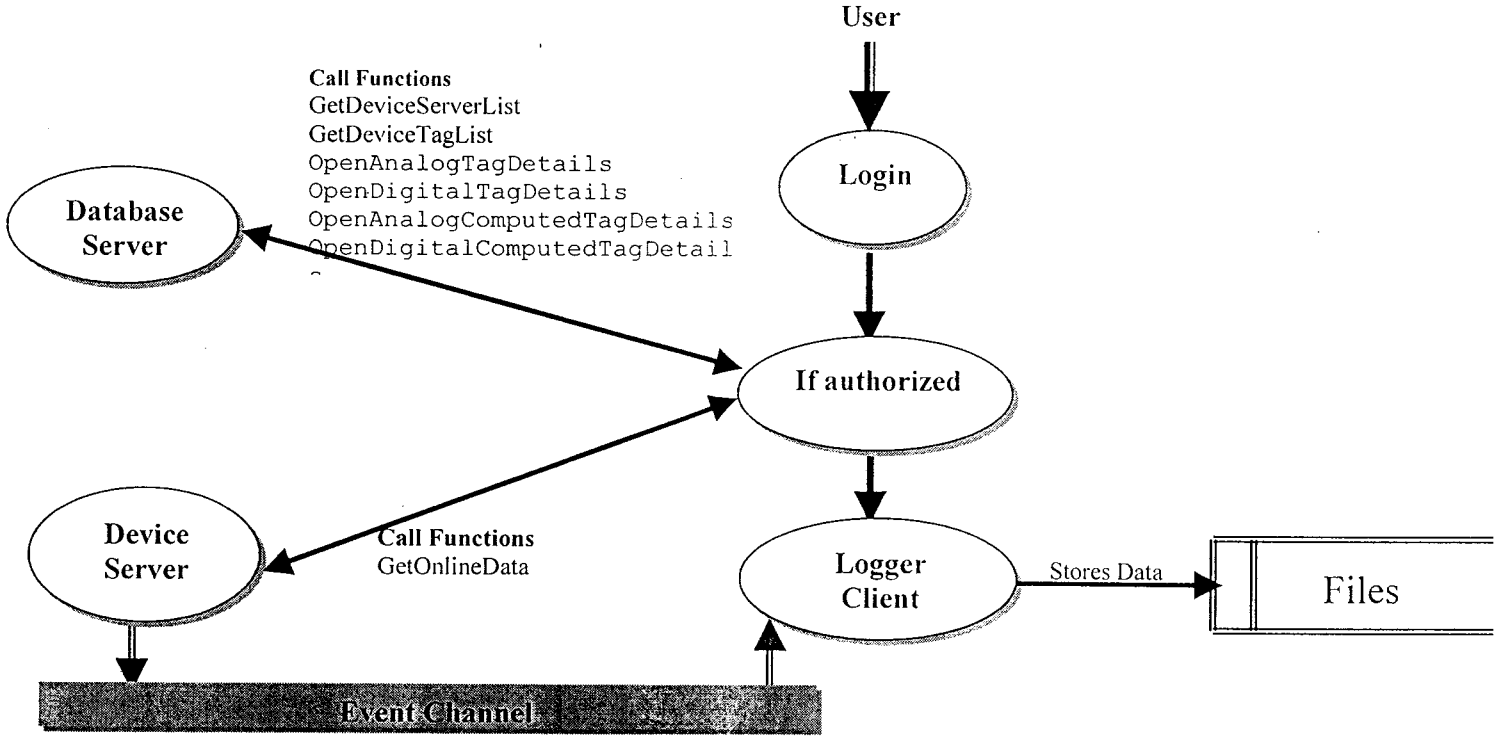
Flow Description

1. The device server registers all the devices that are activate to the database server.
2. The device configuration client gets the active devices and its configuration list from the database server
3. The altered configurations are stored back to the files where the database server is running.
4. Logger registers with the database server and get all the devices and the configuration list
5. Logger connects with device server and gets the online data when it first starts
6. When ever the devices pushes tag value, the device server pushes those value to the event channel.
7. The logger gets those values from the event channel ie binded with the device name.
8. These values are stored in the files where the database server is running.
9. As per the hmi client's request the looger server to retrieves value from the logger file
10. The logger server retrieves those values from the logger file
11. Those retrieved values are given back to the hmi client
12. the reports are given to the End Users

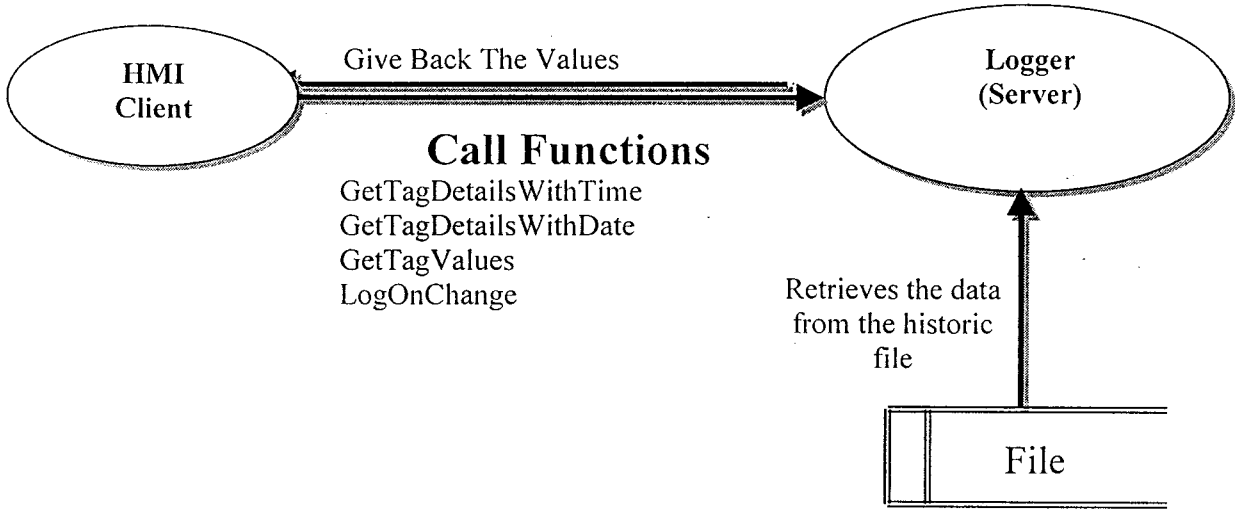
Data Flow Diagram For Device Configuration Client



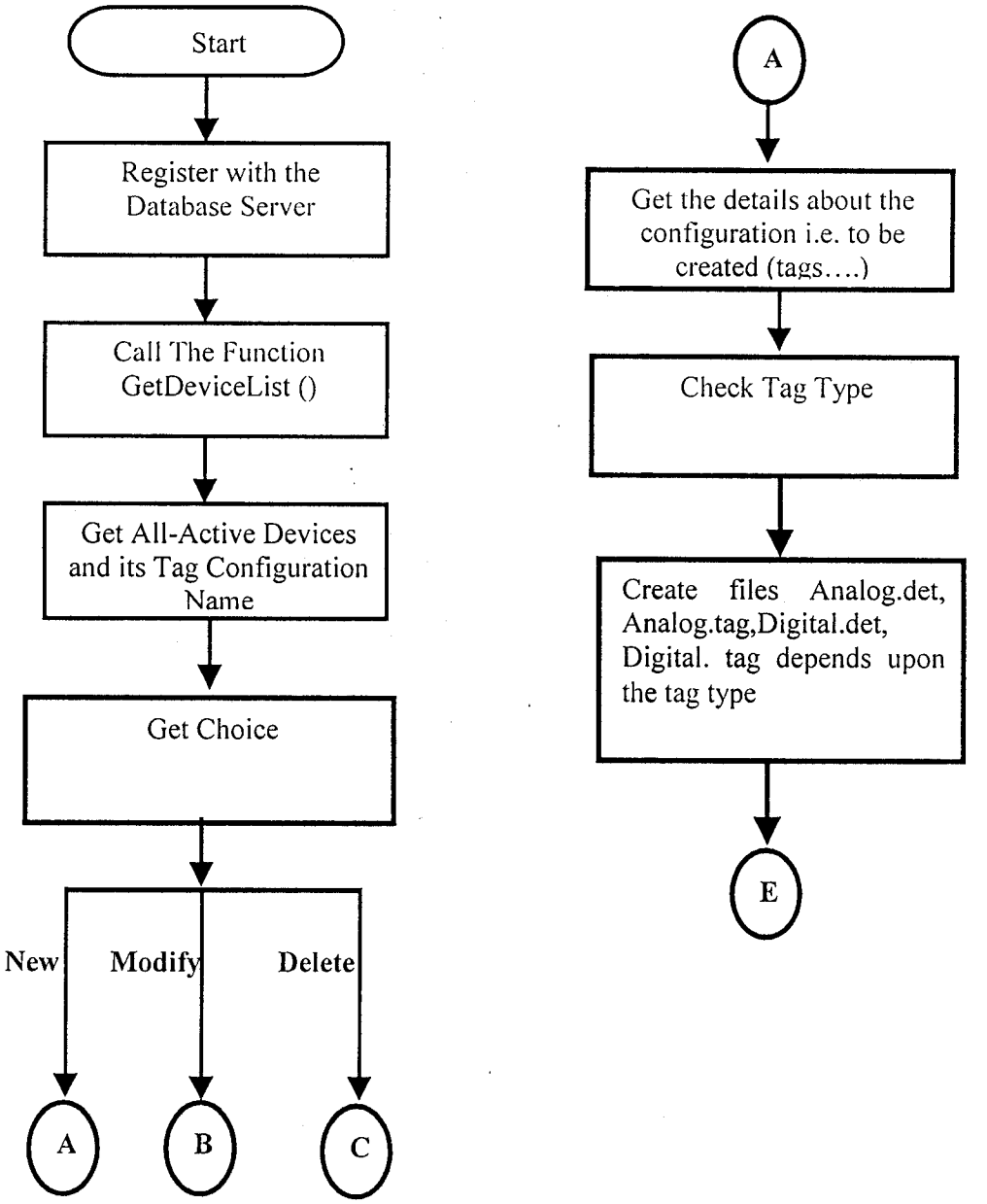
Data Flow Diagram For LOGGER Client

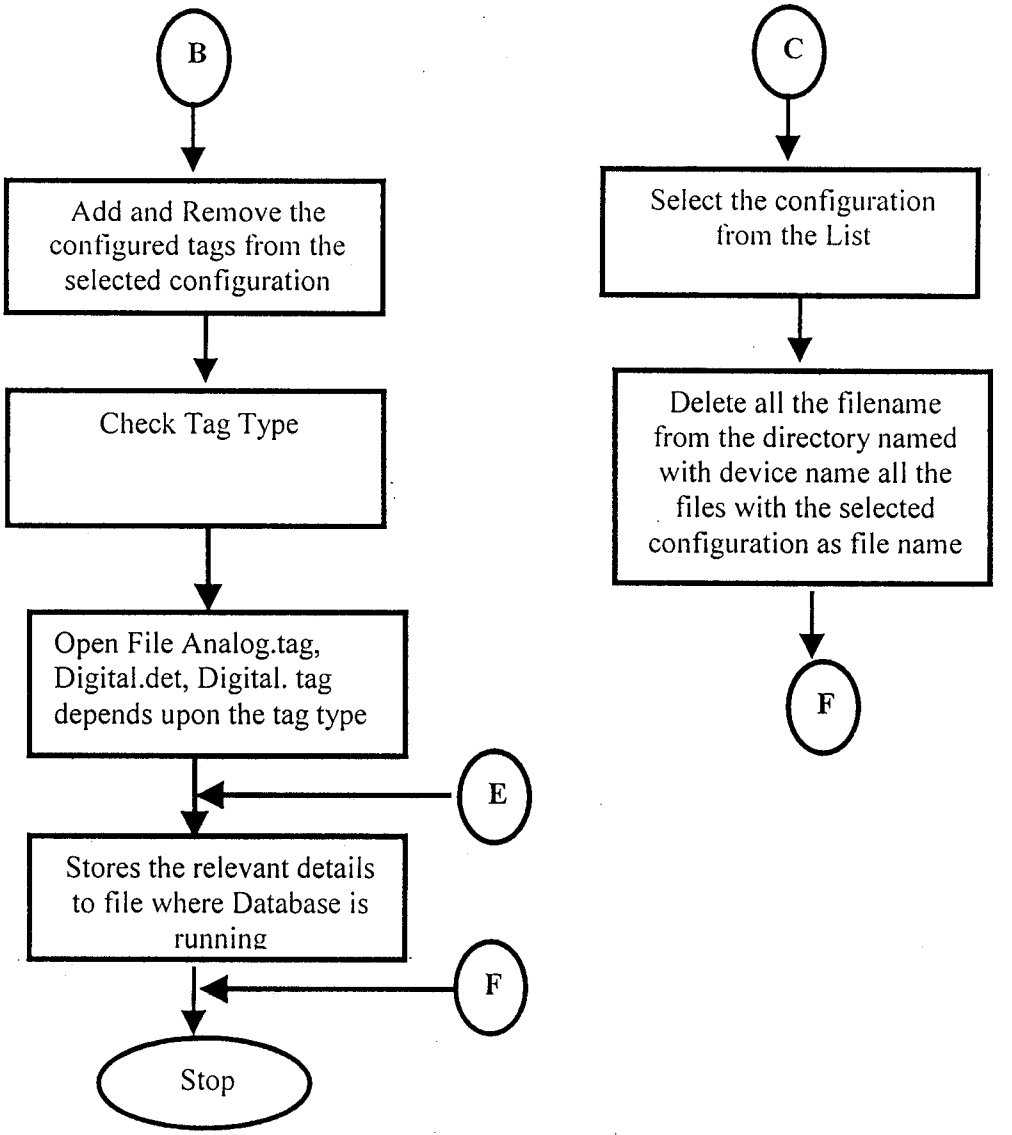


Data Flow Diagram For LOGGER Server

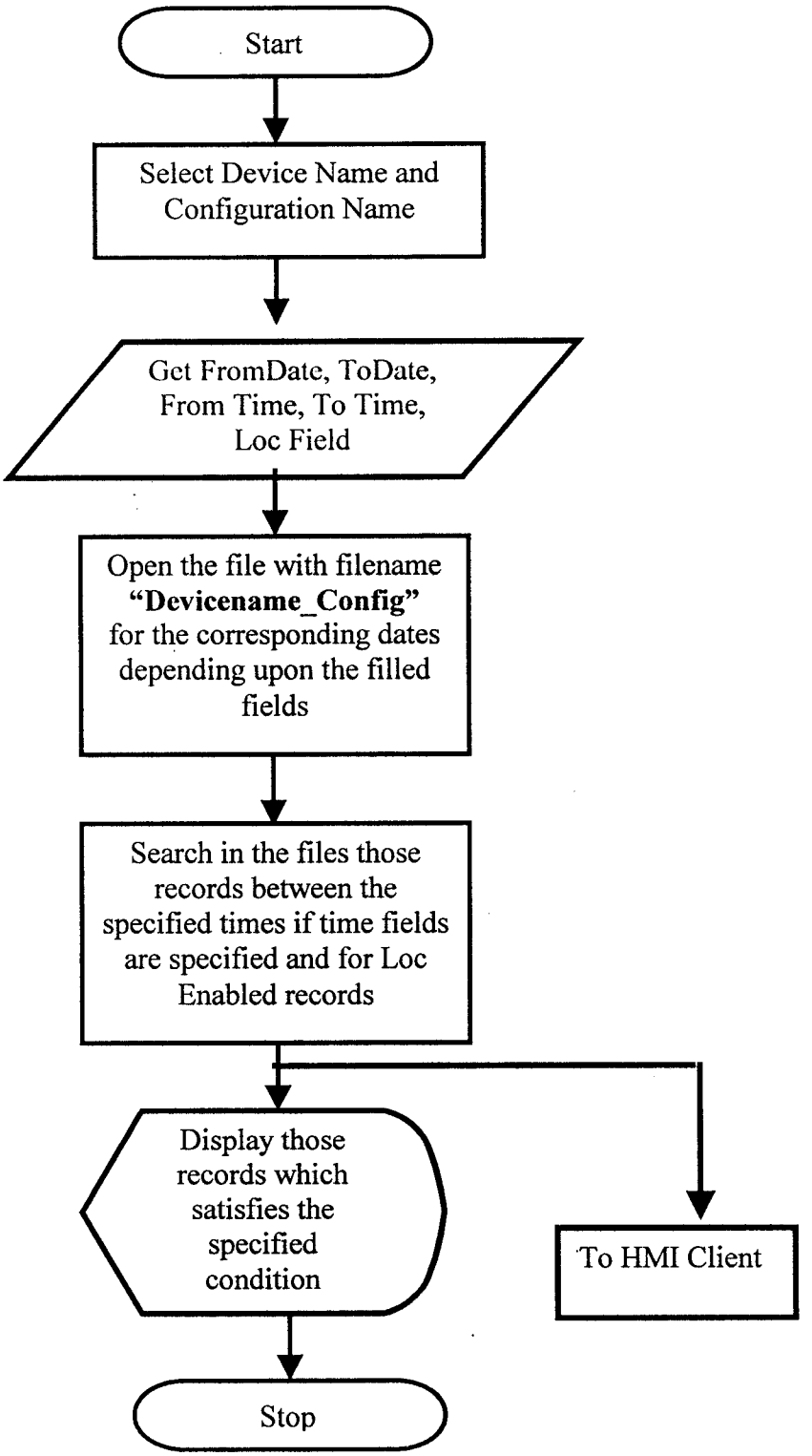


Flow Chart For Device Configuration

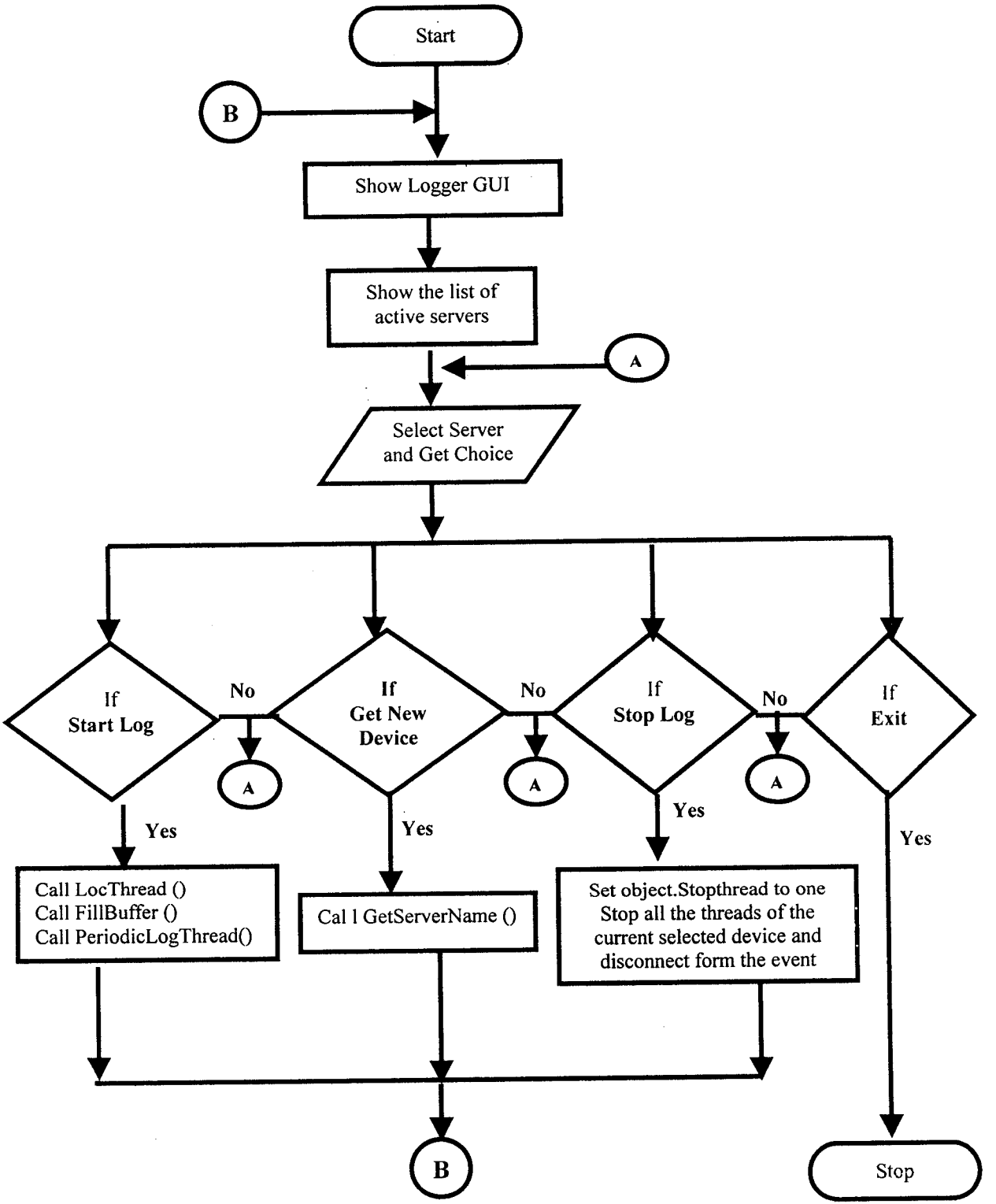



B
Add and Remove the configured tags from the selected configuration
Check Tag Type
Open File Analog.tag, Digital.det, Digital. tag depends upon the tag type
E
Stores the relevant details to file where Database is running
F
Stop
C
Select the configuration from the List
Delete all the filename from the directory named with device name all the files with the selected configuration as file name
F

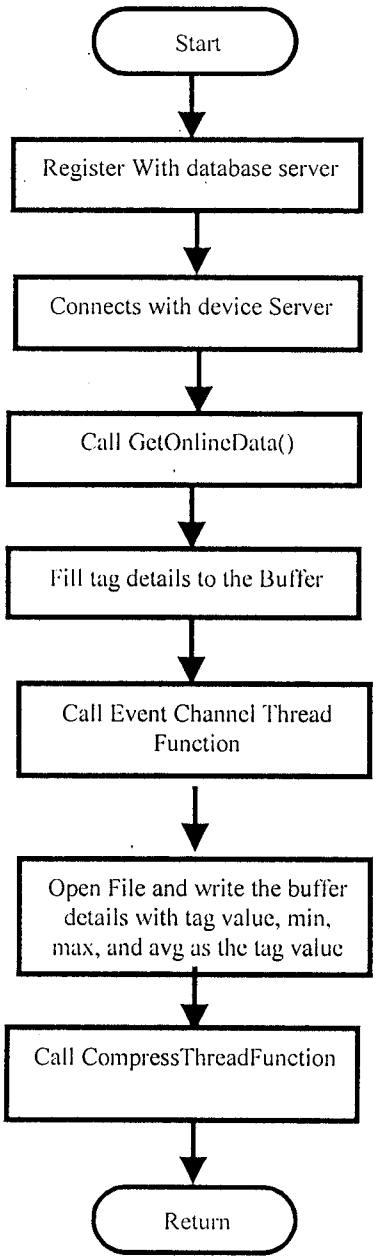
Flow Chart For Logger Server



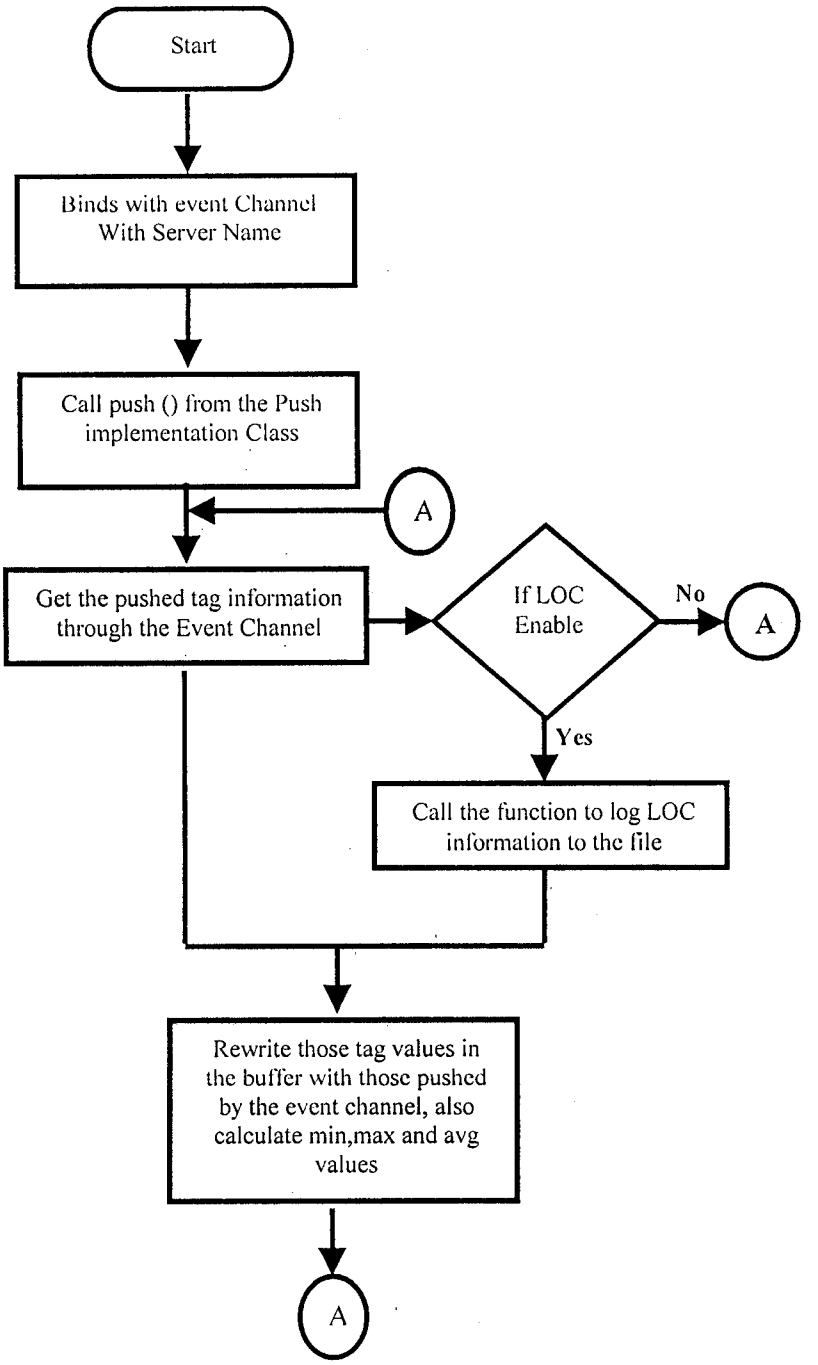
Overall Program Flowchart for Logger Client



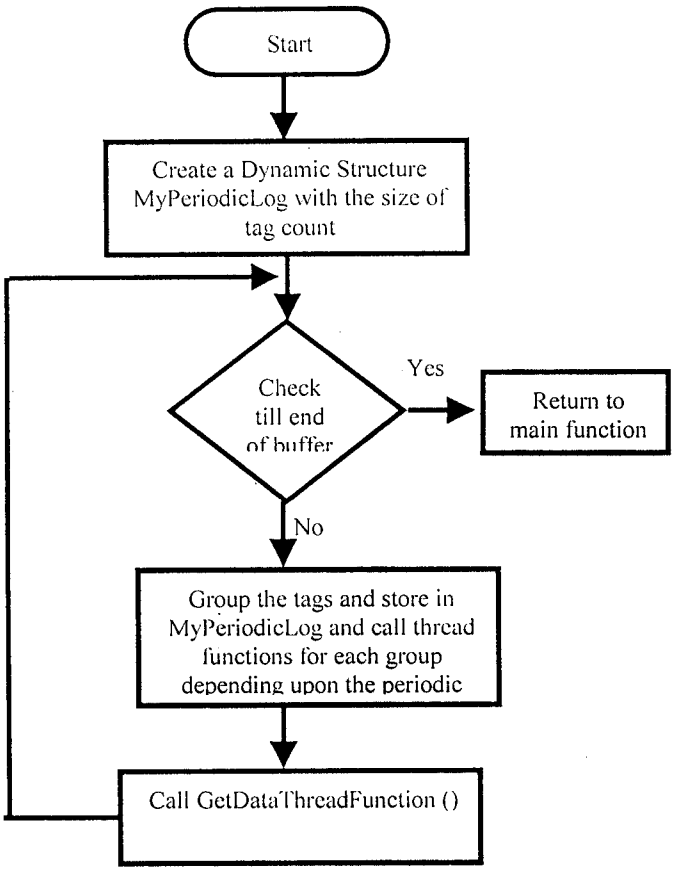
Flow chart For Fill Buffer



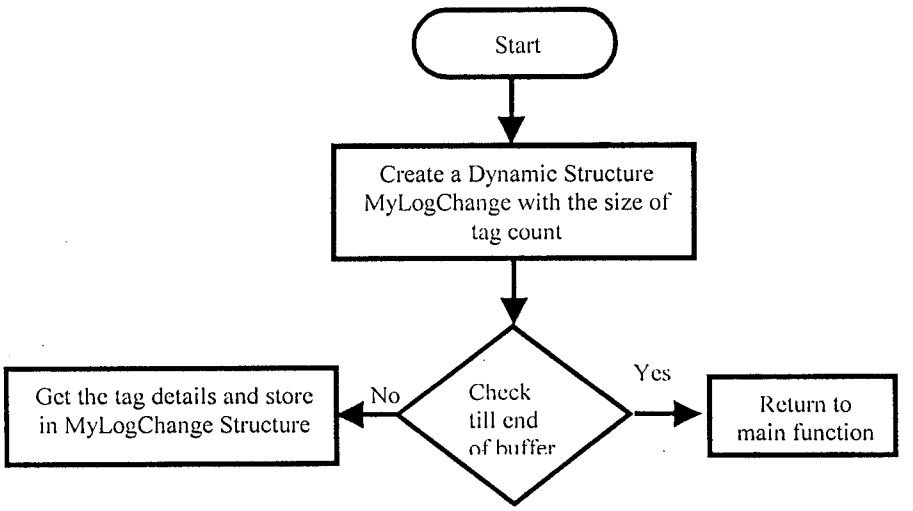
Flow chart For Event Channel Thread Function



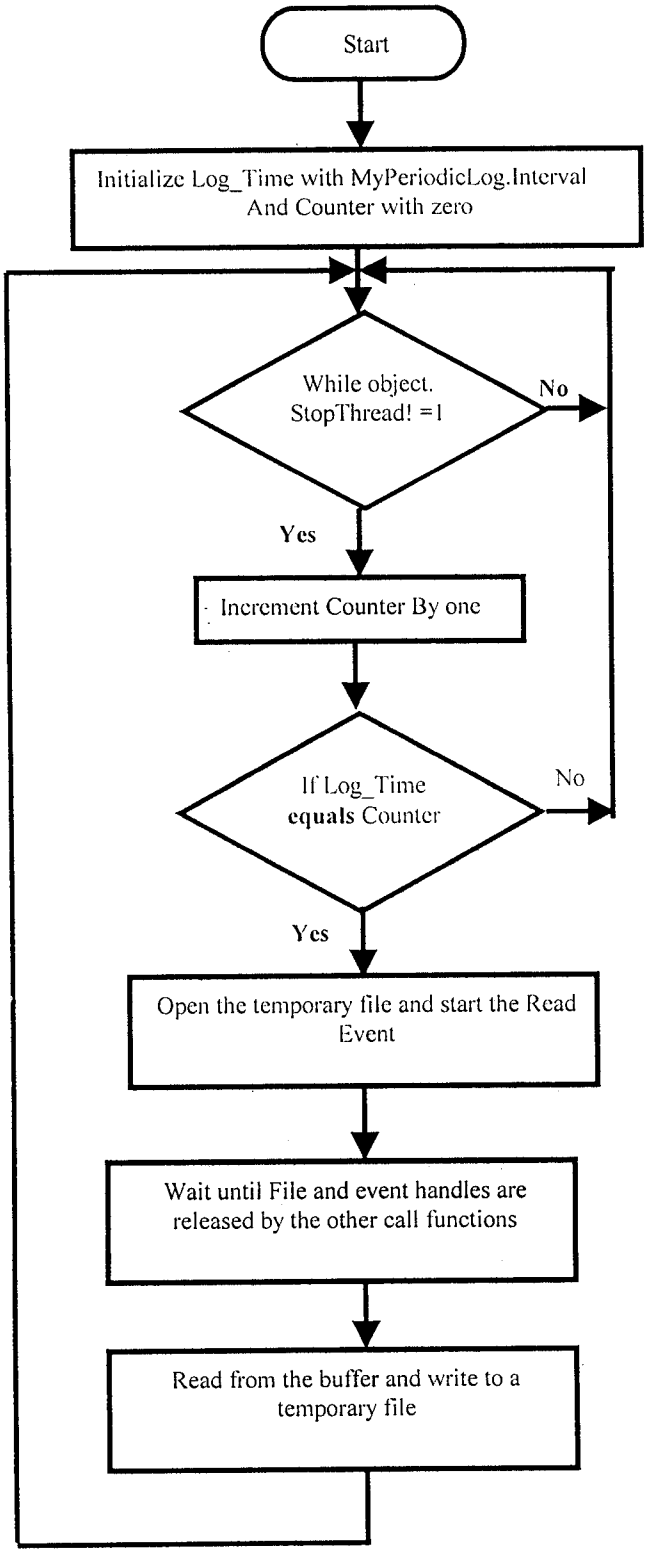
Flow chart For Periodic Log Enable Thread Function



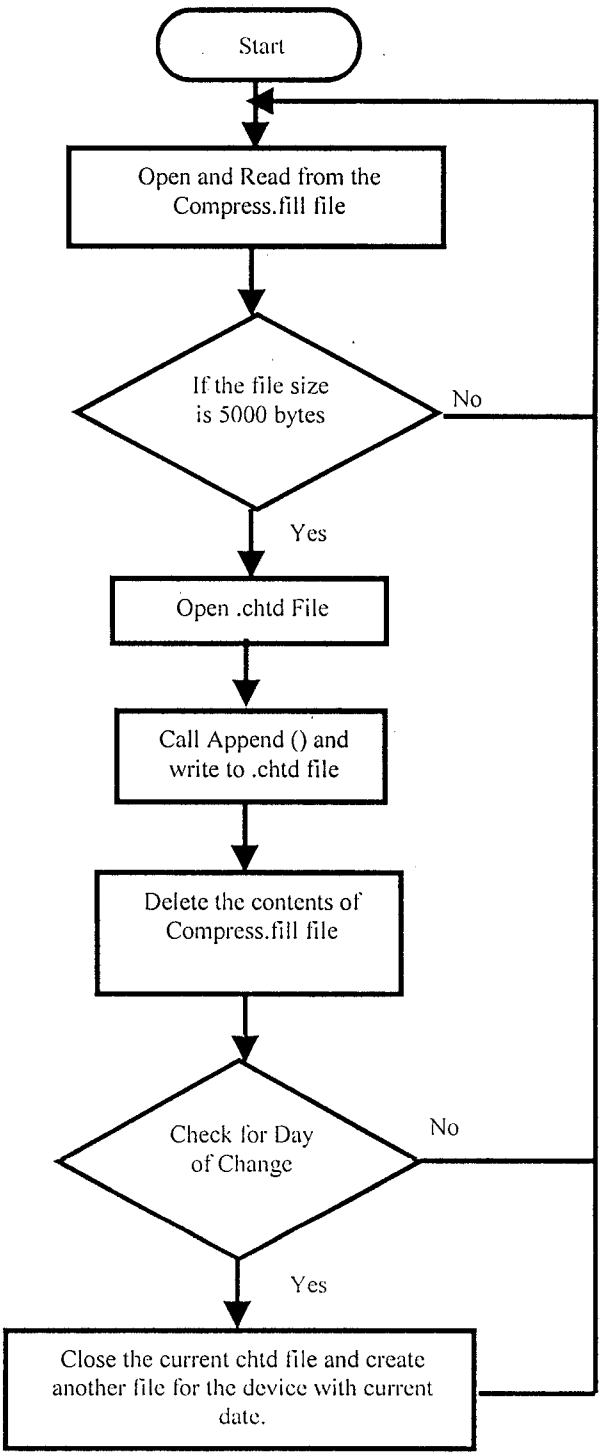
Flow chart For LOC Thread Function



Flow chart For GetDataThreadFunction



Flow Chart For CompressThreadFunction



4.3 File Design

When considering a system there should be given provision to store the relevant details of the system in an appropriate manner in such a way that information can be retrieved and processed in an easier and efficient way.

Coming to CORSCADA all the information are stored in Flat Files.

Files used with User Authorization

All the details about the CORSCADA users are needed to be stored. The file used for storing the User list is

- User.dat

Directory structure

This file is kept in the system where the Database Server is running. All the Configuration details are stored in the **C:\\> prompt** in the Directory with the name **CORSCADA.**

- *C :\CORSCADA\User.dat*

The file structure used for this file is

```
struct USER
{
        char UserName[USER_PWD_LENGTH];
        char UserDesc[DEVICE_INFO_LENGTH];
        char Password[USER_PWD_LENGTH];
        octet Privilege ;
}UserInfo;
```

Files used with Device Configuration Client

The devices exposes mainly 2 types of tag types namely Analog, Digital. So information regarding each type has to be saved differently.

The Device Configuration module uses 4 different types of files for specific uses. The files used are

- **Analog.tag**
- **Analog.det**
- **Digital.tag**
- **Digital.det**

Directory structure

These files are stored in the system where the Database Server is running. All the Configuration details are stored in the **C:\\> prompt** in the Directory with the name **Database Server.** Here in this directory there will be subdirectories for each and every device. And all the files are stored in the directory.

- *c :\DatabaseServer\DeviceName*\ConfigName*\Analog.tag*
- *c :\DatabaseServer \ DeviceName*\ ConfigName*\Analog.det*
- *c :\DatabaseServer \ DeviceName*\ ConfigName*\ Digital.tag*
- *c :\DatabaseServer \ DeviceName*\ ConfigName*\Digital.det*

Analog.tag

This file is used to store all the analog tag names that are configured.

Char tagname[255];

Analog.det

This file stores all the details about the Analog tags for the particular devices. This file is created when any of the Analog tag is selected for the configuration.

Digital.tag

This file is used to store all the digital tag names that are configured.

Char tagname [255];

Digital.det

This file stores all the details about the Digital tags for the particular devices. This file is created when any of the Digital tag is selected for the configuration

The file structure used to store data in both Analg.tag det and Digital.det are as follows

```
struct TAGCOMMONINFO
        {
                char            Name[DEVICE_TAGNAME_LENGTH];
                char            Description[DEVICE_TAGDESC_LENGTH];
                char            IOName[DEVICE_TAGNAME_LENGTH];
                octet           AlmEnable;
                octet           PerLogEnabled;
                long            PerLogInterval;
                octet           LOCEnabled;
                long            LOCChange;
}CommonTagInfo;

typedef struct DIGITAL
        {
                TAGCOMMONINFO           CommonDigTagData;
                char    DigitalMsg[DEVICE_UNIT_LENGTH];
        } DigitalTag;

typedef struct ANALOG
        {
                TAGCOMMONINFO CommonAnaTagData;
                float                   AnalogMaxVal;
                float                   AnalogMinVal;

                float                   AnalogAlmLoLo;
                float                   AnalogAlmLo;
                float                   AnalogAlmHi;
                float                   AnalogAlmHiHi;
                octet                   AnalogEnableLL;
                octet                   AnalogEnableL;
                octet                   AnalogEnableH;
                octet                   AnalogEnableHH;

                float                   DeadBand;
                octet                   DeadBandEnabled;
        } AnalogTag;
```

Files used with Logger

The logger uses flat files to store all the values exposed by the devices. The tags exposed by the devices should be stored according to its periodic interval and depends on the Log On Change field. Usually the devices are configured with 100's of tags. And among these ones tags are configured for 60, 120...seconds. So there are chances that logger file size can come up to several MB even it is made work for a few minutes. So the values should be compressed and stored in the logger files.

Logger files uses 3 types of files

- **Compress.fill**
- **ServerName_ConfigurationNameddmmyyyy.htd**
- **ServerName_ConfigurationNameddmmyyyy.chtd**

Directory structure

These files are stored in the system where the Database Server is running is running. All the logged data details are stored in the Directory with the name **Logger** in the **C:\\> prompt.**

Whenever a device starts logging the logger client checks for the directory with the device name in the system where database server is running. If not created a new subdirectory will be created for that particular device under the directory Logger. And all the Logger files are stored in the directory.

- *c :\Logger\DeviceName*\ DeviceName_ConfigurationNameddmmyyyy.chtd*

Compress.fill

This is the temporary file where the online tag details are stored. This file is used to log the online values form the device and will be opened during logging. When the file size reaches a specified size the data inside the files are compressed and write to the compressed file. And all the data in the fill file is deleted. When the logger stops logging of any particular devices the Compress.fill file is automatically deleted.

The file structure used to store data in the Compress.fill file is:

```
struct TagDataDetails
{

        char  TagName[30];
        float Value;
        float TagMin;
        float TagMax;
        float Average;
        int hour;
        int min;
        int sec;
        int LocEnabled;

}TagDataDetailsSeq
```

DeviceName_ConfigurationNameddmmyyyy.htd

The **historic trend data** (htd) is a temporary file used for uncompressing the details from the compressed file and stores it. The Logger Server as well as the HMI Client mainly uses this file. When the User request for a file it searches for the compressed file and decompressed the file using the specified algorithm and decompressed values are stored in the .htd file. Once the file is closed this historic trend files are deleted

The file structure used to store data in the Historic trend data file is:

```
struct TagDataDetails
{
        char  TagName[30];
        float Value;
        float TagMin;
        float TagMax;
        float Average;
        int hour;
        int min;
        int sec;
        int LocEnabled;
}TagDataDetailsSeq
```

DeviceName_ConfigurationNameddmmyyyy.chtd

The Compressed historic data file is the main logger file and the contents are in a compressed format.

4.4 IDL Design

The Interface Definition Language used in CORBA to describe the interface of objects. IDL is the language that an implementer uses to specify the operations that an object will provide and how they should be invoked. IDL defines the modules, interfaces and operations for applications.

The IDL can be mapped to a variety of programming languages. we use the `idl2cpp` compiler to generate stub routines and servant code from the IDL specification. The stub routines are used by your client program to invoke operations on an object. You use the servant code, along with code you write, to create a server that implements the object. The code for the client and object, once completed, is used as input to your C++ compiler to produce a client application and an object server.

IDL for Database Server (DBSERVER.idl)

This is the file defining the CORBA interface for the database server object in the CORSCADA system.

IDL Definition

```
#define DEVICE_TAGNAME_LENGTH 255
#define DEVICE_INFO_LENGTH 100
#define DEVICE_UNIT_LENGTH 25
#define USER_PWD_LENGTH 10
module DatabaseServer

{
//Common structure to both digital and analog

        typedef struct TAGCOMMONINFO
                {
                        char Name[DEVICE_TAGNAME_LENGTH];
                        char Description[DEVICE_TAGDESC_LENGTH];
                        char IOName[DEVICE_TAGNAME_LENGTH];
                        octet AlmEnable;
                        octet PerLogEnabled;
                        long PerLogInterval;
                        octet LOCEnabled;
          long LOCChange;
        }CommonTagInfo;
```

```
typedef struct DIGITAL
        {
                TAGCOMMONINFO CommonDigTagData;
                char DigitalMsg[DEVICE_UNIT_LENGTH];
        } DigitalTag;

typedef struct ANALOG
        {
                TAGCOMMONINFO CommonAnaTagData;
                float AnalogMaxVal;
                float AnalogMinVal;
                float       AnalogAlmLoLo;
                float       AnalogAlmLo;
                float       AnalogAlmHi;
                float       AnalogAlmHiHi;
                octet       AnalogEnableLL;
                octet       AnalogEnableL;
                octet       AnalogEnableH;
                octet       AnalogEnableHH;
                float       DeadBand;
                octet       DeadBandEnabled;
        } AnalogTag;

typedef struct USER
{
        char UserName[USER_PWD_LENGTH];
        char UserDesc[DEVICE_INFO_LENGTH];
        char Password[USER_PWD_LENGTH];
        octet Privilege ;
}UserInfo;

typedef sequence<AnalogTag>AnalogTagSeq;
typedef sequence<DigitalTag>DigitalTagSeq;
typedef sequence<octet>OctetFileSeq;
typedef sequence<UserInfo>UserInfoSeq;


interface DeviceConfig
{
 long GetDeviceList(inout any DeviceList);//Getting Device Names
 long GetDeviceServerList(inout any DeviceServerList); //Available Server list (Active)
 long GetDeviceConfigList(in string Device,inout any DeviceConfigList);//Available
                                                        configuration list


// read from .tag file
long GetDeviceTagList(in string DeviceName, in string ConfigName,in string
TagType,inout any DeviceConfigList);
```

```
//reading from *.det
long OpenAnalogTagDetails(in string DeviceName, in string ConfigName, inout
AnalogTagSeq AnalogTagList, out long Count);
long OpenDigitalTagDetails(in string DeviceName, in string ConfigName, inout
DigitalTagSeq DigitalTagList,out long Count);


//Saving analog, digital tag details as *.det,.tag for all tags and
//Analog.det,Digital.det
long SaveAnalogTagDetails(in string DeviceName, in string ConfigName, in
AnalogTagSeq AnalogTagList, in long Count);
long SaveDigitalTagDetails(in string DeviceName, in string ConfigName, in DigitalTagSeq
DigitalTagList,in long Count);


// saving ,opening and deleteing configuration file
long SaveConfiguration(in string DeviceName, in string ConfigName, in string FileName
,in OctetFileSeq ReadWrite,in long FileSize);
long DeleteConfiguration(in string DeviceName, in string ConfigName);

};
interface UserLogin
{
        //check password from the file
        long Login(in string UserName,in string Password, out string Message);
        long GetUserList(inout any UserList);
        long GetUserDetails(in string UserName,inout UserInfo UserList);
        long SetUserDetails(inout UserInfo UserList);
        long DeleteUser(in string UserName);
};

};
```

This DBSERVER.idl includes functions for configuring new devices, functions which provides online devices, functions to read and save tag files, functions for checking user authorization and providing privileges from them. This IDL also provides functions for setting user authorization, and providing privileges to each and every user, deleting the users, setting and changing password and privileges of the users.

IDL for Logger Server (Logger.idl)

This is the file defining the CORBA interface for the logger server object in the CORSCADA system.

IDL Definition

```
module tagdetails{

 struct TagDataDetails1
   {
                char  TagName[30];
                float Value;
                long hour;
                long min;
                long sec;
                long LocEnabled;
        };

 struct Time
        {
         long hour;
         long min;
        };
 struct Date
        {
         long Day;
         long Month;
         long Year;
        };

 typedef sequence <TagDataDetails1> TagValue;

 interface Tags
{
  long GetTagValueWithTime(in Time fromtime,in Time totime,in Date OnDate,inout
        TagValue tagvalue);
 long GetTagValueWithDate(in Date fromdate,in Date todate,inout TagValue tagvalue);
        long GetLocEnabledTags(inout TagValue tagvalue,in Date OnDate);
        };
 };
```

The HMI Client calls these functions. And this Idl Provides functions for retrieving logged details of tags form specified files with the criteria specified.

4.5 Process Design

Process Design of User Authorization

Only registered users can use the services provided by the current CORSCADA system. In this package the user are given privileges for accessing the modules. The privileges are classified into 4 groups.

The Administrator has the sole privilege for accessing all the modules of the CORSCADA package and is given a privilege 1.the users with the privilege 2 can access and start Database Server as well the Devices server. Logger Client users are given with a privilege of 3.other users are given with the privilege of 4. They can able to view the reports.

Administrator monitors all the users logged on to the CORSCADA package. The details of the users are stored in the file **User.dat.** The administrator retrieves the user details from the database when required. The administrator is having options for new user registration, changing passwords

Call Functions to Database Server from Administration Client

- long Login(in string UserName,in string Password, out string Message);
- long GetUserList(inout any UserList);
- long GetUserDetails(in string UserName,inout UserInfoSeq UserList);
- long SetUserDetails(in string UserName,inout UserInfoSeq UserList);
- long DeleteUser(in string UserName);

long Login(in string UserName,in string Password, out string Message);

This function is used to check the privilege of the logged user. The inputs to the function are Username and Password. The Database server will opens the file **User.dat** and checks whether s/he is authorized or not, by checking the user privilege field. And passes a message to client stating whether he is authorized or not.

The Messages passed are:

- **CORSCADA_ADMIN**
- **CORSCADA_DATABASEPLUSDEVICE**
- **CORSCADA_LOGGER**
- **CORSCADA_USER**

For the logged user with privilege 1,the Message send is CORSCADA_ADMIN.

For the logged user with privilege 2,the Message send is CORSCADA_DATABASE PLUSDEVICE

For the logged user with privilege 3,the Message send is CORSCADA_LOGGER

For the logged user with privilege 4,the Message send is CORSCADA_USER

long GetUserList(inout any UserList);

This function shows the list of all users in the CORSCADA package. The input to the function is ANY type variable. This function opens the **User.dat** file and gets all the usernames and their privilege number and filled to the variable UserList.

The output is got in the format

⇒ *UserName_Previlage, UserName_Previlage*

E.g. Lancy Thomas_1,Santha Devi_1,John_4...

long GetUserDetails(in string UserName,inout UserInfo UserList);

This function shows the details about a specified user. The input to function is the username. This function opens the **User.dat** file and searches for the specified user from that get the details of the user and store in the Structure UserList and passes to the Administrator Client.

The UserList structure definition is

```
typedef struct USER
{
        char UserName[USER_PWD_LENGTH];
        char UserDesc[DEVICE_INFO_LENGTH];
        char Password[USER_PWD_LENGTH];
        octet Privilege ;
}UserInfo;
```

The structure is filled with the username, description of the user i.e. actual name of the user, password and his privilege number. The users who have the privilege value as 1 can call this function. Here username should be Unique.

long SetUserDetails(in UserInfo UserList);

The users who have the privilege 1can access this function and is used to set a new user to the CORSCADA package. The UserList structure is filled with Username, User description, password and his privilege no and is passed to the server. The Database server will opens the file User.dat and checks any duplication for username and stores this new record to the file.

long DeleteUser(in string UserName);

The users who have the privilege 1can access this function and are used to delete a registered user form the CORSCADA package. The Username is passed to the sever. The Administrator server will opens the file User.dat and remove the user from the file.

Process Design of Device Configuration Client

This GUI based program used for creating, modifying and deleting configurations of different type of device servers. Configurations are varying according to the type of device servers.

When this client starts it registers with the database server and gets the available active server list and its configuration. Once the users are given with the necessary information, provision will be given for creating a new configuration for the selected device, Modify and delete the already an existing configuration.

This also provides provision for selecting tags and setting tag properties. Once these functions are successfully done the data are stored in the file depends on the tag type selected.If the selected tag type is Analog, then the Configured tag names are stored in the file **Analog.tag**. And the configuration details are stored in the file **Analog.det**.If the selected tag type is Digital, then the Configured tag names are stored in the file **Digital.tag** . And the configuration details are stored in the file **Digital.det**.

All these files are saved at the C:\ DatabseServer*\DeviceName*\ConfigName*\ where the database server is running.

Call Functions to Database Server

// read from .tag file

- long GetDeviceTagList(in string DeviceName, in string ConfigName,in string TagType,inout any DeviceConfigList);

//reading from *.det

- long OpenAnalogTagDetails(in string DeviceName, in string ConfigName, inout AnalogTagSeq AnalogTagList, out long Count);
- long OpenDigitalTagDetails(in string DeviceName, in string ConfigName, inout DigitalTagSeq DigitalTagList,out long Count);

// saving and deleteing configuration file

- long SaveConfiguration(in string DeviceName, in string ConfigName, in string FileName ,in OctetFileSeq ReadWrite,in long FileSize);
- long DeleteConfiguration(in string DeviceName, in string ConfigName);

long GetDeviceTagList(in string DeviceName, in string ConfigName,in string TagType,inout any DeviceConfigList)

The configuration client to get the configured tag names from the .tag file calls this function. The inputs passed to this function are Device name, Configuration Name, tag type i.e. analog or digital. The database server opens the file **Analog.tag or Digital.tag** depends upon tagtype and fill the contents in an ANY variable DeviceConfigList and passed to the Configuration Client.

The output is got in the format

⇒ *Tag, Tag, Tag...*
E.g. Sine, Counter, Hour,.....

long OpenAnalogTagDetails(in string DeviceName, in string ConfigName, inout AnalogTagSeq AnalogTagList, out long Count);

When this function called with device name and configuration name the file **Analog.det** from the database running machine is opened. All the file contents are filled in a structure sequence **AnalogTagList** and are passed to the Configuration client with the tag count.

long OpenDigitalTagDetails(in string DeviceName, in string ConfigName, inout DigitalTagSeq DigitalTagList,out long Count);

When this function called with device name and configuration name the file **Digital.det** from the database running machine is opened. All the file contents are filled in a structure sequence **DigitalTagList** and are passed to the Configuration client with the tag count.

long SaveConfiguration(in string DeviceName, in string ConfigName, in string FileName ,in OctetFileSeq ReadWrite,in long FileSize);

This function is called when the client press the **save configuration** button of the Configuration Client. This is used to save the files in the system where the database server is running. When the configurations are made all the details are stored in a temporary file and when the user click the **save configuration** button the temporary file contents is transferred to an Octet sequence structure. This function is called with device name, configuration name, octect sequence, and file size. Once this function is called the temporary file is deleted.

long DeleteConfiguration(in string DeviceName, in string ConfigName);

This function is called when the user press the **Delete** button of the **Device Configuration Setting Window**. This function takes device name and configuration name as input parameters and passed to the database server. The Database Server deletes directory with the name of the selected configuration from the drive
C:\ DatabseServer*\DeviceName*\

Process Design of Logger Client

Only registered users can start the logger. Administrator Server monitors all the users logged on to the CORSCADA package. When the user logs in it gets connected to the Database Server and call the function **Login ()** with username and password. The Database server will open the file **User.dat** and checks whether the user is authorized. If the user is authorized person then logger Client is started and shown with the GUI screen.

When the Logger starts it *register with the* **Database Server.** Once the connection is established, it calls the function **GetDeviceServerList ()** that exposes all the active devices and its configuration in a specified format.

⇒ *DeviceName_Configuration.*

E.g. Thics_Area1

Once the device names are listed, select devices to be logged by pressing the **Start Log** button of the Logger GUI. The **LocThread** function is called and it calls **OpenAnalogTagDetails, OpenDigitalTagDetails ()** from the database server and the Log On Change tags are stored in the structure struct **MyLogChange** and stores the tag information in appropriate structure sequence separately. **FillBuffer()** is called to Create a buffer with the structure

```
struct TagDataBuffer
{
    char  TagName[30];
    float Value;
    float TagMin;
    float TagMax;
    float Average;
    float TagAppear;
}*TagBuffer;
```

The buffer size is dynamic in nature and will depend upon the tag counts. Connects with the Device Server and call **GetOnlineData ()**. Which fills the sequence structure with the given tag names and its device exposed online values and is stored in the **TagBuffer.** This function also calls a thread function to attach logger with event channel of the particular device. The thread function called is **EventChannelThread function.** When this thread function is called logger binds with the Event Channel of device server.when ever the device server pushes any tag details, logger checks whether the pushed tag has LOC. If the LOCEnable is true the currently got value is checked against with the values in the TagBuffer Content. And if the Value is greater than the specified percentage then that tag is written to a temporary file named **Servername_configname eddmmyyyy.htd**. Otherwise the content of the TagBuffer is rewritten with the new Tag value. If channel pushes values for specified tag, the min, max and sum of pushed values are calculate and rewrite in the

TagBuffer structure for the particular tag. The TagAppear is used to count the no: of time the event channel pushes values for particular devices.

The function PeriodicLogThread function group the tags according to the periodic time interval and is stored in the structure **MyPeriodicLog** and for each group a thread function is called. The thread function implemented here is **GetDataThreadFunction ()**.The time interval for periodic log tags are checked, and if it has reached is logged by monitoring a counter. If the counter reached specified time the tag details form the TagBuffer will be stored back to the **Servername_configname ddmmyyyy.htd** file.. When the specified time interval reaches the average is calculated and stored back to TagBuffer.

When the Size of the **Servername_configname ddmmyyyy.htd** reaches 10000 bytes, a ReadEvent is set to read the contents from the file. Once file and the event handles are released , the file contents are compressed and a WriteEvent is set for storing the compressed data to the logger file. The logger file are created for particular devices with the current date and the **Servername_configname ddmmyyyy.htd** file contents are deleted

- *ServerName_ConfigurationName ddmmyyy.chtd*
  E.g. Dummy_Config24022002.chtd

After writing to the **.chtd** file logger checks for day change. if there is change of day, then the current **.chtd** file is closed and a new logger file is created with the format specified above. **Get New Devices** button is used to get new devices name after the logger has started logging. When this button is pressed logger connects with database Server and calls **GetDeviceServerList**

Call Functions to Database Server

When the logger starts it gets registers with database server, the database server exposes these functions to the logger

- long GetDeviceServerList (inout any DeviceServerList);
- long OpenAnalogTagDetails(in string DeviceName, in string ConfigName, inout AnalogTagSeq AnalogTagList, out long Count);
- long OpenDigitalTagDetails(in string DeviceName, in string ConfigName, inout DigitalTagSeq DigitalTagList,out long Count);

long GetDeviceServerList (inout any DeviceServerList);

This function exposes all the active devices with the configuration of list of tags to be logged. The parameters passed to this is of the type ANY. The output will a list of active severs and its configurations in the below format separated by Comma in the variable *DeviceServerList*.

⇒ ***ServerName_Configuration***

E.g. Dummy_Config

long OpenAnalogTagDetails(in string DeviceName, in string ConfigName, inout AnalogTagSeq AnalogTagList, out long Count);

When this function called with device name and configuration name the file **Analog.det** is opened. All the file contents are filled in a structure sequence **AnalogTagList** and are passed to the logger client with the tag count. This function exposed all the analog tag with its configuration details

long OpenDigitalTagDetails(in string DeviceName, in string ConfigName, inout DigitalTagSeq DigitalTagList,out long Count);

When this function called with device name and configuration name the file **Digital.det** is opened. All the file contents are filled in a structure sequence **DigitalTagList** and are passed to the logger client with the tag count. This function exposed all the digital tag with its configuration details.

Call Functions From Device Server

Once the logger registers with the database server and executes all the above functions, logger will register with device server for getting the real time data for the configured tags from the selected logging devices.

- long GetOnlineData(inout TagData TotalTagData););

long GetOnlineData(inout TagData TotalTagData);

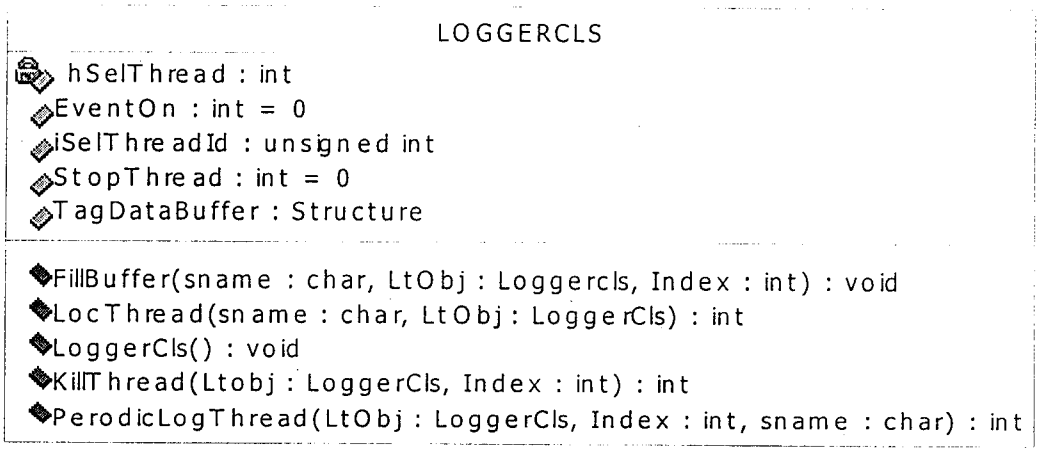
This function exposes the data from the devices through the device server to the LOGGER.The parameter to this function is of structure type *TagData. Here* the tag names are given as input to the function and after the execution of this function this fills the *TagType with values*

```
struct strTagData
  {
        char TagName [255];
        float Value;
  };
typedef sequence < strTagData > TagData;
```

Logger Main Class

```
//structure for contolling periodic enabled tags
        struct PeriodicLog
        {
                char IoName[255];
                long interval;
        }*MyPeriodicLog;

//structure for contolling loc enabled tags
        struct LogChange
        {
                char IoName[255];
                long percentage;
        }*MyLogChange;
```



Structure used as buffer for storing tag details

```
struct TagDataBuffer
{
        char  TagName[30];
        float Value;
        float TagMin;
        float TagMax;
        float Average;
        float TagAppear;
}*TagBuffer;
```

Structure used to store data's on the servername_configname ddmmyyyy.htd file

```
struct TagDataDetails
{
        char  TagName[30];
        float Value;
        int hour;
        int min;
        int sec;
        int LocEnabled;
};
```

LoggerCls()

This is the logger constructor class which will initialize two variables StopThread and EventOn to zero.

void FillBuffer (char* sname, LoggerCls Lobj, int index);

This function is called to initialize TagBuffer which will have the size of total count of tags for a particular device. This function will also fill this buffer with values got from GetOnlineData function of DeviceServer. Once tagBuffer is filled it initialize the thread **EventChannelThread** () that will bind Logger with Event channel of that particular device.

EventChannelThread () also checks the pushed data from the event channel. If the pushed tag is there in MyLocChange then the pushed TagValue is checked with the TagValue of the in theTagBuffer.If the percentage difference is greater than or equal to the LOCPercentage then that value along with Tagbuffer contents are logged with locEnabled field set to one. The TagBuffer Contents are updated with the new pushed information.

int PeriodicLogThread (char* sname, LoggerCls Lobj, int index);

This function is used to group the periodic log enabled tags. The grouped information is stored to a temporary structure array and for each group separate threads are called for logging purpose. The thread function used is GetDataThreadFunction().

int LocThread (char* sname, LoggerCls Lobj);

This function is used to group the Loclog enabled tags. The grouped information is stored to a temporary structure array.

Int KillThread(LoggerCls Lobj,int Index);

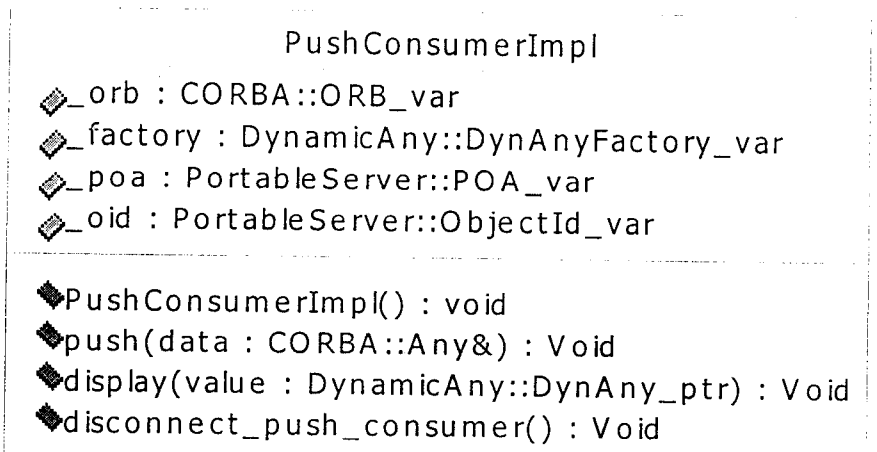
This function when called will stop all the threads initialized for a particular device.

PushConsumerImpl Class

This Class is used to create consumer distributed objects that use the push model of communication.

PushConsumerImpl()

This is constructor function which will be used to fill the variables orb, factory, oid and poa.



Push(data::CORBA::Any&)

The consumer implemented in the logger will call this function once. This will make a connection with Proxy Supplier and which will continually supply data to our consumer. This function is implemented in the logger module and in Push() function **Display()** function is called. **Display** function will take an **ANY** variable as input and will split this **ANY** to sub parts automatically.

Disconnect_push_consumer ()

The Disconnect_push_consumer () method is used to disconnect the consumer from the event channel.

Process Design for Logger Server

This module is used to retrieve information form the logger files. The logger files are stored in the place where the Database server is running. The functions are called by the HMI to show the reports. When these functions are called the specified compression file is decompressed into **htd** file. And this decompressed is used to retrieve values for the reports. After the use the htd file is delete.

The functions called from logger server by the HMI Client are

- long GetTagValueWithTime((in string DeviceName,in Time fromtime,in Time totime,in Date OnDate,inout TagValue tagvalue);
- long GetTagValueWithDate(in string DeviceName ,in Date fromdate,in Date todate,inout TagValue tagvalue);
- long GetLocEnabledTags(in string DeviceName ,inout TagValue tagvalue,in Date OnDate);

long GetTagValueWithTime(in string DeviceName ,in Time fromtime,in Time totime,in Date OnDate,inout TagValue tagvalue)

This function retrieves from the specified file those records, which lie between from time and to time for the specified server.

long GetTagValueWithDate(in string DeviceName ,in Date fromdate,in Date todate,inout TagValue tagvalue);

This function retrieves tag information from those files of the specified device that lie between from date and to date.

long GetLocEnabledTags(in string DeviceName ,inout TagValue tagvalue,in Date OnDate);

This function retrieves information of those tags which has Log On Change Field Enabled.

Function used for Compression and Decompression

Two header files JCompress.h and BPEExpnd.h were used for compression and decompression. The method used was BPE method. This header file was provided by ERDC itself.

The functions called for compression are

JCompressedFile(char *szFileName);

This is the constructor of header file and here the file name used to store compressed data is specified.

int Append(unsigned char *Buffer, int Length);

This function is actually used for compression. Here Logger from the temporary file will fill buffer. Length denotes the total bytes read from temporary file.

The function called for Decompression is

BPExpandFile(argv1,argv);

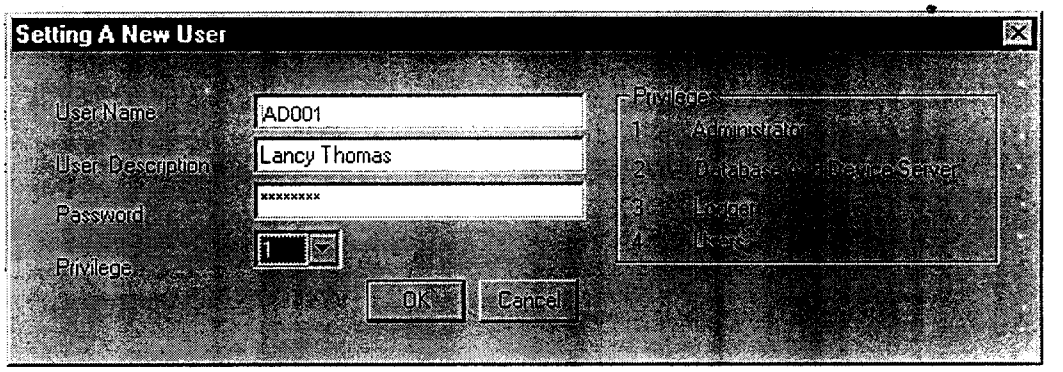
Here argv denotes the destination-uncompressed file to be created and argv1 denotes the source-compressed file.

4.6 Input Design

User authorization

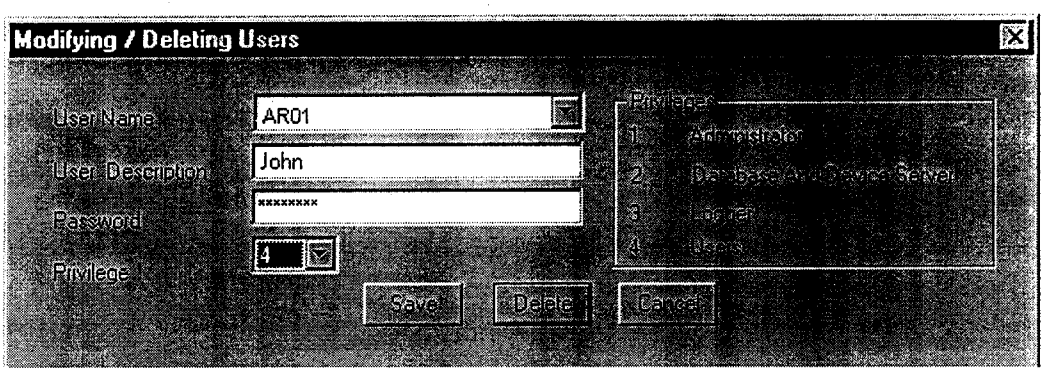
Only registered users can use the services provided by the current CORSCADA system. In this package the user are given privileges for accessing the modules. The privileges are classified into 4 groups.

The input screen for setting a new user is



When the ok button is pressed the new user information is stored in the User.dat file where the database server is running.

This window is used for modifying the user information and for deleting users from the CORSCADA package.



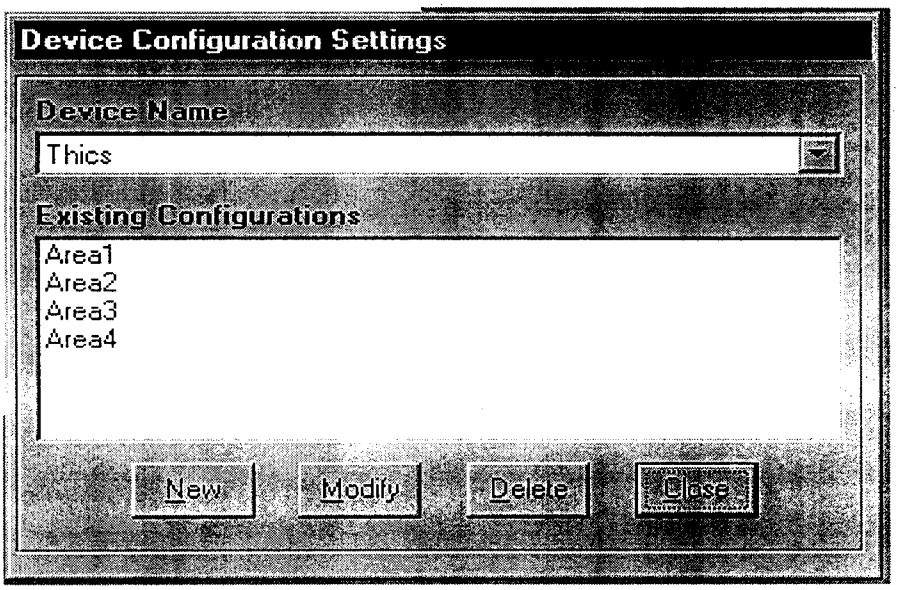
When the save button is pressed the entire entered user information are stored back to the **User.dat** file. Delete button is used to delete an already existing user from the CORSCADA package. When the user presses the delete button a confirmation messages to shown when the confirmation is got the user will be removed

Device Configuration Client

This GUI based program used for creating, modifying and deleting configurations of different type of device servers. Configurations are varying according to the type of device servers.

Device Configuration Setting Window

This screen helps the user to create a new configuration or to modify existing configurations of a particular server.



This has 4 buttons, a combo box showing the plants active devices and a list box, which shows the existing configuration of the selected devices.

⇒ **New**

⇒ **Modify**

⇒ **Delete**

⇒ **Close**

The **New Button** is used for creating a new configuration setting for the selected device. When this button is clicked a window is displayed for setting up a new configuration of tags.

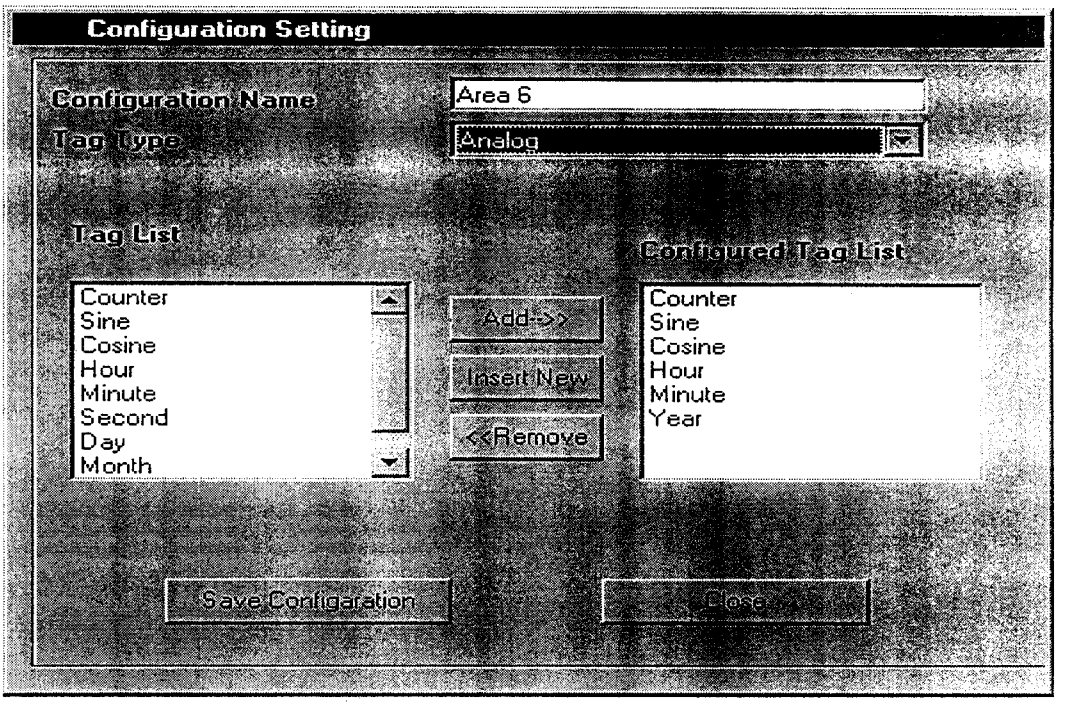
The **Modify Button** is used for modifying an existing configuration setting for the selected device. When this button is clicked a window is displayed for modifying the configuration of tags.

The **Delete Button** is used for removing an existing configuration setting for the selected device. When this button is clicked a delete confirmation is prompted to the user and when the confirmation is got the configuration is removed.

The **Close Button** is used for closing the configuration-setting window.

When the New / Modify button is clicked the user will be showed by the configuration setting window.

Configuration Setting Window



This window is used for Creating or modifying a new or existing configuration setting. The buttons used here are

⇒ **Save Configuration**

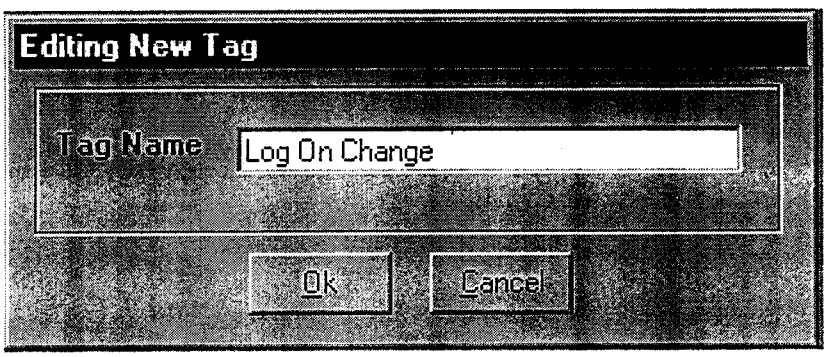
⇒ **Add-->>**

⇒ **Insert New**

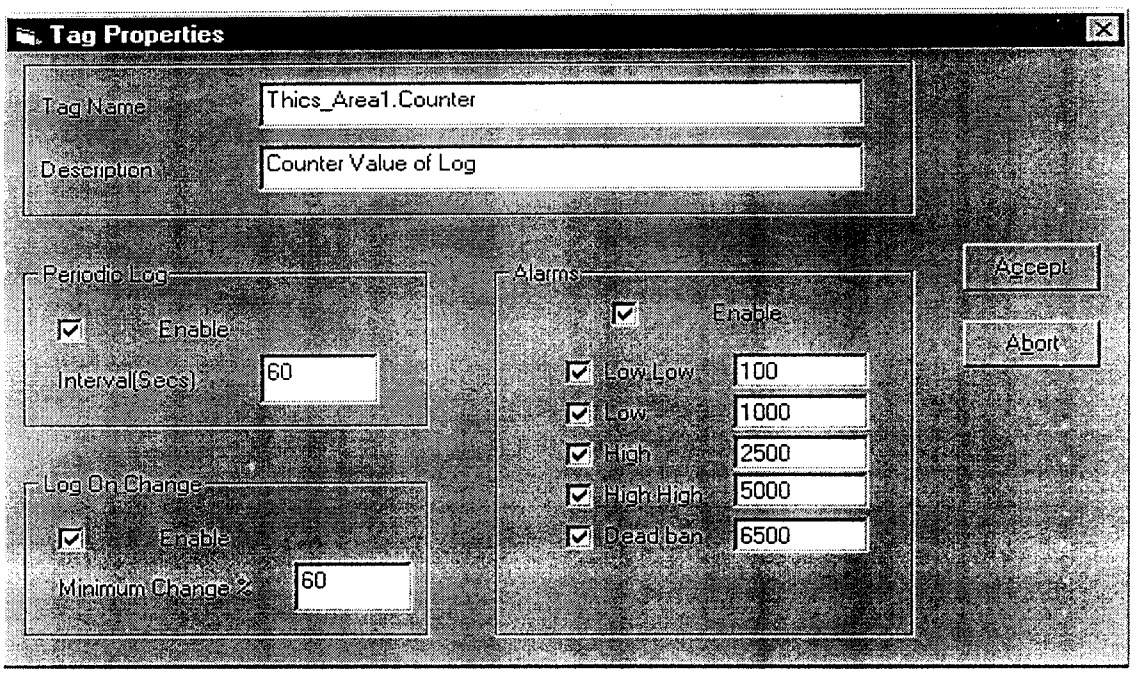
⇒ **<<Remove**

⇒ **Close**

The users have to give an appropriate configuration and select the tag type for the configuration. The tag list shows the tag name available for the selected tag type. User can select those tags that they needed for the current configuration and press the **Add-->>** button. All the selected tag is displayed on the configured tag list. The **<<Remove** button is used to remove those tags that are not needed. The **Insert New** button is used to create new tags, which are not in the tag list. When the **Insert New** button is clicked, a new window is displayed through which a new tag can be added.



Once a new configuration is configured **Save Configuration Button** is clicked to save the configuration. Clicking the tag name from the Configured Tag List from the Configuration Setting Window can configure the tag properties.



Here the tag name can't be edited. The description field shows what that specific tag name corresponds to. The periodic Log field shows whether that tag has periodic log or not. The interval field specifies the time interval for that specific filed in seconds. The Log on Change enables shows whether the tags have the LOC and the percentage shows the % value that is to be checked with the logged value. During the logging process if the value of any tag is increased or decreased by a specified percentage it should be logged as individual record.

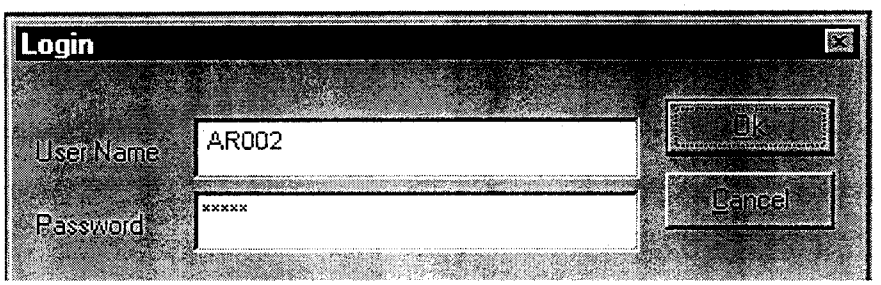
The alarm field shows the low, high and dead ban values of the log values of the specified type of the current configuration.

The **Accept** button is to save the tag properties and **Abort** is to cancel the current changes.

Logger Client

When the logger is activated it first shows up with a login screen for user authorization.

Login Window



Only the users with privileges 3 can start the logger. When the **Ok** Button is pressed the inputs are checked with the contents of the file **User.dat**. If there is no valid user or if the password is wrong the user will be notified with appropriate messages. Otherwise privilege for that particular user is checked and if it is 3 Logger Main Window is shown to the user.

When the logger starts it connects with the database server and starts getting the details about the devices (system) that are currently running through Database Server. Select from the active server list those devices, which are to be logged. Once the logging for specific devices started logger connects with an event channel. And through the event channel the logger pushes and gets the online information about each and every configured tag from the Device Server.

The inputs for the logger consists of

- **Active Server Names**
- **Configured Tag list**

The *Active Server Names* are got through the Database Server. And this contains the names of all the plant's active devices currently running. The Database Server exposes the server names in the format specified below. Here Dummy is the device name and Config is the current configuration i.e. selected for logging purpose

⇒ *ServerName_Configuration*

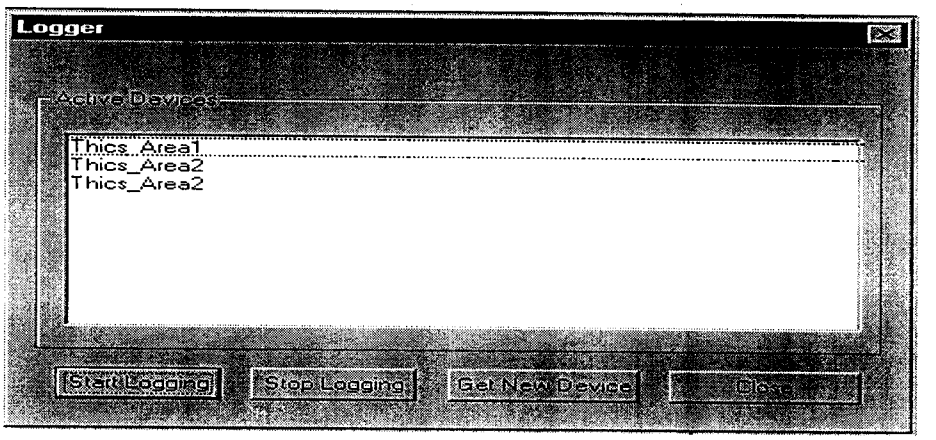
E.g. Dummy_Config

The *Configured Tag List* is extracted from the Configuration specified through the Active Server Names. To get the tags that are to logged for the particular devices first get the configuration name for the specified device from the Active Server List and extract the names of tags from the configuration. This contains a sequence of tag names and these tag values are used for getting the online values from the device.

⇒ ***Tag1, Tag2, Tag3,.....***

E.g. Counter, Sine, Cosine, min, sec, hour

Logger Main Window



When the logger is invoked the logger main screen is activated and is displayed.
This has 4 buttons and a list showing the plants active devices, which are configured for logging.

⇒ **Start Log**

⇒ **Stop Log**

⇒ **Get New Device**

⇒ **Close**

The user has to select those devices that have to be logged and press the **Start Logging** button. From that time, the logger starts logging of the particular devices to the

corresponding logger file. The logged file will be automatically created and logged informations are written off.

The file name convention used here is

⇒ *ServerName_ConfigurationNameddmmyyy.chtd*

E.g. Thics_Area312032002.chtd

The **stop Logging** button is for stops the logging of particular devices of a group of devices. Once the user clicks the stop logging button the file created for the particular server is closed. The **Get New Device** Button is pressed to get those new devices that have be started after the logger is started. The **close** button is to close the Logger window

4.7 O/P design

The normal procedure is to design the outputs in detail first and than to work back to the inputs. The outputs can be in the form of operational documents, lengthy reports. The input records have to be validated, edited, organized and accepted by the system before being processed to produce the outputs.

Computer output is the most important and direct source of information to the users. Designing the computer output should proceed in an organized, well thought out manner. The right output must be developed while ensuring that each output element is designed so that people will find the system easy to use efficiently. When analyst design computer output they identify the specific output needed to meet the information requirement.

Logger Outputs

The output from the Logger module consists of reports about the logged data of the plant's devices that are logged or under logging.

The Logger Server on request by the HMI Client will retrieves the necessary information from the historic file / Logger files and exposes to the HMI Client what they needed.

The important reports generated by the Logger are

- **Based on specific dates**
- **Based on Specific time Interval**
- **Based on Log On Change**

When ever a request is given to the logger server the server first get that compressed file form the system where the database server is running and decompress the file and save that file with the extension .htd file. And query that files depends on the inputs given to the Logger Server.

5.System Implementation & Testing

5.1 System Implementation

Implementation includes all those activities that take place to convert from the old system to the new. The new system may be totally new, replacing an existing system. Proper implementation is essential to provide a reliable system to meet the organizational requirements. Successful implementation may not guarantee improvement in the organization using the new system, as well as, improper installation will prevent any improvement.

The implementation phase involves the following tasks:

- Careful Planning
- Investigation of systems and constraints
- Design of methods to achieve the changeover phase.
- Training of staff in the Changeover phase.
- Evaluation of Changeover.

The method of implementation and time scale to be adapted was found out initially. Next the system is tested properly and at the same time the users were trained in the new procedure.

The implementation process begins with preparing plans and activities for the implementation of the system. Discussions are made for preparations of activities. New equipments are acquired for implementing the new system. Design and coding is done according to the requirement of the user and is tested and debugged.

The successful of the implantation of the new system depends on the involvement of the staff working in the department. In the case of Distributed Logger the department staff are given with the detailed information about the logger's working. The procedure for installing the module is taught.

Change over is the stage of moving from existing system to the proposed system. The new distributed logger using CORBA by the Direct Change Over Method is replacing the current system.

5.2 System Testing

System testing is an expensive but critical process for the software development. The test cases are developed for the DistributedLogger with this in mind. A test case is a set of data that the system will process as normal input. However, the data are created with the express intent of determining whether the system will process them correctly

Some of the test data includes was

- Invalid Date
- Invalid Time

In all these cases the system is found to be reliable. The following are the test performed on the system.

Password Testing

Administrator as well as the user has to specify username and password for entering into system, it will be validated with the values in the database. If they do not match, access is denied to login to the system, thereby providing a strong security.

Unit Testing

In unit testing the attention is diverted to individual modules, independently to one another to locate errors. This testing was carried out during programming stage itself. In the testing step, each module is found to be working satisfactorily as regards to the expected output from the module.

Code Testing

Here the general logic of the program is tested. Test cases were developed that result in executing every instruction in the program module, here every path through the program is tested.

System Testing.

Here the integration of all modules with in the system is tested thoroughly. This verifies whether all the specifications are appropriate. All the modules are working accurately when the buttons are invoked through the GUI windows.

Specification Testing.

The specification stating what the program should do, and how it should perform under various conditions were determined. Then the test cases were developed for each condition and combination of conditions were processed and found satisfactorily.

Output Testing:

The outputs generated or displayed by the system under consideration are tested by asking the users about the same required by them, Here, the output format on the screen is found to be correct as the format was designed in the system design phase according to the user needs.. Hence, output testing has not resulted in any correction in the system.

In all these aspects, the system was found to be reliable.

The results of the testing phase are taken as hard copy and are attached in the appendix section of the report.

6.Conclusion & Scope for future development

6.1 Achievements

The system Distributed Logger was successfully developed to fully satisfy the objectives set by the user. The system was tested with real data and all the reports were successfully taken and were found to satisfy the needs of the concern. A good amount of user-friendly GUI features have been incorporated and it is possible for anyone to exploit these features to get maximum benefits.

The system takes care of effective storage and retrieval of information. The system performs the right procedures properly, presents information in the acceptable and effective way, produces accurate results, provides an acceptable interface and a method of interaction and perceived by users as a reliable systems.

6.2 Scope For Future Development

- It can be further developed to include more operations and analysis, as changes are required in the system to adapt to the internal developments such as new activities.
- Further enhancements can be made to this system at any later point of time.
- Coding procedures can be modified according to the needs of the user. The system code is so well designed that it will form the basis for further enhancement and also new operations can be included in the system.
- Reports can be represented in all necessary perspective. Added options can be included in designing reports.

Reference

Books

Robert Orfali & Dan Harkey , "The Essential Client/Server Survival Guide, 2nd edition". Object Management Group-1997,

Robert Orfali & Dan Harkey ,"The Common Object Request Broker Architecture, V2.0, Architecture and Specification".

Object Management Group – 1997, "The Common Object Request Broker Architecture, V2.0, CORBA Services".

Suhail M.Ahmed, "CORBA Programming Unleashed" Sam Publishing, 1st Edition ,1999

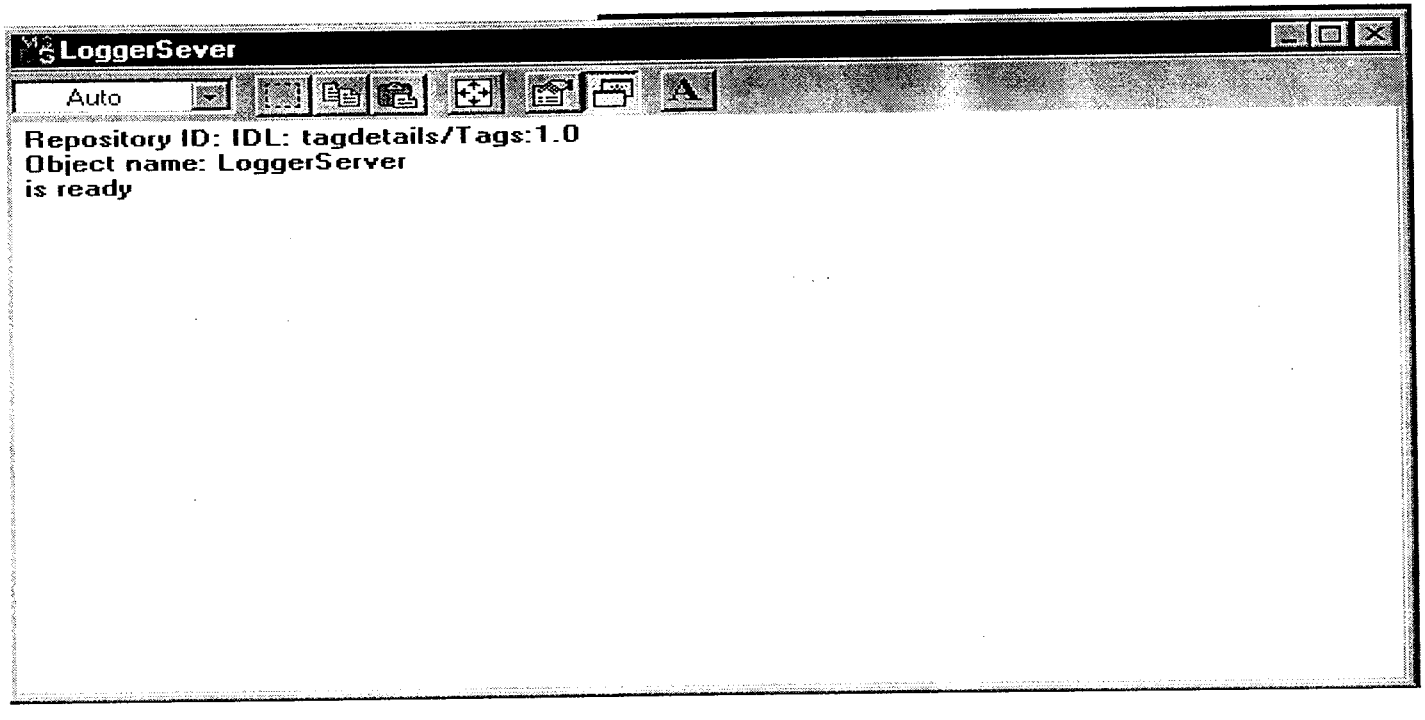
Michael J Young, "Mastering Visual C++ 6" BPB Publications

Sites visited

www.omg.org

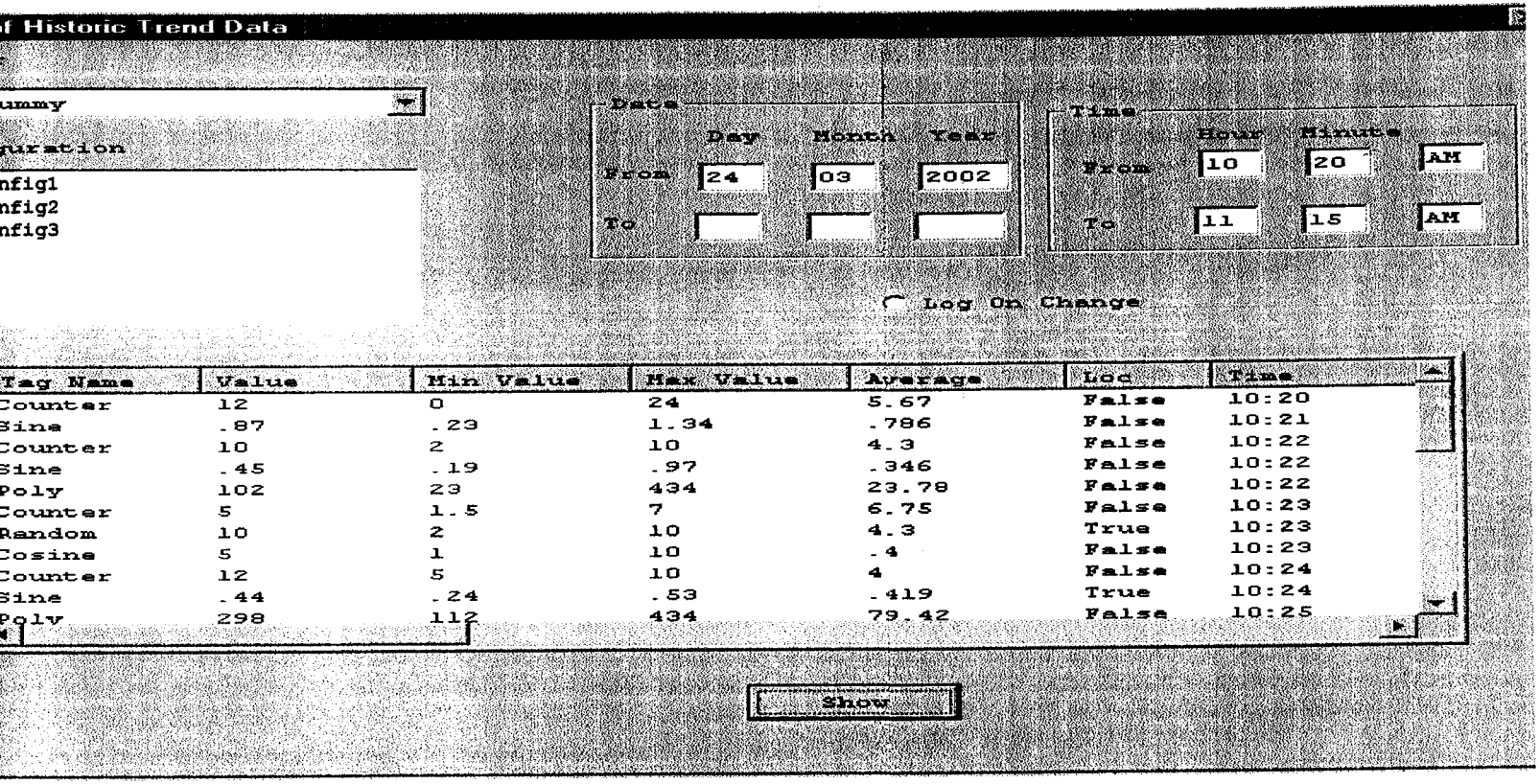
www.codeguru.com

www.funducode.com



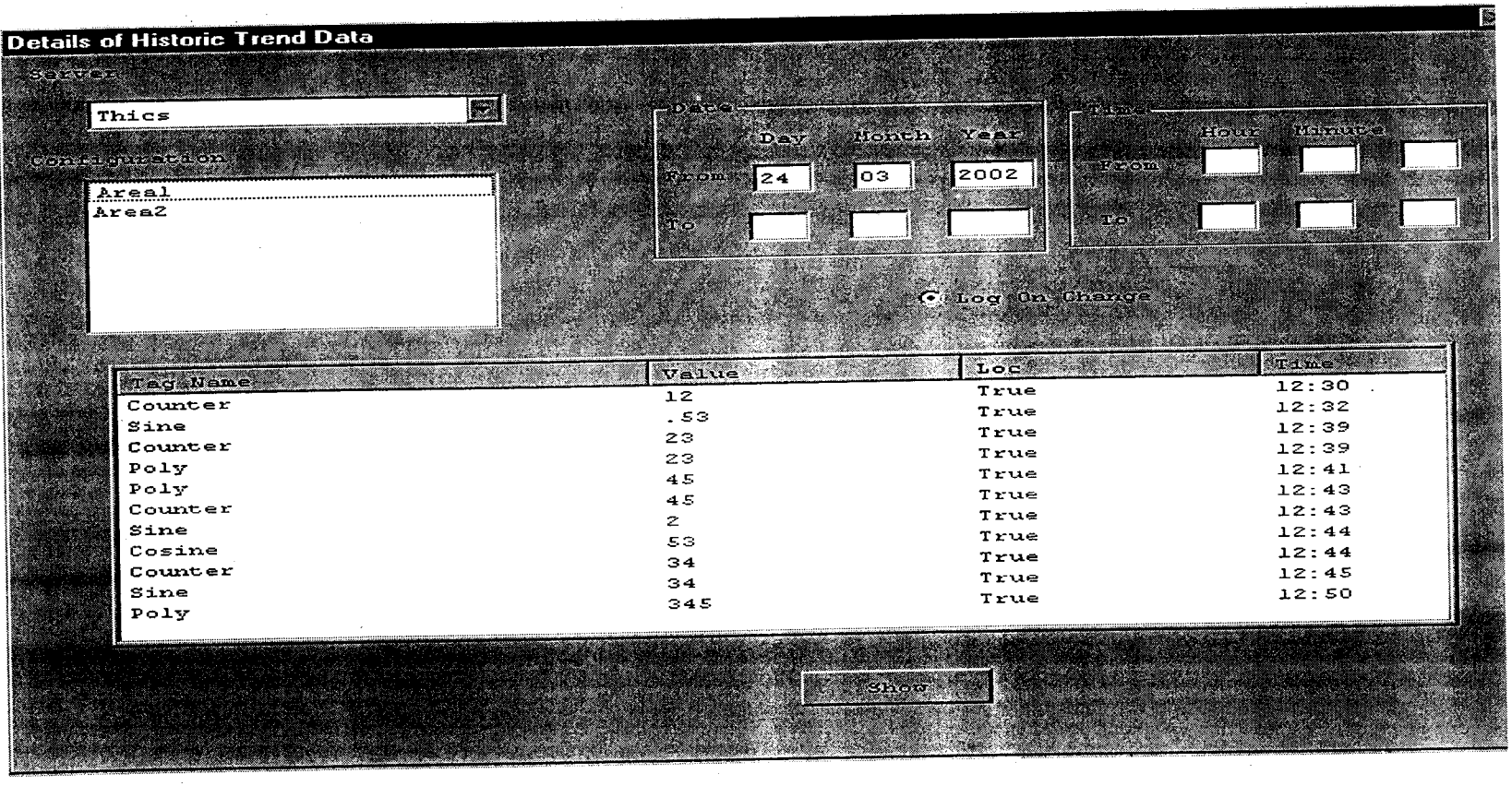
logger Server when it is started

Historic data from the file Dummy_Config224032002.htd files



Here all the tag information are got form a single file dummy_Config224032002.htd

Historic data from the file Thics_Area124032002.htd file



This report shows all the tags with in the specified file who's LOC is true