DOE/MC/29113 . -- 5599
(DE97002246)

Interactive Computer-Enhanced Remote Viewing System (ICERVS)

Final Report
November 1994 - September 1996

Work Performed Under Contract No.: DE-AC21-92MC29113

For

U.S. Department of Energy
Office of Environmental Management
Office of Technology Development
Washington, DC

U.S. Department of Energy
Office of Fossil Energy
Morgantown Energy Technology Center
Morgantown, West Virginia

By
Mechanical Technology Inc.
Latham, New York

HH

Disclaimer

DOE/MC/29113 -- 5599
(DE97002246)

Interactive Computer-Enhanced Remote Viewing System (ICERVS)

Final Report
November 1994 - September 1996

Work Performed Under Contract No.: DE-AC21-92MC29113

For

U.S. Department of Energy
Office of Environmental Management
Office of Technology Development
1000 Independence Avenue
Washington, DC 20585

U.S. Department of Energy
Office of Fossil Energy
Morgantown Energy Technology Center
P.O. Box 880
Morgantown, West Virginia 26507-0880

By
Mechanical Technology Inc.
968 Albany-Shaker Road
Latham, New York 12110

PREFACE

The work documented in this report was performed by Mechanical Technology Incorporated (MTI) for the United States Department of Energy (DOE) under Phase III of Contract DE-AC21-92M29113 in support of the DOE Environmental Restoration and Waste Management Program under the direction of the DOE Morgantown Energy Technology Center (METC) in Morgantown, West Virginia. It was performed during the period of November 1994 through September 1996. All communication regarding this report should be directed to the Contract Reports Receipt Coordinator at METC.

ABSTRACT

The Interactive Computer-Enhanced Remote Viewing System (ICERVS) is a software tool for complex three-dimensional (3-D) visualization and modeling. Its primary purpose is to facilitate the use of robotic and telerobotic systems in remote and/or hazardous environments, where spatial information is provided by 3-D mapping sensors. ICERVS provides a robust, interactive system for viewing sensor data in 3-D and combines this with interactive geometric modeling capabilities that allow an operator to construct CAD models to match the remote environment. Part I of this report traces the development of ICERVS through three evolutionary phases: 1) development of first-generation software to render orthogonal view displays and wireframe models; 2) expansion of this software to include interactive viewpoint control, surface-shaded graphics, material (scalar and nonscalar) property data, cut/slice planes, color and visibility mapping, and generalized object models; 3) demonstration of ICERVS as a tool for the remediation of underground storage tanks (USTs) and the dismantlement of contaminated processing facilities. Part II of this report details the software design of ICERVS, with particular emphasis on its object-oriented architecture and user interface.

SUMMARY

The Interactive Computer-Enhanced Remote Viewing System (ICERVS) has been developed by Mechanical Technology Incorporated (MTI), under contract to the U. S. Department of Energy (DOE), to support the robotic remediation of hazardous environments such as underground storage tanks (USTs), buried waste sites, and contaminated laboratory and processing facilities. The success of remote robotic missions into these environments will depend greatly upon the ability to construct reliable geometric descriptions in order to effectively perform task planning, path planning, and collision avoidance. The overall purpose of the ICERVS development project was to produce a mature system that provides the interactive visualization and modeling of the three-dimensional (3-D) data needed to construct the required geometric descriptions. ICERVS is a software program that runs on Silicon Graphics (SGI) graphics/computing platforms. It is a general-purpose system that can:

- Input data from a wide range of sources (mapping sensors, CAD files, direct operator input)
- Provide robust, integrated 3-D visualization of sensor data and geometric object models
- Provide interactive tools for creating and editing geometric object models
- Output geometric models to model-based robotic control systems.

With these capabilities, robotic system operators can maximize the insights gained from sensors in remote environments and can construct 3-D CAD ("world") models to match the physical environment. These CAD models can be exported to robotic control systems, such as Deneb's IGRIP, where they will allow system users to plan and preview robot tasks in a virtual environment that represents the best model available of the remote environment. The major features of ICERVS are summarized in Figure S-1.

The development of ICERVS was performed by MTI in three phases. In Phase I, MTI developed first-generation software for integrated visualization and modeling. As an important first step in the ICERVS design, MTI enumerated and described representative mission profiles for the remediation of USTs, buried waste sites, and contaminated processing facilities. From these profiles, a set of system requirements for ICERVS were developed. The software was designed in C++ source code using object-oriented methodology to promote modularity, code reuse, and reduced life-cycle cost. Approximately 50 software classes were implemented in Phase I, and the resulting software provided orthogonal views of dimensional data with hidden surface removal, view translation and scaling, and interactive creation and editing of simple prismatic shapes. The Phase I software was demonstrated using data provided by Oak Ridge National Laboratory (ORNL) from two waste tanks at Fernald that were mapped by a structured light sensor system.

In Phase II, MTI expanded the ICERVS visualization and modeling capabilities to address more of the system requirements identified in Phase I. The visualization capabilities were expanded to include features such as general viewpoint rendering, color mapping of material property data, and surface shading (versus wireframe rendering) of object models. Modeling capabilities were expanded to include more general prismatic shapes and the output of object models formatted as IGRIP part files. Also in Phase II,

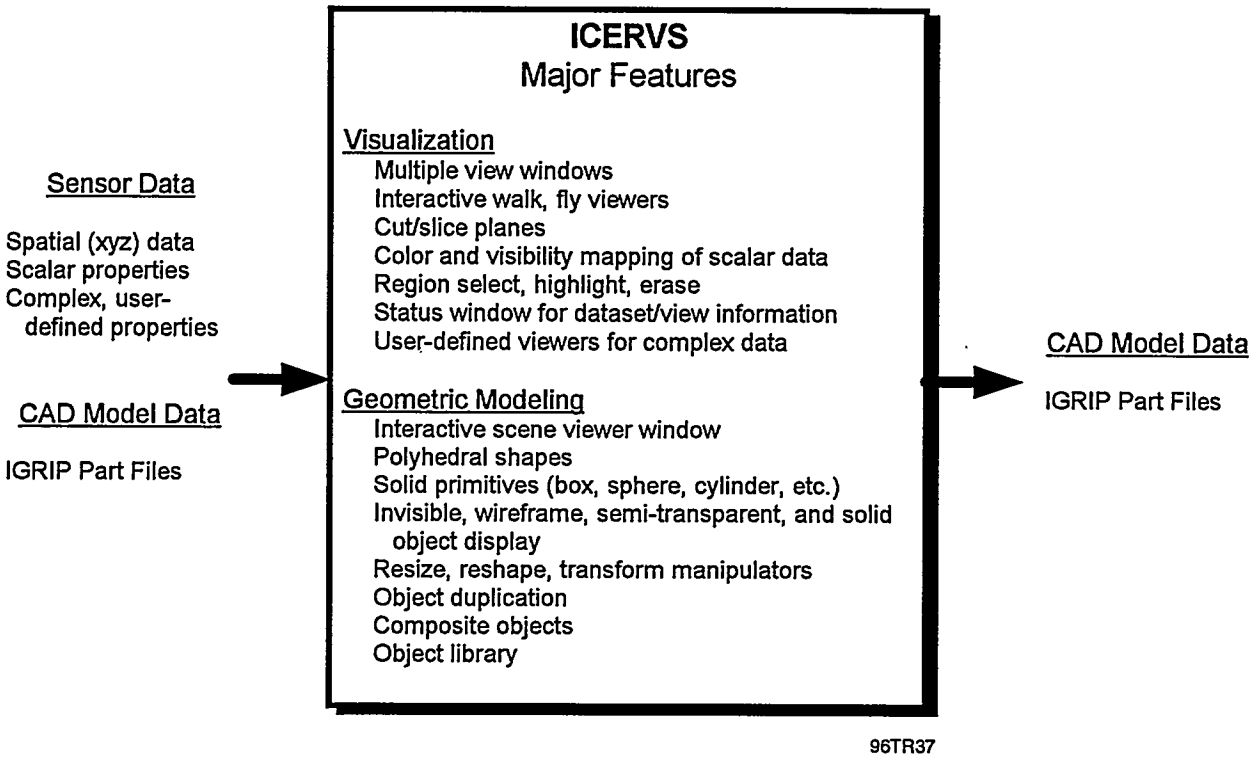


Figure S-1. Major Features of ICERVS

MTI restructured the ICERVS architecture into separate but interoperable subsystems based on Unix client-server interfaces. This restructuring allowed integration of the ICERVS software into higher-level robotic systems that incorporate model-based control, remote manipulators and sensors, and world model construction. The Phase II system was implemented in approximately 200 software classes. The system was demonstrated in laboratory mock-up situations to show performance in three remediation environments: USTs, buried waste, and facility dismantlement. For the UST demonstration, ICERVS was integrated with a structured light mapping sensor developed by MTI under a Cooperative Research and Development Agreement (CRADA) with ORNL.

In Phase III, MTI completed full-scale development of ICERVS. A major accomplishment of this phase was that the system was changed from direct volume rendering to surface-based rendering in order to obtain significantly faster throughput. Image rendering is now done using the SGI Open Inventor, an object-oriented 3-D toolkit. Open Inventor uses the 3-D graphics hardware in SGI platforms to provide interactive throughput for datasets with over 100,000 points. Additional visualization capabilities were implemented in Phase III, including immersive (fly, walk) viewers for navigation through a dataset, color and visibility mapping based on scalar property data, storage of complex data such as images or frequency spectra with display via external application software, cut and slice planes for obtaining cross-section views, and a status area that displays viewing parameters, color legend, and a 3-D icon indicating the view's rotational orientation.

Additional modeling capabilities implemented in Phase III include more general polyhedral shapes plus second order primitives (sphere, cylinder, etc.); on-screen manipulators, transform panels, and numerical parameter editors for modifying object size, shape, and position; control of individual object color and display as invisible, wireframe, semi-transparent, or shaded surfaces; grouping of objects into a single composite object; an object library that lets users select and store objects for reuse; and importing of object models from IGRIP part files. These capabilities were demonstrated at ORNL in conjunction with UST and facility decontamination and dismantlement (D&D) applications.

At the end of Phase III, ICERVS met or exceeded all success criteria established for its development: (1) ICERVS effectively conveys to end users the 3-D characteristics of remote workspaces and other sensor-provided characteristics of workspaces; (2) ICERVS provides the functionality required to produce 3-D CAD models of features in mapped workspaces; (3) ICERVS seamlessly integrates with external sensor systems and model-based robot controllers.

At the end of Phase III, the ICERVS software was delivered to the DOE and is now available for use. In summary, ICERVS is a software system that inputs data from multiple sources, provides robust 3-D visualization of both sensor data and geometric object models, provides interactive tools for creating and editing geometric object models, and exports models to robotic control systems for use in task and motion planning. These capabilities are needed by robotic system operators to interpret sensor data and effectively use it to construct or update 3-D CAD descriptions of the work environment. Because the severe demands of the DOE's environmental cleanup needs often mandate the use of remote robotic operation, the capabilities of ICERVS are a critical element in meeting the cleanup challenges that lie ahead.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

Part I
Program Overview

1.0 INTRODUCTION

Part I of this report presents an overview of the work performed during the ICERVS development project. In Part I, Sections 2.0, 3.0, and 4.0 summarize the Phases I, II, and III activities, respectively, and Section 5.0 presents the project conclusions. Part II of this report presents a technical overview of the final ICERVS design and implementation, as completed in Phase III. In Part II, details are provided on the system design (Section 2.0), ICERVS dataset (Section 3.0), dataset manager (Section 4.0), dataset view (Section 5.0), sensor data input (Section 6.0), geometric objects (Section 7.0), analysis features (Section 8.0), IGRIP interface (Section 9.0), and major class descriptions (Section 10.0). A glossary of terms is also provided for reference in Section 11.0.

1.1 Background/Technical Needs

The cleanup of hazardous waste sites within the U.S. nuclear weapons production complex presents a major challenge that will require extensive use of many advanced technologies including robotics. It is estimated that 381,000 cubic meters of high-level waste are stored in USTs, many of which are suspected of leaking contents to their surrounding environments. Another 190,000 cubic meters of transuranic solid waste are in buried sites, many of which are considered potential threats to their surrounding environment. There are also thousands of contaminated buildings across the complex that require D&D. Remediation of these waste sites will require retrieval of hazardous materials for their subsequent decontamination and/or storage in controlled facilities. Available technologies are not considered adequate to cost effectively solve the problems at hand. Advanced robotic and other technologies are needed to clean up the thousands of contaminated sites that presently exist [1].*

Robotic systems used in environments such as these must meet numerous requirements not commonly found in mainstream robotic applications. Systems must operate in preexisting, hostile environments without significantly altering them (beyond the retrieval of waste). There is often only partial and uncertain knowledge of the environment due to incomplete and/or missing records, physical and chemical changes that occur over time, and limited access due to safety concerns. In many applications, all robotic actions must be performed with high confidence because of extreme human and environmental safety concerns. The robotics, for example, must not compromise the integrity of existing structures and containers.

To address these advanced needs, a model-based control architecture has been developed by the DOE [2]. As shown in Figure 1-1, the architecture includes a human operator in the control loop to direct all tasks and major motions. Human insights are needed to effectively and safely respond to changing conditions and understand what is a continually evolving depiction of the remote environment and its characteristics. An integrated robotic system controller coordinates operator commands among various robot manipulators and subsystems that interact with the remote environment. The remote hardware includes mapping and video sensors to provide feedback from the work environment, and these data are used to construct a 3-D CAD model which is presented to the operator.

* Numbers in brackets designate references presented at the end of this section.

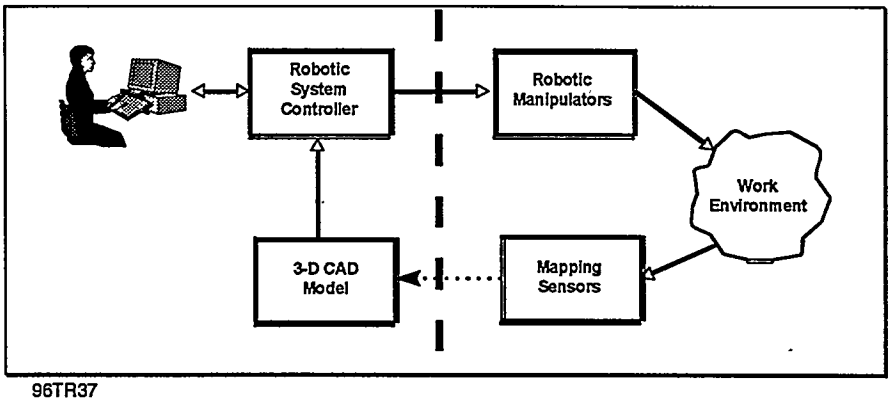


Figure 1-1. Simplified Model-Based Control Architecture

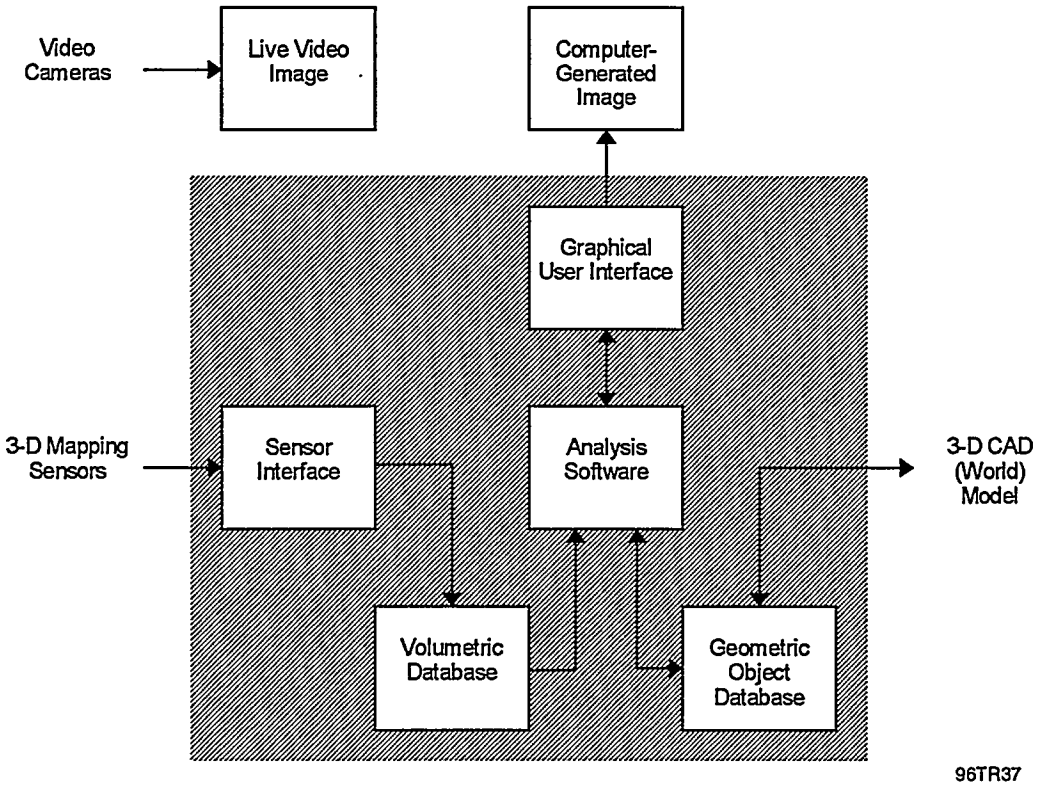


Figure 1-2. ICERVS Functional Model

One of the technology needs in this system architecture is a capability for constructing and verifying 3-D models from the data gathered by remote sensors. Customary triangulation methods [3] yield models with false surfaces and missing "holes," and also produce excessively large CAD models which strain computing resources. Automated object recognition techniques are not sufficiently mature for applications of "real world" complexity. The most effective approach is to provide an interactive system that combines sensor data visualization with 3-D geometric modeling. With an interactive system, operator insight can be leveraged to recognize physical objects and quickly construct CAD models to match them. As a result, reliable and efficient CAD models can be constructed in a cost-effective way.

The ICERVS was conceived as a way to accomplish this need for interactive visualization and modeling. A software program that runs on SGI graphics/computing platforms, ICERVS is a general-purpose system to:

- Input data from a wide range of sources (sensors, CAD files, direct operator input, etc.)
- Provide robust, integrated 3-D visualization of sensor data and geometric object models
- Provide modeling tools for creating and editing geometric object models
- Output geometric object models to model-based robotic control systems.

A functional model of ICERVS is shown in Figure 1-2, with the system elements described below.

The *sensor interface* uses an ASCII file format with the data arranged as an n-dimensional vector (x, y, z, property 1, property 2, ..., property n) to input data from mapping sensors.

In the *volumetric database*, ICERVS uses octree technology to efficiently store sensor data. The octree data structure is a hierarchical data representation that spatially sorts data for fast display and analysis [4]. It uses less memory than conventional means such as uniform grids or triangle networks.

The *geometric object database* stores object models in CAD format. These can be interactively created by operators or can be imported from robotic controllers such as Deneb's IGRIP. The contents of the geometric database can also be exported back to robotic controllers.

The *analysis software* and *graphical user interface* support visualization and modeling based on the SGI Open Inventor geometry library. Visualization is provided by a set of 3-D viewers and geometric modeling by a set of interactive manipulators, menus, and control panels. If the remote hardware includes video cameras, then the live video imagery can be compared to a computer-generated image of the same scene.

1.2 ICERVS Project Objectives and Organization

The overall objective of the ICERVS development project was to produce a mature system for the interactive visualization and modeling of the 3-D data needed for remote robotic remediation of hazardous environments. The development of ICERVS was organized into three phases, each with an increasing level of technology maturity, that culminated in the demonstration of a full-scale system at DOE field sites. In Phase I,

ICERVS was developed to Maturity Level III, Subscale Major Subsystems, for those technologies that had been previously demonstrated at the research laboratory scale. The majority of this work involved development of visualization and analysis software. Specifically, this involved creating an octree data structure from simulated sensor data, rendering face view displays of the sensor data, and creating and manipulating geometric objects to denote features of interest in the sensor data space.

In Phase II, ICERVS was developed to Maturity Level IV, Subscale Integrated System, in which ICERVS was demonstrated at limited size to enable successful development of a full-scale system. The result of Phase II was a system that demonstrated enhanced data analysis and visualization tools, software that provided enhanced geometric modeling capabilities, and an integrated user interface. The system was integrated and demonstrated with a structured light mapping system and was also demonstrated using software simulations of other sensor systems.

In Phase III, the objective was to reach Maturity Level V, Full-Scale Integrated System. Phase III involved the addition of enhanced data analysis, visualization, and geometric modeling capabilities and demonstration at DOE sites for UST and D&D applications.

1.3 References

1. U.S. Department of Energy Office of Environmental Restoration and Waste Management Five-Year Plan, January 1993.
2. B. Christensen, B. Griebenow, W. Drotning, and B. Burks. "Graphical Model Based Control of Robotic Systems for Waste Remediation." *Proceedings of the ANS Fifth Topical Meeting on Robotics and Remote Systems*, Knoxville, Tennessee, April 1993.
3. Hebert, M., R. Hoffman, A. Johnson, and J. Osborne. "Sensor-Based Interior Modeling." *Proceedings of the ANS Sixth Topical Meeting on Robotics and Remote Systems*, American Nuclear Society, Monterey, CA, pp. 731-37, 1995.
4. Meagher, D. J. "A New Mathematics for Solids Processing." *Computer Graphics World*, Penwell Publishing Company, October 1984.

2.0 PHASE I: SUBSCALE MAJOR SUBSYSTEMS

The primary objective of Phase I was to develop the ICERVS technologies that had been previously prototyped but not demonstrated as a working subsystem. In this phase, MTI developed a first-generation system for integrated visualization of sensor data and geometric object models with graphical editing of object models. This section describes the Phase I development activities, results, and conclusions. A more detailed description of Phase I is available in MTI 93TR25, "ICERVS Phase I Topical Report," (August 1993).

2.1 Design

The Phase I design activities included identification of the ICERVS mission profiles and system requirements, software design, rapid prototype, and code and unit test, which are described in the following subsections.

2.1.1 Mission Profiles and System Requirements

The ICERVS design was based on a set of mission profiles that were identified and enumerated in conjunction with personnel at DOE field sites and national laboratories. Discussions were held with personnel at:

- Idaho National Engineering Laboratory (INEL) - site of buried waste pits
- ORNL - experience in mapping USTs at Fernald Environmental Management Project and developing robotic systems
- Sandia National Laboratory - experience in the development of structured light sensors, sensor minilab, and robotic control systems
- Hanford Tank Operations - UST site.

Three missions of broad scope were chosen to ensure generality in the design of a suitable architecture: remediation of USTs, buried waste retrieval, and D&D of production facilities. Each mission was described and documented as a sequence of process steps that would be performed to conduct the mission as it is presently conceived. From the mission profiles, a set of software requirements were defined, covering 10 categories (data representation, object modeling, computer graphics display, video display, manipulation and analysis, miscellaneous functions, data interface, operator interface, sensors, and site environment). These requirements were used to guide the software development and were updated throughout the project. (The final version of the ICERVS requirements is summarized in Section 4.1.1. of Part I of this report).

2.1.2 Software Design

The Phase I software was defined as a single Computer Software Configuration Item (CSCI) with three Computer Software Components (CSCs): user interface CSC, octree engine CSC, and object modeling CSC. From the ICERVS requirements, a set of classes

was defined and specified for each CSC. The class descriptions are documented in MTI 93TR24, "ICERVS Subsystem Design Report."

2.1.2.1 User Interface CSC. The user interface CSC encompasses all functions that display data on the monitor and take input from the keyboard and mouse. The user interface is graphical with pull-down menus, windows, and dialog boxes, and relies heavily upon two commercial software products from Rogue-Wave: Tools.h++ and Views.h++ class libraries. The Tools.h++ library provides a broad spectrum of tools such as collections, string and character manipulation functions, date and time handling, file I/O, virtual I/O streams, and virtual arrays. The Views.h++ library provides a complete encapsulation of the Open Software Foundation's Motif graphical user interface and includes tools for constructing menu-based systems.

The user interface CSC is made up of 34 classes organized into 6 groups: utility classes, main window classes, system parameter classes, view window classes, object modeling interface classes, and octree engine interface classes. The utility classes provide a set of common services available to the other software components and are part of the user interface CSC for convenience only. Utility classes include, for example, a text/file browser window, a text editor window, a manager for program configuration files, a filename selection window, an ordered list of string items, and an ordered triplet of values to represent the xyz coordinates of a point in 3-D space.

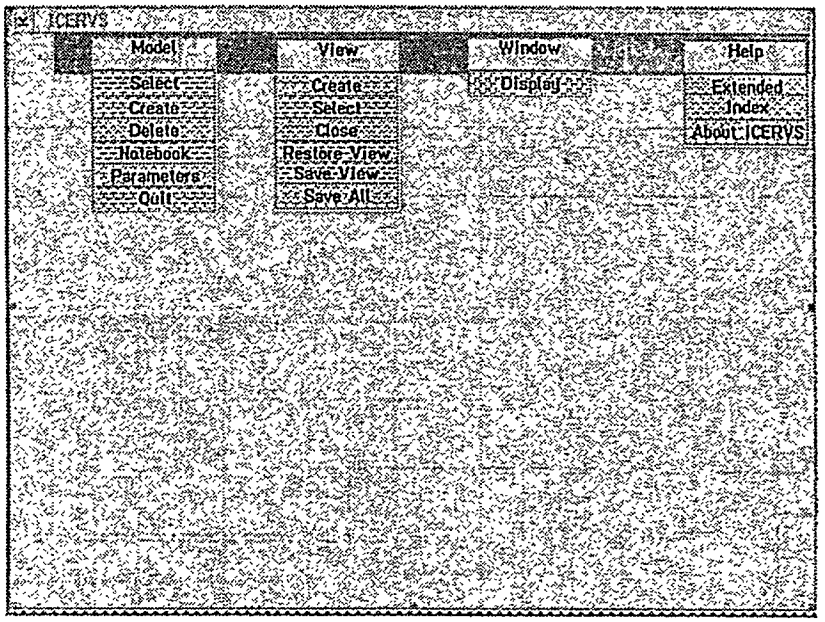
The main window classes provide the top-most program interface to the operator. This set of classes manages the main menus, establishes the workspace context, accepts and dispatches top-level commands, launches the view windows, maintains lists of active windows, and sequences the orderly shutdown of the analysis software. Many of the main window classes are derived from Rogue-Wave Views.h++ classes. The ICERVS main window display and menus, implemented by these classes, are illustrated in Figure 2-1.

The system parameter classes provide control and access to the system parameter and log files. This class enables the operator to select a specific workspace for analysis and also maintains a system logbook file.

The view window classes make up a major portion of the user interface CSC, handling all aspects of presenting the octree and geometric object data stored for a workspace to the operator. The data are presented in image form in a view window, which is a child window of the program main window. Several view windows may exist at one time and display the same or different portions of a workspace. The view window display and menus are illustrated in Figure 2-2. The view window includes scrolling controls for image translation and a slider bar for image scaling. The view window also provides drawing tools to enable the operator to interactively create object models.

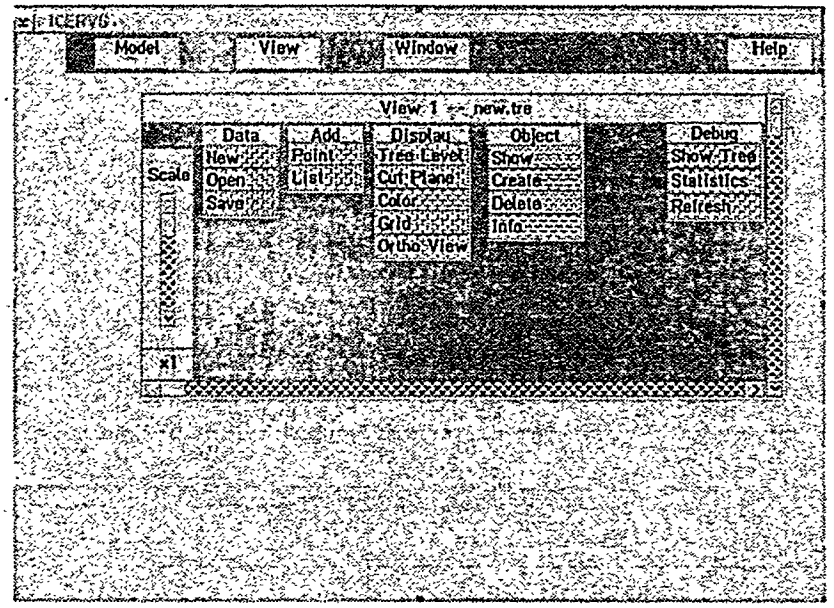
The object modeling interface classes establish and control communications between the user interface and object modeling CSCs. The interface between these CSCs is encapsulated in a single class to provide isolation and promote modularity in the design.

In a similar manner, the octree engine interface classes establish and control communications between the user interface and octree engine CSCs. The interface between these CSCs is also encapsulated in a single class to provide isolation and promote design modularity.



97015.cdr

Figure 2-1. Phase I ICERVS Main Window



97015.cdr

Figure 2-2. Phase I ICERVS View Window

2.1.2.2 Octree Engine CSC. The octree engine CSC encompasses all functions relating to the representation, storage, and management of empirical data for each ICERVS workspace. The data is stored in octree format. The octree engine CSC is made up of 11 classes organized into two groups: tree structure and tree traversal.

The tree structure classes implement the overall structure and management of octrees. This includes the creation and deletion of trees; the creation, deletion, and maintenance of individual tree nodes; scaling of dimensional data between user interface CSC units and tree units; and file input/output of trees. The octree data is structured as a specialized linked list of nodes. The nodes are maintained in a spatially sorted order to enable efficient tree traversal. A tree is generated by creating new nodes as individual data points are added.

The tree traversal classes provide functions for accessing tree data. There is a class that provides a single tree traversal based on depth-first order of nodes. This is used for computing tree statistics (e.g., number of nodes at each tree level), generating the image corresponding to a 2-D quadtree, and printing a tree structure diagram. There is also a dual tree scanner used to generate a quadtree that represents the image of an octree from an operator-prescribed viewpoint. The quadtree image is generated with hidden surface removal.

2.1.2.3 Object Modeling CSC. The object modeling CSC encompasses all functions relating to the representation, storage, and management of 3-D geometric objects generated by the operator for an ICERVS dataset. The object modeling CSC is designed with four classes. One class maintains a collection of geometric objects for the current workspace. A second class is used as an abstract base class for all geometric objects; specific types of geometric models are established by deriving classes from the abstract base class. The third class provides for a set of common parameters (e.g., name, color, creation date/time, and operator ID) assigned to a specific geometric object. The fourth class represents the shape of the geometric object.

In Phase I, geometric objects are limited to constrained prismatic shapes with attached text attributes. The shape, illustrated in Figure 2-3, is characterized by two parallel polygon faces with interconnecting facets. In later project phases, more general versions of this shape were implemented, as were other geometric primitives such as rectangular parallelepipeds, cylinders, and cones.

2.1.3 Rapid Prototype

Octree technology had the lowest maturity level at the start of the ICERVS project and, as such, was considered to be the highest technical risk item. For this reason, the Phase I effort included a software rapid prototype of the octree engine CSC. The rapid prototype established an early confidence level by demonstrating the ability to create trees from point data, store trees in memory and disk, and display tree data.

The rapid prototype was implemented using quadtrees, which are the 2-D equivalent to octrees. Working in two dimensions provided the desired insight and confidence level without the complexity of 3-D geometry. Test data were generated by simulating a set of geometric primitives (circles and lines were used) in space and extracting surface points. Figure 2-4 shows some sample test data, a quadtree structure, and a quadtree display.

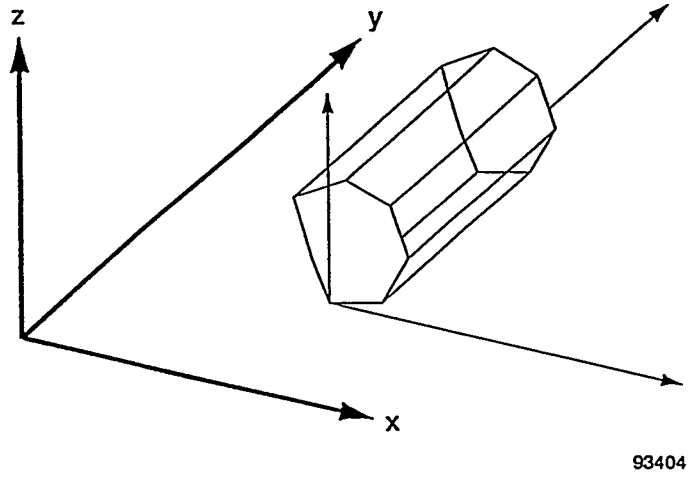
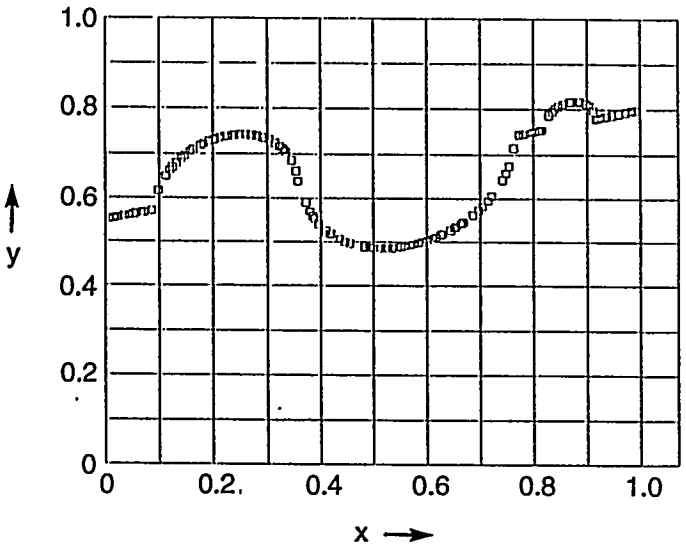


Figure 2-3. Prismatic Shape Used in Phase I ICERVS




```
Total number of nodes in the tree: 68180
    Empty nodes: 21280
    Partial nodes: 17044
    Full nodes: 123
    Unknown nodes: 29733
Maximum level in the tree: 9
Range of property values: 0 - 3
Largest leaf node (level): 2

===Entry Data===
Resolution:      1:512 (i.e. 9 levels)
Max Data Size:  1.0 x 1.0
X-Axis Limit:    0.00 to  1.00
Y-Axis Limit:    0.00 to  1.00
Sculpting is turned: ON

===Node Contents===
 0(0)PPPP
 1(0)  UUUP
 2(3)    UUUP
 3(3)      UUUP
 4(3)        UUPP
 5(2)          UUUP
 6(3)            UUPU
 7(2)              UUPU
 8(2)                FUEU   2 x 2 x
 5(3)          UUPP
 6(2)            UPUP
 7(1)              UUUP
```

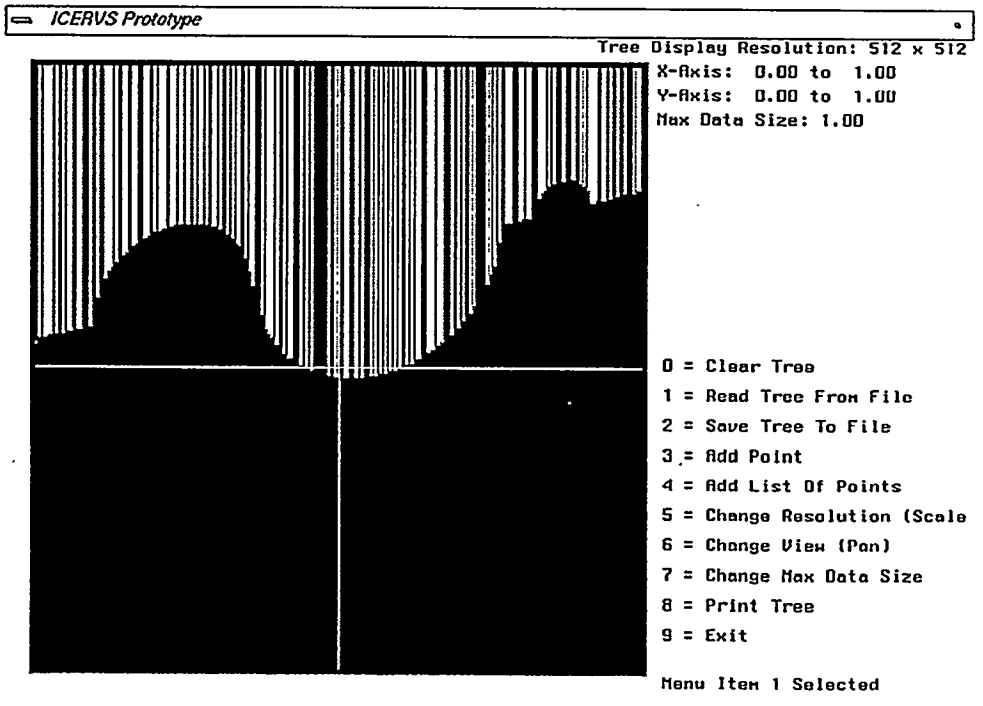


Figure 2-4. Phase I ICERVS Rapid Prototype Test Case

The design of the rapid prototype was done using object-oriented methodology. The basic design is shown in Figure 2-5, which is a class structure diagram using Booch notation. The rapid prototype was made up of 11 classes. The tree structure and geometry are managed by the CQuadArea class, which is derived from the CQuadNode class. CQuadNode contains the CNodeData and CNodeNext classes as member data to make up individual nodes in the tree structure. The CQuadArea class also uses a CSquare class to represent the geographic size and position of the area associated with a node in the tree. The CScaling class provides dimensional scaling between operator units (meter, inch, etc.) and program internal units. Tree traversal functions are provided by the CQuadScan class, which is a base class to CQuadFile and CQuadView. CQuadFile provides functions to read and write trees to disk. CQuadView is used as a base class to CQuadDisplay, which generates an image of the quadtree for display. The Display Window class provides for the display of data on the computer screen, such as was shown in Figure 2-4c. Although it was not originally planned, many of the rapid prototype classes were easily extended to 3-D and reused in the octree engine CSC as part of the final Phase I analysis software.

2.1.4 Code and Unit Test

Source code was written and tested for each of the three Phase I CSCs. All ICERVS source code was written in C++. The user interface CSC was written and tested directly on the computing platform. The octree engine and object modeling CSCs were first written on PC/DOS platforms, then ported to the computing platform. Each class was unit tested prior to CSC integration.

2.2 Test/Results

As described in this section, the Phase I ICERVS was tested to demonstrate its performance. Testing was based on the Phase I success criteria:

- ICERVS will accept, integrate, store, and display dimensional data input in the form of discrete 3-D points. Conversion will be done on a point-by-point basis. Display will consist of three orthogonal face views, with pan and zoom, cut planes, and pseudo-coloring.
- ICERVS will provide interactive synthesis, storage, and display of geometric objects. The Phase I models are restricted to simple prismatic shapes defined by a convex polygon in one view and front/back positions defined in an orthogonal view. Each geometric object may have text data associated with it. Display consists of wireframe images in three orthogonal face views, with pan and zoom.
- ICERVS will provide concurrent display and modeling of dimensional data and geometric objects through a cohesive, integrated user interface. User synthesis of geometric objects to denote features in the dimensional data will be accomplished in a timely and effective manner.

All of the ICERVS tests were completed and a demonstration of the system was presented to DOE in June 1993. All Phase I success criteria were demonstrated, as described in the following paragraphs.

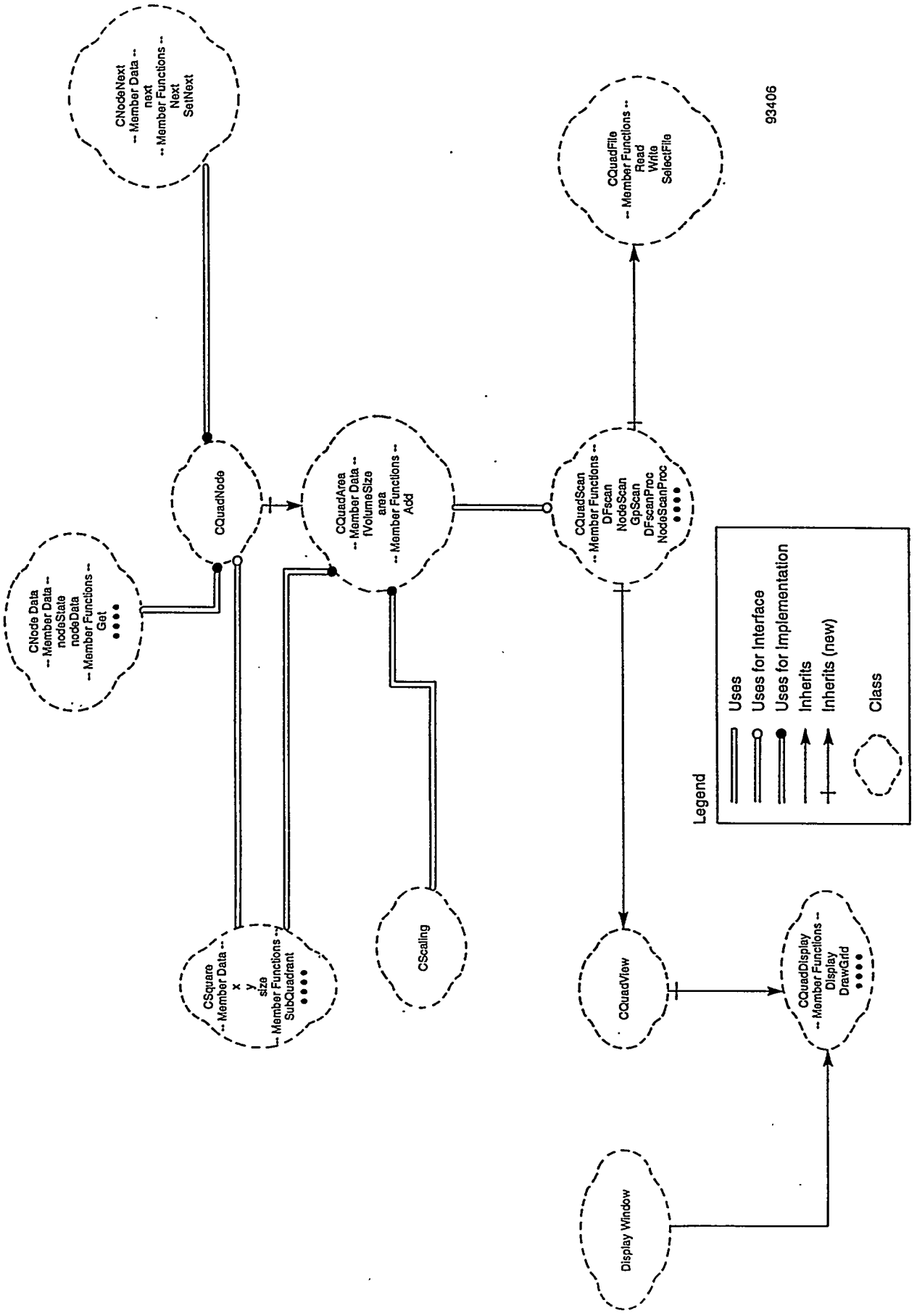


Figure 2-5. Phase I ICERVS Rapid Prototype Design

The first test procedure demonstrated the generation of a tree representation from a collection of input data points. The data points were generated to effect a simulated 2-D representation. A total of 500 data points were generated from an analytic description made from straight-line and circular-arc primitives.

The second test procedure demonstrated the display of an octree generated from an analytic description of a half-cylinder and "bump" feature. Independent pan and zoom, cut planes, and pseudo-coloring features were demonstrated for each view. Pan and zoom were demonstrated by control bars at the periphery of each view window. A pair of cut planes were enabled and disabled by menu command, and manipulated (translated) by mouse control within each view. Pseudo-coloring was enabled/disabled by menu command for each view. Surface data were colored in gray scale according to Z-coordinate (height). The system allows the user to set the range (minimum/maximum Z values) over which coloring is applied.

The third test procedure demonstrated the display of an octree generated from actual sensor data. The data, provided courtesy of ORNL, was measured by a structured light sensor at Fernald Environmental Management Project. The ORNL data were used to generate an octree representing the surface inside a UST before and after the deposition of a 1-ft bentonite cap. The ICERVS cut plane features were used to view sections of the data, showing the bentonite cap and underlying waste surface. A typical screen display is shown in Figure 2-6. The left-hand view window shows a top view of the data with two cut planes set to extract a cross section of the data. A front view of that cross section is shown in the right-hand view window. It was generally agreed that this method of display effectively provided insight to human operators

The final test procedure demonstrated the interactive synthesis of geometric objects to represent features in the dimensional data. Again, the sensor data from ORNL were used. In this case, MTI used only the bentonite sensor data, plus MTI-generated additional data points to "add" two simulated features: a vertical pipe protruding out of the bentonite, and a section of the tank wall with a small anomaly ("bump"). The ICERVS was used to view each feature in detail and interactively generate a geometric object (prismatic shape) to enclose each feature. A third object was generated to enclose the feature observed in the sensor data itself. The resulting screen display is shown in Figure 2-7. Informal timings showed the "average time per object" to be less than 3 min.

2.3 Conclusions

The major conclusions from the Phase I ICERVS results were:

- The Phase I ICERVS met or exceeded all of the success criteria established for this phase of development: integration and display of dimensional data derived from input 3-D points; interactive synthesis and display of geometric objects; and effective operation through an integrated user interface. Based on these results, ICERVS was demonstrated at Maturity Level III, that is, all ICERVS subscale major subsystems were demonstrated.
- The octree technology used in ICERVS provides the basis for a reliable description of a workspace based on input data from empirical measurements. Octrees with thousands of elements or nodes were generated and quickly displayed by ICERVS. Octree size is only limited by the storage capacity of the computing platform.

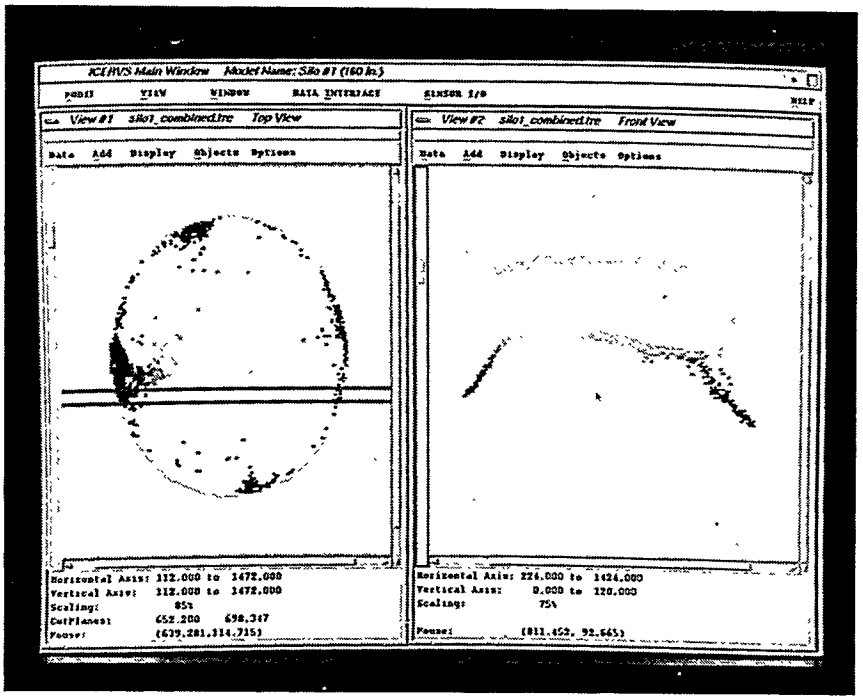
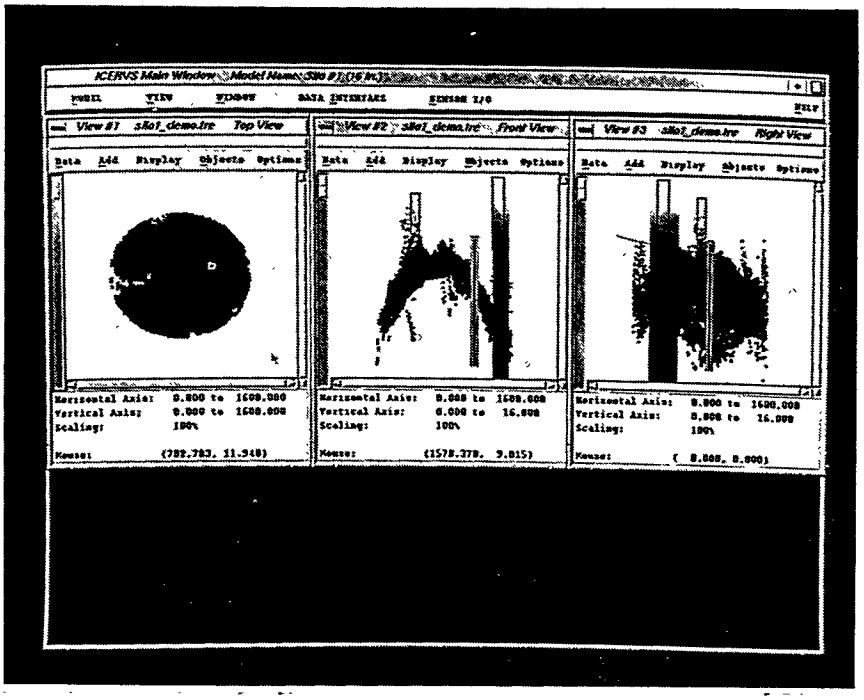


Figure 2-6. Phase I Display of UST Data



V93-166

Figure 2-7. Phase I Display with Objects

- The ICERVS modeling tools provide an effective means for user-interactive synthesis of geometric objects to denote features of interest in a workspace. Successful demonstration of the first level of modeling was shown using prismatic shapes.
- ICERVS provides the integration and display of both empirical- and geometric-based data representations to describe a workspace. This capability of merging empirical and abstract geometric data is an important step toward more productive scene analysis, which will lead to more cost-effective robotic remediation. With ICERVS, operators will be able to compare xyz data from a sensor with surface models of the same region in space and view both sets of data in the same coordinate system.

With the successful demonstration of ICERVS at Maturity Level III (subscale major subsystems) and the general interest indicated by personnel at DOE field sites, it was decided to continue development of ICERVS to reach Maturity Level IV (subscale integrated system) in project Phase II.

3.0 PHASE II: SUBSCALE INTEGRATED SYSTEM

In Phase II, the primary objectives were to further advance the ICERVS visualization and modeling technologies and integrate them with different sensors. This section describes the Phase II development activities, results, and conclusions. A more detailed description of Phase II is available in MTI 95TR1, "ICERVS Phase II Topical Report," (November 1994).

3.1 Design

3.1.1 Mission Profiles and System Requirements

Previously in Phase I, MTI established a set of mission profiles in conjunction with personnel at DOE field sites and national laboratories. In Phase II, MTI increased interaction with DOE personnel in order to develop an improved understanding of the operational and user interface needs associated with using ICERVS in typical environmental restoration and waste management (EM) missions. A number of informal focus groups were held at DOE sites based on their experience and background. From these interactions, a number of important distinctions and revisions were made. For example, MTI learned that at Hanford system operators are not intimidated by sophisticated graphical user interfaces, and, in fact, prefer them to command-line systems. At the Idaho National Engineering Laboratory (INEL), MTI learned that buried waste retrieval will be performed in a sequence of discrete "dig face" steps, and that sensor data interpretation will involve estimating gross features instead of detecting individual objects. At ORNL, MTI learned that sensor-based models are important for providing "camera views" that are not physically available and that tools to improve operator productivity are a primary need. In general, MTI learned that the sensor suite will be configured specific to each remediation site, based on the characterization results and remediation plan. Based on the information obtained from these discussions, MTI revised the mission profiles and system requirements to provide the baseline for Phase II development.

3.1.2 Software Design

In discussing the visualization and modeling needs at DOE field sites, it became evident that ICERVS can be used in applications with differing data requirements and therefore using different sensors, processing algorithms, and data stores. To address these differing remediation environments and needs, the Phase II ICERVS system design was matured to a more open, flexible architecture.

The system configuration implemented in Phase II is shown in Figure 3-1 and consists of seven subsystems. Three of these subsystems, the demonstration, volumetric data, and sensors manager subsystems, jointly provide the visualization and modeling capabilities. The structured light sensor subsystem was developed outside the ICERVS program and loaned to the project in order to support Phase II testing and demonstration. Two simulated sensor subsystems were implemented in order to demonstrate the display of data with scalar material properties, and the remote viewing subsystem was implemented to show how video feedback will be used in conjunction with other sensor data. Table 3-1 summarizes the Phase II ICERVS subsystems.

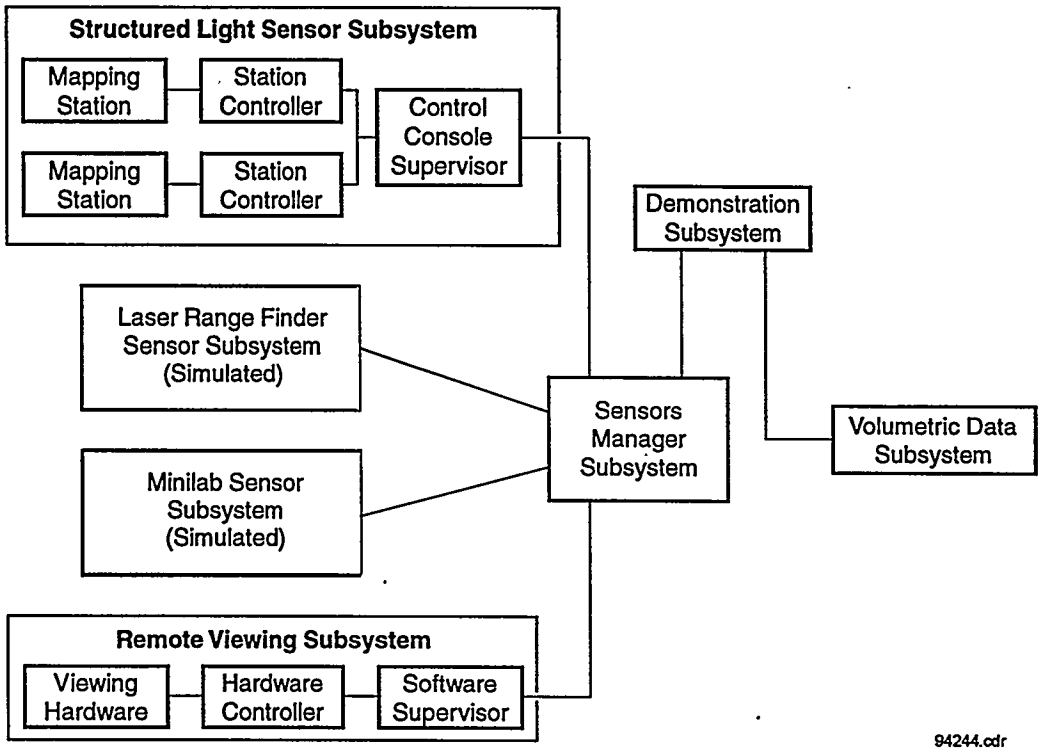


Figure 3-1. Phase II ICERVS Configuration

Table 3-1. Phase II ICERVS Subsystems

Phase II Subsystem	Primary Function	Hardware/Software Configuration
Demonstration	Top-level user access to other subsystems	Software only (SGI platform)
Volumetric Data	Data integration, visualization, analysis, and geometric modeling	Software only (SGI platform)
Sensors Manager	Manages data transfers between sensor subsystems and application/client(s)	Software only (SGI platform)
Structured Light Sensor	Maps interior surfaces of USTs	• Mapping station (hardware) • Station controller hardware and software (PC/DOS platform) • User interface software (Sun platform)
Simulated Laser Range Finder	Generates representative sensor data from files	Software only (SGI or Sun platform)
Simulated Minilab Sensor	Generates representative sensor data from files	Software only (SGI or Sun platform)
Remote Viewing	Provides remote, live video view of workspace	• Viewing camera, lens and pan/tilt • Hardware controller • User interface software (SGI platform)

96TR37

In Phase I, it also became clear that ICERVS will not be used as a stand-alone system, but rather will be integrated into a higher-level, overall remediation system. For that reason, in Phase II MTI matured the original ICERVS design from a monolithic architecture to an open system made up of configurable/modular subsystems. Each of the seven subsystems in Figure 3-1 is an independent Unix process, and communicates externally through client-server connections based on Unix-network sockets. With this design, ICERVS subsystems can be readily integrated and used in higher-level systems. The individual subsystems can be run transparently across a network of Unix workstations.

3.1.2.1 Demonstration Subsystem. In Phase II, the demonstration subsystem provided overall control of ICERVS, serving as a client application to both the volumetric data subsystem and the sensors manager subsystem. The demonstration subsystem is a single CSCI that runs on the SGI platform. In a complete remediation system, the functions of this subsystem would typically be provided by an overall supervisory controller. In Phase II, this subsystem essentially provided the means for coordinating and demonstrating the other ICERVS subsystems.

The demonstration subsystem maintains a set of sites (a site being a distinct remediation workspace such as a UST, buried waste pit, or D&D facility) so that ICERVS can be demonstrated for a variety of remediation applications. The demonstration subsystem provides user commands to open the volumetric data subsystem and sensor subsystems that are on line. It also provides a "connections" capability to support automatic routing of data from one sensor subsystem to a selected data set within the volumetric data subsystem. The demonstration subsystem provides the top-level user interface for the Phase II system. This user interface is a main window with a set of top-level menus as shown in Figure 3-2. The demonstration subsystem also maintains an application data structure, which includes system-level parameter files and a structured set of parameter and data files for each site.

The demonstration subsystem was implemented in 55 MTI software classes, 22 associated with the human interface component and 33 associated with the problem domain component.

3.1.2.2 Volumetric Data Subsystem. The volumetric data subsystem provides the data storage, 3-D visualization and analysis, and geometric modeling functions required by remediation systems. Most of the Phase II visualization and modeling capabilities were implemented using the TrueSolid library developed by Octree Corporation of Cupertino, California. TrueSolid provides a broad range of volumetric modeling capabilities based on a patented octree technology. Benchmark tests run by MTI early in Phase II confirmed that the TrueSolid library provided significant performance and memory advantages over other volumetric software packages. Overall, the volumetric data subsystem is implemented in 75 MTI software classes, 58 associated with the human interface component, and 17 associated with the problem domain component. This subsystem functions as a server process. A key software class in this subsystem is the dataset, which stores the collection of sensor data and geometric objects that characterize a particular site at a discrete time or state.

The volumetric data system provides a variety of features to let users create, manipulate, and analyze ICERVS datasets. Operator functions are provided in a set of graphical user interface windows. These windows provide data visualization, data analysis, and graphical editing of geometric objects. A typical view window is shown in Figure 3-3. The figure shows a topographical map of a UST in Fernald, Ohio. The data was provided by

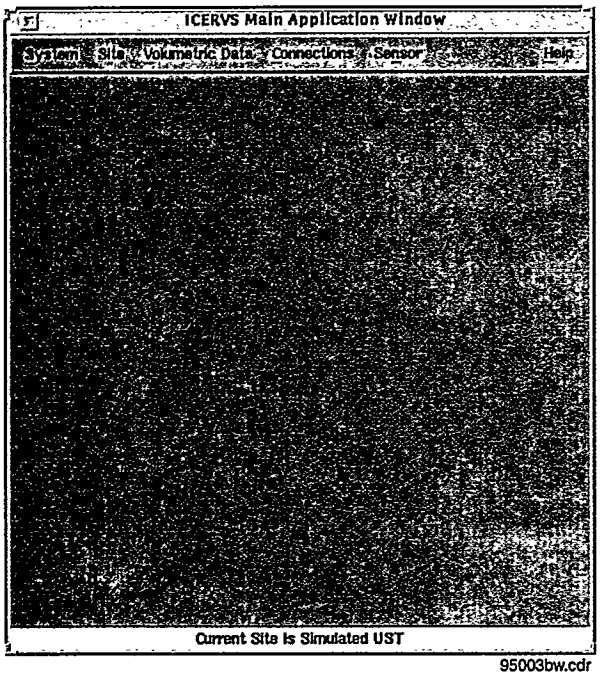


Figure 3-2. Demonstration Subsystem Top-Level User Interface Window

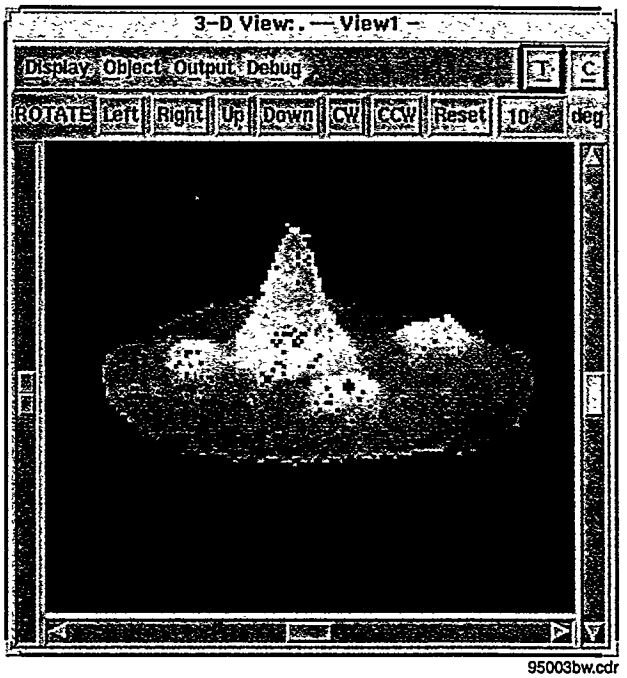


Figure 3-3. Typical Volumetric Data Subsystem View Window

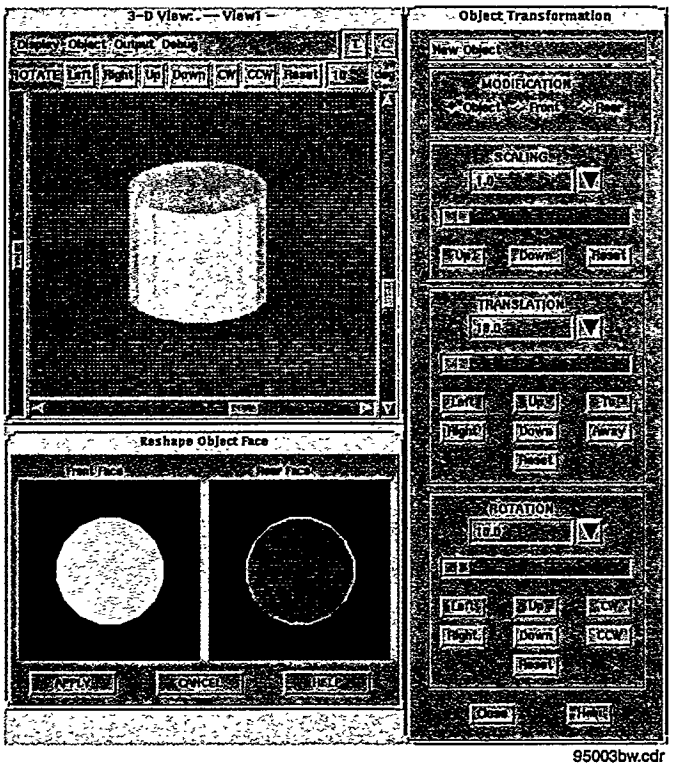


Figure 3-4. Typical Geometric Editing Windows

ORNL engineers. The display options include arbitrary viewpoint (rotation, translation, scaling) and pseudo-coloring of the data based on elevation. MTI implemented a software class to encapsulate the interface to TrueSolid. This software class may have reuse value in other DOE applications that involve volumetric modeling.

Graphical editing of geometric objects was provided by a set of classes built on top of the Rogue-Wave RWCanvas.h++ graphics library. The primary geometric primitive is a prismatic shape, characterized by a front and back "face," each having the same number of vertices, connected by simple facets. A typical set of editing windows is shown in Figure 3-4. The entire object or either (front or back) face can be rotated, translated, or scaled via a control panel. The individual vertices on either face can be interactively edited in separate 2-D windows. Text data, such as name, operator ID, general description, and category can be attached to each object by the user. The set of geometric objects for a dataset can be exported as a file in IGRIP format.

In Phase II, MTI also expanded the dataset capabilities to store scalar material properties such as temperature, elevation, and radiation level. The Phase I design, which provided for a single property value to be associated with each dataset, was expanded to provide up to sixteen user-defined properties. The design was updated so that, for each view window, the user can select a material property for coloring the display.

3.1.2.3 Sensors Manager Subsystem. The sensors manager subsystem provides a link between sensor subsystems and the demonstration subsystem (or other applications). Implemented in software on the SGI platform, this subsystem provides a server interface to the demonstration subsystem and a client interface to each sensor subsystem. It also manages data transfer between sensors and applications, and provides an architecture for preprocessing of sensor data. The sensors manager does not implement a user interface.

To facilitate communications, the sensors manager includes C++ base classes for implementing ICERVS clients and servers. These base classes are used by all ICERVS subsystems. The classes use Ethernet and Unix sockets based on TCP/IP; however, the design is modular to allow other protocols to be used. The base classes provide command formatting and parsing. Overall, the sensors manager subsystem is implemented in 23 MTI software classes.

3.1.2.4 Structured Light Sensor Subsystem. Developed outside the scope of the ICERVS program, the structured light sensor subsystem was jointly developed by MTI, Sandia National Laboratories and ORNL under two CRADA agreements. The purpose of this subsystem is to map the interior surfaces of USTs. A second-generation prototype was first demonstrated in February 1994 and subsequently loaned to the ICERVS project to serve as the primary data source for testing and demonstration in Phase II. The structured light sensor subsystem demonstrated an accuracy of 0.28 in. and an estimated mapping time of 2 hr for a typical tank.

This structured light sensor subsystem includes both hardware and software. As shown previously in Figure 3-1, the subsystem is made up of three main components. The mapping station, which is the hardware that is placed in the tank, is shown in Figure 3-5. It consists of separate laser and camera modules assembled in a long tube that fits down the 4-in. diameter risers in typical tanks. Each module has an optical mirror with an independent tilt motor to provide scanning of tank surfaces. The overall mapping station has a third motor to provide pan motion. The vertical displacement between camera and laser modules establishes the basic geometry for measuring 3-D coordinates by triangulation.

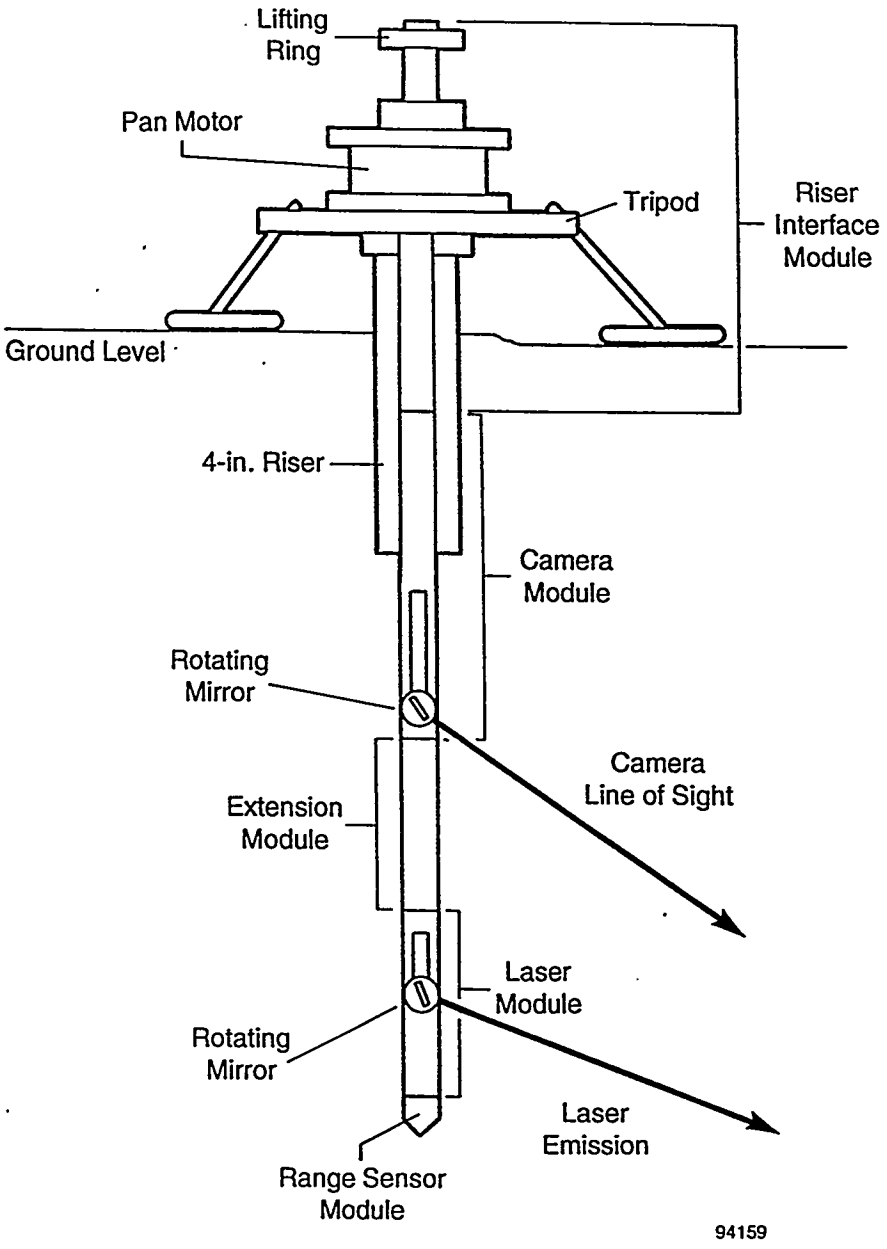


Figure 3-5. Structured Light Sensor Mapping Station

The mapping station is connected by an umbilical cable to the station controller, which consists of a 486/PC, developmental software, and electronics modules to support the mapping station motors, camera, and laser. Included in the 486/PC is a video digitizer and image processor to convert the camera module images of the laser beam into pixel coordinates. The station controller also includes software to convert the pixel coordinates into calibrated xyz coordinates.

The station controller sends its xyz data by Ethernet to a Sun workstation that runs the control console software. The control console provides the main operator interface and map data storage/display. The main user interface, shown in Figure 3-6, presents plan and elevation views of the mapped workspace. The workspace views are colored to indicate the height (z-coordinate) of the measured data in the plan view and distance from the sensor in the elevation view.

To operate in ICERVS, the control console software was upgraded to allow it to be run as a Unix network-server process, using the socket-based classes developed for the sensors manager subsystem.

3.1.2.5 Simulated Laser Range Finder Subsystem. Because the ICERVS design supports diverse sensor types, it was desirable to incorporate multiple sensors in the Phase II demonstration. One sensor that is likely to be used extensively in DOE environmental remediation missions is a laser range finder. These sensors work by modulating a laser beam and scanning it over a surface of interest. The reflected image of the laser is detected, and the measured (relative) amplitude or frequency/phase shift is proportional to the line-of-sight distance. Lacking access to actual laser range finder hardware, MTI implemented software to simulate the behavior of a representative system. This software runs on either the SGI or Sun computing platform and implements a Unix network-server process using the base class developed for the sensors manager subsystem. The software reads data files that were generated from laser range finder measurements and transfers them to the client application under user control. The software provides a simple user interface window to let users enter data filenames and initiate data transfer to the client application.

3.1.2.6 Simulated Minilab Sensor Subsystem. Another sensor subsystem that will be used extensively in DOE environmental remediation missions is the Minilab developed by Sandia National Laboratories. The Minilab is sophisticated data acquisition system that can control a broad range of devices measuring dimensional, physical/geophysical, chemical, and radiological properties. It was desirable to include a Minilab subsystem in Phase II to demonstrate the material property capabilities of ICERVS.

Again, lacking access to or availability of actual hardware, MTI implemented a software subsystem to simulate the behavior of a typical Minilab. This software is very similar to the simulated laser range finder subsystem, and the two subsystems share some common source code. The software runs on either the SGI or Sun computing platform and implements a Unix network-server process using the base class developed for the sensors manager subsystem. It provides the same user interface window to let users enter data filenames and initiate data transfer to the client application.

For Phase II, data files were provided to MTI by INEL. These data files were generated from five experiments performed at the INEL Cold Test Pit, which was constructed for testing buried waste characterization and retrieval strategies. In the experiments, INEL generated a series of dig faces by excavating several feet into the ground and scanning

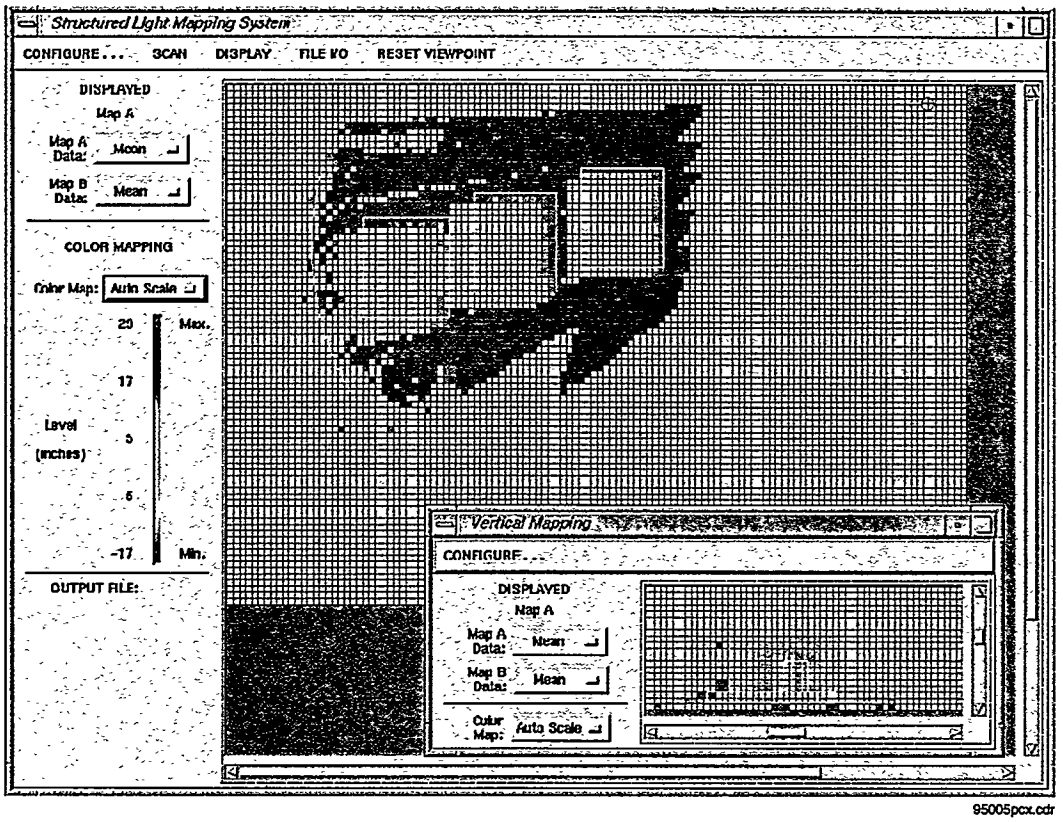


Figure 3-6. Structured Light Sensor User Interface

at different depths. The equipment used include magnetic, electromagnetic and volatile organic compound (VOC) sensors.

3.1.2.7 Remote Viewing Subsystem. The remote viewing subsystem was implemented to demonstrate the expected use of remote cameras in typical remediation systems and their relation to the ICERVS display. This subsystem includes both hardware and software. As shown earlier in Figure 3-1, the remote viewing subsystem has three main elements: the viewing hardware is a commercial video camera with zoom lens and pan/tilt motion; the hardware controller is a commercial electronics package that provides motion control; and the software supervisor is software that runs on the SGI workstation. The software provides simple keyboard and spaceball controls for pan, tilt, and zoom control.

3.2 Test/Results

3.2.1 Unit and Subsystem Testing

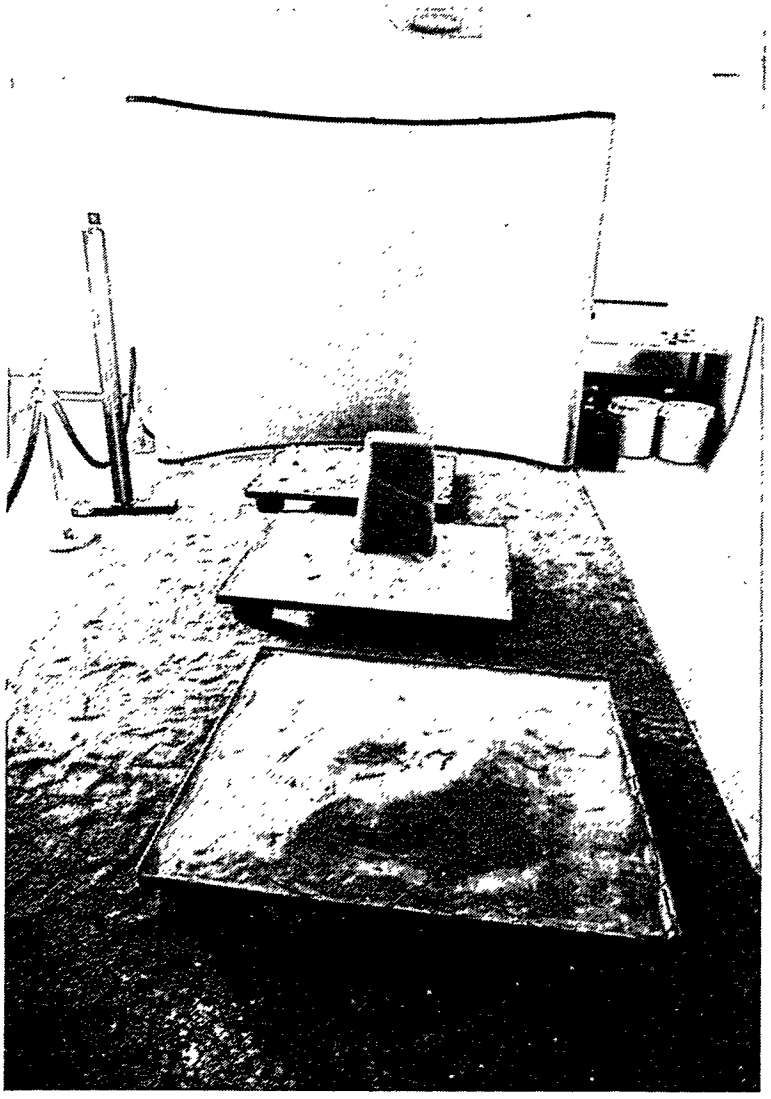
ICERVS testing was performed at the class, subsystem, and system level. More than 200 classes were written in C++ source and unit tested using local test code included in each class source file. Each subsystem was integrated and tested independently. To facilitate subsystem testing, Unix client and server programs were written to connect to a subsystem and provide user control and data access. Overall system integration and testing were completed using the system demonstration plan described below.

3.2.2 ICERVS Phase II Demonstration

The ICERVS Phase II demonstration was held in September 1994. To realistically present the ICERVS features, the system demonstration was divided into three parts portraying remediation activities at a UST, a buried waste site, and a D&D facility. The demonstration took place in MTI's large-scale test facility (see Figure 3-7), which includes the structured light sensor described in Section 3.1.2.4 and a mock-up UST with simulated waste.

The first part of the system demonstration portrayed the use of ICERVS supporting the remediation of a UST. The structured light sensor was used to map pallets of simulated waste and a simulated tank wall. The volumetric data subsystem was used to provide 3-D visualization of the sensor data. Figure 3-8 shows a typical view of the data, which consisted of 20,000 points. The remote viewing subsystem was used to demonstrate how the scene would be viewed remotely, and how the volumetric data display can be made to match the same viewpoint as the remote viewing subsystem. During the demonstration, a portion of the simulated waste was removed and the region remapped to show how ICERVS can be updated as a retrieval process proceeds. Figures 3-9 and 3-10 show a close-up of this region before and after simulated waste removal.

The second part of the system demonstration portrayed the use of ICERVS to support buried waste retrieval. The major focus was material property capabilities. The simulated Minilab sensor was used to supply test data from the INEL dig face characterization experiments, as described in Section 3.1.2.6. The basic procedure was to read in a sequence of dig faces from one experiment. Different material properties were displayed



96P2

Figure 3-7. MTI's Large-Scale Test Facility for ICERVS Demonstration

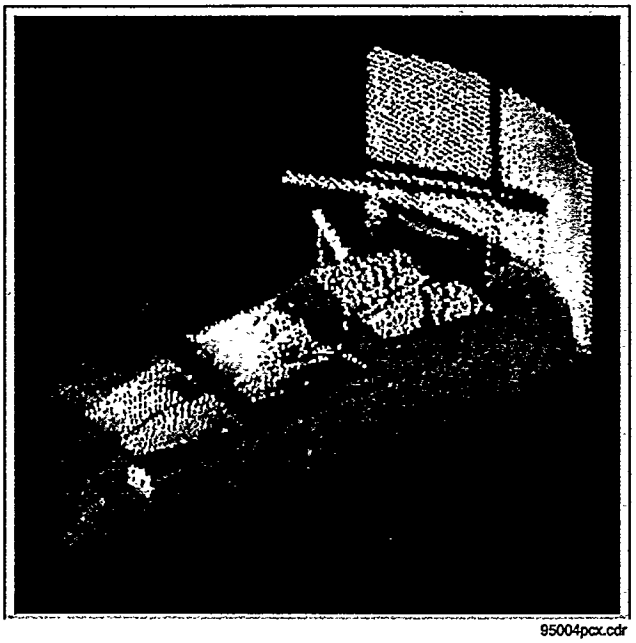


Figure 3-8. MTI Simulated Single-Shell Tank

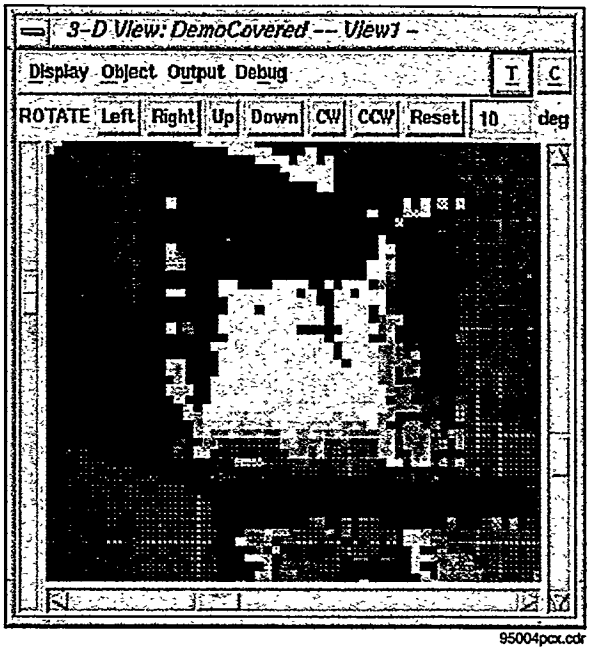


Figure 3-9. Close-up of Bentonite before Removal

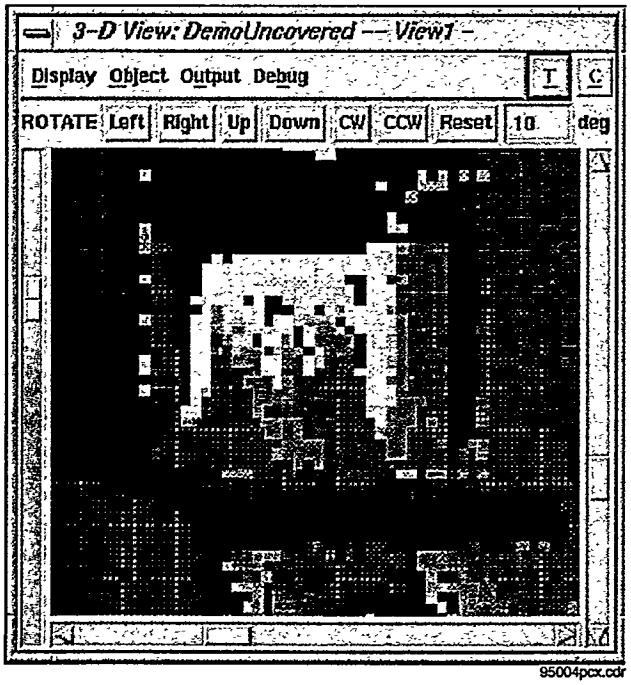


Figure 3-10. Close-up of Bentonite Pallet after Removal

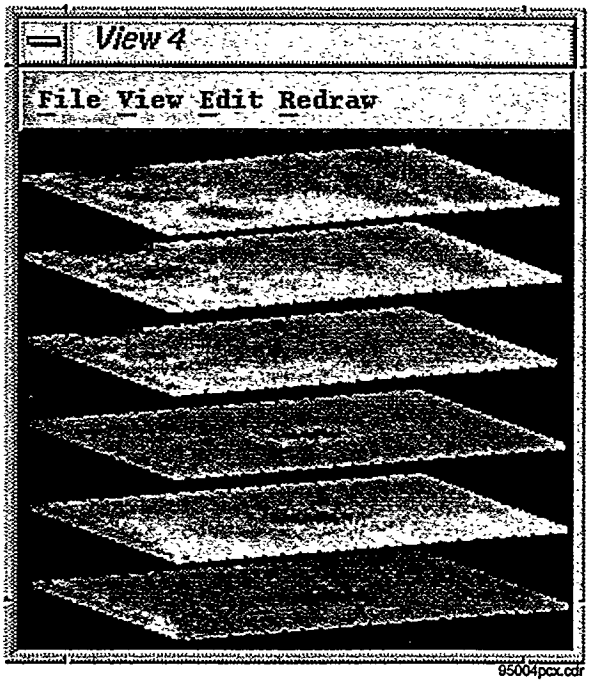


Figure 3-11. Material Property Data at Different Stages of an Excavation

in multiple view. Figure 3-11 shows a typical view of multiple dig faces interpolated and colored by material property.

The third part of the system demonstration portrays the use of ICERVS to support the dismantling of a D&D facility. The simulated laser range finder sensor was used to supply test data from a small pipe assembly. In the volumetric data subsystem, data from different sensor poses were combined into one and geometric objects were created and edited to match features of interest in the data set display. Figure 3-12 shows the sensor data.

During testing and demonstration of the Phase II ICERVS, it was evident that major improvements in usability were needed. The Phase II system met the function-based success criteria, however, image rendering times were excessive and the user interface was difficult to understand. A major redesign was planned and became the main focus of the next phase of ICERVS development.

3.3 Conclusions

The major conclusions from the Phase II ICERVS results were:

- Successful demonstration of the Phase II system showed that ICERVS reached Maturity Level IV, Subscale Integrated System. From a technical perspective, the system was ready to proceed to full-scale development.
- The Phase II system met or exceeded all success criteria established for this phase of development: automatic mapping of a realistic workspace (simulated UST); creation of a volumetric database for dimensional and material property sensor data; creation and editing of geometric models to represent physical objects in the workspace; and correspondence between live video and the computer model of the workspace.
- The octree technology on which ICERVS is based provided the means for the visualization and analysis of general 3-D sensor data such as will be developed in environmental remediation missions. Features such as color mapping and arbitrary viewpoint make ICERVS particularly well suited for data interpretation tasks. Additional development was needed to improve the speed and effectiveness of ICERVS' visualization capabilities.
- The geometric modeling tools, based on generalized prismatic shapes, provide an effective way to generate geometric objects that can be passed to robotic controllers for incorporation into their world model. Further development was needed to make modeling faster and easier for operators.
- The increased interaction with DOE site personnel greatly influenced and improved the design of the Phase II system. This close interaction continued throughout the ICERVS development and greatly benefited ICERVS itself and the EM program.
- The use of commercial software (TrueSolid, Uniras agX/Toolmaster and UIM/X, and the Rogue Wave class libraries) greatly facilitated the software development and improved the quality of the overall system.

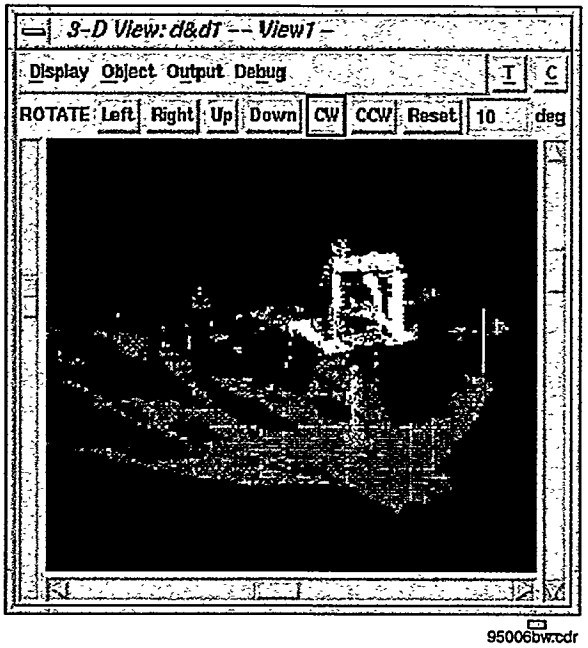


Figure 3-12. Pipe Assembly Display

4.0 PHASE III: FULL-SCALE INTEGRATED SYSTEM

The primary objective of Phase III was to further advance the ICERVS visualization and modeling capabilities to the level of maturity needed for initial deployment in "live" remediation missions, and to demonstrate the ICERVS technology at DOE field sites.

4.1 Design

4.1.1 System Requirements

At the start of Phase III, MTI reviewed the ICERVS requirements and the prior results which indicated that major improvements in usability were needed. The system requirements were revised in Phase III to reflect this need. A number of new requirements were defined, including:

- A dataset manager window to provide the user with a centralized place for managing datasets, with functions such as new, save, copy, delete, etc.
- Object manipulators that let users graphically edit object size, shape, position, and orientation.
- Nonscalar data, which allow users to attach files to spatial points in the dataset to store complex data such as images, frequency spectra, etc
- Immersive viewers that allow users to navigate through the data.

The results of this requirements work, i.e., the final ICERVS requirements/specifications, are summarized in Table 4-1.

4.1.2 Software Design

In designing the Phase III software to incorporate the revised requirements, a key decision was made to adopt the SGI Open Inventor 3-D toolkit for object representation and image rendering. In Phases I and II, image rendering was implemented by direct volume rendering of sensor data and object models. While direct volume rendering consumes significant computational resources, it was the only way to obtain adequate image quality. Open Inventor, which became available in 1994, works in conjunction with the graphics hardware in SGI platforms, and was initially identified as a tool for providing high-speed "previewing" of datasets at limited image fidelity. Through experience and innovation, MTI improved upon the basic capabilities of Open Inventor to the point where image quality was judged sufficient for mainstream ICERVS use. For this reason, the direct volume rendering software used in Phase II was replaced with Open Inventor software.

A second key decision in Phase III was to partition the software development into three sequential releases. This allowed DOE to receive beta releases that could be used for applications prior to the final software release and the opportunity for DOE to evaluate usability issues and continually improve the software design throughout Phase III.

Table 4-1. Summary of Phase III ICERVS Requirements/Specifications

General Specifications	
Computing Platform	Silicon Graphics, IRIX 5.3 or higher
Dataset Manager	Dataset open, new, copy, archive, etc.
Dataset Archiving	Transfers datasets between systems
Multiple System of Units	inch, foot, meter, millimeter
Online Manual	html file
Sensor Data	
Input format	Ascii text file
Spatial (xyz) points	Unlimited number per dataset
Scalar Material Properties	Set of floating point values associated with each spatial point; up to 30 sets per dataset
Nonscalar Material Properties	User-defined data types associated with one spatial point each Up to 64K per dataset External software viewers
Geometric Object Models	
Object models	Unlimited number per dataset
General polyhedral shapes	Unlimited number of vertices and faces
Solid primitives	Box, sphere, cylinder, cone
Composite objects	Multiple objects grouped together
Object library	Stores commonly used shapes
Associated text	Name, annotations, etc.
Visualization and Modeling	
Interactive viewpoint control	Mouse direct, thumbwheel, transform panel
Multiple view windows	Unlimited number
Status window	Parameter display Color legend Rotational orientation cue Selected point coordinate (xyz) display
Cut/Slice planes	5 sets
Multiple viewer types	Immersive (walk, fly) Nonimmersive (orthographic, perspective)
Save/Restore view parameters	Viewpoint, color scheme, cutplanes, etc.
Color and visibility mapping of scalar material property data	Interactive control panel
Indicators for nonscalar data	Colored spheres
Object display attributes	Invisible, wireframe, semi-transparent, solid
Object editing	Resize, reshape, transform manipulators Transform panel Parametric (numerical) editor Snap to xyz point Object duplication
Analysis Functions	
Region selection/highlight	Points enclosed by selected objects
Region erase	Points enclosed by selected objects
Connectivity	From selected (seed) point
IGRIP Part File Interface	IGRIP format version 11 First-order geometry

96TR37

Key parts of the Phase III design are described in the following subsections. More detailed descriptions of the ICERVS design and technical capabilities are provided in Part II of this report.

4.1.2.1 Dataset Disk/File Representation. End user data are stored in *datasets,* which are general collections of data acquired from, or constructed to describe, a 3-D workspace. Dataset contents can include spatial (xyz) data, scalar properties, nonscalar properties, and geometric object models. Datasets can be used, for example, to characterize a specific work environment at a discrete time or state. Datasets can be collected into *groups* so that, for example, the history of a work environment can be stored as a sequence of datasets. Within the Unix file system, groups and datasets are directories. The organization of files within a dataset is shown in Table 4-2.

4.1.2.2 Dataset Manager. To improve usability, a dataset manager was implemented in Phase III. The dataset manager provides high-level functions for controlling datasets and groups. As shown in Figure 4-1, the dataset manager displays a list of groups and list of datasets within a selected group. Within the dataset manager, users can perform functions such as create and delete groups, create and delete datasets, open datasets for viewing, and copy datasets. The dataset manager window opens when ICERVS is started, and includes the quit function for exiting the software.

4.1.2.3 Dataset View Windows. The ICERVS mechanism for dataset visualization and modeling is the view window, an example of which is shown in Figure 4-2. The view window is derived from the Open Inventor "Full Viewer" software class. The image in the view window is rendered by Open Inventor, and the menus above the image, status area below the image, and toolbar buttons on the left margin have been added by MTI to provide the Phase III functionality.

The 3-D visualization capabilities of the ICERVS view window include:

- Each view window can be set to walk or fly viewing for immersive navigation through the data, or to a conventional (through the window) scene viewer with independent rotation, translation, and scale controls.
- Sensor data can be color mapped to a selected scalar property such as elevation or temperature. Similarly, data visibility can be mapped according to the same property. This capability has been used, for example, to prune low-confidence points from large datasets.
- Complex data such as images or frequency spectra can be stored at discrete xyz points and viewed using external application software. To facilitate the use of complex data, specially colored "data indicators" are displayed at each location where complex data are stored. When the user selects a data indicator, it provides a list of data available for viewing at that location.
- Cut and slice planes provide viewing of cross sections through the data.
- The view window includes a status area that displays viewing and dataset parameters, a color legend, and a 3-D icon that mirrors the view's rotational orientation.

With visualization capabilities such as these, robotic system operators can maximize the insights gained from sensors in remote environments. With these insights, then, operators can construct CAD models to match the physical environment. The object modeling capabilities provided by ICERVS view windows include:

- Object primitive shapes such as polyhedral, sphere, and cylinder can be created and edited by on-screen manipulators, transform panels, or parameter editors.

Table 4-2. Dataset Files

Group Name/Dataset Name	Dataset specific files
<colormap>.cmap	Colormap Files
<octreename>.oct	Volumetric Data Files
<octreename>.Shape	Volumetric Data Files
<octreename>.<prop>	Property Files
object.iv	Geometric Object File
indicator.iv	Indicator Object File
dataset.prm	Dataset Parameters File
site.def	Site Definition File
view.prm	View Parameters File
<savedviewname>.vew	Saved View Files
property	Non-scalar property files
xxxxx.gif	image files
xxxxx.txt	text files
.....etc.	.....etc.

96TR37

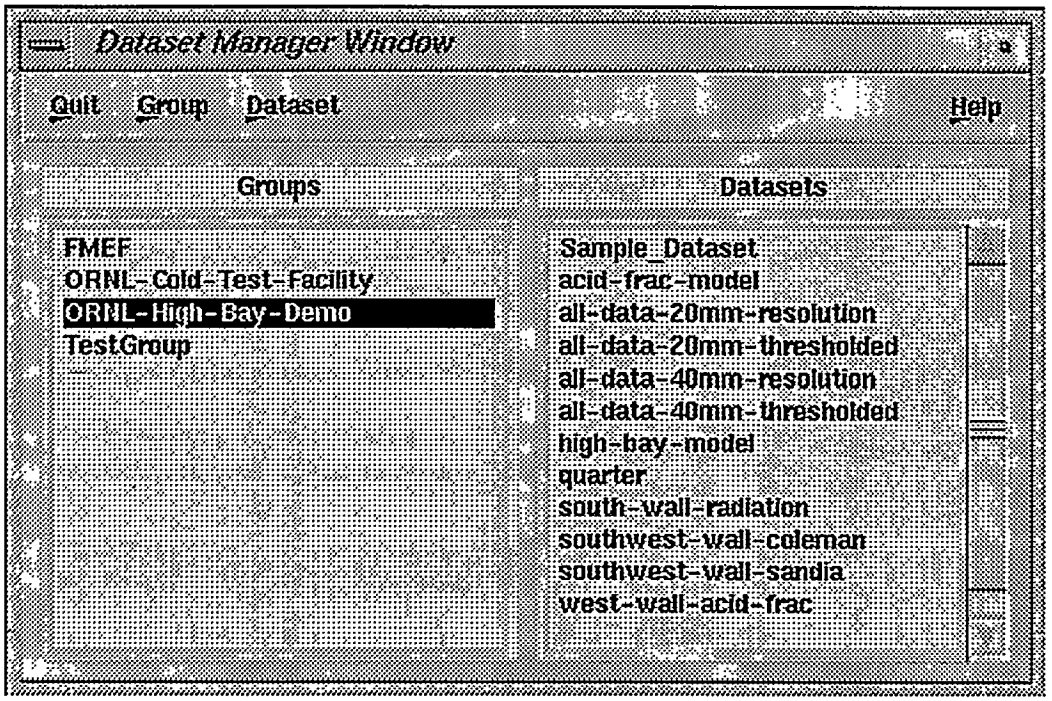


Figure 4-1. Dataset Manager Window

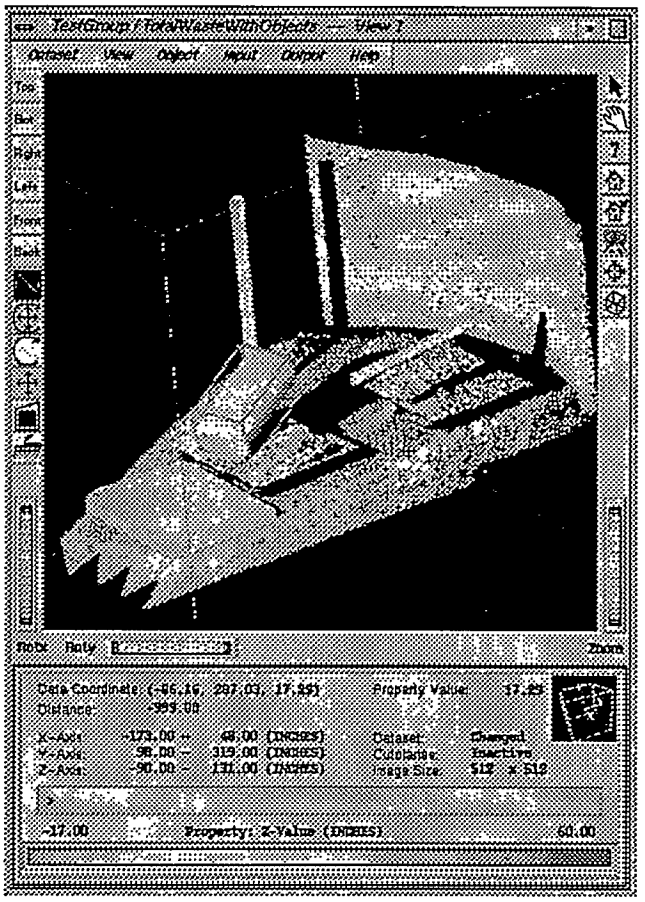


Figure 4-2. ICERVS View Window

- Individual objects can be displayed in one of four forms: invisible, wireframe, shaded, or semi-transparent.
- Objects can be individually colored, so that, for example, walls can be modeled in one color, steel pipes in another, insulated pipes in a third color, waste material in another, etc.
- Multiple object primitives can be grouped and manipulated as a single, composite object.
- ICERVS provides an "object library" for storing objects for reuse.

4.1.2.4 Sensor Data Input. A text file format was defined for inputting sensor data into ICERVS. A short example input file is shown in Figure 4-3. Various headers at the beginning of the file provide descriptive information. The sensor data follow with one data record per line, which can contain up to three types of content: position data, scalar property data, and nonscalar property data. Position data, also referred to as dimensional data or spatial data, consists of the xyz coordinates. Position data are required for each data record and occur in the first three columns. Scalar property data can be used to represent nonspatial data acquired by sensors, for example, temperature, density, or overall radiation level. Up to 30 scalar property values may be attached to each position datum (dimensional point) in the file. Nonscalar data can be used to represent complex or user-defined data types such as images, arrays, or formatted documents. Nonscalar data are input to ICERVS by assigning the filename containing the nonscalar data contents to the position datum.

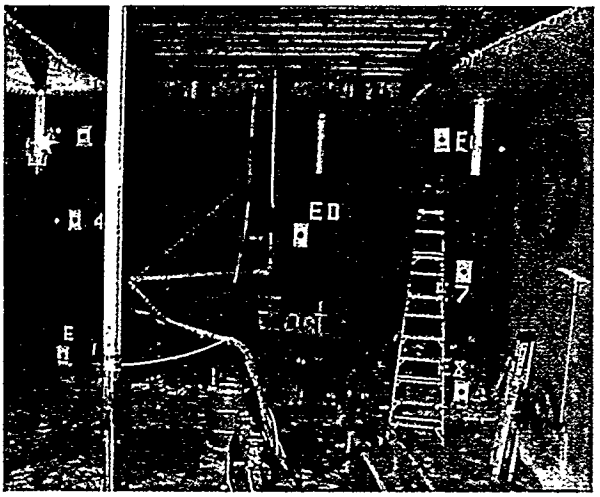
4.1.2.5 IGRIP File Input/Output. To support the interchange of geometric object data with robot control systems, ICERVS provides input and output of IGRIP part files. Because the IGRIP part file format does not support second-order primitives such as spheres and cylinders, ICERVS converts these objects to first-order (polyhedral) approximations prior to writing the output part file. For the same reason, all input objects are converted to the ICERVS polyhedral type.

4.2 Test/Results

Testing and field demonstration of ICERVS was planned and conducted out for both UST and D&D applications. In July 1996, ICERVS was integrated with an MTI-developed Topographical Mapping System (TMS) and used to map a cold test pit at Oak Ridge. Figure 4-4a is a photograph taken from inside the cold test pit that shows the three walls and floor that were mapped by the TMS. Prior to this mapping, a geometric model of the pit, modeled as a set of IGRIP part files, was input to ICERVS to provide a baseline dataset. This IGRIP model is shown in Figure 4-4b, which was captured from an ICERVS view window on the SGI screen. After mapping was complete, the data were input to ICERVS for visualization with both immersive and standard viewers. ICERVS region selection and point removal functions were also used to delete a region where a small number of points were mapped outside the TMS calibration range and were obviously incorrect. Figure 4-4c shows the data in a standard ICERVS view window, and Figure 4-4d shows both sensor data and a CAD (IGRIP) model in an immersive walk viewer. (The sensor data consisted of approximately 50,000 discrete points.) From visual comparison of the sensor data and the IGRIP model, dimensional errors in the original model were found and used to update it. The capabilities demonstrated at ORNL were later presented at the 1996 Robotics Forum in Albuquerque, NM.

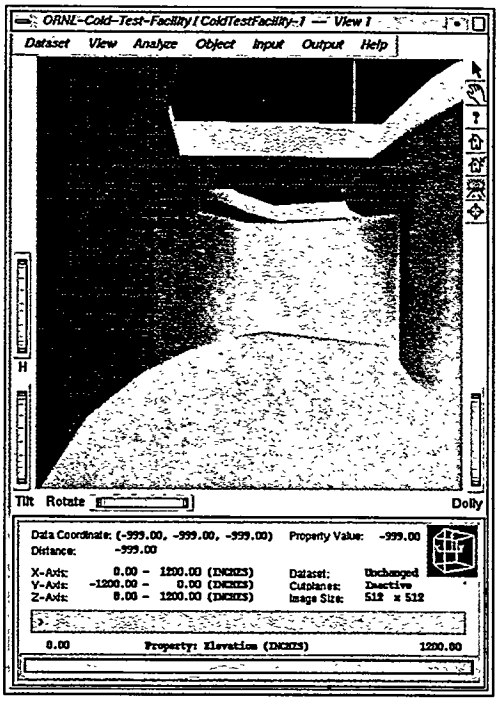
```
*----------------------------------------
* Waste Tank Demo Data
* Mechanical Technology Inc.
* 05/09/96
*----------------------------------------

[ID]
FileVersion= 2.0

[GENERAL]
DimensionalUnits= METERS
Resolution= 0
PathToNonScalars= /usr/local/
NumberOfDataPoints= 5

[PROPERTIES]
NumberOfScalarProperties= 2
Property1= Temperature
Units1= DegC
Property2= Radiation
Units2= Rad

 [DATA]
    -1.9232    3.0362    -1.2427    60.2467    .3946    image.gif    notes.txt
    -1.9386    3.0604    -1.2413    60.2365    .3948
    -1.9558    3.0876    -1.2409    60.2856    .3937
    -1.9739    3.1161    -1.2409    60.2464    .3956    numbers.arr
    -1.9918    3.1443    -1.2406    60.2587    .3963
```

Figure 4-3. ICERVS Input File



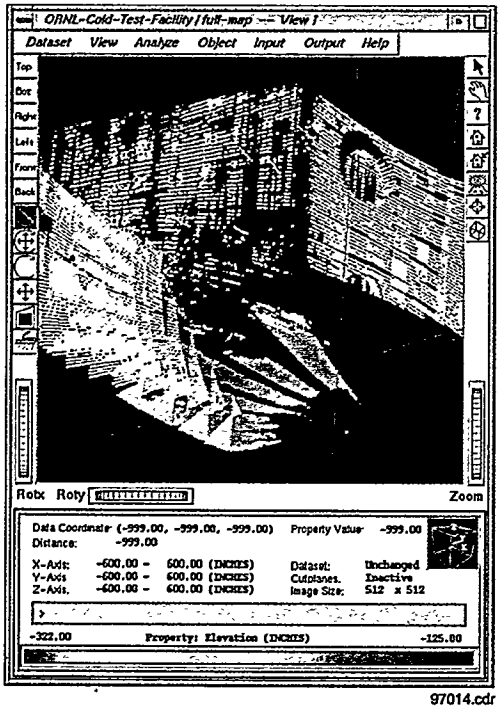
eastwall.tif

(a) Cold Test Pit Maps



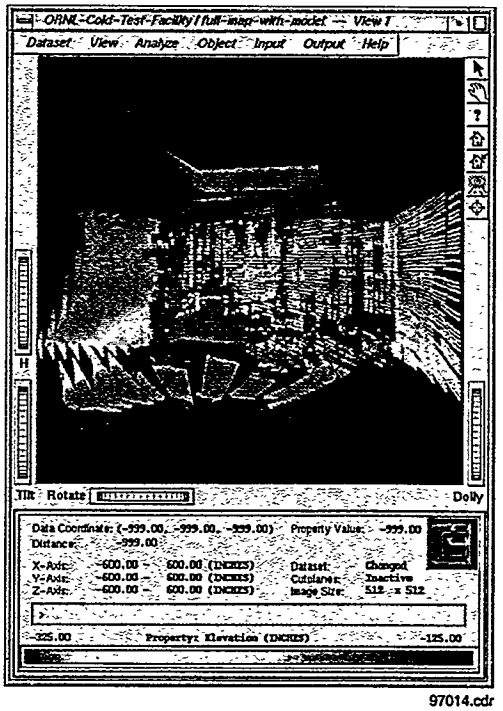
97014.cdr

(b) IGRIP Model



97014.cdr

(c) Data in Standard ICERVS Window



97014.cdr

(d) Sensor Data and IGRIP Model

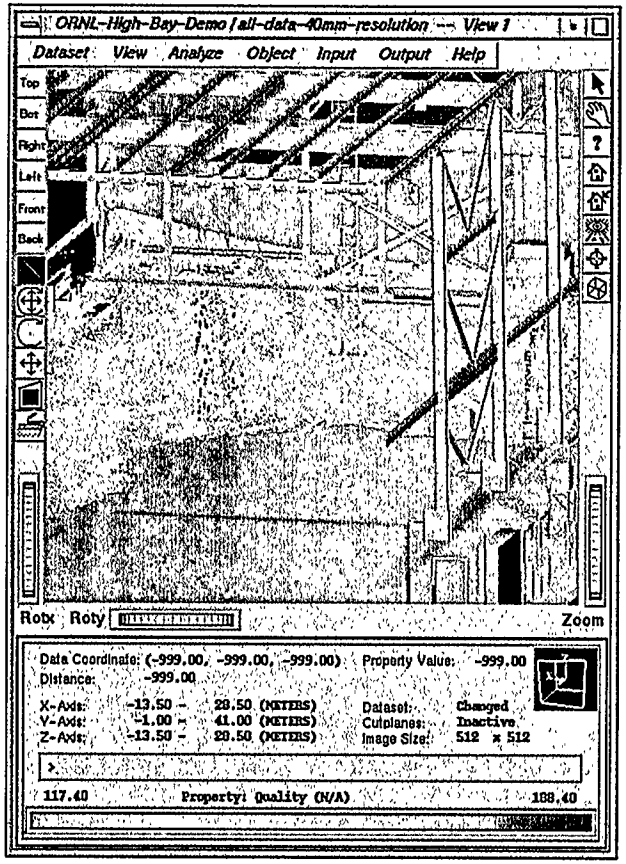
Figure 4-4. Cold Test Pit Maps

In August 1996, MTI presented a demonstration of ICERVS in conjunction with a Mobile Mapper System demonstration at ORNL. For this demonstration, engineers at ORNL had previously mapped three walls in the Robotics Technology Assessment Facility (RTAF), a high-bay area in Building 7601. The data were produced by a laser range finder mounted on a mobile robotic platform. In all, over 500,000 discrete points were mapped. ORNL had also constructed a fairly detailed CAD model of the facility via photogrammetry. The model was formatted as an IGRIP part file and comprised over 20,000 polygons. Both IGRIP model and sensor data were input into an ICERVS dataset. Two view windows from this dataset are shown in Figure 4-5. In Figure 4-5a, all of the laser range finder data are shown, including many points with "low quality" value, which were considered wild points resulting from multipath interference. They are most evident in the corners of the facility as clouds in front of the actual walls. The ICERVS color/visibility mapping capabilities were used to threshold out points with low quality value, and the results are shown in Figure 4-5b. In this view, 25 percent of the data has been removed, providing a more realistic depiction of the facility.

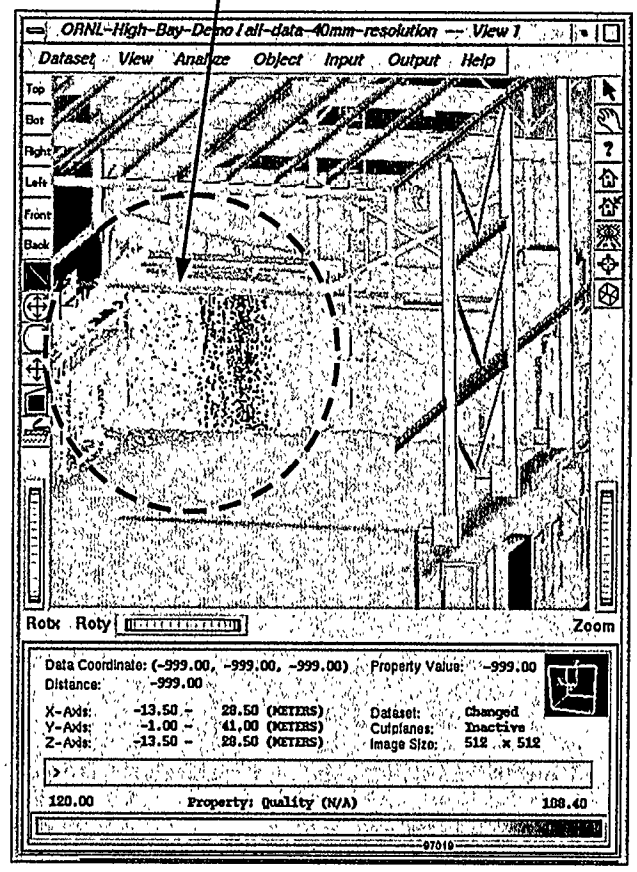
It was further observed that these large datasets, with hundreds of thousands of data points, required considerable computational capabilities and taxed the limits of desktop workstations. Using an SGI Indigo^2 with Extreme Graphics and 128 MB of memory, frame rates of 1Hz were typical, and certain operations such as point selection and data input took considerably longer. To address the huge scale of D&D projects, it is clear that further improvements in computational efficiency are required. Considerable (one or two orders of magnitude) improvement can be obtained by applying known technologies such as spatial partitioning, demand-based file paging, and level of detail. This is recommended as an important area for further ICERVS development.

In addition to the visualization described above, MTI also demonstrated geometric model construction and manipulation at ORNL. Figures 4-6 and 4-7 show some of the results from the south and west wall map data, respectively. Figure 4-6 shows the laser range finder data acquired from the south wall. To demonstrate some of ICERVS' color mapping capabilities, a simulated radiation pattern was superimposed on the data, concentrated on two barrels shown in the lower right-hand side of Figure 4-6a. The other data are colored on a green background level, to indicate that their data is below a threshold radiation level. To facilitate model construction, the data in Figure 4-6a was subsequently color mapped by range (from the viewpoint), and the result is shown in Figure 4-6b. To demonstrate rapid modeling, MTI constructed a number of simple rectangular and cylindrical shapes that are shown in Figures 4-6c (with the sensor data) and 4-6d (objects only). These object models were generated in only a few minutes.

To demonstrate more complex shapes, MTI constructed a model of an "acid fractionator mockup" assembly on the west wall using composite shapes that had been previously constructed and stored in an ICERVS model library. The sensor data, color mapped by range, is shown in Figure 4-7a. MTI constructed several tanks, pipes and elbows, and various structural parts to arrive at the model shown in Figures 4-7b (with the sensor data) and 4-7c (objects only). The parts were retrieved from the library and placed in the dataset in only a few minutes. The resulting model was also exported to an IGRIP part file, which was subsequently added to ORNL's model of the RTAF. This work demonstrated the use of ICERVS to refine CAD models of remote environments so that they accurately represent the "as-is" condition, one of the key capabilities needed for remote robotic operation.

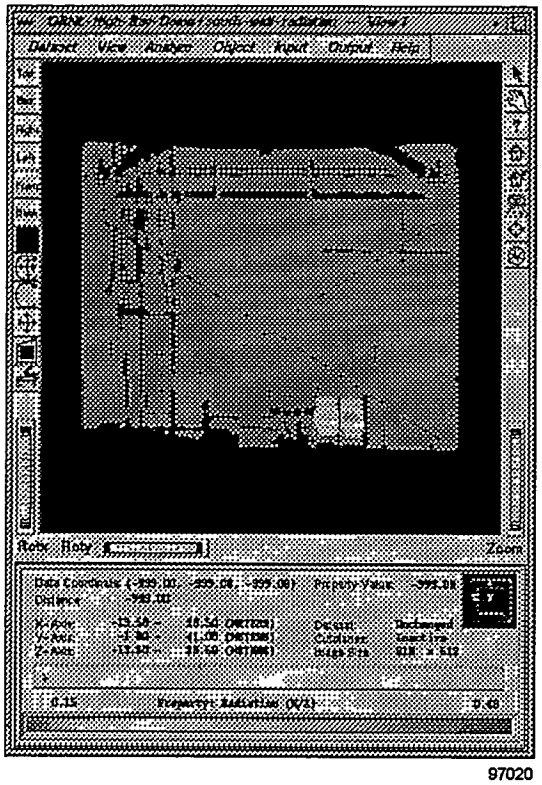


(a) Laser Range Finder Data

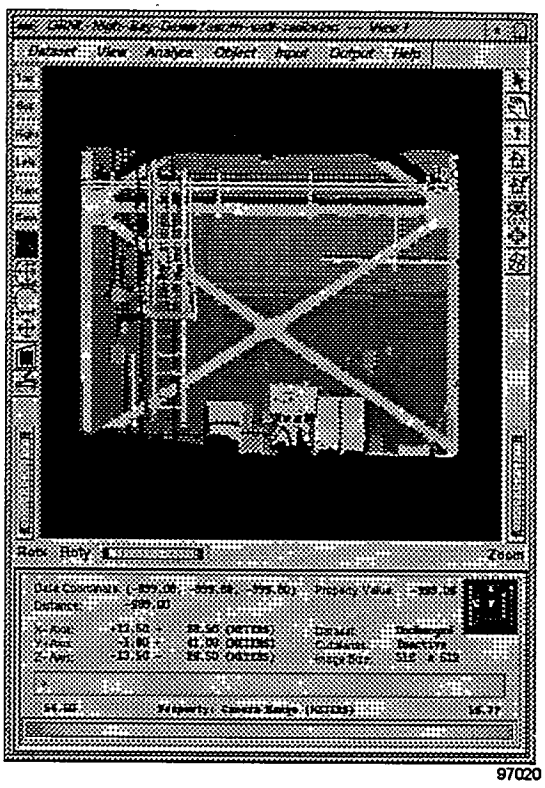


(b) Low Quality Value Points Thresholded Out

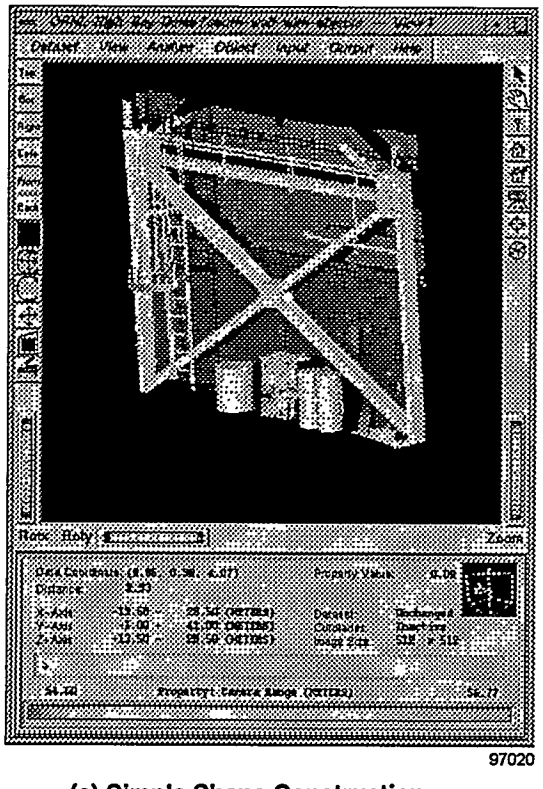
Figure 4-5. Laser Range Finder Data: Two View Windows



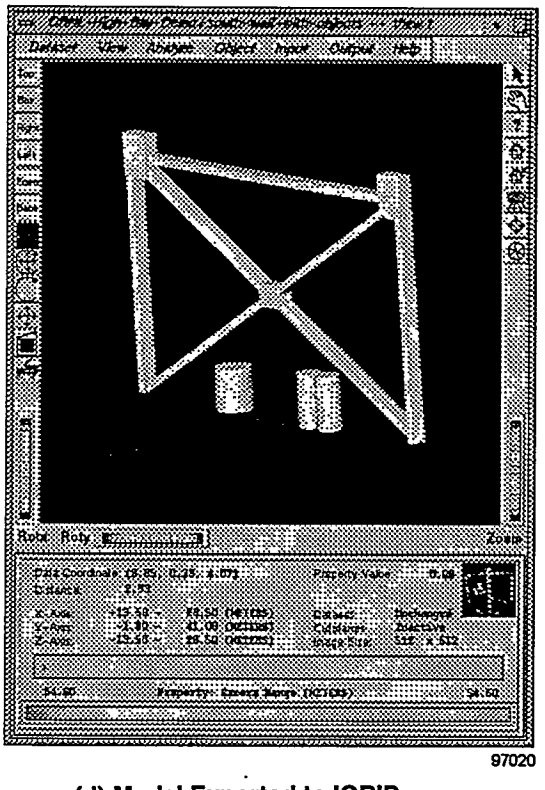
(a) Simulated Radiation Pattern



(b) Data Color Mapped by Range

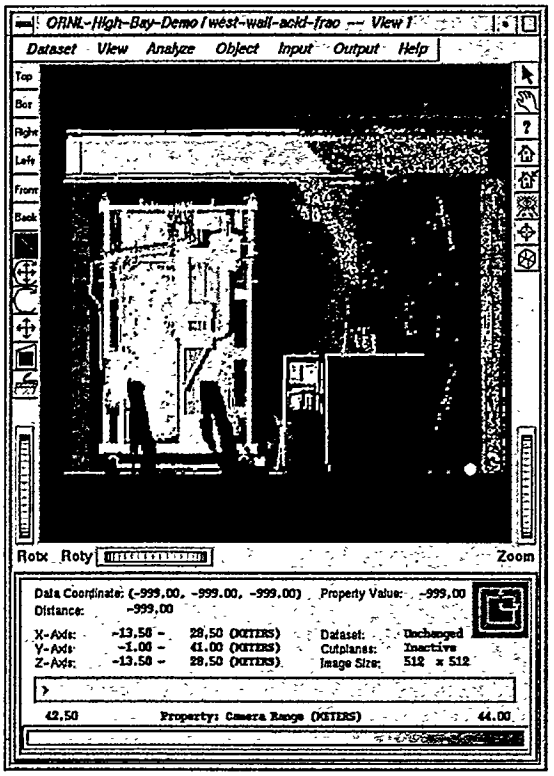


(c) Simple Shape Construction



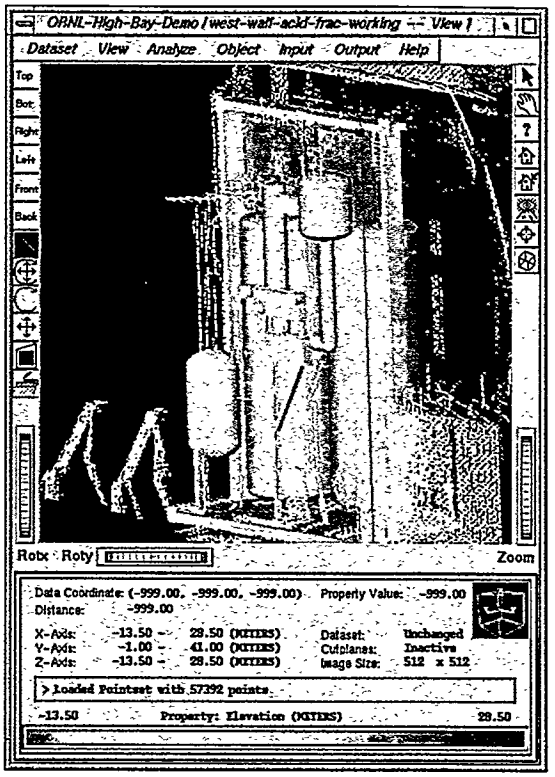
(d) Model Exported to IGRIP

Figure 4-6. South Wall Map Data



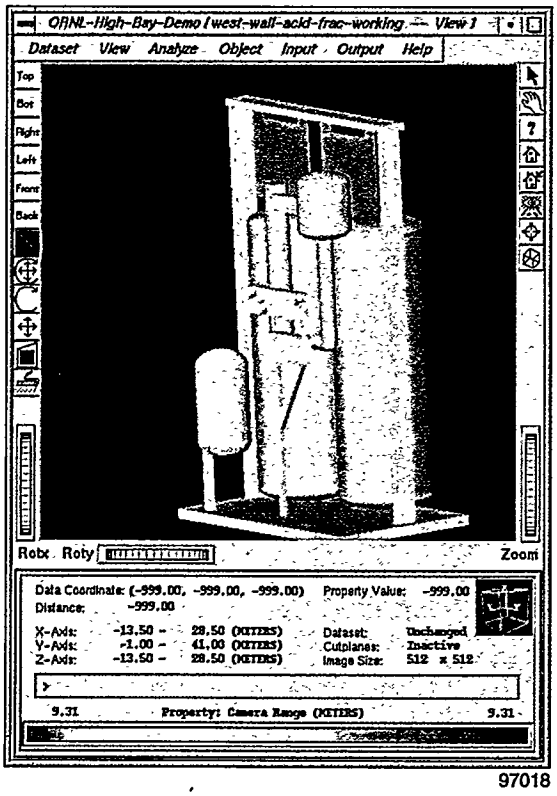
97018

(a) Sensor Data Color Mapped by Range



97018

(b) Model with Sensor Data



97018

(c) Model with Objects Only

Figure 4-7. Model for a Fractionator Mockup

4.3 Conclusions

The major conclusions from the Phase III ICERVS development are:

- Successful demonstration of ICERVS has shown that the technology has reached development Maturity Level V, Full-Scale Integrated System. The software is ready for use in "live" applications where remote robotics are needed for environmental cleanup.
- The architectural changes made in Phase III were successful in achieving high-speed display and interaction for datasets with more that 100,000 points.
- The Phase III system met or exceeded all success criteria established for this phase of development: (1) ICERVS effectively conveys to end users the 3-D characteristics of remote workspaces and other sensor-provided characteristics of workspaces; (2) ICERVS provides the functionality required to produce 3-D CAD models of features in mapped workspaces; (3) ICERVS seamlessly integrates with external sensor systems and model-based robot controllers.
- The integration of interactive visualization and modeling capabilities remains an important part of remote, robotic systems for environmental remediation. With ICERVS, system operators will be better equipped to manage the formidable complexities of task and path planning in remote environments, and construct world models that reflect the best information available.
- Continued development of ICERVS technologies will lead to even greater payback to the DOE in the future. Such development will add registration and data fusion capabilities to ICERVS in order to combine data for minimal error.

5.0 PROJECT CONCLUSIONS

Because of the unstructured and hostile nature of many environmental cleanup sites, new robotic control strategies are being developed that will enable remediation work to be done remotely, so that operational personnel can be kept in safe environments. These robotic control strategies critically rely on 3-D sensors to accurately map remote scenes and require the construction of 3-D CAD or world models to match the remote environment. To bridge the present technology gap between sensor data and CAD model representation, MTI has developed ICERVS, a software program that provides integrated, interactive 3-D visualization and modeling of sensor and CAD data. With ICERVS, operators can perform immersive, virtual walkthroughs of the available data and thereby effectively perceive the full spatial nature of the remote site. With ICERVS, they can also create and/or edit CAD object models to represent physical features in the remote site. The resulting geometric models can be sent to downstream systems and used for task and path planning, where the CAD models enable automatic collision avoidance.

The new capabilities provided by ICERVS will directly lead to improved cleanup operations across the DOE nuclear weapons complex, with potential benefits in UST, buried waste, and D&D applications. ICERVS' interactive visualization of sensor and CAD data will enable *better* operator understanding of remote work sites, leading to *faster* assessments of site characteristics and development of remediation strategies and tactics. By providing CAD models that reflect the best information available from remote site, ICERVS will also lead to *safer* robotic cleanup operations through model-based robotic control. All of these benefits will ultimately contribute to *cheaper* cleanup of the weapons complex, making ICERVS a key technology in the DOE's arsenal for environmental stewardship.

Part II
Technical Overview

1.0 INTRODUCTION

Part I of this report presents an overview of the work performed during the ICERVS development project. In Part I, Sections 2.0, 3.0, and 4.0 summarize the Phases I, II, and III activities, respectively, and Section 5.0 presents the project conclusions. Part II of this report presents a technical overview of the final ICERVS design and implementation, as completed in Phase III. In Part II, details are provided on the system design (Section 2.0), ICERVS dataset (Section 3.0), dataset manager (Section 4.0), dataset view (Section 5.0), sensor data input (Section 6.0), geometric objects (Section 7.0), analysis features (Section 8.0), IGRIP interface (Section 9.0), and major class descriptions (Section 10.0). A glossary of terms is also provided for reference in Section 11.0.

As described in Part I, ICERVS is computer software that combines two different data modalities — sensor data and CAD models — into a common computing environment for interactive visualization and modeling. This capability enables robotic system operators, for example, to acquire 3-D sensor data from a remote environment and use that data to construct and/or verify CAD or computer "world" models of the environment. Such capability will reduce the exposure of human operators to hazardous, contaminated, and radioactive environments, while improving the productivity with which robotic tasks can be performed.

2.0 SYSTEM DESIGN

The ICERVS software provides storage, retrieval, manipulation, and visualization of spatial, property, and geometric object data. It is implemented as a UNIX process with an X Windows user interface, and 3-D visualization and modeling using Open Inventor. The design is grouped into a main module and 5 main objects: Dataset Manager, Dataset, Dataset Viewer, View, and Render Model. The main program initializes the client server connection, sets up the global parameters, invokes the Dataset Manager Window, and starts the main event loop. See Figure 2-1 for an ICERVS object block diagram.

The Dataset Manager Object manages groups and datasets created by the operator. It is the "Database Manager" of ICERVS and handles all user interfaces to the database. There is only one Dataset Manager object in the system.

The Dataset Objects represent the various sets of data that have been input into the ICERVS. They are the "Database" of the ICERVS and handle all interfaces to the data itself. Datasets can include spatial data and property data stored in the form of octrees, non-scalar data stored in user-defined format, and geometric object data stored in Open Inventor file format. There is one Dataset object for each operator-selected group/dataset request from the Dataset Manager object.

The Dataset Viewer Objects interface the Dataset objects to the various View objects which are viewing the dataset. They contain all elements which are shared among all views of the dataset. There can be many Dataset Viewer objects in the system. There is one Dataset Viewer object for each Dataset object.

The View Objects represent the views of the data in the datasets. They contain all elements which are unique to each view of the dataset, and handle all user interfaces to the views. The various View Window classes within this object implement the various behaviors of a view. There is one View object in the system for each Dataset and Dataset Viewer object pair.

The Render Model Objects represent the rendering of the data in the datasets They contain all elements necessary for rendering the data and handle all interfaces to Open Inventor and TrueSolid. In effect, they allow visualization of the Dataset objects in the View objects. There is one Render Model object in the system for each Dataset and Dataset Viewer object pair.

2.1 Computing Platform

The following sections describe the ICERVS computing platform.

2.1.1 Hardware

The ICERVS software executes on a Silicon Graphics (SGI) platform. The system should be (minimally) configured with 32-MB RAM, 1-GB disk, and 128-MB swap space.

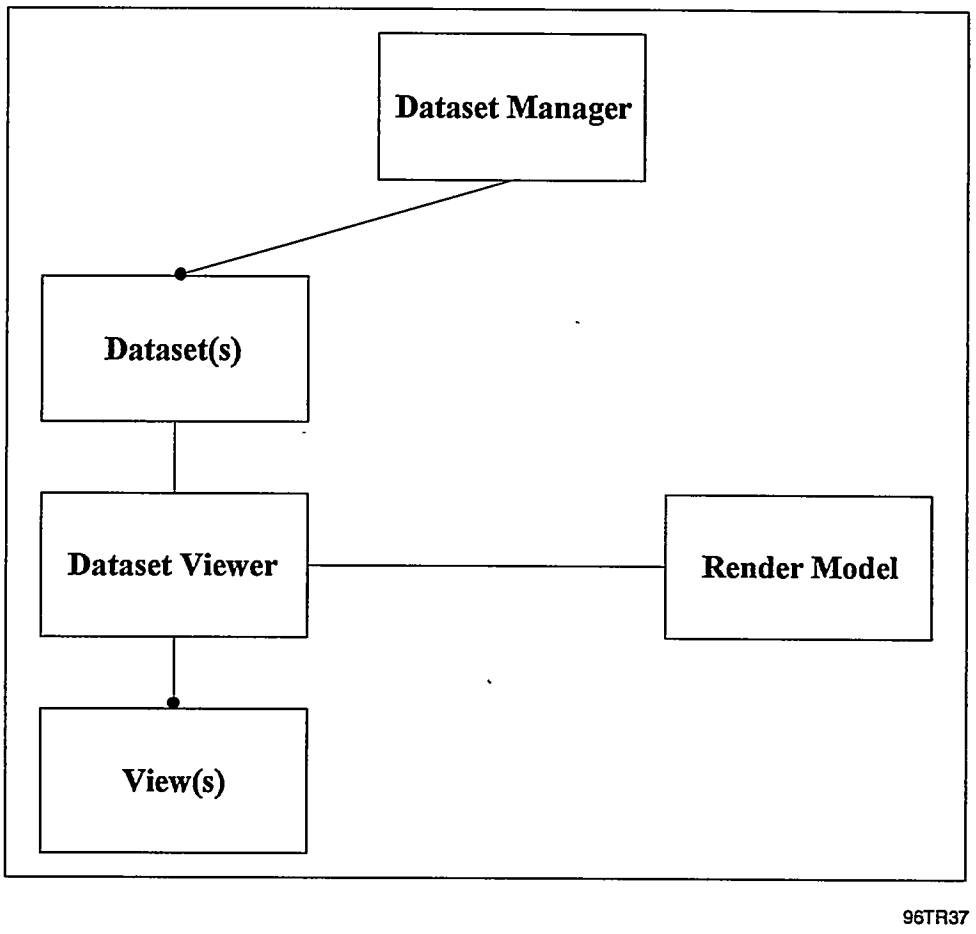


Figure 2-1. ICERVS Object Block Diagram

2.1.2 Software

The system requires the following software products be installed on the SGI workstation for system execution:

Operating System: IRIX Version 5.3 (or higher). Included as extensions of the operating system are X Windows and Motif, which are used to provide a robust interface for creating and manipulating windows.

Open GL: A graphics language, provided by SGI, that enables fast and efficient graphic rendering.

Open Inventor (Version 2.1): An SGI 3-D graphics toolkit, built upon *Open GL,* used for rendering and manipulating data and geometric objects.

2.1.3 Licenses

The ICERVS requires the following software licenses to be installed on the SGI workstation for system execution:

TrueSolid: Octree Corporation's "volumetric" solid modeler software package.

For information about TrueSolid contact:

Octree Corporation
7337 Bollinger Rd.
Cupertino, CA 95014
(408)257-9013
http://www.octree.com

2.2 ICERVS System Directories

The ICERVS directory structure is organized with two main directory paths: the $ICERVSROOT path and the $ICERVSDATA path which are specified by environment variables. The $ICERVSROOT path contains system-wide files such as system default files and object library files. These files come installed with the system. The $ICERVSDATA path contains end user data which is organized into group and dataset directories (see Section 3.1). The general organization and contents for the system directory structure is shown in Table 2-1.

2.3 Main Execution Control

Before starting an ICERVS session, the $ICERVSROOT and $ICERVSDATA environment variables must be set. The $ICERVSROOT variable should be set to the ICERVS root directory which was created during installation. The $ICERVSDATA variable should be set to the path at which the groups of datasets are to be located. This path could be the same for all users on a system, or each user may have their own ICERVS group/dataset directories.

Table 2-1. ICERVS Directory Structure

$ICERVSROOT	Root of Static ICERVS Data Structure
bin	Contains executable programs
etc	Contains templates and general files
config.vds	Program configuration file
view.prm	Default View Parameters File
site.def	Default Site Definition File
inset.iv	View orientation template
mail.cap	Metamail configuration file
mime.types	Metamail file
send-bug	Script for sending bug reports
start-help	Script for displaying on-line user manual
lib/icervs/objectlib	Contains Object Library files
LIBRARY1	Contains library-specific objects for 1st library
<object1>.iv	Library object file #1
.	
<objectn>.iv	Library object file #n
LIBRARYn	Contains library-specific objects for nth library
<object1>.iv	Library object file #1
.	
<objectn>.iv	Library object file #n
lib/icervs/help	Contains User Manual files in HTML format
<name>.htm	html file of manual text
<name>.gif	image files
lib/icervs/bitmaps	Contains bitmap files
<name>.xbm	bitmap file
share/icervs	Extraneous files

96TR37

2.4 Session Startup

An ICERVS session may be started by typing:

$ICERSVROOT/bin/vds

or, if $ICERVSROOT/bin is in the path, type:

vds

The system will respond by starting the Dataset Manager window described in Section 3. Since the initial system is installed with no existing Groups or Datasets, the User must first create a Group in order to get started. Once a Group is created, the User may create a new empty Dataset, or the User may retrieve an archived dataset from the $ICERVSROOT/share/icervs directory.

2.5 Session Termination

An active ICERVS session may be terminated by selecting the *Quit* option from the Dataset Manager menu bar.

3.0 ICERVS DATASET

An ICERVS dataset stores the collection of data (volumetric, scalar property, non-scalar property, geometric) that characterizes a particular physical site at a discrete time or state. A dataset is a subset of a group which represents a workspace. An example of a group might be an individual waste storage tank. An example of a dataset might be data taken for the initial state of the waste tank before any excavation was performed. For each group, there can be unlimited number of datasets.

A dataset is stored on disk as a set of files in a dataset directory (e.g., *$ICERVSDATA/Group1/Dataset1).* When the first view on a dataset is created, the dataset is read from disk and stored in memory. See Figure 3-1 for a description of the dataset representation. The highlighted symbols on the figure represent the dataset files and memory images. All other symbols represent dataset objects (groups, datasets, views).

The dataset is represented on disk as a set of files in a dataset directory. These files include: the volumetric data file (octree files: xxxx.oct and xxxx.Shape), the scalar property files (xxxx.<property name>), the non-scalar property files, the geometric objects file (object.iv), the indicator objects file (indicator.iv), and the dataset parameters file (dataset.prm). These files are highlighted in Figure 3-2. The following sections describe the main elements of a dataset on disk:

- The octree files are TrueSolid format files which are read and written by TrueSolid. They contain a direct translation of the memory version of the volumetric and property data. They also contain indexes of the non-scalar property data for each spatial data point contained in the octree.
- The non-scalar property data are a set of files containing any type of information. An index entry in the octree is used to link a spatial point to the actual sets of non-scalar property data files.
- The geometric objects file is an Inventor format file which is read and written by Inventor and is used to build a memory-resident Inventor tree representing the geometric objects in a dataset. It contains a direct translation of the memory version of the geometric objects.
- The indicator objects file is an Inventor format file which is used to build a memory-resident Inventor tree representing the points that are linked with non-scalar property data.
- The dataset parameter file contains a list of all the scalar properties maintained for the dataset.

3.1 Directory Structure

The ICERVS directory structure is organized with two main directory paths: the $ICERVSROOT path and the $ICERVSDATA path which are specified by environment variables. The $ICERVSROOT path contains system-wide files such as system default files and object library files. These files come installed with the system. (See Section 2.2) The $ICERVSDATA path contains end-user data which is organized into group and dataset

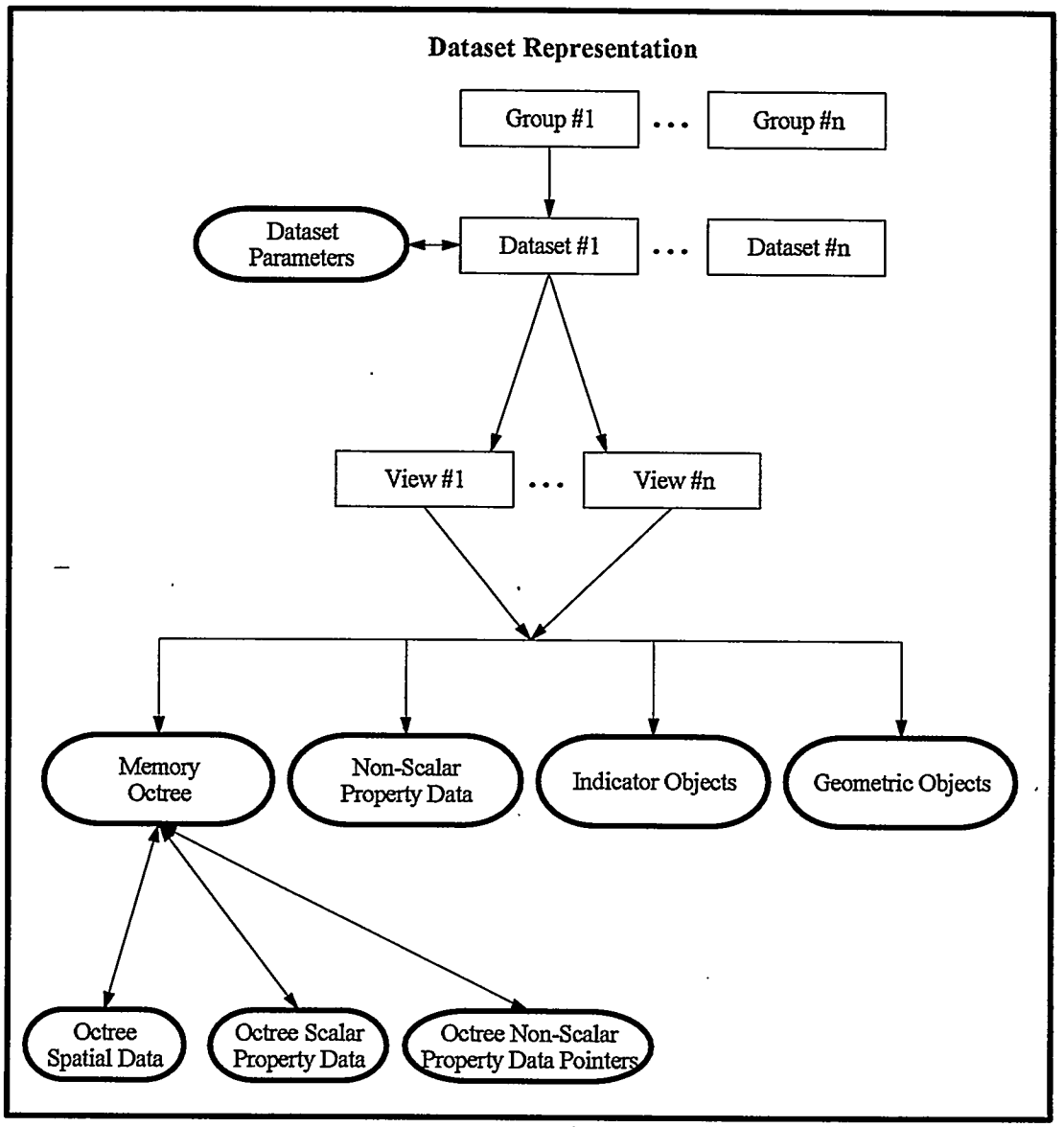


Figure 3-1. Dataset Representation

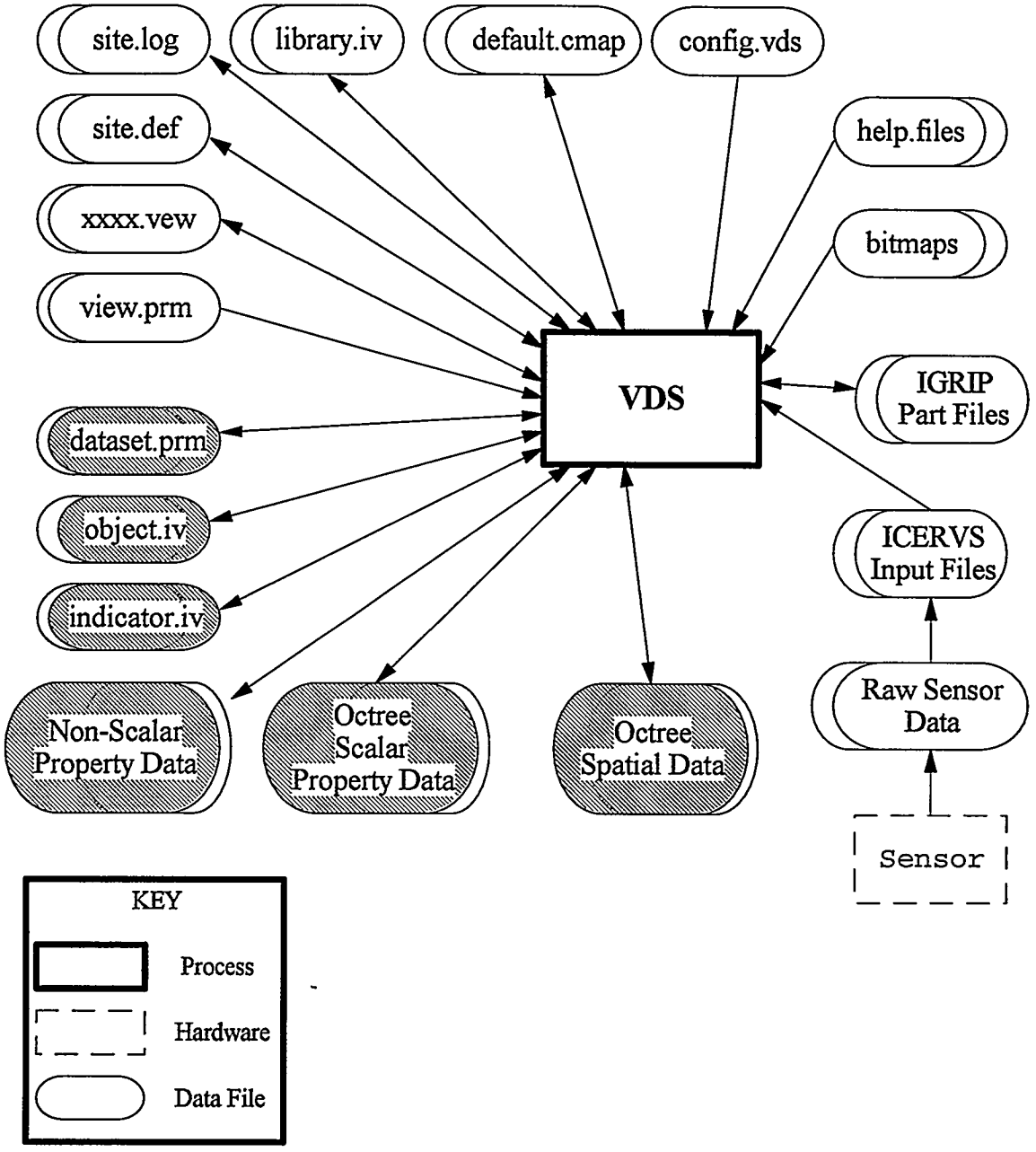


Figure 3-2. ICERVS Data Interaction Diagram

directories. Each group directory is named with the operator-specified group name given when the group was created. Under each group directory, many dataset directories can exist. Each dataset directory is named with the operator-specified dataset name given when the dataset was created. The general organization and contents for the dataset directory structure is shown in Table 3-1.

The following subsections detail each dataset level file.

3.1.1 Group-Specific Files

Default Site Definition File (site.def): This file is an ASCII text file that contains all the group-specific parameters, specifically, the dimensional description of the site. It is initially copied from the system level file *site.def.* when a new group is created and can be modified by an ordinary text editor.

Site Log File (site.log): This file is an ASCII text file that contains operator log entries for the particular group. It is initially created by the system when a new group is created. No restrictions are placed on the content or the format of the entries. The operator may browse or append to these files as desired. One of these files is stored in each group directory.

3.1.2 Dataset-Specific Files

Colormap Files (xxxxxx.cmap): These files contains colormap information used to color sensor data. Each file contains a dataset-specific colormap which was previously created using the ICERVS View/Colormap menu function.

Dataset Parameters File (dataset.prm): This file contains a list of the dataset material properties that have been stored in the dataset. It is initially created programmatically when a new dataset is created. The contents of this file are not editable and are maintained automatically by the VDS. One of these files is stored in each dataset directory.

Site Definition File (site.def): This file contains some dataset-specific parameters, specifically, the dimensional description of the site that can be modified by the user with a text editor. Whenever a dataset is created in the group, a copy of the group's site definition file is placed in the new dataset directory. One of these files is stored in each dataset directory.

View Parameters File (view.prm): This file contains the default parameters for view windows, consisting of viewpoint and cutplane definitions. These parameters may be altered in memory for each view, which allows each view to be unique. The unique view parameters may be saved permanently in a saved view file for later retrieval. The current view parameters can also be saved as the default view parameters, and will be the parameters used whenever a new dataset is created.

Geometric Objects File (object.iv): This file contains the user-defined 3-D geometric objects for a specific dataset in a specific group. The objects were either created by the operator, or imported from IGRIP via IGRIP part files. All of these objects are dimensionally described and are referenced by object name. It is assumed that their units match the units defined in the dataset. The file is in Open Inventor format. The Open Inventor ASCII file format is described in *The Inventor Mentor* book. The file is

Table 3-1. ICERVS Dataset Directory Structure

$ICERVSDATA	Root of Dynamic ICERVS Data Structure
GROUP1	Contains group-specific files for first group
site.def	Group Default Site Definition File
site.log	Site Log File
GROUP1/DATASET1	Contains dataset specific files for first dataset
<colormap>.cmap	Colormap Files
dataset.prm	Volumetric Data Files
site.def	Volumetric Data Files
view.prm	Property Files
object.iv	Geometric Object File
indicator.iv	Indicator Object File
<octreename>.oct	Dataset Parameters File
<octreename>.Shape	Site Definition File
<octreename>.<prop>	View Parameters File
<savedviewname>.vew	Saved View Files
property	Contains non-scalar property files for first dataset
<propertyname>	Non-scalar Property Files
GROUP1/DATASET2	Contains another set of data files
GROUP2	Contains group-specific files for second group

96TR37

read the first time a view is created on a dataset and is added to the Inventor tree being built in memory as a node under the root node. Conversely, when the dataset is saved, this node and all "geometric object" nodes underneath it, get written to the Geometric Objects File on disk. One of these files is stored in each dataset.

Indicator Objects File (indicator.iv): This file contains the non-spatial data indicator objects for a specific dataset in a specific group. These objects represent the location of some non-scalar property in the sensor data. These objects are created whenever non-scalar property data are added to the dataset. All of these objects are dimensionally described, and it is assumed that their units match the units defined in the dataset. The file is in Open Inventor format. The file is read the first time a view is created on a dataset and is added to the Inventor tree being built in memory as a node under the root node. Conversely, when the dataset is saved, this node and all "indicator object" nodes underneath it, get written to the Indicator Objects File on disk. One of these files is stored in each dataset directory.

Volumetric Data Files (xxxxxx.oct): These files contain a description of the octree in which the sensor data is stored. They are ASCII files in TrueSolid format. Their filename comes from the dataset parameter file. One of these files is stored in each dataset directory.

Volumetric Data Files (xxxxxx.Shape): These files contain the sensor data which is stored in the octree. They are binary files in TrueSolid format. Their filename comes from the dataset parameter file. One of these files is stored in each dataset directory.

Volumetric Data Files (xxxxxx.<propertyname>): These files contain the property data which is stored in the octree. They are binary files in TrueSolid format. Their filename comes from the dataset parameter file. Their file type is named after the scalar property contained within. Several of these files may be stored in each dataset directory, one for each defined scalar property.

Saved View Files (xxxxxx.vew): This file contains the view parameters that define a saved view including window location, view type, translation, scaling, rotation, colors, cutplane information, etc. It is in the same format as the View Parameters File. It is an ASCII file that is initially created when the operator executed the View "Save" function. The filename is given by the operator when the view was being saved, and the file extension is given by the system. A Saved View Parameters File may be restored into an existing view with the View "Restore" function and the view parameters contained in the file will overwrite the view's current view parameters. Multiple files may be stored in each dataset directory.

3.1.3 Non-Scalar Property Files

(<non-scalar_property_name>): These files contain the non-scalar property data. The files can be any type of file. The <non-scalar_property_name> is given by the operator when the file is stored. All non-scalar property files are stored in their corresponding Group/Dataset subdirectory */property.*

3.2 Memory Structure

The dataset is represented in memory as a TrueSolid octree, a set of indexes to non-scalar property files, a reference to the geometric object node in the Inventor Tree, a

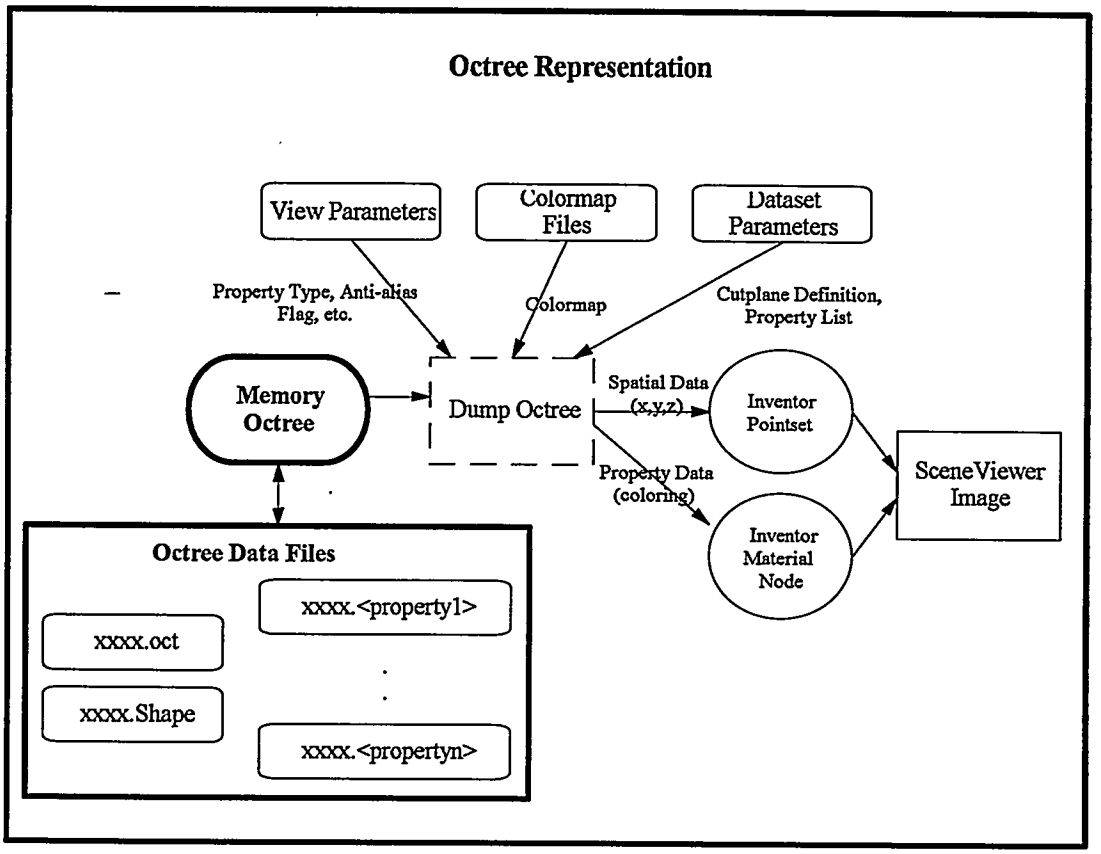
reference to the indicator object node in the Inventor Tree, and memory resident dataset parameters. The following describes the main elements of a dataset in memory:

- The octree contains the current volumetric and property data (scalar property data and pointers to non-scalar property data). All views on the dataset are connected to the same octree in memory. See Figure 3-3 for a description of the octree representation. The highlighted symbols on the figure represent the octree data files and memory image. All other symbols represent objects that the octree must interact with in order to render an image.
- The octree contains a special property field called *non_scalar_index* (not shown in Figure 3-3). This property contains index numbers which correspond to names of indicators in the scenegraph. An info node for the data indicator in the scenegraph contains the list of non-scalar filenames for that particular data indicator. This is the way each spatial data point which contains non-scalar property data can access the appropriate non-scalar property data files.
- The Geometric Objects for a dataset are stored in an Inventor format file (not shown in Figure 3-3), and when it is read into the Inventor tree, it gets added to a child node off the root node. Refer to Section 3.3 for details of the geometric object nodes. An instance of the Geometric Object class *C3dObject* is created, and the pointer to this Inventor Geometric Object node is stored in the geometric object along with other object information.
- The Indicator Objects for a dataset are stored in an Inventor format file, and when it is read into the Inventor tree, it gets added to a child node off the root node. An indicator object marks the point in the spatial data where non-scalar property data are stored. This child node is a group node named "Indicator Objects." Refer to Section 3.3 for details of the indicator object nodes. An instance of the Indicator Object class *C3dObject* is created, and the pointer to this Inventor Indicator Objects node is stored in the indicator object along with other object information.
- The dataset parameters are stored in a file and read into memory the first time a view of a dataset is created. An instance of the Dataset Object class *CVDSDataset* is created and the list of material properties from the file is stored as a data member in the object. In addition, the site parameters from the site.prm file are stored as data members in the object.

3.2.1 Sensor Data

Sensor data consists of three types: volumetric data, scalar property data, and non-scalar property data.

Volumetric data represents the spatial data acquired by the sensors. These data represent a 3-D space and are stored in octree files on disk. The octree file is really two files: an octree file <name>.oct, (where <name> is the octree name from the dataset file) which stores general octree information, and a shape file, <name>.Shape, which



96TR37

Figure 3-3. Octree Representation

contains information about the octree shape. Each dataset contains an octree file. When a dataset is selected for viewing, the octree is read into memory and dumped into an Open Inventor pointset.

A pointset is an Inventor structure that describes a set of 3-D points. Since the actual rendering of the volumetric data is done using an Open Inventor scenegraph, the volumetric data must be contained in a pointset before rendering can occur.

Scalar Property data represents the non-spatial data acquired by the sensors. These data represent property values acquired at spatial locations and are stored in octree property files on disk. These properties are represented in the octree as floating point values. There is one octree property file for each property acquired. These files are named <name>.<property> (where <name> is the octree name from the Dataset Parameter File and <property> is the property name from the Dataset Parameter File). Each dataset contains an octree file with several property files that are part of the octree. When a dataset is selected for viewing, the octree is read into memory along with all its property information. The property data in memory are then dumped to an array to be used for coloring the pointset being rendered in the Open Inventor scenegraph.

There are at least two special properties which are always in the octree. The first is a property called *z-value*. It represents the elevation or z spatial value at that point in the octree, and is used as a means for default coloring of volumetric data. The second special property is called *non-scalar_index*. It is a field containing index information about the non-scalar property values that are stored at that point in the octree. These two special properties cannot be accessed by the operator and are known only to the software. The remaining user property fields contain all the property types that exist for the particular dataset. Up to 14 user properties may be stored per dataset.

Non-scalar property data provide the user with a means to associate complex data with a spatial data point (xyz). Complex data are any type of data that are more than a single data value (i.e. scalar property). These data can be in the form of text, images, formatted documents, spread sheets, etc. A special integer property in the octree exists called *non_scalar_index* which contains an index value for each spatial point associated with some non-scalar data. This index is actually part of a name for a data indicator which marks the location of the non-scalar data in the view. The data indicator is represented in the system as a geometric object defined in the scenegraph. All data indicators in the scenegraph are actually sphere shapes which contain an info node containing the list of non-scalar properties associated with the data indicator. This is the way that non-scalar properties are kept track of in the system.

3.2.2 Geometric Object Models

Geometric objects are supported in ICERVS for modeling objects which exist in the data. These objects include geometric primitives, polyhedrons, and composites.

All geometric objects are represented internally in ICERVS as 3-D geometric objects in Open Inventor. They describe physical matter that is affected by property (color, material type, etc). During a rendering operation on the tree, they cause their shape to be drawn on the screen. ICERVS uses Open Inventor for both storage (memory and disk) and manipulation of objects.

The following discusses the specific types of objects supported and their internal representation.

Geometric Primitive - This is an object type that include spheres, cones, cylinders, and boxes. It is created in ICERVS parametrically with either a radius (sphere); a radius and length (cone, cylinder); or a length, width, and height (box). It is represented in ICERVS using one of the Open Inventor classes SoSphere, SoCone, SoCylinder, or SoCube, which are derived from SoShape. It can be manipulated interactively using the manipulator icons supplied on the left side of the *View Window* (see Section 5.2.2). However, the *Reshape* manipulator may not be used on geometric primitives. It can be edited either interactively via the *Object Transform Editor Window* (see Section 6.4.2), or parametrically via the *Object Parametric Editor Window* (see Section 6.4.1).

Polyhedron - This is an object type with an arbitrary number of polygonal faces, each of which can be arbitrarily shaped. The simplest form of this type is a right regular prism, which can then be edited with the *Reshape* manipulator to achieve an arbitrary shape. It is created in ICERVS parametrically with a radius, length, and number of vertices input. It is represented in Open Inventor as a set of faces, or polygons. Each polygon is made up of a set of vertices or coordinates, which are ordered by connectivity indexing. The polygon is represented by a set of indices into the set of vertices. The index ordering of the vertices corresponds to traversing the vertices of the polygon in a counterclockwise direction, applying the right-hand rule so that the normal (right-handed) to the polygon points outward from the object. The Open Inventor class name for this type is SoIndexedFaceSet and is derived from SoVertexShape which is derived from SoShape. In addition, Inventor stores the vertices of the object as a SoCoordinate3. It can be manipulated interactively using the manipulator icons supplied on the left side of the *View Window* (see Section 5.2.2). It is restricted from being parametrically edited via the *Object Parametric Editor Window.* However, it can be interactively edited via the *Object Transform Editor Window* (see Section 7.4.2).

Composite - This type is a collection of two or more objects of the above types grouped together to form one common object. This common object can be manipulated as one object. It is created with the View Window's *Object - Build Composite* menu function and can be taken apart with the *Object - UnBuild Composite* menu function. It is represented in Open Inventor as a group node containing a collection of SoShape nodes (one for each object in the composite object). Each SoShape may be any one of the Open Inventor shapes described above. It can be manipulated interactively using the manipulator icons supplied on the left side of the *View Window* (see Section 4.2.2). It is restricted from being parametrically edited via the *Object Parametric Editor Window* or reshaped interactively via the *Reshape* manipulator. However, it can be interactively edited via the *Object Transform Editor Window* (see Section 6.4.2).

All object types except polyhedron are represented the same way in Open Inventor as they are in ICERVS when they are parametrically created. Because of this, polyhedron performs a format translation during storage into or retrieval from an Open Inventor tree. The object knows how to convert itself to and from Inventor format. This conversion translates parametrically represented polyhedron (ICERVS format) to sets of polygons (Open Inventor format) and visa versa.

All objects in IGRIP part files are stored as sets of polygons (or faces). All object types except geometric primitives are represented in a similar fashion in Open Inventor as they

are in IGRIP part files. Because of this, geometric primitives perform a format translation when they are being exported to IGRIP. The *CFaceSet* class represents an Inventor-indexed face set and contains methods for format translation between polyhedron in ICERVS format and IGRIP or TrueSolid formats.

Some object's types are changed automatically when they are exported to IGRIP and later imported back to ICERVS. These restrictions are necessary because of the inconsistencies between IGRIP part file format and Open Inventor file format, but can be misleading to the operator. The following scenario depicts some of the limitations.

A geometric primitive (cylinder) is created and parametrically edited. It is then exported to IGRIP and later imported from IGRIP with some minor changes made to it. Since IGRIP doesn't support parametric representation of objects, the object had to be converted into a polyhedron before it could be exported to IGRIP. When it is later imported from IGRIP, it is now known in ICERVS as a polyhedron. The operator is disallowed from parametrically editing it because it is represented as a polyhedron.

3.2.3 View Parameters

View parameters exist in ICERVS in one of three ways: default view parameters, current view parameters, and saved view parameters.

Default view parameters are parameters used by the view for display purposes. A default set of system-wide view parameters ($ICERVSROOT/etc/view.prm) is copied by each dataset when it gets created. It contains parameters such as the default viewpoint and the default background color of the View Window. This default file may only be edited off-line.

Once the default file is copied into the dataset directory, it becomes the current working version of view parameters for that dataset ($ICERVSDATA/<Groupname>/<Datasetname>/view.prm). This is the version that is used each time a dataset is viewed. The operator may change the view parameters. (in memory) via several menu options, and save them as the dataset-specific default view parameters from the View Settings Window button "*Save As Default*" if desired.

If the operator wishes to store the current memory version of the view parameters onto disk, it can be done with the View "*Save*" menu function. This will allow the current view parameters to be stored in an operator named file for later retrieval via the View "*Restore*" menu function. These saved view parameters contain all the information required to restore a view window to its current state (including cutplane information). If a view is being restored and another view already exists for that dataset, the operator will be asked if the cutplane information in the restore file should be used for all other views on that dataset. If so, all other views will update their view parameters to be the restored view parameters. If not, the restored view parameters will not include its cutplane information, and the restored view will use an existing view's cutplane information. This logic is used to maintain cutplane compatibility between View Windows. See Figure 3-4 for an example View Parameters File.

3.2.4 Dataset Parameters

Dataset parameters exist in the ICERVS as a combination of dataset parameters and site parameters. The dataset parameter file ($ICERVSDATA/<Groupname>/<Datasetname>/

```
#-------------------------------------------------------------
# VIEW PARAMETERS FILE
#-------------------------------------------------------------

[OPTIONS]
ShowObjects=                1
ManualRefresh=              0
StatusWindow=               1
HighResolutionDisplayMode= 0
ApplyCutplanes=             0

AntiAliasing=               0
PropertyType=               NONE
ThresholdFile=              default.cmap
CameraType=                 ORTHOGRAPHIC

RenderAreaSizeX=               512
RenderAreaSizeY=               512
WindowPositionX=               13
WindowPositionY=               33

ColorBackgroundWindow=         0 0 0
ColorObjects=                  1 1 1
ColorSelectedRegion=           1 0 0
ColorSelectedObject=           1 0 0
ColorLowResolutionPoints=      0.93 0.86 0.51

ColorIndicators=               1 0 1
SizeIndicators=                0.05
MaxSizeIndicators=             0.1
ShowIndicators=                1


[CAMERA]
Position=                   8.83108  56.3731  2.54924
Orientation=                9.80615e-07  0.707106  0.707107  3.14159
AspectRatio=                1
NearDistance=               15.3576
FarDistance=                57.4305
FocalDistance=              36.3731
Height=                     8.10477
HeightAngle=                -9999

[CUTPLANES]
NumberOfCutplanes=   5

[CUTPLANE1]
Enabled=             0
ShowCuttingSurface=  1
ReverseDirection=    0
Type=                12
SlicePlane=          0
Thickness=           0
Translation=         0 0 0
Rotation=            0 0 1 0

[CUTPLANE2]
Enabled=             0
ShowCuttingSurface=  1
ReverseDirection=    0
Type=                12
SlicePlane=          0
Thickness=           0
      Translation=         0 0 0
```

96TR37

Figure 3-4. View Parameters File

```
Rotation=              0 0 1 0

[CUTPLANE3]
Enabled=               0
ShowCuttingSurface=    1
ReverseDirection=      0
Type=                  12
SlicePlane=            0
Thickness=             0
Translation=           0 0 0
Rotation=              0 0 1 0

[CUTPLANE4]
Enabled=               0
ShowCuttingSurface=    1
ReverseDirection=      0
Type=                  12
SlicePlane=            0
Thickness=             0
Translation=           0 0 0
Rotation=              0 0 1 0

[CUTPLANE5]
Enabled=               0
ShowCuttingSurface=    1
ReverseDirection=      0
Type=                  12
SlicePlane=            0
Thickness=             0
Translation=           0 0 0
Rotation=              0 0 1 0
```

96TR37

Figure 3-4. (continued)

dataset.prm) is created by the system each time a new dataset is created. It contains the set of material properties that currently exist in the data. Initially, this set is blank. In addition, a site definition file ($ICERVSDATA/<Groupname>/<Datasetname>/site.def) is copied from the associated group's default site definition file ($ICERVSDATA/<Groupname>/site.def). This group-specific default site definition file was copied from the system-wide site definition file ($ICERVSROOT/etc/site.def), and edited by the operator each time a new group was created. This Site Definition File contains site-specific information such as dimensions and units. The group-specific site definition is editable from the Dataset Manager Window via the Group "*Edit Defaults*" menu function. The dataset-specific site definition is editable from the Dataset Manager Window via the Dataset "*Edit Parameters*" menu function. See Figure 3-5 for an example Dataset Parameters File. See Figure 3-6 for an example Site Definition File.

3.3 Open Inventor Representation

The dataset is visualized in a View Window using an Open Inventor Examiner View Window which is attached to the Open Inventor tree (or scenegraph).

The Inventor tree is a tree structure with nodes or branches which represent different parts of the image to be viewed. The tree contains a collection of 3-D objects. All information about these objects, their shape, size, coloring, surface texture, location in 3-D space, is stored in the tree. The objects in the tree can be rendered, picked, highlighted, and manipulated as discrete entities. They can also be printed, searched for, read from a file, and written to a file.

The Examiner View Window is an XT component of Open Inventor which automates much of the view window interactively with the operator. The Examiner View window automatically contains a *Camera* node which handles all the viewpoint manipulation. Therefore, a *Camera* node is not needed in the Inventor tree.

Multiple views of the same dataset all share the same octree and Inventor tree. However, since each view has its own set of view parameters, the display of data can be different for each View Window. Because the same Inventor tree is connected to all view windows, certain nodes on the tree must be changed each time a different view renders the scenegraph. For example, one view may have objects turned off and one view may have objects turned on. This situation is handled by creating a callback node attached to the tree's root node. The callback method is executed each time the tree is rendered. This callback method figures out which view window initiated the render, and sets certain nodes in the tree to the rendering view's view parameters. For example, when the first view window (which has objects turned off) initiates a render event, the root node's callback method is executed using this view's *objectshowflag* view parameter. Since the *objectshowflag* value is OFF, the method sets the "geometric object" group node in the tree to ignore all of its child nodes when rendering the tree. This has the effect of not displaying any geometric nodes when the tree is rendered. When the second view window (which has objects turned on) initiates a render event, the root node's callback method is executed using this view's *objectshowflag* view parameter. Since the *objectshowflag* value is ON, the method sets the "geometric object" group node in the tree to render all of its child nodes when rendering the tree. This has the effect of displaying all geometric object nodes when the tree is rendered.

```
#-------------------------------------------------------------
# DATASET PARAMETERS FILE
#-------------------------------------------------------------

[GENERAL_DATASET]
 Name=                  south-wall-radiation
 Description=           South wall with radiation
 CreateDateTime=        08/22/96 19:21:10
 ModifyDateTime=        08/22/96 19:35:34
 OctreeFilename=        octree
 LastUsedNonscalarIndex 0


[PROPERTY1]
 #-------------------------------------------------------------
 # PROPERY TYPE #1
 #-------------------------------------------------------------
 Name=                  Radiation
 Description=           none

 Units=                 N/A
 MinimumRange=          0
                            MaximumRange=        1.4028
```

96TR37

Figure 3-5. Dataset Parameters File Example

```
#-------------------------------------------------------------
# SITE DEFINITION FILE
#-------------------------------------------------------------

[GENERAL_SITE]
 Name=                          SouthWall
 Description=                   South Wall w/ Radiation

[UNITS]
 DataUnits=                     METERS
 DataResolution=                .02

[DIMENSIONS]
 HeightZ=                       25.
 WidthX=                        25.
 LengthY=                       42.

 HeightOffset=                  -5.
 WidthOffset=                   -5.
 LengthOffset                   -1.
```

96TR37

Figure 3-6. Site Definition File Example

The Inventor tree contains a root node and several child nodes to represent all images of the dataset contents (i.e. volumetric data, geometric objects, data indicators, etc.) See Figure 3-7 for a description of the Open Inventor tree representation. The highlighted symbols on the figure represent the tree nodes that must be updated for each View Window when rendering occurs. The following sections describe the child nodes in detail:

The PointSet child node represents the volumetric and property data in the dataset. It represents the volumetric data as a shape made up of a set of Inventor points. This set of points is obtained by dumping out all nodes in the octree using the TrueSolid function *ts_for_all_nodes,* and building a point set shape in Open Inventor. This shape can be thought of as a geometric object for all practical purposes, except that it cannot be manipulated by the operator. However, since Inventor knows about this object, the rendering of it is automatic whenever Inventor needs to re-render its tree. See Figure 3-8 for a diagram of the Inventor tree representation of the Pointset Objects.

The 6 Cutplane child nodes handle the manipulation and display of cutplanes. These nodes are built using the cutplane definitions contained in the View Parameters File. These definitions define the transformation of the cutplanes, the type of cutplanes, and whether or not each cutplane is displayed on the view. A view parameters option also defines whether or not cutplanes will be enabled on the view. See Figure 3-9 for a diagram of the Inventor tree representation of the CutplaneObjects.

The Geometric Object child node handles the manipulation and display of geometric objects. This node is built directly from the Geometric Objects File on disk which gets read into memory and attached as a group node (SoSeparator) under the root node. This group node itself contains child nodes, each of which represents the individual geometric objects. The individual geometric object nodes actually contain a shape node, a material node, a texture node, a complexity node, a transform node, and several information nodes which together define a geometric object. See Figure 3-10 for a diagram of the Inventor tree representation of the Geometric Objects.

The Indicator Object child node handles the manipulation and display of indicator objects. This node is built directly from the Indicator Objects File on disk which gets read into memory and attached as a group node named "indicator objects" under the root node. This group node itself contains child nodes, each of which represents the individual indicator objects. The individual indicator object nodes contain the same nodes as the geometric objects. In addition, they contain an info node used for listing the corresponding non-scalar property filenames. See Figure 3-11 for a diagram of the Inventor tree representation of the geometric objects.

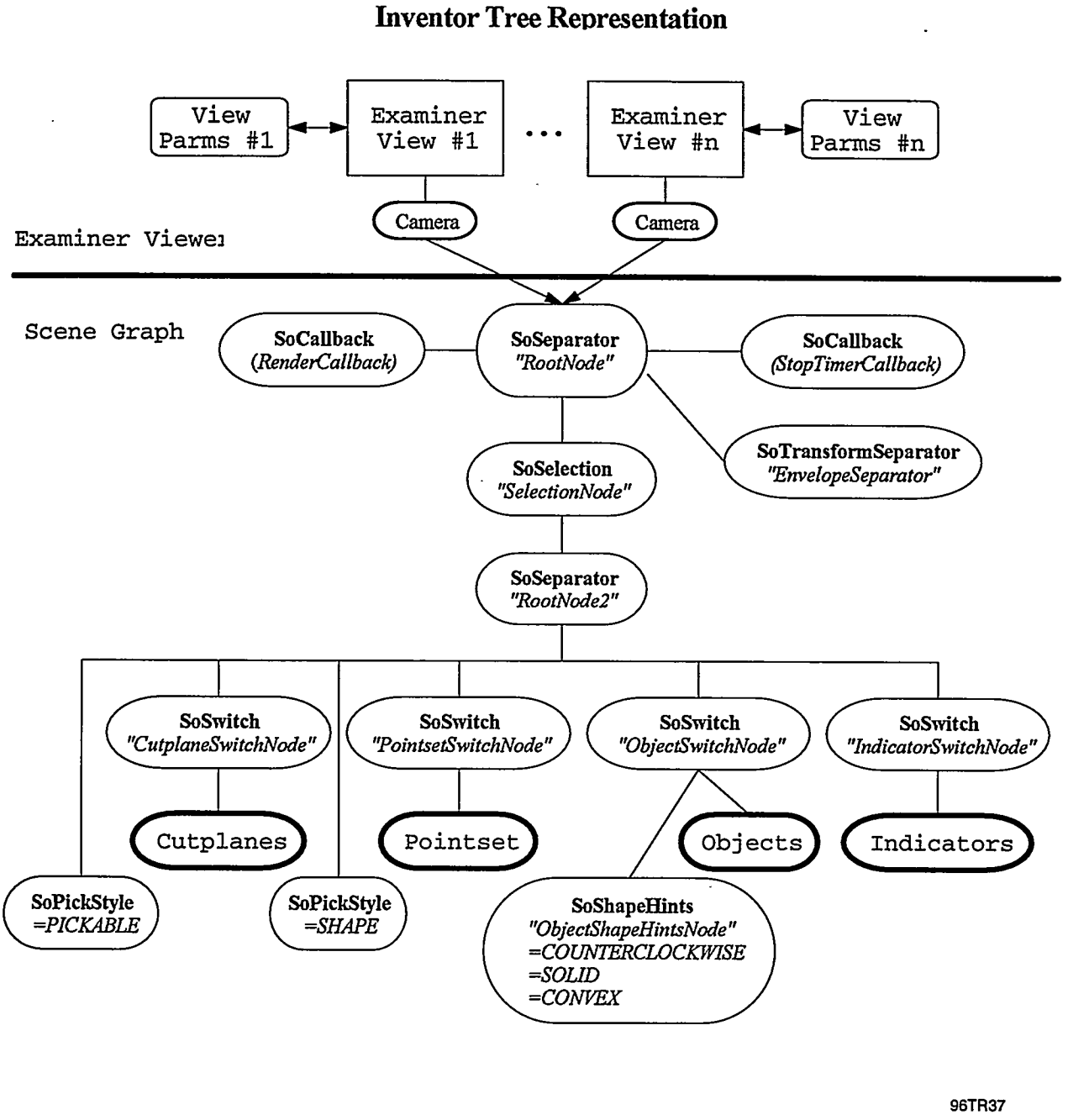


Figure 3-7. Open Inventor Tree Representation

Open Inventor Pointset Representation

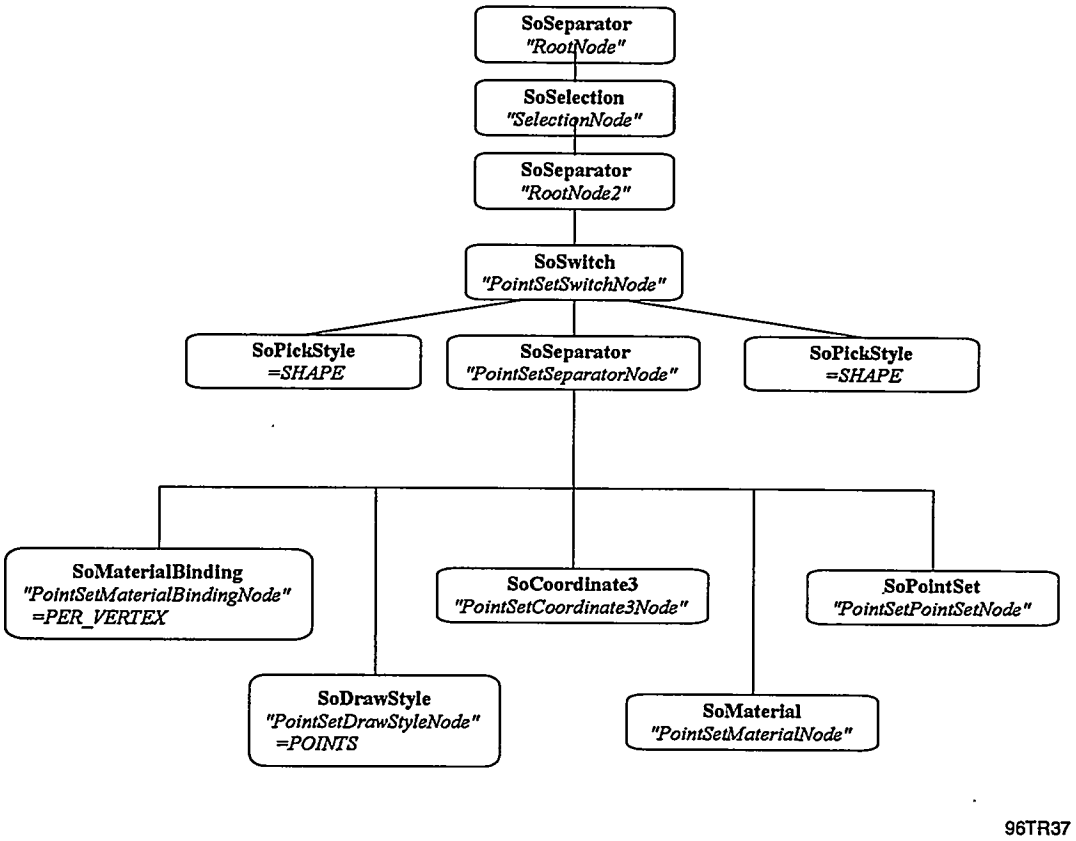
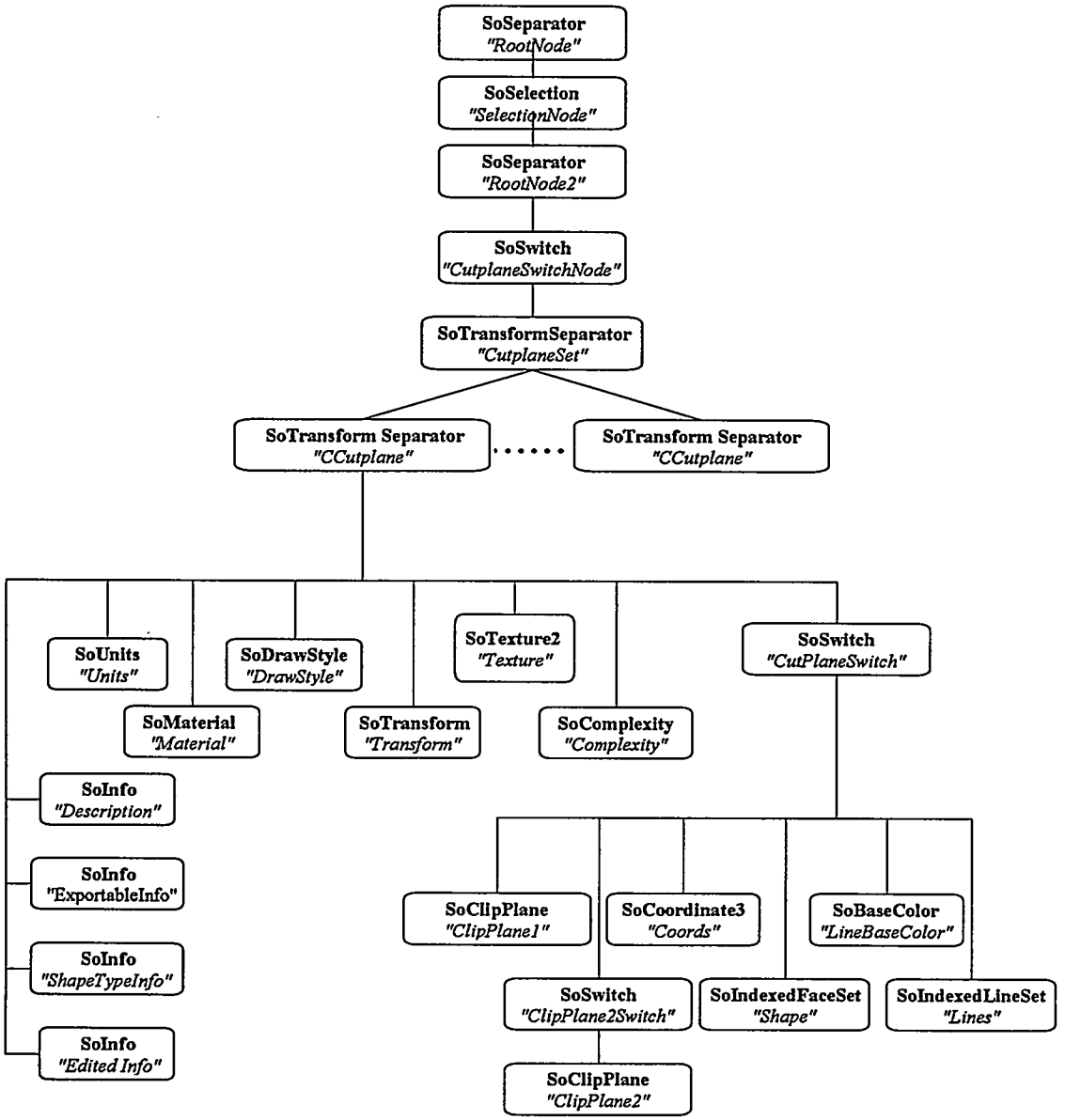


Figure 3-8. Open Inventor Pointset Representation

Open Inventor Cutplane Representation



96TR37

Figure 3-9. Open Inventor Cutplane Representation

Open Inventor Geometric Object Representation

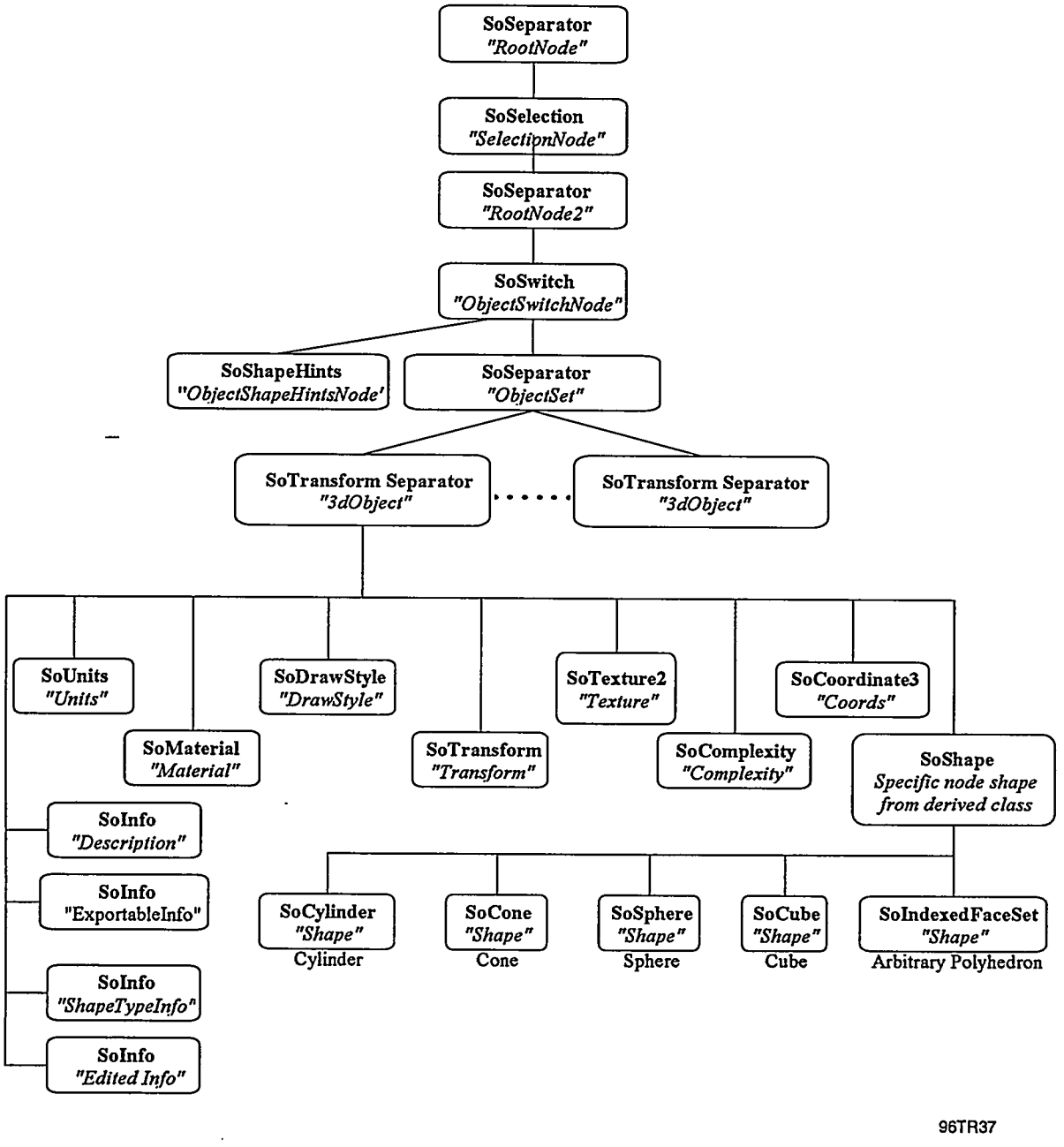
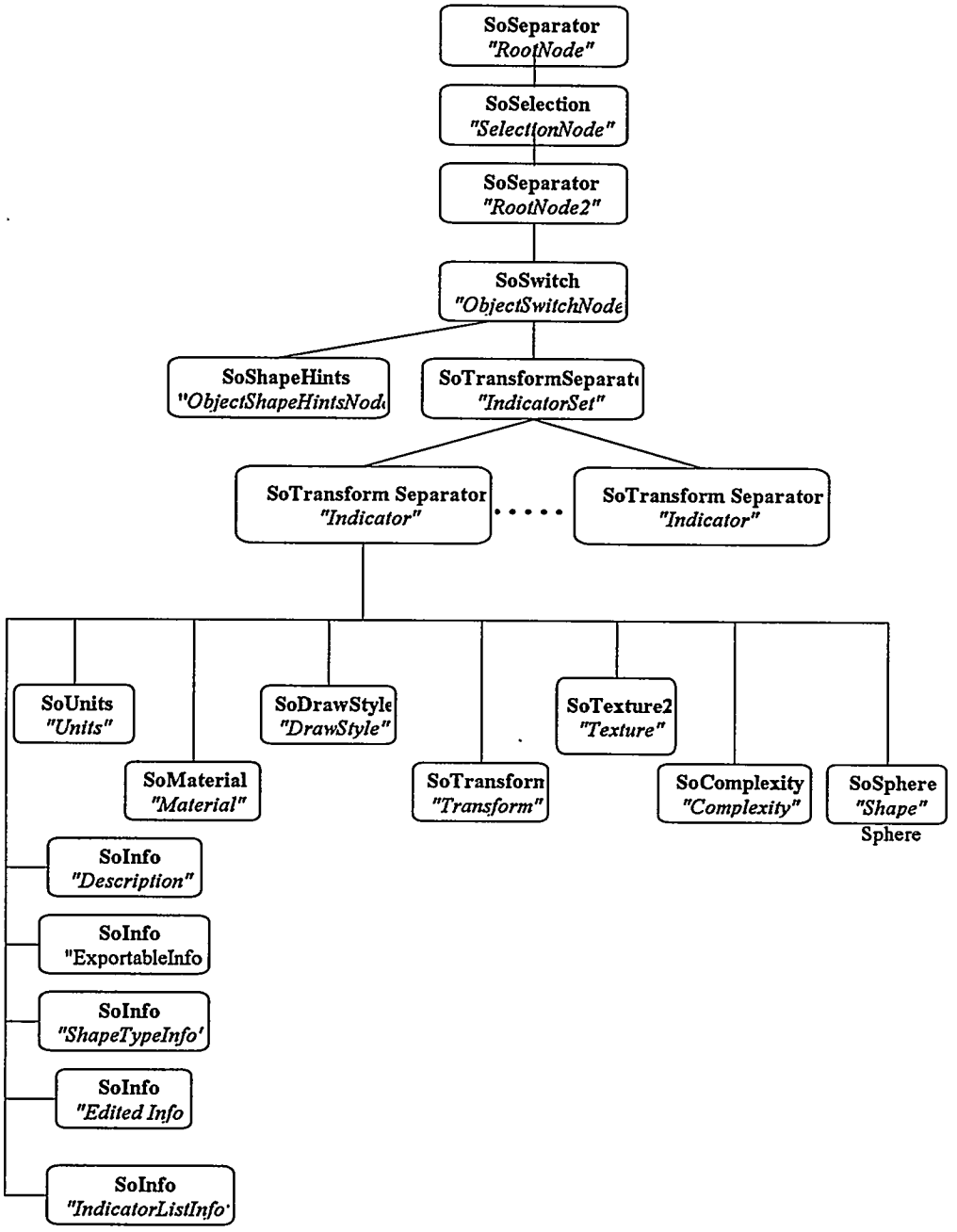


Figure 3-10. Open Inventor Geometric Object Representation

Open Inventor Indicator Representation



96TR37

Figure 3-11. Open Inventor Indicator Representation

4.0 DATASET MANAGER

4.1 Group and Dataset Definitions and Relationship

A "dataset" is a general collection of data acquired from, or constructed to describe, a 3-D workspace. Datasets can include volumetric data, scalar properties, non-scalar properties, and geometric objects. Datasets can be used, for example, to characterize a specific workspace at a discrete time or state.

A "group" is a collection of datasets. As an example, a group might represent a specific workspace, and datasets contained within the group might contain data taken at different times during remediation of the workspace. The number of datasets within a group is unlimited.

4.2 Dataset Manager Layout and General Characteristics

The Dataset Manager Window (Figure 4-1) contains the functions pertaining directly to a group or a dataset. The window provides four menu bar pull-down menus. The *Quit* pull-down terminates an active ICERVS session. Functions associated with the group management (i.e., creation, modification, deletion) are contained on the *Group* pull-down menu. The *Dataset* pull-down menu supports functions related to the management and visualization of datasets. The *Help* pull-down provides an on-line User Manual.

Clicking the left mouse button on a group or dataset name selects that entry. Subsequent functions for the selected entry can be obtained from the menu bar button labeled *Group* or *Dataset*. Table 4-1 identifies the Dataset Manager Window menu functions.

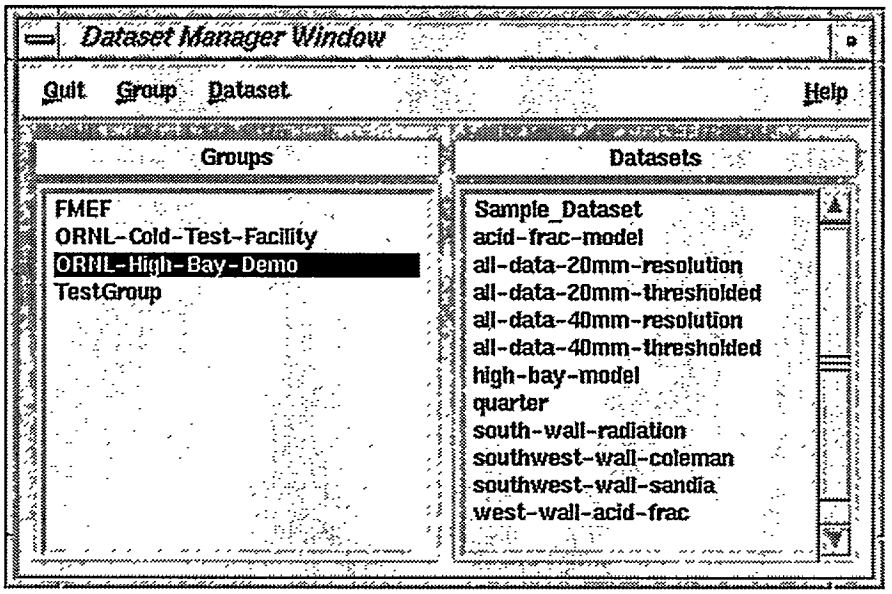
4.3 Group Management

4.3.1 Creating Groups

The creation of a new ICERVS group is initiated by selecting the *New* function from the Dataset Manager Window *Group* pull-down menu. A window titled *GROUP NEW* (Figure 4-2) will appear within which the group name is entered. The window may be dismissed without creating a new group by selecting *CANCEL*. The entered name becomes that of both the newly created group and the new group directory. Once a name is entered and the *OK* button is pressed, a group is created. A text editor window appears within which the default dataset parameters are set for the newly created group (see Section 4.3.2).

4.3.2 Setting Default Group Parameters

The default group parameters are initially defined with the creation of a new group. This initial group definition may be modified by selecting the *Edit* Defaults option of the *Group* pull down. Editing of the default dataset parameters for the selected group is performed via a text editor window, as shown in Figure 4-3.



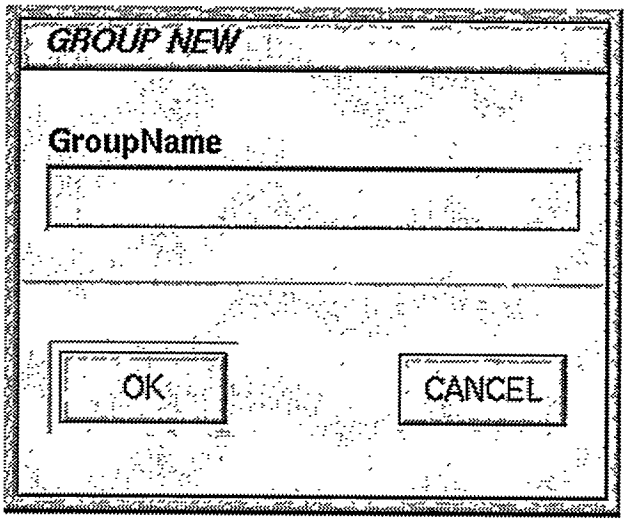
96TR37

Figure 4-1. Dataset Manager Window

Table 4-1. Dataset Manager Window Pull-Down Functions

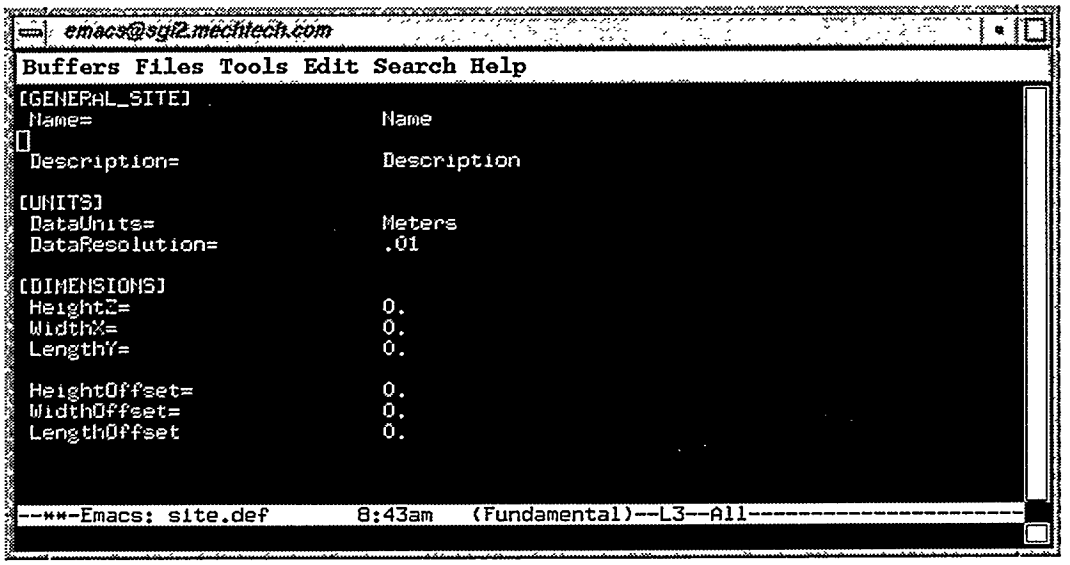
Quit	Group	Dataset	Help
Quit	Delete	View	User Manual
	Edit Defaults	Copy	Problem Report
	Edit Log	Delete	About
	New	Edit Parameters	
		Info	
		Archive	
		Retrieve	
		New	

96TR37



96TR37

Figure 4-2. Group New Window



96TR37

Figure 4-3. Default Group Parameters in a Text Editor Window

The [GENERAL_SITE] data block contains descriptive information of the group. The group name and description entries are optional.

The [UNITS] data block contains the definition of the (group) data units and the desired data resolution. When inputting sensor data to an ICERVS dataset, the software verifies that the sensor data units, as specified in each input sensor data file, matches the dataset units. The data resolution represents the grid spacing used for digitizing the input data. For example, a value of .01 implies that the input data points are stored at a grid resolution of .01 units.

The [DIMENSIONS] data block defines the origin and the dimensions of the group's world space (volume). Definition of the parameters contained within the data block is required. The offsets (height, width, and length) define the volume origin (z, x, and y coordinates, respectively) The height, width, and length values correspond to the extents from the origin in the z, x, and y directions.

4.3.3 Removing Groups

A selected group on the Dataset Manager window may be deleted by selecting the *Delete* function from the *Group* pull down (see Section 4.2). A confirmation message window will then appear. Confirmation deletes the ICERVS group directory and dataset subdirectories.

4.3.4 Information

Notes may be appended to the site log file for the selected group by selecting the *Edit Log* function from the Dataset Manager window *Group* pull down (see Section 4.2). The notes are entered via the text editor (see Section 4.3.2).

4.4 Dataset Management

4.4.1 Creating Datasets

A dataset is created under the selected group by selecting the *New* function of the Dataset Manager window *Dataset* pull down (see Section 4.2). The dataset name and an (optional) description are entered on the Dataset New window (see Figure 4-4). The software opens a text editor window to set the dataset parameters. The default group parameters for the selected group become the default dataset parameters for the newly created dataset. When the new dataset has been defined, a View window will be automatically displayed for the newly created dataset.

4.4.2 Copying Datasets

The Dataset function "*Copy*" copies the selected dataset to a new dataset within the same group. This new dataset will have a user-supplied name and description. (This the same function as the Dataset *Save As* function in the View Window).

4.4.3 Removing Datasets

The Dataset function *Delete* deletes an entire dataset with user confirmation.

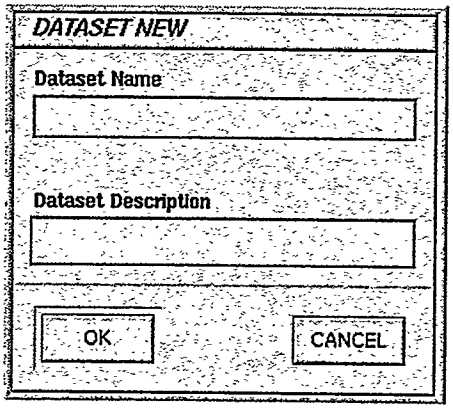


Figure 4-4. Dataset New Window

4.4.4 Modifying Dataset Parameters

The Dataset function *Edit Parameters* allows the editing of the group parameters that are specific to a dataset via a text editor window (see Section 4.3.2). The saved edited parameters are applied to the existing dataset. Modifying Dataset parameters is disallowed if a view window is open for that dataset. (Note that changing the data resolution and dimensions may invalidate data previously stored in the dataset).

4.4.5 Viewing Dataset Information

The dataset function *Info* generates a Browser Window containing dataset specific information (e.g., dataset name, description, date/time created/modified, existing properties, data units, resolution, and world dimensions).

4.4.6 Saving Datasets to File

The Dataset function *Save* saves the data currently displayed in the View Window, including dimensional data, scalar property data, non-scalar property data, and geometric objects. The Dataset function *Save As* saves the data currently displayed in the view window to another group and dataset. A *Dataset - Save As Window* is displayed to allow the User to select a group for storing the dataset and to enter a new dataset name and description. If other views exist for the same dataset, then these views will also display the newly created dataset.

4.4.7 Transferring Datasets to Other Systems

The dataset functions *Archive* and *Retrieve* allow a dataset to be saved to an external file and the subsequently restored dataset to another group/dataset (possibly on another computer).

4.4.7.1 Dataset Archive. The Dataset function *Archive* archives the selected dataset to a user-defined file. A *Dataset Archive Window* is displayed to allow the user to select a filename to archive the dataset to. The default file path is the user's home directory, and the default file name is the dataset name being archived.

4.4.7.2 Dataset Retrieve. The Dataset function *Retrieve* retrieves the selected archived dataset into the currently selected group. A *Dataset Retrieve Window* is displayed to allow the user to select the archived dataset filename to be restored. The retrieved dataset will automatically create a new dataset in the selected group with the same name as the archived dataset being retrieved. If the dataset name already exists in the selected group, the dataset will be overwritten with the archived dataset being retrieved.

5.0 DATASET VIEW

The *View Window* is created from the Dataset Manager's *Dataset - View* Function or when a new dataset is created. All functions pertaining to the viewing of a dataset are contained in this window. *View Windows* contain images that represent the 3-D volume. The *View Window* renders an image of the spatial data and the set of geometric objects that model the known entities in the volumetric data. Figure 5-1 illustrates a sample *View Window.*

The title bar on the *View Window* is labeled with information identifying the group name, the dataset name, and view identifier (i.e., number). The view number increments with each view that is created (see Section 5.5). The menu bar contains pull-down buttons for the following functions.

Dataset	Contains all functions related to the dataset displayed in the view
View	Contains all functions related to viewing the dataset
Analyze	Contains all analysis functions
Object	Contains all geometric object functions
Input	Contains all functions related to inputting data into the dataset
Output	Contains all functions related to outputting the image in the view
Help	Contains all help functions

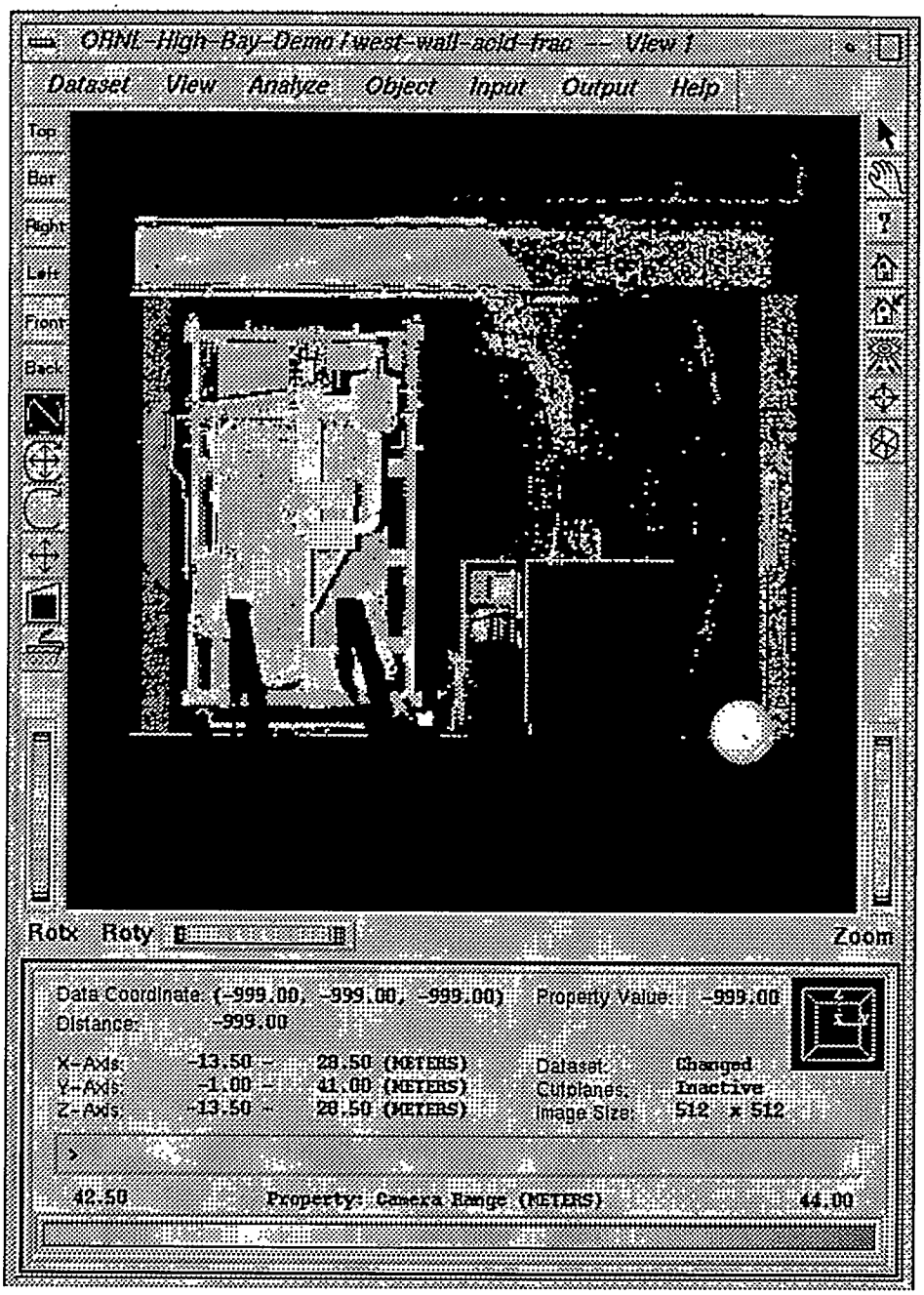
Table 5-1 identifies the functions selectable from each function menu in the *View Window.*

The main area in the center of the view window is used to display the dataset contents. Dataset contents are displayed three ways:

- Scalar data, in which each point is defined by an xyz triplet and optional list of scalar values, is displayed with a small pixel field for each point. The color of the pixels are determined by one selected scalar value. The default scalar value is the triplet's elevation (z-value).
- Non-scalar data, in which any type of file is attached to a specific xyz location, is not displayed directly. Instead, data "indicators" are used to show the locations where non-scalar data are stored. Data indicators appear as spherical objects for which the user can specify size and color (one can be seen in the lower right-hand corner of Figure 5-1).
- Object data, in which each item is a geometric primitive. Objects can be displayed as shaded surfaces, semi-transparent surfaces, wireframe, or not displayed (i.e., invisible) (see Section 5.4.1).

5.1 Open Inventor Capabilities

Much of ICERVS' display and interactive functionality is built upon the SGI Open Inventor graphics toolkit. Open Inventor is a set of building blocks that enables programs to take advantage of powerful graphics hardware features with minimal programming effort. Based on OpenGL, the Open Inventor toolkit defines a tree-based data structure for representing 3-D models as a collection of primitive shapes with geometric transforms



96TR37

Figure 5-1. Sample View Window

Table 5-1. View Window Pull-Down Menus

Dataset	View	Analysis	Object	Input	Output	Help
Save	Settings	Select Region	Select	Sensor Data	Debug	User Manual
SaveAs	Colormap	Erase Region	Attributes	Non-Scalar		Problem Report
	Transform	Print Region	Create			About
	Indicators	Set Connectivity	Duplicate			
	Examiner	Clear Connectivity	Snap to Point			
	Walk	Set Color By Range	Delete			
	Fly	Clear Color By Range	Composite			
	Headlight Edit	Erase Invisible Points	IGRIP			
	Headlight Show		Edit			
	Cutplane		Transform			
	Cutplane Show		Color			
	Copy		Info			
	Save					
	Restore					
	Close					

96TR37

and other modifiers. This data structure is used as the basis for the Virtual Reality Modeling Language (VRML).

The ICERVS *View Window* is implemented using an Open Inventor *Examiner Viewer* Window with custom software that provides additional *ICERVS View Window* functionality. A sample *View Window* is shown in Figure 5-1.

5.2 Overview of Display Controls

5.2.1 Mouse Controls

Several types of mouse controls are implemented in the View Window. The mouse behaves differently depending upon whether "View" or "Pick" mode is active.

When the window is in view mode (see Section 5.2.2), the left mouse button rotates the camera viewpoint, and the middle mouse button translates the camera viewpoint. In addition, the rotation thumbwheel controls rotate the view in the x and y axis. The zoom thumbwheel control changes the camera height (when in orthographic viewpoint mode) or the camera height/angle (when in perspective viewpoint mode). The dolly thumbwheel control changes the camera distance (when in perspective viewpoint mode). A ViewPoint Editor is available from the "*View - Transform*" menu function and allows parametric editing of the camera viewpoint.

When the window is in pick mode (see Section 5.2.2), a left mouse button click picks an object. An object may be picked for 1 of 2 reasons, either for object selection or object manipulation.

Picked objects are selected for the purpose of performing an object function via the View Window's *Object* pull-down menu. An object may be selected if the "No Manipulator" icon is currently selected (see Section 5.2.2). A selected object will appear with a red

wireframe box surrounding the picked object. Multiple object selection can be performed by clicking the left mouse button while pressing the SHIFT key. Objects are unselected by clicking in an empty region of the View Window. Picked objects may be manipulated various ways, depending upon the currently selected manipulator. (see Section 5.2.2). With the manipulators provided, an object may be translated, rotated, scaled, and reshaped. An object picked for manipulation will appear with a white manipulator surrounding the object. Only one object can be manipulated at a time.

Single clicking on a dimensional point with the left mouse button will update the current mouse position and property value in the status window (if it is being displayed). (Note that dimensional points can only be picked with the mouse when "No Manipulators" are currently selected.) Picking dimensional points in manipulator mode gets in the way of object manipulation and is therefore disabled when manipulators are active.

Single clicking on a data indicator with the left mouse button will display all property values (both scalar and non-scalar properties) associated with the data indicator's position.

When the window is in either view or pick mode (see Section 5.2.2), a right mouse button displays an Examiner View pop-up menu of various features and settings for the viewer.

5.2.2 Toolbar Functions

Several toolbar icons are displayed on the sides of the *View Window*. The icons on the right are standard for all Inventor Examiner Viewer Windows, and the icons on the left are unique to ICERVS.

The Inventor toolbar functions, available on the right-hand side of the *View Window* are, from top to bottom:

 Pick Mode	Allows user to pick objects in the view.
 View Mode	Disallows user selection of objects in the view. Only the viewpoint of the entire rendering area can be manipulated in this mode.
 Help	Provides a Help facility for the Inventor toolbar functions.
 Home	Changes the viewpoint to the currently defined home position.
 Set Home	Sets the Home position to the current viewpoint


View All — Moves the viewpoint so that all data is visible.


Seek — Moves the selected point to the center of the window.


Viewpoint type — Sets the viewpoint type to either perspective or orthogonal.

The ICERVS toolbar functions, available on the left-hand side of the *View Window* are, from top to bottom:


Top view — Displays the viewpoint as a top view


Bottom view — Displays the viewpoint as a bottom view


Right view — Displays the viewpoint as a right view


Left view — Displays the viewpoint as a left view


Front view — Displays the viewpoint as a front view


Back view — Displays the viewpoint as a back view


No Manipulator — Remove the current object manipulator. This icon has no effect if the viewer is not In Inventor "PICK" mode


Transformbox — Translates, rotates, and/or uniformly scales a selected object. Active in pick mode (only). Manipulator displays a wireframe box around the object and allows the user to translate the object with the left mouse button. The object can be uniformly scaled in all axes by clicking on a vertex of the box and dragging the vertex to a desired location. The object can be rotated about a selected axis by clicking on an edge of the box and dragging the edge in the direction of desired rotation.

 Trackball Manipulator	Rotates a selected object. Active in pick mode (only). Manipulator displays a wireframe sphere around the object and allows the user to rotate the object with the left mouse button.
 Tabbox	Translates and/or non-uniformly scales a selected object. Active in pick mode (only). Manipulator displays a tabbed box around the object and allows the user to translate the object with the left mouse button. User uniformly scales by grabbing the boxes edge and dragging to a new location. (Note that unexpected results may occur with this manipulator due to known bugs in Inventor.)
 Reshape Manipulator	Reshapes a selected object (polyhedral shape only). Active in pick mode (only). Manipulator displays a dragger that can be attached to a single object face or vertex for translation (with the left mouse button) by the user.
 Refresh	Refreshes the *View Window.*

5.3 Changing the Viewpoint

ICERVS provides several different ways to control the viewpoint at which a dataset is displayed in the *View Window.*

5.3.1 Perspective versus Orthographic Projection

ICERVS provides display using either perspective (which matches human vision and cameras with standard lenses) or orthographic projection. The projection mode can be switched using the Open Inventor ViewPoint Type toolbar control located on the right side of the *View Window* (see Section 5.2.2).

5.3.2 Predefined Viewpoints

The ICERVS toolbar functions, available on the left side of the *View Window* and described in Section 5.2.2, can be used to set the viewpoint to top, bottom, front, back, left, and right orthogonal views.

5.3.3 View Mode Tool

When the *View Window* "view mode" is active (as described in Section 5.2.1), the viewpoint can be interactively adjusted via mouse control. Moving the mouse with the left button down rotates the viewpoint. Moving the mouse with the middle button down translates the viewpoint.

5.3.4 Rotx, Roty, Zoom, and Dolly Controls

Thumbwheel controls for rotating the viewpoint about the horizontal (*Rotx*) and vertical (*Roty*) axes are available on the bottom left section of the window. Thumbwheel controls for zooming the image and dollying the viewpoint are available on the bottom right section of the window. The zoom control changes the camera height (when in orthographic viewpoint mode) or the camera height/angle (when in perspective viewpoint mode). The dolly control changes the camera distance (when in perspective viewpoint mode). The dolly slider is available only when in perspective mode.

5.3.5 Viewpoint Transform Command/Window

A ViewPoint Editor is available from the "*View - Transform*" menu function and allows parametric editing of the viewpoint.

5.3.6 Viewer Types

Three type of viewers are available from the *View* menu function as a toggle button option: ExaminerViewer, Viewer, and FlyViewer.

ExaminerViewer. This viewer allows the user to rotate the camera around a point of interest using a virtual trackball. It uses the camera focal distance to figure out the point of rotation, which is usually set to be the center of the scene. In addition, this viewer allows the user to translate the camera in the viewer plane, and dolly (move forward or backward) to get closer to or further away from the point of interest. The viewer also supports seek to quickly move the camera to a desired object or point. This is the default type of viewer in ICERVS. The previous sections all describe the functionality of this type of viewer.

WalkViewer. The paradigm for this viewer is a walkthrough of an architectural model. Its primary behavior is forward, backward, and left/right turning motion while maintaining a constant "eye level." It is also possible to stop and look around at the scene. The eye level plane can be disabled, allowing the viewer to proceed in the "look at" direction, as if on an escalator. The eye level plane can also be translated up and down - similar to an elevator.

FlyViewer. This viewer is intended to simulate flight through space, with a constant world-up direction. The viewer only constrains the camera to keep the user from flying upside down. No mouse buttons need to be pressed in order to fly. The mouse position is only used for steering, while mouse clicks are used to increase or decrease the viewer speed. The viewer allows the user to tilt their head up/down/right/left and move in the direction they are looking (forward or backward). The viewer also supports "seek" to quickly move the camera to a desired object or point.

5.4 View Options

5.4.1 View Settings

View-specific parameters can be changed by selecting the *Settings* function from the *View Window View* pull-down menu. A *View Settings Window* (see Figure 5-2) is displayed to allow user editing of view parameters. Included in these parameters are:

Property to be displayed. The scalar property type that is displayed in the view. The default is to display elevation (z-value) of the spatial data.

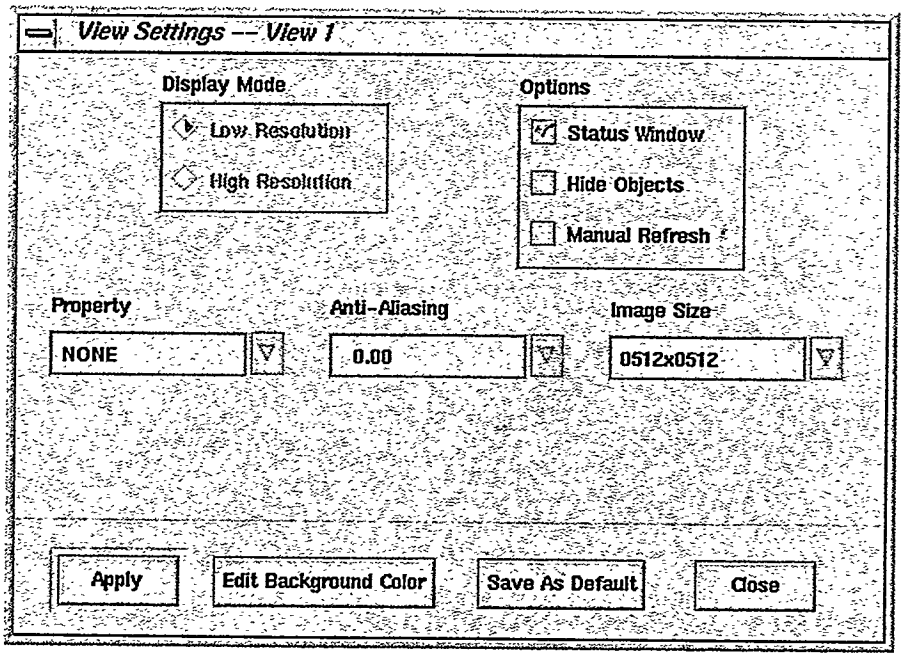
Display mode. This value is not currently used.

Anti-Aliasing value. A value from 0 to 255 that determines the amount of aliasing to use for display. A value of zero means no aliasing. Note that any non-zero value greatly increases the display time. The default is 0.

Image size. The size of the view window in pixels. The available choices are 128x128, 256x256, 512x512, 1024x1024. The default is 512x512.

Background color. This is the color of the background window and is selected by using the color selection widget. The default is black.

Status window display option. An On/Off flag indicating whether the status window is displayed at the bottom of the view window. The default is ON.



96TR37

Figure 5-2. View Settings Window

Manual refresh option. An On/Off flag indicating whether to only update the view window manually (i.e., by clicking on the refresh icon at the left of the view window). This is used in cases where the view window takes a long time to update. The user can make several changes to the view and then display the changes all at once. The default is OFF.

Hide objects option. An On/Off flag indicating whether the geometric object will be invisible in the view window. The default is OFF.

When the *APPLY* button is clicked, the new display settings take effect immediately. *The Save As Default* button allows the current view settings to become the default settings for subsequent viewing of the dataset. Be aware that all view settings, including some that do not appear on this window such as current viewpoint, are also saved as the default.

5.4.2 Status Window

The Status Window (see Figure 5-1) is attached to the bottom of the View Window and may be enabled or disabled from the View *Settings* function. It contains View Window and Dataset status. These include:

Data Coordinate. The last dimensional point that was clicked with the left mouse button is displayed here. Dimensional point picking can only done in Pick mode when the "No Manipulator" icon is selected.

Property Value. The property value that is associated with the last dimensional point clicked on with the left mouse button is displayed here.

Distance. The difference between the last two dimensional points that were clicked on with the left mouse button is displayed here.

X-Axis. The range of the x-axis in world units is displayed here with the current units.

Y-Axis. The range of the y-axis in world units is displayed here with the current units.

Z-Axis. The range of the z-axis in world units is displayed here with the current units.

Dataset. The current changed status of the dataset. If any changes were made to the dataset (i.e., new geometric objects, new data indicators, imported objects, newly input spatial data), this flag will be set to *Changed.*

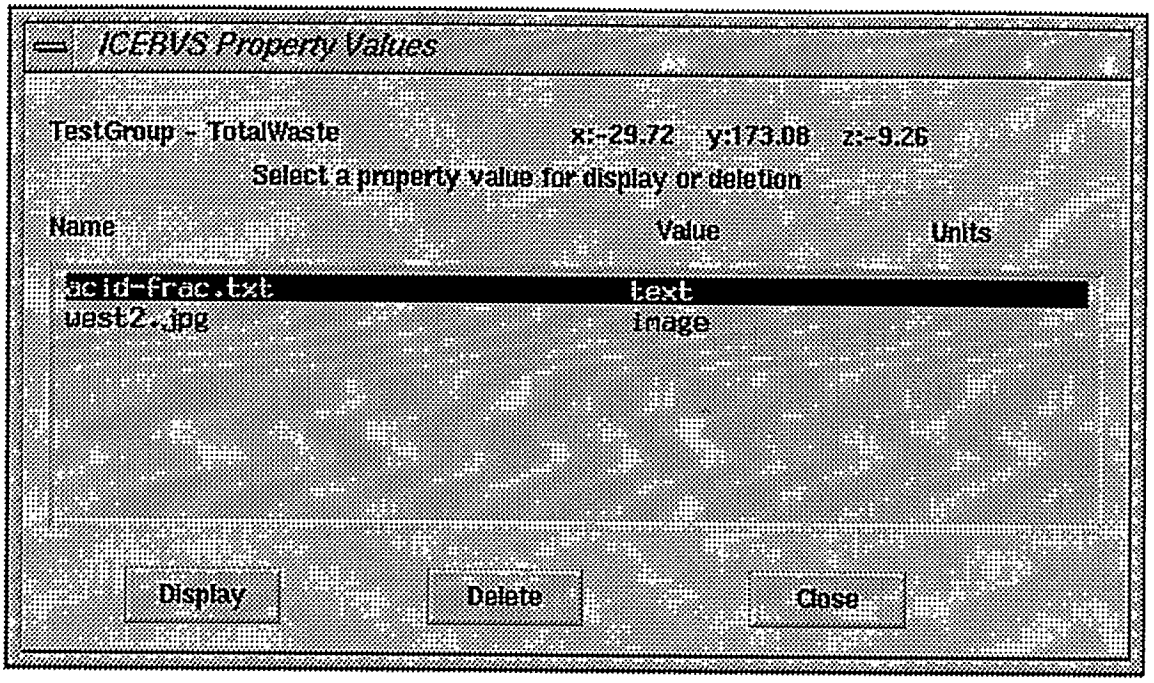
Cutplanes. The current cutplane status of the dataset. If cutplanes are enabled, this flag will be set to *Active.*

Image Size. The current image size of the View Window.

Property. The name, units, minimum value, and maximum values of the current property are displayed here.

Colorbar. The colorbar for the currently displayed property is displayed here. The leftmost color represents the lowest property value and the rightmost color represents the highest property value (see Section 5.4.4).

Rotational Orientation Cue. A visual aid for rotational orientation is displayed on the upper right section of the status window. It updates in real time along with the image in the render area of the View Window.



96TR37

Figure 5-3. Property Display Window

5.4.3 Sensor Data Property Display

All properties (both scalar and non-scalar) associated with a data indicator, which has been picked with the left mouse button on the View Window, are displayed in a *Property Display Window* (see Figure 5-3). The dimensional point associated with the displayed property list is shown at the top of the window. The scalar property data in the list represents the non-spatial data acquired by sensors. The non-scalar property data in the list is represented by array, text, and image files, each of which is associated with a dimensional data point in the dataset. The *Property Display Window* will automatically update when another indicator is picked and will display the properties associated with the new data indicator.

5.4.3.1 Scalar Data. Any scalar property data associated with the dimensional point represented by the currently picked data indicator is displayed in the *Property Display* Window. Only the scalar properties' names are displayed in the window. The scalar property's type and units are shown as *N/A.* Currently, scalar property data values are not displayed except on the status window, which shows the value of the current property for the dimensional point last clicked on with the left mouse button.

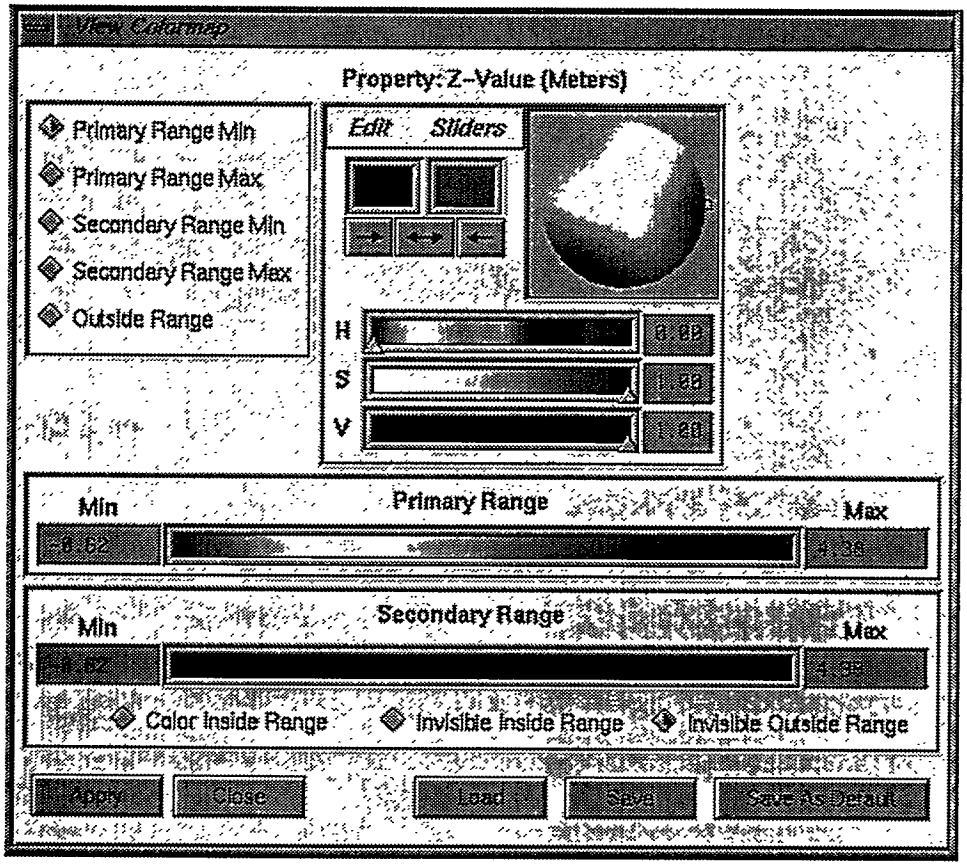
5.4.3.2 Non-Scalar Data. Non-scalar property data is displayed using an established, and widely accepted technique for associating various forms of data with viewers called MIME (Multipurpose Internet Mail Extensions). ICERVS uses an application called *Metamail* (a public domain software application) to display all types of non-scalar files. *Metamail* uses a configuration file called *mailcap,* located in the $ICERVSROOT/etc directory, which contains associations between file types and viewing applications. This file may be edited to allow any user-desired display application to be associated with their non-scalar file. Each of the non-scalar property types may be selected and viewed by clicking on the Property Display Window's *Display* button. In addition, selected non-scalar data can be deleted permanently from the dataset.

5.4.3.2.1 View Indicators. The *View* function *Indicators* allow the User to customize the indicators on the *View Window*. When a view is created, indicators are automatically displayed at each dimensional point to which non-scalar property data has been attached. Indicators appear in the *View Window* as spheres (shown previously in Figure 5-1). When the *View Indicators* function is clicked, the *View Indicator Window* is displayed allowing the user to turn on/off the display, change the color, or change the size of all indicators in the view. When the last non-scalar property is deleted at a given indicator location, the indicator will also be deleted.

5.4.4 Colormap

The View function *Colormap* allows the user to define the colormap for the spatial data in the view.

5.4.4.1 Defining Colormaps. A View Colormap Window (see Figure 5-4) is displayed to allow the user to create a colormap. Colormaps are defined by selecting a property range, and then selecting the colors to apply to the property range. Property min/max



96TR37

Figure 5-4. View Colormap Window

values (range endpoints) can be selected anywhere within the property range. The selected color will be linearly interpolated from min to max in hue. The resulting colors are displayed in the color legends on the View Colormap Window.

The View Colormap Window allows the colormap to be split into two ranges: a *Primary* range and a *Secondary* range. The primary range is the main colormap. The secondary range is used to apply enhancements to the primary range (see below).

To create a colormap, the appropriate radio button that corresponds to the endpoint is selected. Then the color wheel or sliders are adjusted to the desired color. The property range min/max values must also be set. In most cases, the secondary range can be left untouched. When the *APPLY* button is clicked, the view will redisplay using the newly created colormap.

The secondary range is used to apply enhancements to the primary range. These enhancements include a second colormap or making data invisible. A second colormap is created by selecting the *Color Inside Range* radio button. An example of a second colormap could be a neutral color, such as a dark gray, while the primary colormap colors are a selected region of data. This will effectively highlight a feature in the data, without losing the context of the surrounding data. Making data invisible is helpful in eliminating noisy data from the view (this will not eliminate data from the dataset). If the secondary range is set to the range of "good" data, and the *Invisible Outside Range* radio button is set, then data outside the secondary range will be removed from the view.

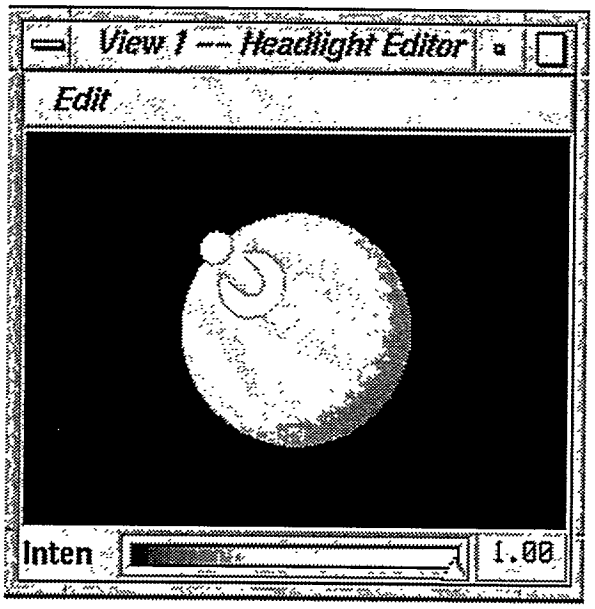
When using the secondary range, proper adjustment of the primary range and secondary range min/max values is critical. When the colormap is applied, the primary range takes precedence over the secondary range. Therefore, if both property ranges are set the same, the secondary range will never be seen. This can also be stated as follows – the only portion of the secondary range that will be seen is that portion which does not overlap the primary range.

By default, the colormap is set as follows:

1. The Primary Range property min/max values are set to extents of the property.
2. The Primary Range color min/max is set to blue/red, magenta
3. The Secondary Range property min/max values are set to extents of the property.
4. The Secondary Range radio button is set to Invisible Outside Range.

To change this default, see Section 5.4.4.2.

5.4.4.2 Saving Colormaps. Colormaps may be saved using the *Save* button from the *View Colormap Window.* A file selection window will be displayed for the user to select a filename for the colormap file to be saved into. All colormap files are automatically given the extension *".cmap"* when saved. The current colormap may also be saved as the default colormap for the dataset. Subsequent viewing of the dataset will use this default colormap to view the data initially.



96TR37

Figure 5-5. Headlight Editor Window

5.5 Headlight Editor

The View function *Headlight Edit* brings up an editor window (see Figure 5-5) to allow the operator to change the directional lighting in the View Window. The View function *Headlight Show* turns ON or OFF the directional lighting in the View Window.

5.6 Copying View

The View function *Copy* makes a copy of a view window. The only difference between the original and the copied View Window is that the copied view's title has a different view number.

5.7 Saving Views

The View function *Save* saves both the view specific view parameters (see Section 5.4.1) and the shared view parameters (cutplane definitions) in a saved view file for later restoration. A file selection window will be displayed for the user to select a filename for the saved view file to be saved under. All saved view files are automatically given the extension *".vew"* when saved.

5.8 Restoring Views

The View function *Restore* restores both the view specific view parameters (see Section 5.4.1), and the shared view parameters (cutplane definitions) from a saved view file, and applies them to the current view. To restore a view, the user selects the saved view from the file list (see Figure 5-6) and clicks on the *OK* button. To dismiss the

ICERVS File Selection Window, the *Cancel* button is pressed. Note that when more than one view exists for the same dataset, those views will automatically take on the same cutplanes as the restored view. Also note that when more than one view exists for the same dataset, those views which have the same property type as the view being restored will automatically take on the same colormap as the restored view.

5.9 Closing a View

The View function *Close* closes the view window. The window may also be closed by double clicking on the upper left corner of the window. If the dataset displayed in the view window has been changed and not saved, the user is given the option of saving the dataset before closing the view.

5.10 Problem Reporting

Because ICERVS is a complex software system, the occurrence of anomalies cannot be precluded. A problem-reporting utility is, therefore, provided to enable the user to report the existence of (ICERVS) system problems or irregularities. To report a problem, the user selects the *Problem Report* function from the *Help* pull down menu on the *View Window.* A text editor window (Figure 5-7) will be displayed wherein the user enters information describing the problem (summary, type, severity, and a detailed description). The user is required to provide identifying information (name, organization, telephone number, and mail address) to facilitate communications between the system developers and the user (should it become necessary in order to resolve the problem). The date and software version number are entered by the system. When the problem report is saved by the user, an e-mail message is sent to the factory and assigned to a technical expert for resolution.

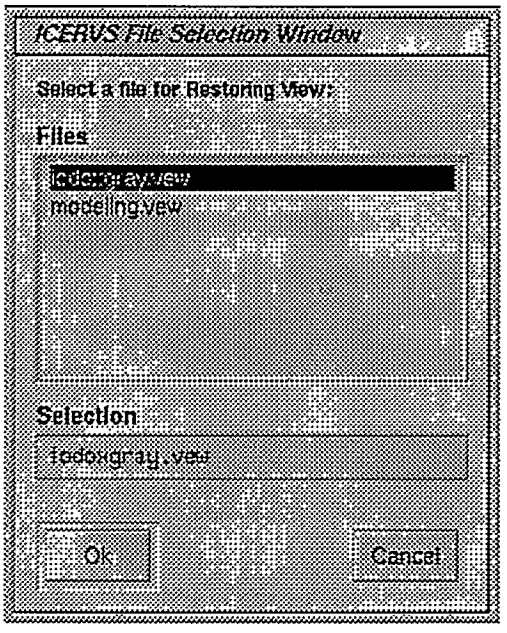
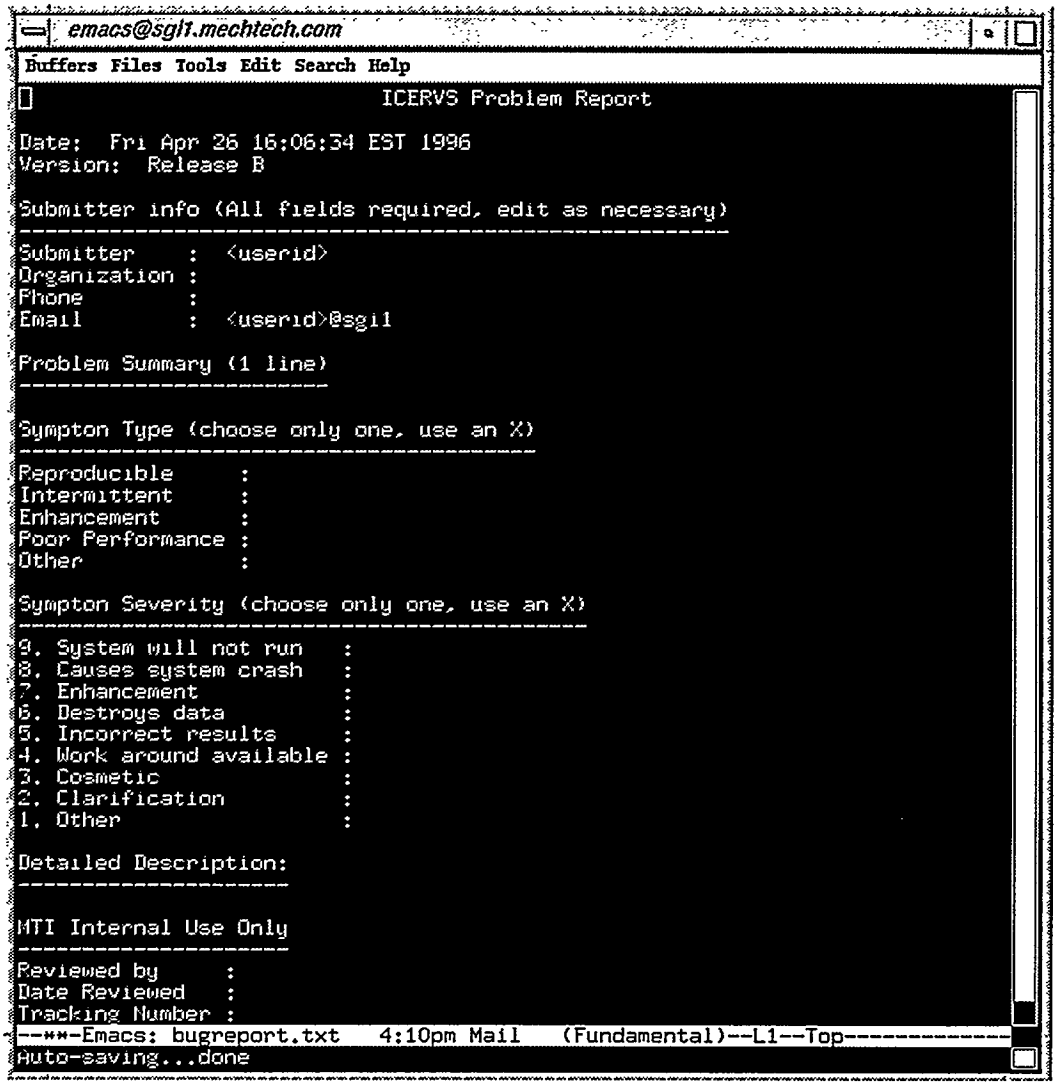


Figure 5-6. ICERVS File Selection Window (Restoring View)



96TR37

Figure 5-7. Problem Report Input Window

5.11 On-Line User Manual

ICERVS contains an on-line HELP facility. The facility is a hypertext based system using HTML. To activate the HELP facility, the user selects the *User Manual* function from the *Help* pull down menu on either the *View Window* or the *DatasetManager Window*. When the function has been selected, a WEB browser is spawned to display the User Manual. The operator may quickly view various sections of the manual by clicking on the table of contents entry for the desired section.

6.0 SENSOR DATA INPUT

6.1 Sensor Data Types

Three types of sensor data may be input into ICERVS: position data, scalar property data, and non-scalar property data.

6.1.1 Position Data

Position data, also referred to as dimensional data or spatial data, is xyz data obtained from sensors and represents a 3-D space. The data universe (or maximum allowable data range) for this type of data is defined in the dataset parameters. If a data point comes in that is outside this universe, it will not be added to the dataset. Position data cannot, at present, be deleted from the dataset unless the entire dataset is deleted.

6.1.2 Scalar Property Data

Scalar property data represents the non-spatial data acquired by the sensors (for example, temperature, density). Each scalar property value is associated with an xyz triplet which represents a dimensional point in 3-D space. Scalar property data is optional, but more than one scalar property type may exist per dimensional data point. There need not be a scalar property value for every dimensional point. Scalar property data is input via the standard ICERVS input file. Up to 14 scalar properties may be defined per dataset, but properties cannot, at present, be deleted from the dataset unless the entire dataset is deleted.

6.1.3 Non-scalar Property Data

Non-scalar property data is intended to provide the user with a means to associate complex data with a spatial data point (xyz). Complex data is any type of data that is more than a single data value. This data can be in the form of text, images, formatted documents, spreadsheets, etc. Non-scalar property data is optional, however, more than one non-scalar property file may exist per dimensional data point. Non-scalar properties are represented in the dataset as data indicators, which are used to show the locations where non-scalar data are stored (see Section 5.4.3.2.1). Non-scalar property data is input either via the standard ICERVS input file (see Section 6.3) or directly via the *View Window's Input* menu functions (see Section 6.4). Non-scalar property data may be deleted via the *Property Display Window* (see Section 5.4.3).

6.2 Sensor Data File Format

The ICERVS sensor data file format is described below in Table 6-1. Some of the fields in the file have been added for future data fusion capabilities and are presently ignored by ICERVS. Other fields in the file are optional and need not be specified. In order to correctly identify the file for ICERVS, the ID parameter "*FileVersion*" should be specified as 2.0. The fields marked with an asterisk (*) are the only ones absolutely necessary for ICERVS. If scalar properties are to be specified, then the additional fields marked with a

double asterisk (**) are also necessary. If non-scalar properties are to be specified, then the additional fields marked with a triple asterisk (***) are also necessary. The rest are optional. See Figures 6-1 through 6-3 for examples of ICERVS input files.

6.3 Inputting Sensor Data Files

Sensor data that are expressed in the ICERVS standard format, as defined in Section 6.2, may be input to a dataset by selecting the *Sensor Data* function from the *Input* pull down on the *View Window* menu bar. A *Sensor Data Input File* selection window, as show in Figure 6-4, is displayed wherein the user enters the name of the file for input. When the *Input* button is clicked, the sensor data is read from file and stored in an octree at a level that is computed based on the user-specified dataset resolution. All views of the dataset are redisplayed to render the newly input data. The input data is permanently stored in the dataset when the user selects the *Save* function from the *Dataset* menu pull down.

6.4 Direct Input of Non-scalar Property Data

6.4.1 Quick Text Data Input

The User may interactively input text and attach it to a dimensional point by selecting the *Non-Scalar - Quick Text* function from the *Input* pull down on the View Window menu bar. An *Input Nonscalar - Quick Text Window* is displayed, as shown in Figure 6-5, within which the user defines the Cartesian coordinates of the dimensional point to which the text is to be attached and enters the text to be input.

To input the non-scalar property text in the dataset and display an indicator at the dimensional point to which the text is attached, the user presses the *Input Nonscalar* button. All dataset views are subsequently redisplayed to render the newly created indicator. The non-scalar property data is permanently stored in the dataset by selecting the *Save* function from the *Dataset* pull-own menu on the *View Window.*

6.4.2 Data Input from File

The user may input a file containing non-scalar data (e.g., text, arrays, images) and attach it to a dimensional point by selecting the *Non-Scalar - From File* function from the *Input* pull down on the View Window menu bar. An *Input Nonscalar - From File Window* is displayed, as show in Figure 6-6, within which the user defines the Cartesian coordinates of the dimensional point to which the data is to be attached.

A *File Selection Window* is displayed wherein the user selects the filename of the non-scalar property data file being input. To input the non-scalar property data in the dataset and display an indicator at the dimensional point to which the data is attached, the user presses the *Input Nonscalar* button. All dataset views are subsequently redisplayed to render the newly created indicator. The non-scalar property data are permanently stored in the dataset by selecting the *Save* function from the *Dataset* pull-down menu on the *View Window.*

Table 6-1. Sensor Data Input Format Specification

Item No.	Data Field	Type	Range	Default Value
1*	**File ID Block Header**	String	[____]	[ID]
2*	File Version Number	Float		2.0
3	File Description	String		
4*	**General Data Block Header**	String	[____]	[GENERAL]
5	Originator Name	String		" "
6	Originator ID	String		" "
7	Operator ID	String		" "
8	Comments	String	#_____	" "
9	Date Acquired	String		" "
10	Facility Name	String		" "
11	Site Name	String		" "
12*	Dimensional Units	String		mm
13***	Path to Non-Scalar Properties	String		" "
14*	Number of Data Points	Int	0 to 2^{32}	0
15*	Resolution	Float		0
16	Transmitter Location - (x, y, z) triplet	(3) float (C3dPoint)		(0, 0, 0)
17	Receiver Location - (x, y, z) triplet	(3) float (C3dPoint)		(0, 0, 0)
18	TS Node Type	Int	1 to 6	1
19	Surface Data Flag	Int	0 = Occupancy, 1= Reference	
20	Line of Sight Sensor Flag	Boolean	0 = False, 1 = True	0
21	**Block Data Fusion Module Header**	String	[____]	[DATA FUSION]
22	Sensor Name / Identifier	String		" "
23	Sensor Type	Int	1 = Generic, 2 = CRC Laser Radar, 3 = Perceptron Laser Radar, 4 = Structured Light	0
24	Number of Environmental Parameters	Int	Non-negative	0
	Repeat Items 25 and 26 for each environmental property included in Item 24			
25	Environmental Parameter Type	List of Ints	1 = Temp., 2 = Humidity, 3 = Reflectivity	0
26	Environmental Parameter Value	List of Floats		
27	Number of Fusion Properties	Int	Non-negative	0
28	Data Fusion Property Indices	List of Ints	Non-negative	0
29**	**Properties Block Header**	String	[____]	[PROPERTIES]
30**	Total Number of Scalar Properties	Int	0-15	0
31**	Number of ICERVS Displayable Properties	Int	0-15	0
32**	ICERVS Property Indices	List of Ints	Non-negative	
	Repeat Items 29 and 30 for each scalar property included in Item 30 count			
33**	Name of Scalar Property	String		" "
34**	Property Units	String		" "
35	**Region Block Header**	String	[____]	[REGION]
36	Number of Vertices	Int	0, 3 to 255	0
37	Front Face Vertex Set	(n) Floats		
38	Rear Face Vertex Set	(n) Floats		
39*	***Data Block Header**	String	[____]	[DATA]
	Repeat Item 40 for each data record included in Item 14 count			
40a*	.x position coordinate	Float		
40b*	y position coordinate	Float		
40c*	z position coordinate	Float		
	Repeat 40d for each scalar property in Item 30			
40d**	Scalar Property Value	Float		
	Repeat 40e for each non-scalar property value			
40e***	Non-scalar property data filename including extension	String		" "

*Required for all files.
**Required for files that include scalar properties.
***Required for files that include non-scalar properties.

96TR37

```
*----------------------------------------
* Waste Tank Demo Data
* Mechanical Technology Inc.
* 05/09/96
*----------------------------------------

[ID]
FileVersion= 2.0

[GENERAL]
DimensionalUnits= METERS
Resolution= 0
NumberOfDataPoints= 5

[PROPERTIES]
NumberOfScalarProperties= 0

 [DATA]
    -1.9232     3.0362    -1.2427
    -1.9386     3.0604    -1.2413
    -1.9558     3.0876    -1.2409
    -1.9739     3.1161    -1.2409
    -1.9918     3.1443    -1.2406
```

96TR37

Figure 6-1. ICERVS Input File (Position Data Only)

```
*----------------------------------------
* Waste Tank Demo Data
* Mechanical Technology Inc.
* 05/09/96
*----------------------------------------

[ID]
FileVersion= 2.0

[GENERAL]
DimensionalUnits= METERS
Resolution= 0
NumberOfDataPoints= 5

[PROPERTIES]
NumberOfScalarProperties= 2
NumberOfICERVSProperties= 1
ICERVSProperties=         2
Property1= Temperature
Units1= DegC
Property2= Radiation
Units2= Rad

 [DATA]
     -1.9232     3.0362    -1.2427    60.2467    .3946
     -1.9386     3.0604    -1.2413    60.2365    .3948
     -1.9558     3.0876    -1.2409    60.2856    .3937
     -1.9739     3.1161    -1.2409    60.2464    .3956
     -1.9918     3.1443    -1.2406    60.2587    .3963
```

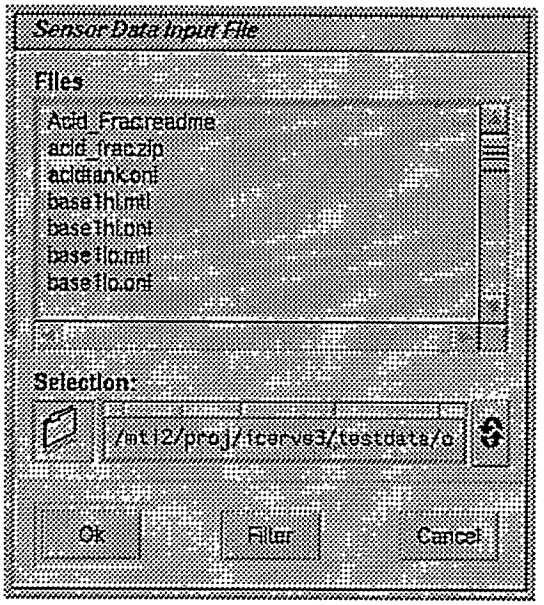
96TR37

Figure 6-2. ICERVS Input File (Position Data and Two Scalar Properties) (Display only the second property)

```
*-----------------------------------------
* Waste Tank Demo Data
* Mechanical Technology Inc.
* 05/09/96
*-----------------------------------------

[ID]
FileVersion= 2.0

[GENERAL]
DimensionalUnits= METERS
Resolution= 0
PathToNonScalars= /usr/local/
NumberOfDataPoints= 5

[PROPERTIES]
NumberOfScalarProperties= 2
Property1= Temperature
Units1= DegC
Property2= Radiation
Units2= Rad

 [DATA]
    -1.9232    3.0362    -1.2427    60.2467    .3946    image.gif
notes.txt
    -1.9386    3.0604    -1.2413    60.2365    .3948
    -1.9558    3.0876    -1.2409    60.2856    .3937
    -1.9739    3.1161    -1.2409    60.2464    .3956    numbers.arr
    -1.9918    3.1443    -1.2406    60.2587    .3963
```

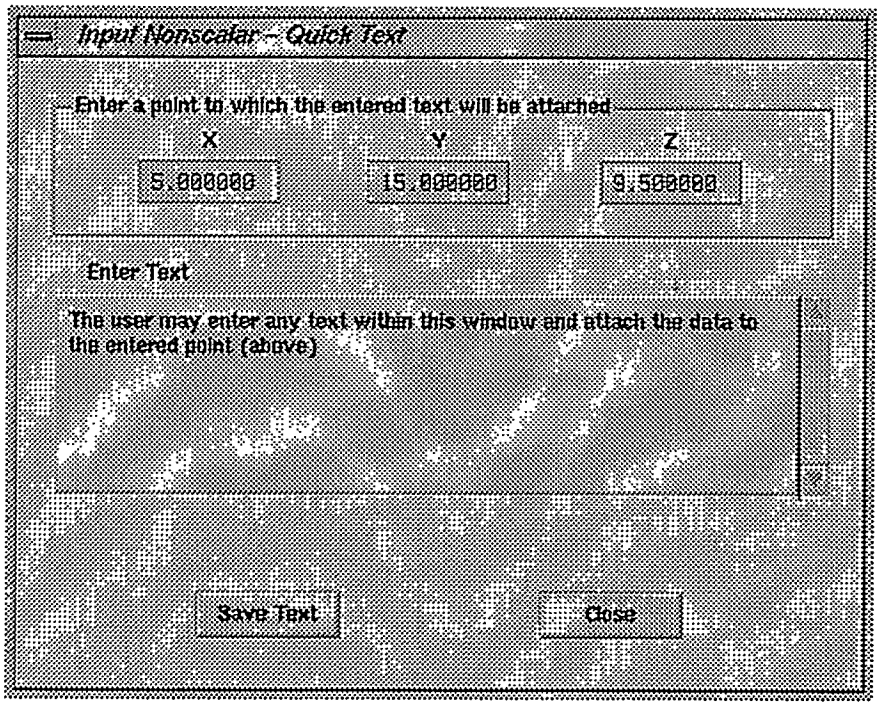
96TR37

Figure 6-3. ICERVS Input File (Position Data, Two Scalar Properties, Non-scalar Properties)



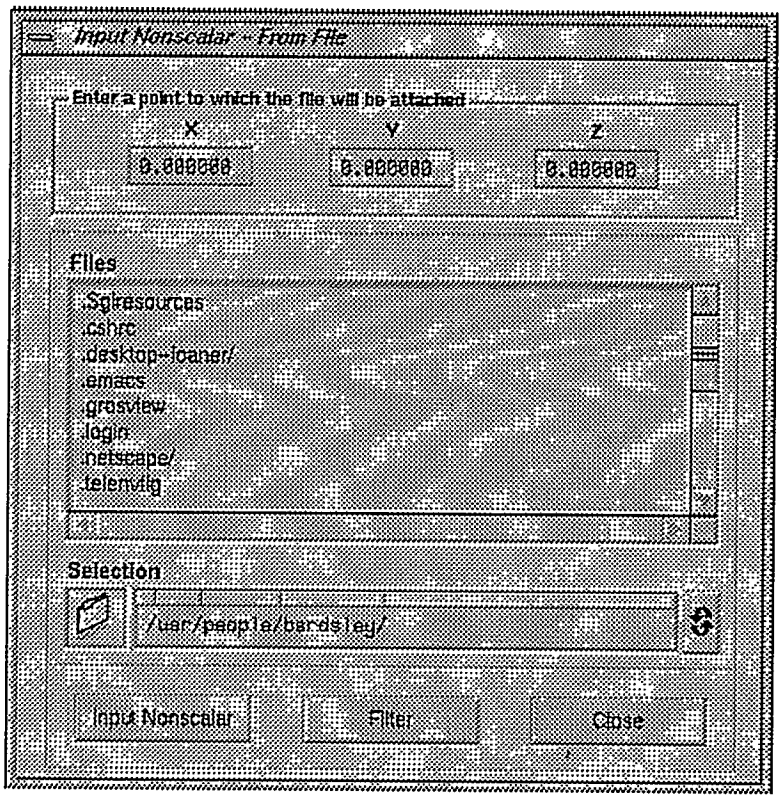
96TR37

Figure 6-4. Sensor Data Input File Window



96TR37

Figure 6-5. Quick Text (Non-scalar Data) Input Window



96TR37

Figure 6-6. Input (Non-scalar Data) From File Window

7.0 GEOMETRIC OBJECTS

Geometric objects are supported in ICERVS for modeling objects which exist in the data. The following sections describe the type of objects supported as well as the functionality of these objects in the system.

7.1 Supported Object Types

ICERVS supports three geometric object types as described in Section 3.2.2: geometric primitive (spheres, cones, cylinders, boxes), polyhedron, and composite.

7.2 Interactively Creating Objects

The *Object* menu pull-down *Create* function supports the creation of geometric objects by the user. When selected, the *Object Creation* Window, shown in Figure 7-1, is displayed wherein the user defines the objects to be created.

To create an object, the user first selects the type (see Section 7.1) of object desired.

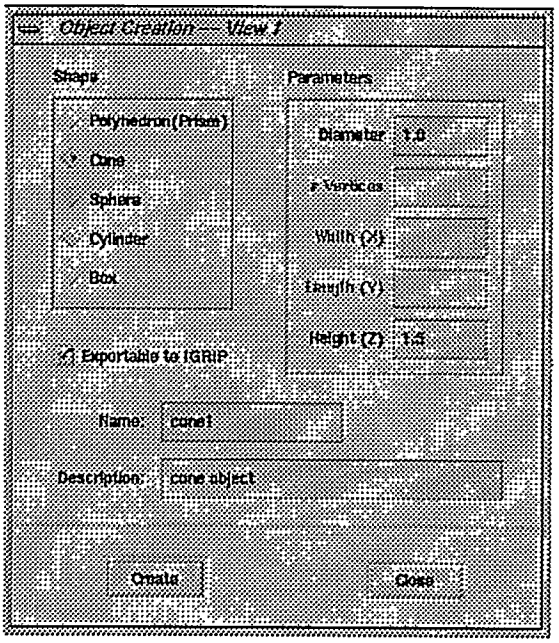
Based upon the object shape selected, the applicable parameter fields will be highlighted in the *Parameters* box on the window. The user defines the values in units consistent with those specified for the group (see Section 4.3.2) for each required parameter and specifies an object name and (optional) description. By default, the "Exportable to IGRIP" flag is set (to True). The user may mouse click on the flag to change its value. Only those objects with the export flag set will be exported to IGRIP.

Clicking on the *CREATE* button creates the object in the center of the View Window. All views of the dataset are redisplayed to render the created object. The user may then transform (see Section 7.4.2) the object as necessary. The created object(s) is not made a permanent part of the dataset until the *Dataset Save* function is performed. The *Object Creation* Window is dismissed by clicking the *Close* button.

7.3 Selecting Objects

The selection of a single object or a set of objects in the view is performed by one of two approaches: mouse selection or menu selection.

In pick mode, a single object in the view may be selected by placing the mouse over the object and clicking the left mouse control button. Multiple object selection is performed by holding down the SHIFT button on the keyboard, positioning the mouse on each object individually, and clicking the left mouse control button. Objects are unselected by clicking the left mouse control button in a region of the view devoid of any geometric objects and spatial data points.



96TR37

Figure 7-1. Object Creation Window

Objects may also be selected from a menu list by selecting the *Select* function from the *Object* pull-down menu. This approach facilitates the selection of objects in the view that are obscured by either other objects and/or spatial data. It is also more expedient for selecting large number of objects. The Object Selection Window, shown in Figure 7-2, displays the list of the dataset object names, from which the user may select.

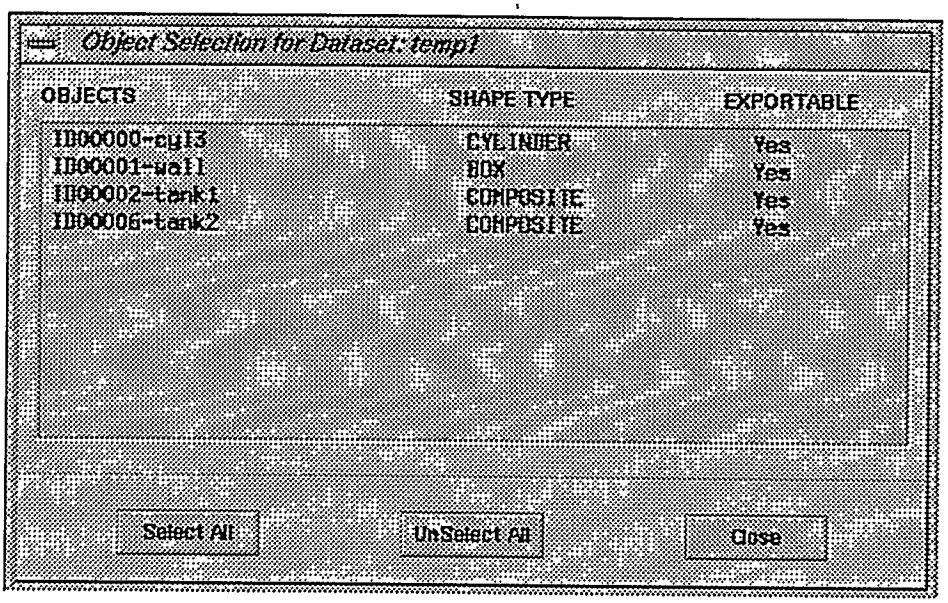
Additional information pertaining to the object shape type and the state of its "Exportable to IGRIP" flag is provided for each object. As an object is selected, it is highlighted in the View Window. Buttons for *Select All* and *Unselect All* are provided to select or deselect all objects in the View Window. The objects selected in the *Object Selection* Window are highlighted by a red bounding frame in the View Window. Clicking on the *Close* button dismisses the *Object Selection* Window.

7.4 Editing Objects

In addition to the interactive editing of objects via the manipulator buttons located on the left side of the View Window (see Section 5.2.2), geometric objects may be either edited parametrically (i.e., length, width, number of vertices, etc.), or transformed via a transform editor.

7.4.1 Parametric Editing

The *Object* menu pull-down *Edit - Parametric* function supports the parametric editing of geometric objects by the user. When selected, the *Object Editing* Window, shown in Figure 7-3, is displayed wherein the user edits the objects.



96TR37

Figure 7-2. Object Selection Window

The object in the window being edited directly reflects the last object picked with the mouse. The name of this object is displayed as part of the title on the window. If no objects are selected for editing, the object name will be "NONE".

As an object is picked, its shape type and name are displayed on the window to indicate the current object being edited. These fields are for information only and may not be changed. The *Parameters* fields, *Exportable to IGRIP* flag, and *Description field* are also displayed for the current object being edited. These fields may be edited by the user as desired.

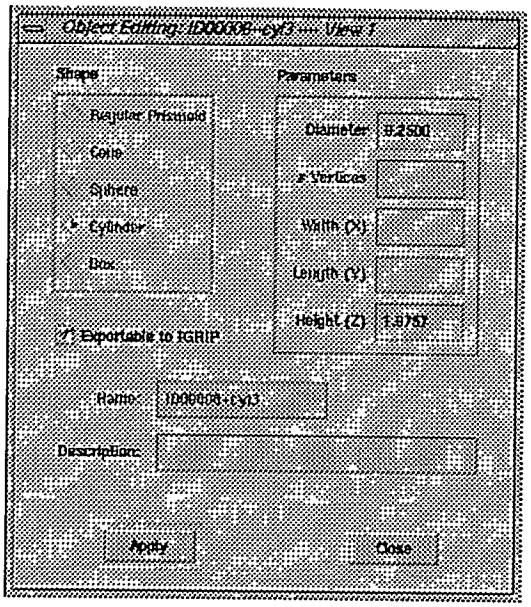
Clicking on the *Apply* button applies the editing changes to the object in the View Window. All views of the dataset are redisplayed to render the edited object. The edited object(s) is not made a permanent part of the dataset until the *Dataset Save* function is performed. The *Object Editing* Window is dismissed by clicking the *Close* button. The changes to the object are not saved in the dataset until the *Dataset Save* function is performed.

7.4.2 Transforming Objects

The *Object* menu pull-down *Transform* function supports the transformation of geometric objects by the user. When selected, the *Object Transform* Window, shown in Figure 7-4, is displayed wherein the user transforms the objects.

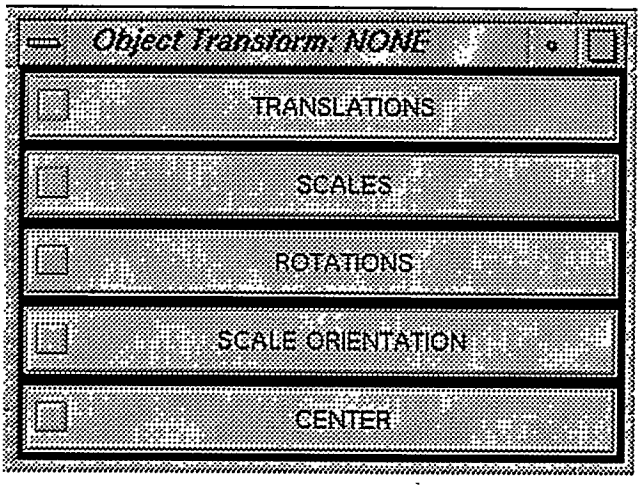
The object in the window being transformed directly reflects the last object picked with the mouse. The name of this object is displayed as part of the title on the window. If no objects are selected for transformation, the object name will be "NONE."

As an object is picked, its transformation values (i.e., translation, scaling, rotation, scale orientation, and center) are displayed on the window to indicate the current object being transformed. These values may be edited by the user as desired.



96TR37

Figure 7-3. Object Editing Window



96TR37

Figure 7-4. Object Transform Window

Transform values can be changed either by entering new transform values or moving the slider bars to interactively transform the object in the View Window. All views of the dataset are redisplayed to render the transformed object. The transformed object(s) is not made a permanent part of the dataset until the *Dataset Save* function is performed. The *Object Transform* Window is dismissed by double clicking with the mouse on the upper-left button on the window. Changes to the object are not saved in the dataset until the *Dataset Save* function is performed.

7.4.3 Reshaping Polyhedral Objects

Arbitrary polyhedron objects may be interactively reshaped using the *Reshape* manipulator button on the left side of the View Window. This is a custom manipulator which allows moving an object's face or vertex to an arbitrary location.

When the *Reshape* manipulator is first clicked on, the manipulator is not automatically attached to the object (unlike the other manipulators which visually change when the manipulator button is clicked on). The user must first select a vertex or face of the object with a mouse click in order to see the manipulator on the object.

If a vertex was selected, the manipulator is attached directly to that vertex. Only that vertex is changed when the manipulator is moved. As a result, all faces that share that vertex are changed. If a face was selected, the dragger is attached to the center of that face. All vertices of the selected face are moved when the manipulator is moved. As a result, any faces attached to the edited face will be changed. Holding down the *SHIFT* key while using the *Reshape* manipulator will constrain motion to a single axis. Any changes to the object are not saved in the dataset until the *Dataset Save* function is performed.

7.4.4 Snapping Objects to Points

Selected geometric objects may be "snapped" to a point in the spatial data. This is a good way to quickly move a newly created object to the correct location. (Newly created objects are centered in two dimensions, but the third dimension can be offset by quite a bit.) The *Object* menu pull-down *Snap to Point* function supports the quick movement of geometric objects by the user. The operator must have previously picked a point in the data with the mouse. The currently picked point will be displayed in the status window under "Data Coordinate" (see Section 5.2.1). The operator must also have one or more objects highlighted. When the *Snap to Point* menu function is clicked, the currently highlighted objects are translated to the currently picked point (object rotational orientation is not affected).

7.5 Coloring Objects

The *Object* pull down menu *Color* function generates an Inventor *Object Color* Editor Window, as shown in Figure 7-5, for the user to select the color of a selected object.

The object in the window being colored directly reflects the last object picked with the mouse. The name of this object is displayed as part of the title on the window. If no objects are selected for coloring, the object name will be "NONE."

As an object is picked, its color is displayed on the *Object Color* window to indicate the current object being colored. This color may be edited by the user as desired by placing the mouse on the color wheel and dragging to the desired color. The *Object Color* Window is dismissed by double clicking with the mouse on the upper-left button on the window. Any changes to the object are not saved in the dataset until the *Dataset Save* function is performed.

7.6 Setting Object Attributes

The attributes of the objects on the View Window may be customized using the *Attribute* function of the *Object* pull-down menu. An *Object Attributes* Window, as shown in Figure 7-6, is displayed wherein the user may specify, for individually selected objects, its display type (e.g., wireframe, solid, semi-transparent, invisible), the rendering complexity (applies to sphere, cone, and cylinder objects only), and the state of its "Exportable To IGRIP" flag.

The object in the window having its attributes set directly reflects the last object picked with the mouse. The name of this object is displayed as part of the title on the window. If no objects are selected for coloring, the object name will be "NONE."

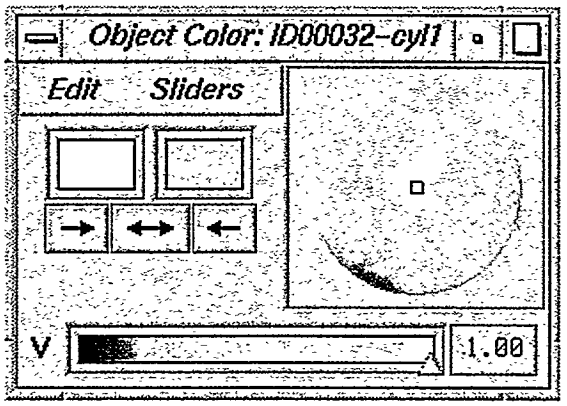
As an object is picked, its attributes are displayed on the *Object Attributes* window to indicate the current object having its attributes set. The user may change any of the attributes for that object. When the *Apply* button is clicked, the new object attributes take effect. All dataset views are refreshed to render the objects according to the newly defined attributes. The *Object Attributes* Window is dismissed by double clicking with the mouse on the upper-left button on the window. Any changes to the object are not saved in the dataset until the *Dataset Save* function is performed.

Note that when multiple objects are built into a composite object, the components retain their initial attributes until the attributes for the composite object are specified. At such time, the component object attributes are redefined to inherit the values specified for the composite. For example, a new composite object will be rendered with the display type of solid (by default), even though its components' display types may be defined as semi-transparent.

If the composite's attributes have not been redefined and the composite is unbuilt, the individual components will have the same attributes that they did before they became composites. However, if the composite's attributes have been redefined, then the component's attributes have also been redefined to match the composite's attributes. When the composite is unbuilt, the individual components will have different attributes than they did before they became composites.

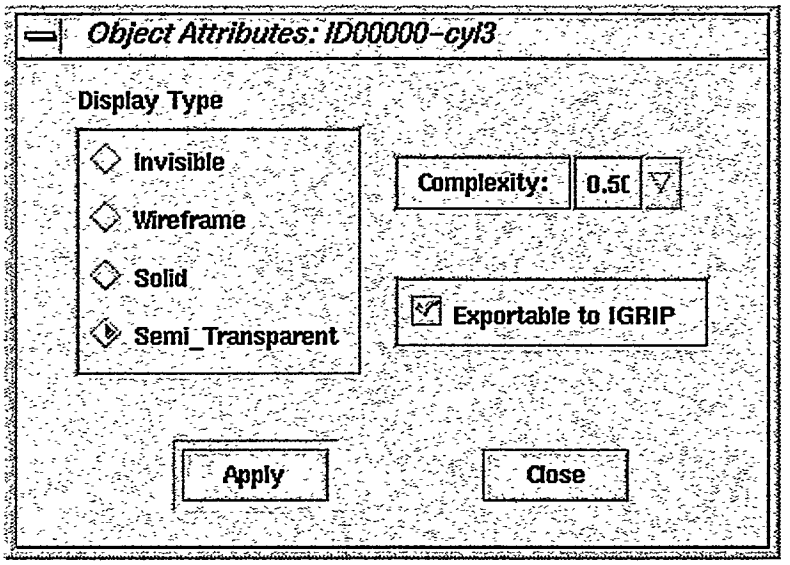
7.7 Composite Objects

Geometric objects may be joined together to become a composite object. Once objects become composites, they become components of the composite object and loose their identity as unique objects. This feature is often used to build larger, more complex shapes out of simpler shapes. Composite objects can be unbuilt, at which time, the component objects once again take on their own unique identity as separate objects.



96TR37

Figure 7-5. Object Color Wheel



96TR37

Figure 7-6. Object Attributes Window

7.7.1 Building Composite Objects

A composite object may be built from a set of selected objects in the View Window using the *Object* menu pull-down *Composite - Build* function. The component objects must be pre-selected by the user. Once built, the composite is rendered on the dataset view(s) as one object, bounded by a single red frame. The Object Selection list (if displayed) is updated to contain the composite object in the place of its individual components. By default, the attributes for composite objects are defined as: solid (display type), 0.5 (complexity), and active ("Exportable to IGRIP" flag). The composite object is not saved in the dataset until the *Dataset Save* function is performed.

7.7.2 Unbuilding Composite Objects

A selected composite object may be unbuilt into its components by selecting the *Composite - Unbuild* function from the *Object* pull-down menu. Once performed, all dataset views are redisplayed to render the set of individual components. The Object Selection list (if displayed) is updated to contain the individual component objects in the place of the (unbuilt) composite object. The "Export to IGRIP" flag setting is set for each component object to the same state as that defined for the composite. The result of a composite object unbuild is not saved in the dataset until the *Dataset Save* function is performed.

7.8 Object Library

A set of Object Libraries can be created to provide a universal set of geometric shapes which may be used among all groups and dataset. For example, one library may contain a set of barrels, and another a set of pipes, etc. The Object Library may be accessed using the *Library* function of the *Object* pull-down menu. An *Object Library* window, as shown in Figure 7-7, is displayed wherein the user may perform function on the library such as Store Library Object, Retrieve Library Object, Delete Library Object, Create New Library, Delete Library. The left-hand side of the window deals with functions which pertain to the library, and contains a selection list of all current object libraries. The right-hand side of the window deals with functions which pertain to the objects in the selected library, and contains a selection list of all current object in the currently selected library. Clicking on the *Close* button dismisses the *Object Library* Window.

7.8.1 Creating Object Libraries

New Object Libraries can be created using the *Library* function of the *Object* pull down menu. An *Object Library* Window, as shown is Figure 7-7, is displayed wherein the user may create a new library in which to store library objects. The user clicks on the *New* button under the *Libraries* section of the *Object Library* Window. *The Object Library New* Window, as shown is Figure 7-8, is displayed for the user to enter the name of the new library being created. This name is used to name the directory in which new library objects will subsequently be stored. When the name is entered, the new library appears in the library selection list.

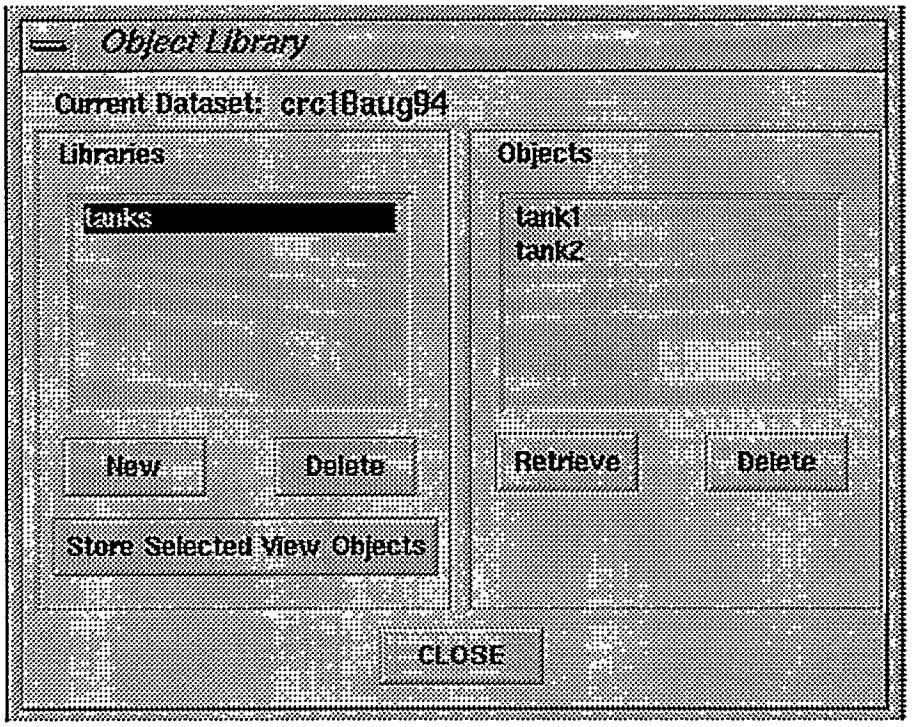


Figure 7-7. Object Library Window

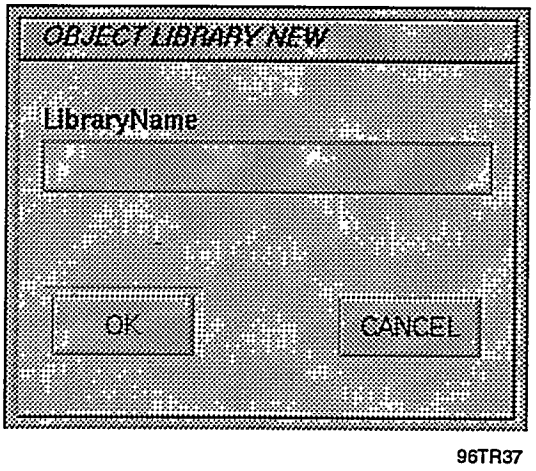


Figure 7-8. Object Library New Window

7.8.2 Storing Objects

Objects may be stored to the object library for subsequent retrieval and/or reuse. The objects in the view to be stored are first selected (see Section 7.2), followed by the *Library* function on the *Object* menu pull-down. The *Object Library* Window (see Figure 7-7) is displayed. From the *Libraries* list on the left side of the window, the user selects the library name within which the objects are to be stored (see Section 7.8.1 for information on how to create a new library). The *Store Selected View Objects* button is pressed. For each selected object, the user is prompted for a name under which the object will be stored. The *Objects* list, on the right side of the window, updates to display the names of the stored objects in the *Objects* list, thereby confirming their storage. The *Object Library* Window is dismissed by clicking the *Close* button. The stored object remains an object in the object library unless explicitly deleted.

7.8.3 Retrieving Library Objects

Objects that have been previously stored in the Object Library may be subsequently retrieved into a dataset by selecting the *Library* function from the *Object* pull-down menu. The *Object Library* Window is displayed. The user selects the library, from the *Libraries* list on the left side of the window, in which the object exists. Once selected, the library name will be highlighted and the objects stored in the specified library will be displayed in the *Objects* list on the right side of the window. The user selects the objects to be retrieved and clicks the *Retrieve* button. The retrieved objects will appear in the center of the View Window and will be given an unique identifier that includes the name by which they were stored. The *Object Library* Window is dismissed by clicking the *Close* button. The retrieved library object is not stored permanently in the dataset until a *Dataset Save* function is explicitly performed.

7.8.4 Removing Library Objects

Objects that have been previously stored in the Object Library may be permanently deleted by selecting the *Library* function from the *Object* pull-down menu. The *Object Library* Window is displayed. The user selects the library, from the *Libraries* list on the left side of the window, in which the objects to be deleted exist. Once selected, the library name will be highlighted and the objects stored in the specified library will be displayed in the *Objects* list on the right side of the window. The user selects the objects to be deleted and clicks the *Delete* button on the *Objects* side of the window. The deleted objects will disappear from the *Objects* list. The *Object Library* Window is dismissed by clicking the *Close* button.

7.8.5 Deleting Object Libraries

Object libraries that have been previously created may be permanently deleted, along with all library objects that are stored in the object library, by selecting the *Library* function from the *Object* pull-down menu. The *Object Library* Window is displayed. The user selects the library to be deleted from the *Libraries* list on the left side of the window. Once selected, the library

name will be highlighted, and the objects stored in the specified library will be displayed in the *Objects* list on the right side of the window. The user clicks the *Delete* button on the *Libraries* side of the window. The deleted library will disappear from the *Libraries* list. The *Object Library* Window is dismissed by clicking the *Close* button.

7.9 Deleting Objects

Selected objects may be deleted from a dataset view using the *Delete* function on the *Object* pull-down menu. A *Confirmation Message* Window is displayed to obtain the user's verification prior to the deletion of the objects from the dataset. Upon confirmation, all dataset views are redisplayed void of the deleted objects. Object deletion is not permanent until the *Dataset Save* function has been performed.

7.10 Duplicating Objects

Selected objects may be duplicated from a dataset view using the *Duplicate* function on the *Object* pull-down menu. When the *Duplicate* function is clicked, all dataset views are redisplayed with the newly duplicated object. Object duplication is not permanent until the *Dataset Save* function has been performed.

8.0 ANALYSIS FEATURES

8.1 Region Selection

Region Selection allows the operator to select a region of data by using predefined geometric objects to encompass the desired data. This capability allows large areas of bad data or undesirable data to be erased. It also allows a selected region of points to be dumped to an ASCII file.

A *Region Selection Window,* as shown in Figure 8-1, is displayed wherein the user may select the objects to be used for bounding the data. Multiple objects may be selected for both inclusion and/or exclusion of data in the defined region of interest. Once bounding objects are selected, the *Select Region* button is clicked to highlight the data points enclosed in the selected region. The user may then opt to unselect the highlighted points or to actually delete the highlighted points from the dataset. Erasing points can be done either from the *Region Selection Window,* or from the *Analysis - Erase Region* menu function. The deleted points are not permanently saved in the dataset until the *Dataset Save* function is performed. The selected region's points may be dumped to an ASCII file by clicking the *Analysis - Print Region* menu function.

8.2 Connectivity

This analysis function allows the operator to highlight all data points that are stored in connected voxels in the dataset's underlying grid (octree). *Please describe what is meant by this sentence - see DOE reviewer comments.* This capability might allow the user to visualize objects in the data if the data points which make up the object are dense enough.

The user must have first picked a point with the mouse previous to this function (See Section 5.4.4). This picked point, which is displayed in the status window, is used as the seed point for the connectivity function. When the *Analysis - Set Connectivity* menu function is clicked, all points touching the seed point, or all points touching a point which touches the seed point will be highlighted. The number of connected points is displayed in the status window's message area. The connected points may be unconnected at any time by clicking on the *Analysis - Clear Connectivity* menu function.

8.3 Erasing Invisible Points

This analysis function allows the operator to delete all data points which are set to invisible by the view colormap scheme. This capability allows ranges of bad or undesirable data to be erased.

The user must have first colormapped a range of data to be invisible (see Section 5.4.4). When the *Analysis - Erase Invisible Points* menu function is clicked, all invisible data points will be deleted from the dataset. The number of deleted points is displayed in the status window's message area. The deleted points are not permanently removed from the dataset until the *Dataset Save* function is performed.

8.4 Coloring By Range

This analysis function allows the operator to color all data points within the current view by range from the virtual camera position using the currently defined colormap. This capability allows the operator to visualize the data by depth from the perspective of the current virtual camera position.

When the *Analysis - Set Color By Range* menu function is clicked, the data are redisplayed colored by range from the camera. The coloring remains the same even if the viewpoint changes. The *Analysis - Set Color By Range* menu function must be clicked again if coloring from another viewpoint is desired. The original colormap can be redisplayed by clicking the *Analysis - Clear Color By Range* menu function.

8.5 Cutplanes

Through the application of cutplanes, selected regions of the view may be rendered as invisible. This capability facilitates the visualization of spatial data and/or geometric objects that may be entirely or partially obscured.

The *View* function *Cutplane* allows the user to define and position up to five cutplane/sliceplane combinations to be positioned in the view. The enabled cutplanes are displayed in the View Window by selecting the *View* function *Cutplane Show.*

The following cutplanes types are supported:

Cutplane - A cutplane is a half space. Data on one side are visible and data on the other side are not.

Sliceplane - A sliceplane is composed of two facing half-spaces separated by user-setable distance. The two half-spaces move together and maintain the specified separation distance as they are moved.

The definition and enablement of cutplanes is performed in the *View Cutplane* Window, shown in Figure 8-2. To create a cutplane, the user must perform the following steps:

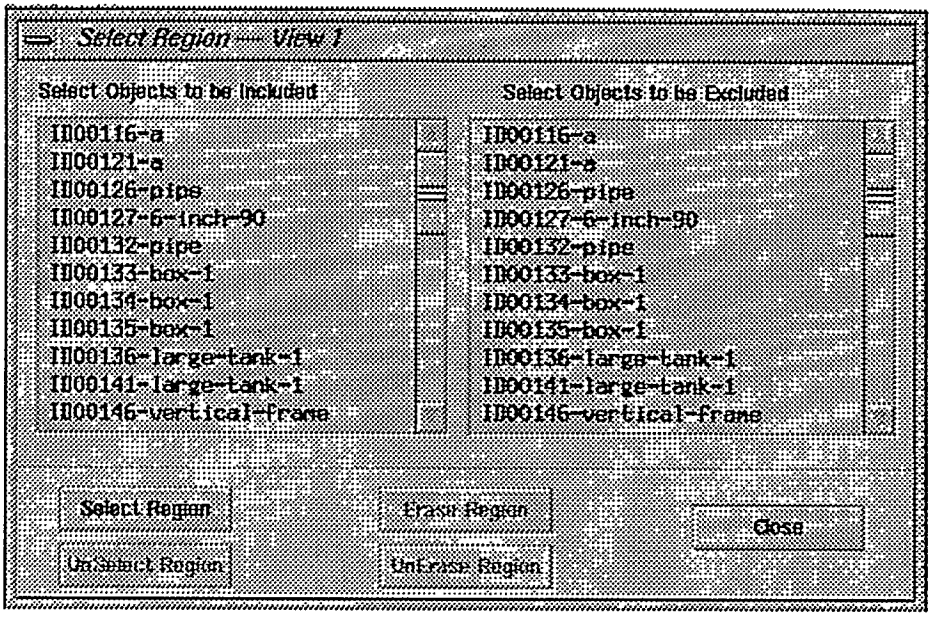
Step 1: Enable the cutplane by mouse clicking on the *Enable* button

Step 2: Mouse click on the *Manipulate* button to transform the cutplane. The selected cutplane may be translated, rotated, and scaled using the *Transform Editor Window* in the same manner used to transform a geometric object (see Section 7.3.2)

Step 3: Select the type of cutplane desired (cutplane or sliceplane)

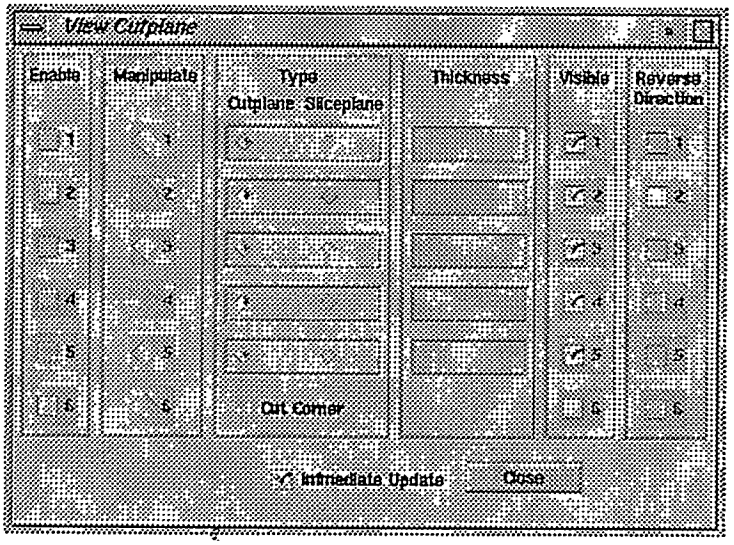
Step 4: Specify a thickness (for sliceplane only).

All views of the dataset for which cutplane display has been enabled (with the *Cutplane Show* function) are immediately redisplayed with the effect of the active cutplanes. The cutplane objects each appear in a unique color that matches the color of the corresponding active check box on the *View Cutplane* Window. When changes are made to any cutplane, all views of the dataset with cutplanes on are redisplayed to show the updated cutplanes.



96TR37

Figure 8-1. Region Selection Window



96TR37

Figure 8-2. View Cutplane Window

9.0 IGRIP INTERFACE

IGRIP (Interactive Graphics Robot Instruction Program) is a software system originally developed by Deneb Robotics to facilitate off-line programming of robotic systems. IGRIP provides an interactive, 3-D graphic simulation tool for design, evaluation, and analysis so that manufacturing processes may be constructed, programmed, and analyzed for cycle time, collisions, and motion constraints. ICERVS provides an interface to IGRIP by supporting the export of ICERVS geometric objects to an IGRIP part file as well as the import of IGRIP part files into ICERVS geometric objects. Further IGRIP information can be obtained from:

Deneb Robotics, Inc.
3285 Lapeer Road West
Auburn Hills, MI 48321
(810)377-6099
http://www.deneb.com

9.1 Interface Mechanism

A software interface has been developed that supports the import (i.e., input) and export (i.e., output) of geometric objects between the ICERVS and the Deneb IGRIP software. The (IGRIP) "part file" format is a data representation used (internally) by Deneb to transfer of data to/from disk from/to its graphic display. ICERVS has implemented this format in its data exchange interface

9.2 Supported Part File Versions/Format

The ICERVS *IGRIP-Output* function produces a text file consistent with Deneb's version 11 part file format. Prior to output, ICERVS geometric objects are converted to arbitrary polyhedrons. With the limitations discussed in Section 9.3, part files of version 11 and older may be input to ICERVS.

9.3 Assumptions/Limitations

The ICERVS - IGRIP interface does not currently support the following IGRIP entities/structures:

1. Multiple Coordinate Systems
2. B-Spline Surfaces
3. B-Spline Curves
4. Text
5. Object Texture Maps
6. Concave Bounded Planes

9.4 Import of Objects to IGRIP

The *Object* function *IGRIP - Import* allows the user to import an IGRIP part file into the current dataset. From the *IGRIP Import* (File Selection) *Window*, shown in Figure 9-1, the user selects the file(s) to be imported. When the *OK* button is clicked, all dataset views are redisplayed to include the imported object(s). The *IGRIP Import Window* is dismissed by pressing the *Cancel* button.

Imported objects are assigned a name and a type by ICERVS. The name is composed of a unique ID number and (a portion of) the file name from which the object was imported. The object type is defined to be either an arbitrary polyhedron (in the case where only one object was contained in the part file) or as a composite (see Section 7.7). Input composite objects may be unbuilt (see Section 7.7.2) to form a set of arbitrary polyhedrons. All imported objects are tagged as "exportable to IGRIP." When a *Dataset Save* is performed, the imported objects are permanently stored in the object dataset.

9.5 Export of Objects to IGRIP

The *Object* function *IGRIP - Export* allows the user to export selected object(s) on the View Window, which have their "Exportable to IGRIP" flag set, to a IGRIP-compatible part file. The user enters the name of the part file to which the exported object definitions will be written in the *IGRIP Export* (File Selection) *Window* (see Figure 9-2). When the *OK* button is clicked, all selected objects are written to the specified IGRIP part file. The *IGRIP Export Window* is dismissed by pressing the *Cancel* button.

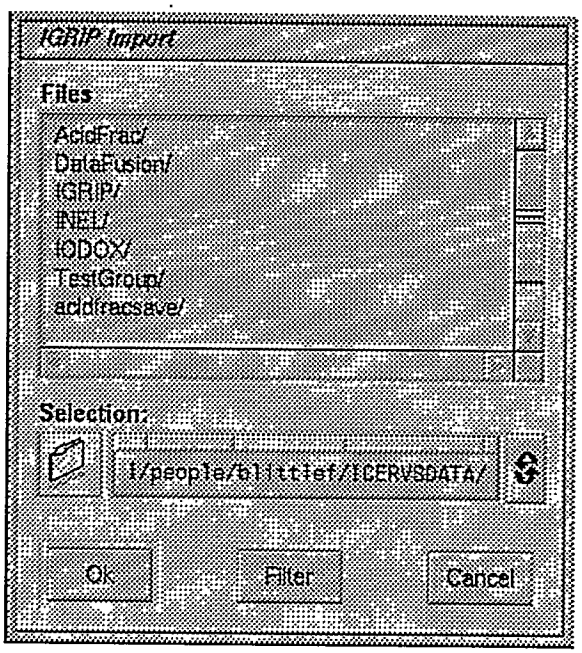


Figure 9-1. IGRIP Import Window

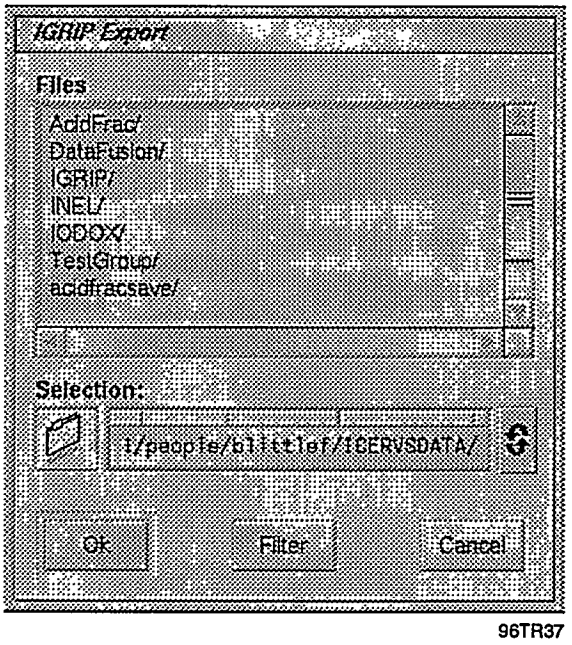


Figure 9-2. IGRIP Export Window

10.0 MAJOR CLASS DESCRIPTIONS

Figure 10-1 illustrates the major classes that comprise ICERVS. Each block contains the name of an object, and a path to the object's header file. The path is relative to the root of the ICERVS source tree. In the diagram, closely related objects have been grouped by a dotted box. Table 10-1 describes each of the objects in the diagram, and is grouped in the same manner. Complete listings of the header files can be found in Appendix A.

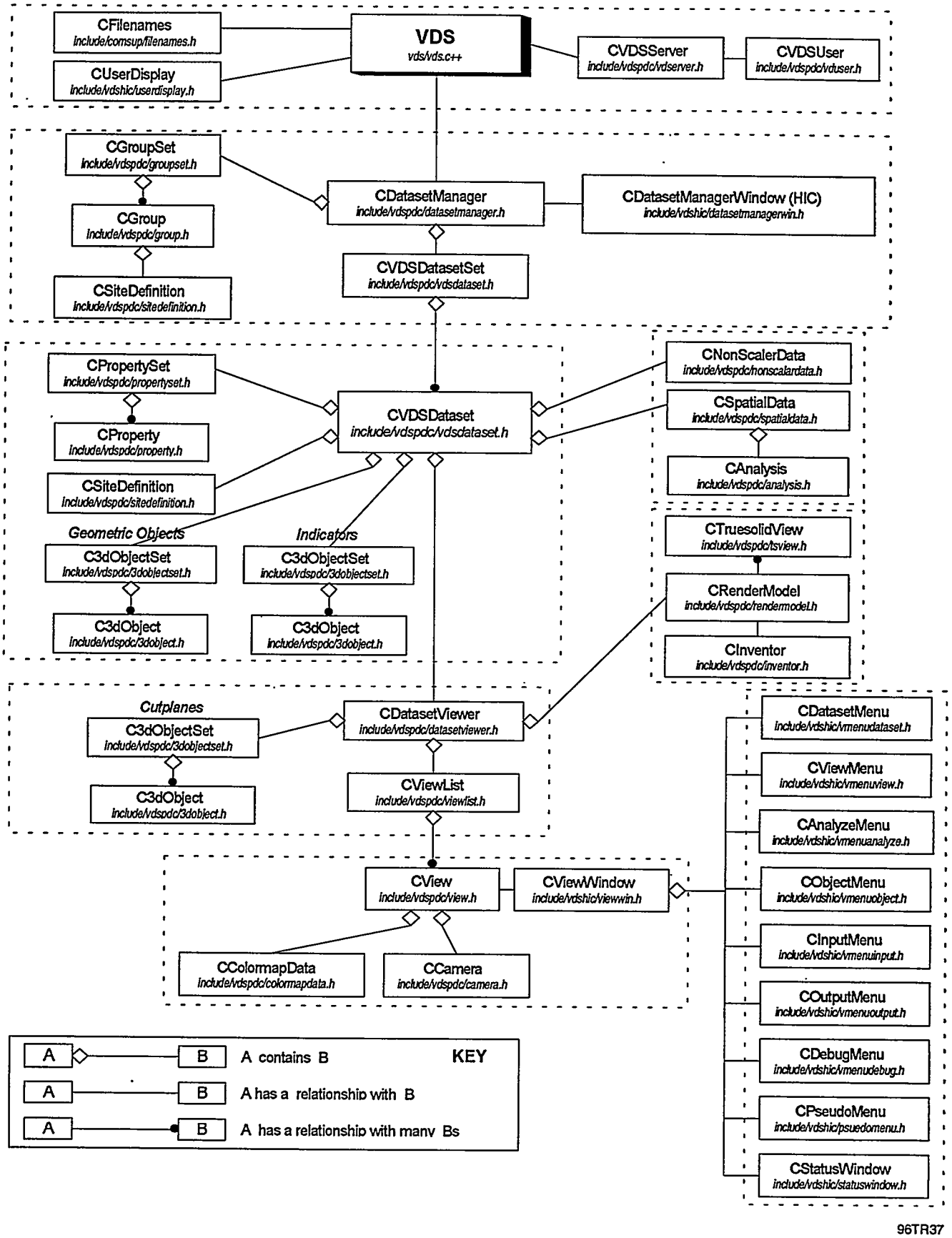


Figure 10-1. Major Classes of ICERVS

Table 10-1. ICERVS Major Classes

ICERVS	
CFilenames	A static object that contains paths to all parts of the execution environment. The contents of this object can be accessed by any other object using scope extension.
CUserDisplay	A static object that contains application-wide information for the graphical user interface. The contents of this object can be accessed by any other object using scope extension.
CVDSServer	This is an object that encapsulates the server and allows clients to communicate with VDS.
CVDSUser	This is an object that a server uses to maintain a client connection. A client object is created for each client.

Dataset Manager	
CDatasetManager	An object which encapsulates all dataset functionality (i.e., creating, deleting, copying, archiving, viewing).
CDatasetSet	An object which contains a list of pointers to all CVDSDataset objects.
CGroupSet	An object which contains a list of pointers to all CGroup objects.
CGroup	An object which encapsulates a dataset group. A group defines a collection of datasets.
CSiteDefinition	An object which defines the site definition for the group (i.e., description of the data universe).
CDatasetManagerWindow	An object which encapsulates all user interface with the DatasetManager. This HIC class defines the Dataset Manager Window, its behaviors, and its functionality.

Dataset	
CVDSDataset	An object which encapsulates all the data acquired from, or constructed to describe, a 3-D workspace.
CPropertySet	This contains a list of pointers to CProperty. The set consists of 1-n properties.
CProperty	This describes a single property. Such attributes as property name, minimum, maximum, etc., are available.
CSiteDefinition	An object which defines the site definition for the dataset (i.e., description of the data universe).
C3dObjectSet	This object contains a list of pointers to a C3DObject objects. The dataset has two such sets. One contains a set of Geometric Objects, and the other contains a set of Data Indicators. Each set can consist of 0-n elements.
C3dObject	This is the base class for all geometric objects that ICERVS uses. The specific geometric object types are: C3dBox, C3dComposite, C3dCone, C3dCylinder, C3dIndicator, C3dPolyhedron, C3dSphere, and C3dCutplane.
CNonScalarData	An object to encapsulate all non-scalar data attributes and functionality.
CSpatialData	An object to encapsulate all volumetric data attributes and functionality. CSpatialData utilizes a TrueSolid Octree library provided by Octree Corp.
CAnalysis	An object to encapsulate all volumetric data analysis functions.

Dataset Viewer	
CDatasetViewer	This object describes and contains "view attributes" that are common among all view windows for a dataset. It contains a number of other objects to achieve this goal, and their descriptions follow. The class was established to handle the problem with cutplanes. Cutplanes cut the data in the dataset, but can be defined differently for each view of the dataset. To handle the dilemma of whether cutplanes should be part of the dataset or the view, the DatasetViewer was created as an interface class between the CVDSDataset and Cview. This class would hold all data that is common among all view windows for a dataset.
C3dObjectSet	This object contains a list of pointers to C3DObject objects. The Dataset Viewer's object set contains a set of 6 cutplanes.
C3dObject	A cutplane object (either cutplane or sliceplane)
CViewList	This object contains the list of open views (view windows) for the dataset.

96TR37

Table 10-1. Continued

View Window	
CView	This object describes and contains "view attributes" that are unique to a particular view of the dataset. It contains a number of other objects to achieve this goal, and their descriptions follow.
CColormapData	This object describes how a property value is colored in the view. A primary and secondary colormap can be applied to the property. The object contains attributes such as min/max range value, min/max range color, etc.
CCamera	This object contains information about the view window's camera.
CViewWindow	This object contains the graphical user interface components of the View Window. It contains such attributes as a menu, a render area, tool bar buttons, etc.
CDatasetMenu	This contains all menu items for dataset operations.
CViewMenu	This contains all menu items for view window functions.
CAnalyzeMenu	This contains all menu items for analysis functions.
CObjectMenu	This contains all menu items for object functions.
CInputMenu	This contains all menu items for data input operations.
COutputMenu	This contains all menu items for data output operations.
CDebugMenu	This contains all menu items for debug operations (primarily used for application development).
CPseudoMenu	This contains all functionality for clicking on a data indicator on the View Window.
CStatusWindow	This contains the graphical user interface components of the View Window's Status Window.

Render Model	
CRenderModel	This object is responsible for all rendering of the dataset, a single view is rendered at a time, and only when necessary. Rendering takes into account both common and unique "view attributes" for a particular view. Also, user interactions with the view's render area are managed. This object is composed of the CTruesolidView and CInventor objects.
CTruesolidView	This object is responsible for volumetric rendering using the TrueSolid Corp. Octree Library. It is used to implement "High Resolution" display mode which is not currently supported by ICERVS.
CInventor	This object is responsible for geometric rendering and user interaction with the Open Inventor elements of the Dataset Viewer.

96TR37

11.0 GLOSSARY

API: Application Programming Interface.

CAD: Generic Computer-Aided Design Tool.

Colormap: a collection of color cells. A color cell is a color structure which contains red, green, blue (RGB) values.

Cutplane: a plane used to segment 3-D space into two regions. Typically one or more cutplanes are used to bound a region of interest. A pair of cutplanes with a user-setable separation that is used to view slices of the 3-D volume is called a sliceplane.

Cylinder: the term "cylinder" by itself is taken to mean a mathematical right circular cylinder which has infinite height. The term "truncated cylinder" refers to a portion of a cylinder which has finite height.

Dataset: a general collection of data acquired from, or constructed to describe, a 3-D workspace. Datasets can include volumetric data, scalar properties, non-scalar properties, and geometric objects.

Dataset Manager Window: a window on the user display for displaying available sites and datasets. Functions can be performed on sites and datasets from within this window.

Dialog Window: an area on the user display for user input/output.

DOE: Department of Energy.

Geometric Object: a geometric object that occupies real or conceptual space of three dimensions. Examples of 3-D geometric objects include cylinders, prismoids, and more general polyhedral shapes. A 3-D geometric object has the property of volume.

Group: a collection of datasets.

GUI: Graphical User Interface.

ICERVS: Interactive Computer-Enhanced Remote Viewing System.

ICERVS File Format: ASCII file format which is read by ICERVS to input data from sensor systems.

IGRIP: a software system developed by Deneb Robotics to facilitate off-line programming of robotic systems. The tools provide 3-D simulation to prototype and visualize robot actions.

Indicators: special markers on the view window denoting some sort of non-scalar property data to be displayed.

IRIX: operating system for Silicon Graphics Workstation. IRIX version 5.3 is currently being used.

Motif: an XWindows-based computer windowing system from Open Systems Foundation. Motif is a standard part of most UNIX-based workstations.

Object, Composite: an aggregation of two or more geometric objects that are treated as a single geometric object.

Object, Geometric: a representation of a contiguous region of conceptual space defined in a mathematical form(s).

OpenGL: a software package from SGI that provides a software interface to graphics hardware.

Open Inventor: a software package from SGI that provides an object-oriented interactive 3-D graphics toolkit

Polygon: a geometric object characterized by a sequence of connected vertices lying in a plane. The straight line connections between adjacent vertices are called edges. A 2-D polygon is *convex* if a line segment joining any two interior points is completely contained within the polygon.

Polyhedron: a 3-D geometric object bounded by plane faces which are polygons.

Property, Non-scalar: a property that takes on composite values such as arrays, text, and images.

Property, Scalar : a property that takes on a single value.

Property Data: for octrees, data used to describe the non-geometric physical characteristics of a region in space corresponding to one node. Property data are generally a single value (temperature, conductivity, weight, color, etc.), an array of values (magnitude and phase, speed vector, etc.), text, or images.

Robotic System: a computer-controlled mechanical mechanism. Typically, the computer coordinates the actions of multiple linked mechanical elements to achieve a programmed path at the working end of the mechanism.

Rogue Wave (Math.h++ and Tools.h++): a software package from Rogue Wave that provides data structure and mathematical tools.

Rotation: for computer graphic display, an angular movement of the image about an axis in 3-D space.

Scaling: for computer graphic display, a change in the size of the image of a computer-modeled space. Scaling may be performed independently in x,y,z directions.

Sensor: a device for measuring some physical quantity, such as position, temperature, weight, pressure, etc.

Sensor, LRF (Laser Range Finder): a non-contact sensor using laser energy to measure range to objects in a scene, which may be 1 to 20 meters from the sensor.

Sensor, TMS (Structured Light): a non-contact, dimensional profiling system that uses triangulation between a laser source, projected onto a region of interest, and an imaging detector such as a video camera.

Sensor Data: the location, orientation, and other attributes of a sensor together with a response set. Sensor data could represent spatial (dimensional) data or non-spatial (property) data.

SGI: Silicon Graphics Inc. (a computer workstation and software company).

Sliceplane: a pair of cutplanes with a user-defined separation used to view slices of the 3-D volume. The pair of cutplanes is controlled as a single object.

Translation: for computer graphic display, a change in the apparent position of the image of a computer-modeled space. Translation, when used in conjunction with scaling, provides a means for viewing a subsection of the world model at a magnified scale.

TrueSolid: a software package from Octree Corporation that provides tools for the building, viewing, and modification of volumetric data sets using octrees.

Units: the physical dimensional units selected by the ICERVS user for display and data input/output.

UNIX: the computer operating system used on most major workstations. UNIX is a multi-user, multi-tasking computing environment.

View Window: a window on the user display for displaying volumetric data. Functions can be performed on the data in the view from within this window.

Volumetric Data: a database containing discrete elements with spatial dimensionality of three (solids). The database is used to describe a 3-D modeling space.

Voxel: a 3-D rectangular subdivision of space.

Widget: an object with related methods. It is associated with a window and belongs to a widget class which inherits the abilities of its ancestor classes.

Wireframe: a method for defining or generating a graphic display of a polyhedron as a set of points and connecting edges.

XWindows: windowing system upon which ICERVS user interfaces are built.

Z-Buffer: graphics software internal buffer used for hidden surface removal. Also called a depth buffer.

Appendix A
Major Class Header Files

MAJOR CLASS HEADER FILES

Refer to Figure 10-1 for a graphical layout of the ICERVS major classes. The following major class header files are listed in this appendix:

3dobject.h
3dobjectset.h
analysis.h
camera.h
colormapdata.h
datasetmanager.h
datasetviewer.h
datasetmanagerwin.h
filenames.h
group.h
groupset.h
inventor.h
nonscalardata.h
property.h
propertyset.h
pseudomenu.h
rendermodel.h
sitedefinition.h
spatialdata.h
statuswindow.h
tsview.h
userdisplay.h
vdsdataset.h
vdserver.h
vduser.h
view.h
viewlist.h
viewwin.h
vmenuanalyze.h
vmenudataset.h
vmenudebug.h
vmenuinput.h
vmenuobject.h
vmenuoutput.h
vmenuview.h

```
//*************************************************************************/
//  File:               3dobject.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      C3dObject
//
//  Class Description:  C3dObject is the base class for all 3d geometric
//                      objects.  This class represents both the geometric
//                      object and its representation in the Inventor tree.
//                      Specific shaped geometric objects are derived from
//                      this class.
//
//
//  Inheritance: C3dObject
//
//  Required Class Libraries: Rogue Wave Math and Tools, Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  --------    --------    --------        -------
//  103:00:00   05/02/95    L. Cowper       Created
//  103:00:01   05/30/95    G. Hirschman    Modified design
//  103:00:02   06/06/95    G. Hirschman    Incorporated review comments
//  103:00:03   06/19/95    G. Hirschman    Added GetShapeTypeString();
//                                          added dictionary
//  103:00:04   06/20/95    G. Hirschman    Changed typedef enum's to enum's
//  103:00:05   07/10/95    G. Hirschman    Removed dictionary,
//                                           (it's in C3dObjectSet)
//                                          Added construct from Inventor tree
//  103:00:05   07/27/95    G. Hirschman    Made class non-abstract.
//  103:00:06   08/02/95    G. Hirschman    Added index stuff
//  103:00:06   09/25/95    L. Cowper       Added saved color instance data
//  103:00:06   11/08/95    L. Cowper       added data indicator type
//  103:00:06   02/02/96    L. Cowper       added cutplane types
//
//*************************************************************************/
#if !defined C3dObject_H_
    #define  C3dObject_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                   //C++ input/output streams (ALWAYS)
#include "standard.h"                   //MTI Standard Definitions  (ALWAYS)
#include "ictypes.h"                    //MTI symbols for types
#include "color.h"                      //color object

#include "transform.h"                  //our transformation object
#include "cvset.h"
//#include "faceset.h"

#include <rw/compiler.h>                    //RW compiler ident
#include <rw/rstream.h>
#include <rw/cstring.h>                     // RWCString class

#include <Inventor/nodes/SoTransformSeparator.h>   //Inventor include files
```



```
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoTexture2Transform.h>
#include <Inventor/nodes/SoTexture2.h>
#include <Inventor/nodes/SoComplexity.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoInfo.h>
#include <Inventor/nodes/SoNormalBinding.h>
#include <Inventor/actions/SoCallbackAction.h>
#include <Inventor/SoPrimitiveVertex.h>
#include <Inventor/fields/SoMFVec3f.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
//class CColor;
class CFaceSet;

//-------------------------------------------------------------------------
//...The variable matrix4 is a 4x4 matrix type.
//...This is a cool way to handle 3 dimensional arrays.
//...Create a 2 dimensional array using a typedef "matrix4".
//...Then when you want to add a third dimension use:  matrix4 *newArray.
//...Now newArray can be referenced like:   newArray[i][j][k].
//...This approach will only work if you know the dimensions of the
//...2 dimensional array.
//-------------------------------------------------------------------------
typedef float matrix4[4][4];

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class C3dObject
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

enum ShapeType  { UNDEFINED             = 0,
                  REGULAR_PRISMOID      = 1,
                  GENERAL_PRISMOID      = 2,
                  SPHERE                = 3,
                  CONE                  = 4,
                  CYLINDER              = 5,
                  BOX                   = 6,
                  ARBITRARY_POLYHEDRON  = 7,
                  SURFACE_MESH          = 8,
                  RECTANGLE             = 9,
                  COMPOSITE             = 10,
                  INDICATOR             = 11,
                  CUTPLANE              = 12,
                  CORNER_CUTPLANE       = 13 } ;


enum DisplayType {      INVISIBLE           = 0,
                        WIREFRAME           = 1,
                        SOLID               = 2,
                        SEMI_TRANSPARENT    = 3} ;

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  int           myObjectID;
```

```
  CColor        savedColor;   //Original color of selected object
                              // (Used for restoring color when unselected)
  //The following are used for creating facesets
  SoMFVec3f  *vertexSet;
  int        numVertices;
  SoMFVec3f  *normalsSet;
  int        numNormals;

protected:      // allow desrived classes access
  SoTransformSeparator *pObjectNode;    //ptr to object node in inventor tree
  static int    nextObjectID;

  //-------------------------------------------------------------------
  //- The following data items are stored in the Inventor tree;
  //- they act like private data as far as gets and sets but do not
  //- actually exist as independent variables in the class
  //-------------------------------------------------------------------
  // SbName       name;           // object name - stored in SoSeparator node
  // SbString     desc;           // object description - stored in SoInfo node
  // ShapeType    shapeType;      // shape type - stored as string in SoInfo node
                                 //    string values equal enum values
  // bool     exportableToIgrip; // Flag - stored as "TRUE" or "FALSE"
                                 //    string in SoInfo node
  // bool     edited;            // Flag - stored as "TRUE" or "FALSE"
                                 //    string in SoInfo node
  // DisplayType displayType     // stored in SoDrawType node as "style"
  // SbColor      color;         // color of object - stored in SoMaterial node
  // CTransform   trans;         // object's transformation in SoTransform node
  // SoComplexity complexity;    // measure of rendering complexity
  // RWCString    textureMap;    // texture map filename in SoTexture node
  // SoSFEnum     units          // units in SoUnits node

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    C3dObject(void);                                //Default constructor
    C3dObject(SoTransformSeparator *pInvNode);  //Constuct from Inventor tree

    ~C3dObject(void);                               //Destructor

public:
    C3dObject(const C3dObject& other);          //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
protected:
//-------------------------------------------------------------------
    SoTransformSeparator   CreateGenericObjectNode(void);
//
//  This method is called by the constructor to build
//  the generic part of the inventor sub-tree for the geometric object,
//  without the specific SoShape node attached.  The derived class
//  constructor then adds the correct derivative of SoShape to complete
//  the sub-tree.
//-------------------------------------------------------------------

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const C3dObject& P)
                                        {P.Print(os); return os;}

public:
```



```
    C3dObject& operator=(const C3dObject& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  //-----------------------------------------------------------
  //...Always make these interface methods use ICERVS types
  //-----------------------------------------------------------

  //-------------------------------------------------------------------------
  //...The following methods allow access to the components of the objectNode
  //...They each update the objectNode which automatiaclly gets updated in
  //...the Inventor tree because the tree is pointing to this instance.
  //-------------------------------------------------------------------------

  boolean       GetExportableToIgrip(void) const;
  virtual void  SetExportableToIgrip(boolean val);

  boolean       GetEdited(void) const;
  virtual void  SetEdited(boolean val);

  RWCString     GetEditedString(void)  const;
  virtual void  SetEditedString(const RWCString& val);

  DisplayType   GetDisplayType(void)  const;
  virtual void  SetDisplayType(C3dObject::DisplayType val);

  RWCString     GetShapeTypeString(void)  const;

//  C3dObject::ShapeType    GetShapeType(void)    const;
  ShapeType GetShapeType(void)    const;
  void      SetShapeType(ShapeType val);

  RWCString GetName(void)             const;
  void      SetName(const RWCString& val);

  RWCString GetDesc(void)             const;
  void      SetDesc(const RWCString& val);

  virtual void  SetColor(const CColor&  color);
  virtual void  SetAmbientColor(const CColor&  color);
  virtual void  SetSelectedObjectColor(const CColor&  color);
  CColor        GetColor(void)       const;

  void      SetSavedColor(const CColor&  color)  {savedColor = color;}
  CColor        GetSavedColor(void)       const      {return savedColor;}

  void          SetTransform(const CTransform& tranformation);
  CTransform    GetTransform(void)   const;

  void          GetTransformMatrix(matrix4 transform);

  void          SetTextureMap(const RWCString& textureMap);
  RWCString     GetTextureMap(void)  const;

  virtual void  SetComplexity(double complexity);
  double        GetComplexity(void)  const;

  void          SetUnits(const RWCString&);
  RWCString     GetUnits(void)  const;
  int           ConvertUnits( RWCString, RWCString&);

  void          SetNormalBinding(SoNormalBinding::Binding val);

//=========================================================================
```

```
// OTHER METHODS
//------------------------------------------------------------------------
public:

    //-------------------------------------------------------------------
    void  Print(ostream& os=cout)  const;
    // Print object data. This prints all common object values.
    // It is supposed to be overridden by the derived classes, but can
    // be explicitly invoked by them within thier Print method.
    //-------------------------------------------------------------------

    //-------------------------------------------------------------------
    virtual IcerClassType IsA(void)        {return _C3DOBJECT;}
    //   This virtual function should be overridden by derived classes
    //-------------------------------------------------------------------

  //---------------------------------------------------------------------
  virtual int  GetFaceSet(CFaceSet& faceSet);
  //...Get the face set of an object
  //   This virtual function should be overridden by derived classes
  //---------------------------------------------------------------------

  //---------------------------------------------------------------------
    static void myTriangleCallback(void* data, SoCallbackAction *cb,
                                   const SoPrimitiveVertex *vertex1,
                                   const SoPrimitiveVertex *vertex2,
                                   const SoPrimitiveVertex *vertex3)
    (((C3dObject *)data)->myTriangleCallback2(cb, vertex1, vertex2, vertex3);};
    void myTriangleCallback2(SoCallbackAction *cb,
                             const SoPrimitiveVertex *vertex1,
                             const SoPrimitiveVertex *vertex2,
                             const SoPrimitiveVertex *vertex3);
    //...Callback on shape node which saves the triangles
  //---------------------------------------------------------------------


  //---------------------------------------------------------------------
    SoTransformSeparator* GetObjectNode(void) const
        {return pObjectNode;}
  //...Return pointer to root of sub-tree defining object
  //...This is used by the object set to add the object to the set of objects
  //---------------------------------------------------------------------

public:
  //---------------------------------------------------------------------
    RWCString MakeNameWithID(const RWCString &baseName) const;
    RWCString MakeNameWithID(const RWCString &baseName, int id) const;
    RWCString GetNameWithoutID(const RWCString &name) const;
  //
  //  These methods are used to add or strip a 5 digit object ID from
  //  Inventor node names
  //---------------------------------------------------------------------

};

#endif
```

```
//*************************************************************************/
//  File:               3dobjectset.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      C3dObjectSet
//
//  Class Description:  C3dObjectSet represents a collection of 3d geometric
//                      objects of class C3dObject or derived classes.
//
//                      NOTE: This class is NOT derived from CSetOfNamedItems
//                            since it implements the set using an Inventor
//                            scene graph tree.
//
//
//  Inheritance: C3dObjectSet
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/26/95    L. Cowper       Modified for Phase III
//  103:00:01   05/31/95    G. Hirschman    Revised design approach
//  103:00:02   06/06/95    G. Hirschman    Incorporated review comments
//  103:00:03   07/08/95    G. Hirschman    Added numberOfObjects;
//                                            IsEmpty()
//
//*************************************************************************/
#if !defined C3dObjectSet_H_
    #define  C3dObjectSet_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                   //C++ input/output streams (ALWAYS)
#include "standard.h"                   //MTI Standard Definitions  (ALWAYS)
#include "ictypes.h"                    //MTI symbols for types

#include <rw/compiler.h>                //RW compiler ident
#include <rw/rstream.h>
#include <rw/cstring.h>                 // RWCString class

#include <Inventor/nodes/SoSeparator.h> // Inventor Includes


#include "3dobject.h"                   //C3dObject class
#include "cstrlist.h"                   //COrderedStringList class

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class C3dComposite;                     //Composite object class
class CDictionary;                      // Dictionary class

//-------------------------------------------------------------------------
// CLASS DEFINITION
```



```
//-------------------------------------------------------------------------
class C3dObjectSet
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

    SoSeparator  *pInventorNode;        // Inventor root node for object set

    CDictionary *pDictionary;           // look up from inventor node to
                                        //  3dObject pointer

    int   currentObjectIndex;           // Inventor child index of current
                                        // object

    int   numberOfObjects;              // Number of objects in set


//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    C3dObjectSet(void);
    ~C3dObjectSet(void);                                //Destructor

private:
    C3dObjectSet(const C3dObjectSet& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const C3dObjectSet& P)
                                                        {P.Print(os); return os;}
private:
    C3dObjectSet& operator=(const C3dObjectSet& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
    //-----------------------------------------------------------------
    SoSeparator* GetInventorNodePtr(void)  const {return pInventorNode;}
    //
    //... Returns pointer to Inventor node which is root of all geometric
    //... objects in the Inventor tree.
    //-----------------------------------------------------------------

    //-----------------------------------------------------------------
    int  GetNumberOfObjects(void)  const {return numberOfObjects;}
    //
    //... Returns current number of objects in object set
    //-----------------------------------------------------------------
```

```
//=============================================================================
// OTHER METHODS
//=============================================================================
public:
    void          Print(ostream& os=cout)  const;      //Print object data
    IcerClassType IsA(void)                const        {return _C3DOBJECTSET;}

    //-------------------------------------------------------------------------
    int  Add(const C3dObject* p3dObject);
    //
    //... Adds the object pointed to by p3dObject to the set.
    //... The object's Inventor node will be added to the set's Inventor
    //... node using AddChild()
    //... Add() will check to make sure there is not an object already in
    //... the set with the same name, and return an error if there is.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    int       Remove(const RWCString& name);
    int       Remove(C3dObject* p3dObject);
    //
    //... Remove the Object named or pointed to from the set.
    //... The object removed will be destroyed.
    //... The object's Inventor node will be removed from the set's Inventor
    //... node and the C3dObject destructor will be called.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    int       RemoveAll(void);
    //
    //... Remove all objects from the set.  An empty set will remain.
    //... Each object removed will be destroyed ??????????
    //-------------------------------------------------------------------------


    //-------------------------------------------------------------------------
    int GetNames(COrderedStringList* aList)   ;
    int GetDescriptions(COrderedStringList* aList)   ;
    int GetShapeTypes(COrderedStringList* aList)   ;
    int GetExportableFlags(COrderedStringList* aList)   ;
    //
    //... Return a list of the names / descriptions / shape types of all
    //... objects in the set.
    //... Shape Types are returned as ASCII strings of the type name.
    //... Exportable Flags are returned as ASCII strings.
    //... These methods are intended for use in putting up a selection list
    //... of objects in the user interface.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject*  Find(const RWCString& ObjectName)   ;
    //
    //... Return a pointer to the object specified by the given name
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject*  Find(SoTransformSeparator* pObjectNode)   ;
    //
    //... Return a pointer to the object specified by the given Inventor node
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject*  Find(C3dPoint point)   ;
    //
    //... Return a pointer to the object specified by the given
    //... translation point (ie x,y,z)
```



```
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject*  FindSubstring(RWCString subString);
    //
    //    Return a pointer to the first object found with the given
    //    substring in the object name.  A regular expression (RE) search
    //    is performed on each name, the *first* match is returned.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    boolean  C3dObjectSet::IsEmpty(void) const;
    //
    //... Returns TRUE if no objects in set
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject*  GetFirst(void)  ;
    C3dObject*  GetNext(void)  ;
    C3dObject*  GetPrevious(void) ;
    C3dObject*  GetLast(void) ;
    //
    //... Return a pointer to the desired (first/next/previous/last) object
    //... in the set.  Each of these methods will update the "current object"
    //... which is the reference for subsequent GetNext() or GetPrevious()
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    int  ReadGeometricObjects(const RWCString& fname);
    //
    //... Read in the entire set of geometric objects from the specified file
    //... The file must be in Invnetor .iv ASCII format and should start
    //... with a Separator node which is the root of all objects.
    //... Any objects previously in the set will be deleted.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    C3dObject  *ReadObject(const RWCString& fname);
    //
    //...  Reads a library object from the specified file into Inventor
    //...  sceneGraph.
    //... The file must be in Inventor .iv ASCII format and should start
    //... with a Transform Separator node which is the root of object.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    int  WriteObject(const RWCString& fname, C3dObject* object);
    //
    //...  Writes an object from the Inventor sceneGraph to the specified file
    //... The file must be in Inventor .iv ASCII format and should start
    //... with a Transform Separator node which is the root of object.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    int WriteGeometricObjects(const RWCString& fname);
    //
    //... Write the entire set of geometric objects to the specified file
    //... in Inventor .iv ASCII format starting with the separator node
    //... which is the root of all objects.
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
//    int WriteIGripFile(const RWCString& fname);
    //
    //====> NOT IMPLEMENTED FOR RELEASE A
    //...First open an IGRIP file and write header info.  Then loop over all
```

```
    //...the "Exportable" objects in set.  For each object, call object
    //...method to append itself to the IGRIP file (ConvertToIgrip).
    //-------------------------------------------------------------------

    //-------------------------------------------------------------------
//  int  ReadIGripFile(const RWCString& fname);
    //
    //====> NOT IMPLEMENTED FOR RELEASE A
    //...First open an IGRIP file and read header info.  Then loop over all
    //...objects in file.  For each object, instantiate a C3dArbPolyhedron
    //...object and call object method to read next section from IGRIP file
    //...and create an object from the IGRIP info (ConvertFromIgrip). Now add
    //...this new object to the object set and to the Inventor tree.
    //-------------------------------------------------------------------

    //-------------------------------------------------------------------
    C3dComposite*  BuildCompositeObject(const RWCString& name,
                                        COrderedStringList objectNames);
    //
    //... First instantiate a C3dComposite object (called compositeObject).
    //... Then loop over all objects in the inputted object name list
    //... (i.e. list of currently selected objects).
    //... For each object in list, add it to the set of objects within the
    //... newly created composite object (compositeObject) and then remove the
    //... object from our set of objects.
    //... Finally add the new composite object to the set of objects
    //... Return the pointer to the composite object just created.
    //-------------------------------------------------------------------

    //-------------------------------------------------------------------
    int    UnBuildCompositeObject(C3dComposite compositeObject);
    //
    //... Loop over all objects in the inputted composite object's set of
    //... objects.  For each sub-object in composite object, add it to the
    //... master set of objects and remove the object from the composite
    //... object.
    //... Return error status (0= OK)
    //-------------------------------------------------------------------

};

#endif
```

```
//*************************************************************************
//  File:               analysis.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CAnalysis
//
//  Class Description:  CAnalysis encapsulates all octree function pertaining
//                      to analysis functions
//
//  Inheritance:
//
//  Required Class Libraries: Rogue Wave Math/Tools, TrueSolid
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//
//*************************************************************************
#if !defined CAnalysis_H_
    #define  CAnalysis_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                       //C++ input/output streams
#include "standard.h"                       //MTI Standard Definitions
#include "ictypes.h"                        //MTI symbols for types

#include <unistd.h>
#include <rw/compiler.h>                    //RW compiler ident
#include <rw/cstring.h>                     //RWCString class
#include <rw/rstream.h>

#include  <Ts/octree.h>
#include  <Ts/Csg_memprim.h>

#include "3dobject.h"
#include "3dobjectset.h"
#include "3dconvert.h"
#include "3dpointset.h"

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CSpatialData;
class C3dPoint;

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CAnalysis
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
```



```
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  CSpatialData*      pSpatialData;          //pointer to parent object
  ts_Csg_ptr         currentRegion;         //ptr to currently selected region

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
   CAnalysis(CSpatialData *pSpatialData);
   ~CAnalysis(void);                                  //Destructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:
  //...Needed to get rid of linker warning
  void       TsCopyright(void)    const                    {if (ts_copyright);}

  ts_octree_ptr  GetTree(void)      const;
  ts_Csg_varptr  GetCsgstuff(void)  const;


//=========================================================================
// OPERATORS
//=========================================================================
public:

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:


//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  //-------------------------------------------------
  //...Select a region of points in the octree
  //...The region consists of a set of included objects
  //...and a set of excluded objects
  //...The selected region node type is set to F2
  //-------------------------------------------------
  int          SelectRegion(C3dObjectSet* includedObjects,
                            C3dObjectSet* excludedObjects);

  //-------------------------------------------------
  //...Unselect a region of points in the octree
  //...This is done by setting node type F2 to unknown F1
  //-------------------------------------------------
  int          UnSelectRegion(void);

  //-------------------------------------------------
  //...Erases a region of points from the octree
  //...This is done by:
  //    setting node type F3 (temporarily erased) to F0 (empty)
  //    setting node type F2 (selected region) to F3 (temporarily erased)
  //-------------------------------------------------
  int          EraseRegion(void);

  //-------------------------------------------------
```

```
  //...UnErases the last erased region of points in the octree
  //...This is done by setting node type F3 (temporarily erased)to F1 (normal)
  //-------------------------------------------------
  int          UnEraseRegion(void);

  //-------------------------------------------------
  //...Create an object set CSG
  //-------------------------------------------------
  ts_Csg_ptr   CreateObjectSetCsg(C3dObjectSet* ObjectSet);

  //-------------------------------------------------
  //...Set the node type for a region to another type
  //-------------------------------------------------
  int          SetTreeRegionNodeType(ts_sourceptr region,
                                     int newType);
  //-------------------------------------------------------------
  //...Builds a CSG for a given geometric object
  //-------------------------------------------------------------
  ts_Csg_ptr   BuildPolyhedralCsg(C3dObject* object);
  ts_Csg_ptr   BuildCylinderCsg(C3dObject* object);
  ts_Csg_ptr   BuildSphereCsg(C3dObject* object);
  ts_Csg_ptr   BuildConeCsg(C3dObject* object);
  ts_Csg_ptr   BuildBoxCsg(C3dObject* object);

  //--------------------------
  //...Data conversion methods
  //--------------------------
  void         ConvertToTrueSolidTransform(float transform[4][4],
                                           ts_Xform& object_xform);
  void         ConvertToTrueSolidTransformSimple(float transform[4][4],
                                                 ts_Xform& object_xform);
  ts_Vector    ConvertToTrueSolidVector(double x, double y, double z);
  void         PrintMatrix(float matrix[4][4]);


  //--------------------------
  //...Volumetric Connectivity
  //--------------------------
  int          Connectivity(C3dPoint* point);
  int          ConnectivityUndo(void);

  //--------------------------
  //...Erase thresholded points
  //--------------------------
  int EraseThreshhold(const RWCString& propertyType,
                      float min,
                      float max );

  int EraseThreshhold(const RWCString& propertyType,
                          float lowerMin,
                          float lowerMax,
                          float upperMin,
                          float upperMax);
};

#endif
```

```
//*************************************************************************/
//  File:                camera.h
//
//  Project Name:        ICERVS Phase III
//
//  File description:    This file contains the declaration for class
//                       CCamera
//
//  Class Description:   CCamera encapsulates the definition of an Inventor
//                       SoCamera. A camera describes a viewpoint for viewing
//                       a dataset in a view window.  The type of camera
//                       (orthoginal or perspective) is defined in the view
//                       parameters class (CViewParameters).  This class handles
//                       all interfaces to the Inventor camera data.
//
//  Inheritance:         <CCamera>
//
//  Required Class Libraries:   Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name          Remarks
//  ---------   --------    --------      -------
//  103:00:00   04/26/95    L. Cowper     Created
//  103:00:01   05/23/96    L. Cowper     Changed Examiner Viewer to
//                                        Full Viewer
//
//*************************************************************************/
#if !defined CCamera_H_
    #define  CCamera_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================

#include <iostream.h>                  //C++ input/output streams (ALWAYS)
#include <standard.h>                  //MTI Standard Definitions  (ALWAYS)
#include <ictypes.h>                   //MTI symbols for types

#include <Inventor/Xt/viewers/SoXtFullViewer.h> //Inventor include file
#include <Inventor/nodes/SoCamera.h>
#include <Inventor/nodes/SoOrthographicCamera.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CViewWindow;
class C3dPoint;
class CView;

// The following definitions are used to adjust the camera so that the
// view takes up as much space as possible in the view window
#define HEIGHT_SCALE_FACTOR .588          //adjustment for orthoginal view
#define HEIGHT_ANGLE_SCALE_FACTOR .625    //adjustment for perspective view

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
```



```
class CCamera
{
//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  CView              *pView;                 //CView class
  SoXtFullViewer     *myViewer;              //Full Viewer
  SoCamera           *myCamera;              //Viewer camera
  double             orientationMatrix[9];   //orientation matrix

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    CCamera(CView *parent);                  //Default constructor
    ~CCamera(void);                          //Destructor

    CCamera(const CCamera& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:
//-----------------------------------------
// Get view parameters filename
//-----------------------------------------
  RWCString GetViewParametersFilename(void)         const;

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, CCamera& P){
      P.Print(os);
      return os;
    }
    CCamera& operator=(const CCamera& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  //...Get the Examiner Viewer's camera
  //SoCamera*     GetCamera(void)                    const  {return myCamera;}
  SoCamera*     GetCamera(void);

  // These  methods access values from the Examiner Viewer's camera
  C3dPoint      GetPositionVector(void);
  void          SetPositionVector(C3dPoint vector);

  C3dPoint      GetOrientationVector(void);
  void          SetOrientationVector(C3dPoint vector);

  void          SetOrientation(C3dPoint vector, double angle);

  double        GetOrientationAngle(void);
  void          SetOrientationAngle(double angle);

  double*       GetOrientationMatrix(void);
```

```
  void          SetOrientationMatrix(double* matrix);

  double        GetNearDistance(void);
  void          SetNearDistance(double value);

  double        GetFarDistance(void);
  void          SetFarDistance(double value);

  double        GetFocalDistance(void);
  void          SetFocalDistance(double value);

  double        GetAspectRatio(void);
  void          SetAspectRatio(double value);

  double        GetHeight(void);
  void          SetHeight(double value);

  double        GetWidth(void);
  void          SetWidth(double value);

  double        GetHeightAngle(void);
  void          SetHeightAngle(double value);


//================================================
// OTHER METHODS
//================================================
public:
   void             Print(ostream& os=cout);            //Print object data
   IcerClassType    IsA(void)                const      {return _CCAMERA;}

  //...Gets and sets the camera type parameter stored in the view object
   RWCString     GetCameraType();
   unsigned      SetCameraType(const RWCString& cameraType);

  //-------------------------------------------------------------
  //...Initialize the camera
  //...Read the camera parameters from the view parameters file
  //-------------------------------------------------------------
  int  Initialize();

  //--------------------------------------
  //...View Parameters File access methods
  //--------------------------------------
  int  ReadViewCameraParameters(void);
  int  WriteViewCameraParameters(void);

  int  ReadViewCameraParameters(const RWCString&);
  int  WriteViewCameraParameters(const RWCString&);

};

#endif
```

```
//*****************************************************************************/
//  File:               colormapdata.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CColormapData
//
//  Class Description:  CColormapData encapsulates the PDC that contains
//                      colormap information
//
//  Inheritance:        CColormapData
//
//  Required Class Libraries: Rogue Wave Tools UNIRAS
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   01/31/96    H. Belawski     Created
//
//*****************************************************************************/
//                          Version Information
// $RCSfile: colormapdata.h,v $
// $Author: cowper $
// $Revision: 1.3 $
// $Date: 1996/02/26 22:16:31 $
//
//*****************************************************************************/
#if !defined CColormapData_H_
    #define  CColormapData_H_

//============================================================================
// REQUIRED INCLUDE FILES
//============================================================================
#include <stddef.h>                     //Standard C include files
#include <iostream.h>                   //C++ input/output streams.
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

//----------------------------------------------------------------------------
// APPLICATION INCLUDE FILES
//----------------------------------------------------------------------------
#include <rw/cstring.h>
#include <color.h>
#include <view.h>
#include <vdsdataset.h>
#include "colormap.h"

//============================================================================
// OTHER DECLARATIONS
//============================================================================
class CViewMenu;

//============================================================================
// CLASS DEFINITION
//============================================================================
```



```
class CColormapData
{

//============================================================================
// DECLARATIONS (enums, defines, etc)
//============================================================================
public:

//----------------------------------------------------------------------------
//  DATA MEMBERS (ALWAYS PRIVATE)
//----------------------------------------------------------------------------
private:
  CColormap* pColormapWindow;
  CView* pView;
  CViewMenu* pViewMenu;

  RWCString     myPropertyType;
  RWCString     myPropertyLabel;
  RWCString     myColormapFilename;
  CColor        myPrimaryMinColor;
  CColor        myPrimaryMaxColor;
  CColor        my2ondaryMinColor;
  CColor        my2ondaryMaxColor;
  CColor        myOutsideColor;
  float         myPrimaryMinVal;
  float         myPrimaryMaxVal;
  float         my2ondaryMinVal;
  float         my2ondaryMaxVal;

  int           myRangeOption;      // 1=primary min 2=primary max
                                    // 3=2ondary min 4=2ondary max 5=outside

  int           my2ondRangeOption;  // 1=color inside 2=invisible inside
                                    // 3=ivisible outside

  // a flag used to indicate that the color has been changed.  This flag
  // should only be set in the Update() method.
  int colorChangedFlag;

  // used to flush the cache in the Update() method.
  int flushCache;

  // called when the property type has changed to fill instance data
  // with the current property ranges, etc
  void NewProperty(void);

//============================================================================
// CONSTRUCTORS AND DESTRUCTORS
//============================================================================
---->
public:
  CColormapData(void);
  CColormapData(CView* view);
 ~CColormapData(void);

//============================================================================
// OTHER METHODS
//============================================================================
  boolean         IsDefined(void)             const;
  IcerClassType   IsA(void)                   const;
  virtual void    Print(ostream& os=cout)     const;

  friend ostream& operator<<(ostream& os,
                             const CColormapData& colormapdata)
                             {colormapdata.Print(os); return(os);}
```

```
  CView*  GetView(void) { return(pView); };

  void        SetViewMenu(CViewMenu* menu) { pViewMenu = menu; };
  CViewMenu*  GetViewMenu(void) { return(pViewMenu); };

  void  SetPrimaryMinColor(const CColor&  val) {myPrimaryMinColor = val;}
  void  SetPrimaryMaxColor(const CColor&  val) {myPrimaryMaxColor = val;}
  void  Set2ondaryMinColor(const CColor&  val) {my2ondaryMinColor = val;}
  void  Set2ondaryMaxColor(const CColor&  val) {my2ondaryMaxColor = val;}
  void  SetOutsideColor(const CColor&  val)    {myOutsideColor    = val;}
  void  SetPrimaryMinVal(const float  val)     {myPrimaryMinVal   = val;}
  void  SetPrimaryMaxVal(const float  val)     {myPrimaryMaxVal   = val;}
  void  SetSecondaryMinVal(const float  val)   {my2ondaryMinVal   = val;}
  void  SetSecondaryMaxVal(const float  val)   {my2ondaryMaxVal   = val;}
  void  SetRangeOption(const int  val)         {myRangeOption     = val;}
  void  Set2ondRangeOption(const int  val)     {my2ondRangeOption = val;}

  const RWCString  GetPropertyType(void)    const {return myPropertyType;}
  const CColor&    GetPrimaryMinColor(void) const {return myPrimaryMinColor;}
  const CColor&    GetPrimaryMaxColor(void) const {return myPrimaryMaxColor;}
  const CColor&    Get2ondaryMinColor(void) const {return my2ondaryMinColor;}
  const CColor&    Get2ondaryMaxColor(void) const {return my2ondaryMaxColor;}
  const CColor&    GetOutsideColor(void)    const {return myOutsideColor;}
  const float      GetPrimaryMinVal(void)   const {return myPrimaryMinVal;}
  const float      GetPrimaryMaxVal(void)   const {return myPrimaryMaxVal;}
  const float      Get2ondMinVal(void)      const {return my2ondaryMinVal;}
  const float      Get2ondMaxVal(void)      const {return my2ondaryMaxVal;}
  const int        GetRangeOption(void)     const {return myRangeOption;}
  const int        Get2ondRangeOption(void) const {return my2ondRangeOption;}

  void  SetHIC(CColormap* hic){ pColormapWindow = hic; };
  void  RemoveHIC(void){ pColormapWindow = 0; };

  int  GetColormapFilename(void);

  int  LoadColormap(const RWCString& filename);
  int  SaveColormap(const RWCString& filename);

  int  Initialize(void);
  void Update(void);
  void UpdateColormapWindow(void);

  // The colorChangedFlag is set *only* when the Update() method is
  // called.  This Get method is intended to be called from the
  // RenderModel when a render occurs.
  void SetColorChangedFlag(int val){ colorChangedFlag = val;};
  int  GetColorChangedFlag(void){ int temp = colorChangedFlag;
                                  colorChangedFlag = 0;
                                  return(temp);};
};
#endif
```

```
//*************************************************************************
//  File:               datasetmanager.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CDatasetManager
//
//  Class Description:  CDatasetManager encapsulates the dataset manager,
//                      and is of primary importance in that most of
//                      ICERVS functionality is related to datasets.
//                      This class is analogous to a file manager
//                      except it works with groups and datasets instead
//                      of directories and files. It manages all the functions
//                      related to groups and datasets.
//
//                      The name of the group given by the operator when a
//                      new group is requested, becomes the directory name
//                      for the group.
//
//                      The name of the dataset given by the operator when a
//                      new dataset is requested, becomes the directory name
//                      for the dataset.
//
//  Inheritance:        <CDatasetManager>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/27/95    L. Cowper       Created
//  103:00:01   07/06/95    J. Hutchison    Revised: removed methods for adding/
---->/                                              and removing datasets and
---->groups
//                                          Added methods to validate names
//                                           and methods to make/remove
//                                           a directory and to copy files.
//
//
//*************************************************************************/
#if !defined CDatasetManager_H_
   #define  CDatasetManager_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                   //C++ input/output streams )
#include <standard.h>                   //MTI Standard Definitions
#include <ictypes.h>                    //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>                         //RWCString
#include <rw/rstream.h>

#include <cstrlist.h>                           //COrderedStringList
#include <csortstr.h>                           //CSortedStringList
#include <vdsdataset.h>
```



```
#include <vdsdatasetlist.h>                     //CVDSDatasetList
#include <datasetmanagerwin.h>                  //CDatasetManagerWindow
#include <groupset.h>                           //CGroupSet

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CVDSDatasetList;
class CVDSUser;
//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CDatasetManager
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  CDatasetManagerWindow  *myDMWindow;   //Pointer to the DM window

  CVDSDatasetList         *myOpenDatasets; //Set of dataset objects, each of
                                           // which is created when the dataset
                                           // is opened and removed  when
                                           // the dataset is closed

  CGroupSet              *myGroups;      //ALL existing groups
                                         //(identified by existence of a
                                         // directory in the ICERVS data path)

  RWCString               operatorName; //Current operator name

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    CDatasetManager(void);
    ~CDatasetManager(void);                             //Destructor

    CDatasetManager(const CDatasetManager& other);      //Copy constructor

public:
  int MakeDirectory(const RWCString& path, const RWCString& name)   const;
  int RemoveDirectory(const RWCString& path, const RWCString& name)   const;
  int RemoveFile (const RWCString& filename)  const;
  int CopyFile (const RWCString& file1, const RWCString& file2)  const;
  int AppendFile (const RWCString& file1, const RWCString& file2)  const;
  int ValidateDirectoryName(const RWCString& name) const;
  int GetDirectoryNumberOfFiles(const RWCString& dirname)  const;

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

  int MakeAddedInfoFile (const RWCString& filename)  const;

  int WriteLogHeader(const RWCString& filename, const RWCString& group) const;

  //...Close all datasets for a given group
```

```
  //...Loop over all datasets in inputted group list
  //...Invoke method in CVDSDataset to close dataset
  //...(this will save dataset if necessary)
  int CloseAllDatasets(CGroup *group);

//========================================================================
// OPERATORS
//========================================================================
public:
    friend ostream& operator<<(ostream& os, const CDatasetManager& P)
                                                {P.Print(os); return os;}
    CDatasetManager& operator=(const CDatasetManager& other);

//========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//========================================================================
public:
  CGroupSet*             GetSetOfGroups(void)          {return myGroups;}
  CVDSDatasetList*       GetListOfOpenDatasets(void)   {return myOpenDatasets;}
  CDatasetManagerWindow* GetDatasetManagerWindow(void) {return myDMWindow; }

//========================================================================
// OTHER METHODS
//========================================================================
public:
  void         Print(ostream& os=cout)  const;   //Print object data
  IcerClassType IsA(void)               const    {return _CDATASETMANAGER;}

            //-------------------------------------------------
            // Initialize set of groups by reading directories in
            // at ICERVSDATA path level
            //-------------------------------------------------
  int Initialize(void);

             //-----------------------------------
             //  Create the dataset manager window
             //-----------------------------------
  int  CreateDatasetManagerWindow(const RWCString& display,
                                  CVDSServer* vdsServer,
                                  CVDSUser* vdsUser);

            //-------------------------------------------------
            // Check for acceptable group name syntax, and for
            // name already in use
            //-------------------------------------------------
  int ValidateGroupName(const RWCString& groupName) const;


            //-------------------------------------------------
            // Check for acceptable dataset name syntax, for
            // existence of group and existence of a dataset (in
            // the group) with this name
            //-------------------------------------------------
  int ValidateDatasetName(const RWCString& groupName,
                          const RWCString& datasetName) const;

            //-------------------------------------------------
            // Initialize set of groups by reading directories in
            // at ICERVSDATA path level
            //-------------------------------------------------
  boolean IsOpen
          (const RWCString& groupName, const RWCString& datasetName) const;


  int WriteLogDelimeter
           (const RWCString& filename) const;
```



```
  int BringUpView(const CVDSDataset* dataset) const;

  void  AttachDataset(CVDSDataset* dataset) const {dataset->Attach();}
  void  DetachDataset(CVDSDataset* dataset)
  {
    CVDSDataset* i;
    if (dataset->Detach() == 0)
     i = myOpenDatasets->Find(dataset->GetTheName(),
                              dataset->GetGroupName());
      if (i != 0)
        myOpenDatasets->Remove(dataset);
      delete(dataset);
  }

  //=============================================
  //...DatasetManagerWindow Group menu functions
  //=============================================

            //-------------------------------------------------
            // Quit -- active from stand-alone mode only
            //-------------------------------------------------
  void  Quit(void);

            //-------------------------------------------------
            // Delete the group directory and everything below it
            //-------------------------------------------------
  int GroupDelete(const RWCString& groupName);


            //-------------------------------------------------
            // Create a directory with name = groupName.
            // Create the goup's log file.  Instantiate a group
            // object and add it to myGroups
            //-------------------------------------------------
  int  GroupNew(const RWCString& groupName,
            const RWCString& groupDesc);


  //=============================================
  //...DatasetManagerWindow Dataset menu functions
  //=============================================

            //-------------------------------------------------
            // Assuming the group and dataset names are valid,
            // and the destination group/dataset directory does not
            // exist, create the group/dataset directory and
            // copy all files from the old to the new
            //-------------------------------------------------
  int DatasetCopy(const RWCString& oldGroupName,
                                const RWCString& oldDatasetName,
                                const RWCString& newGroupName,
                                const RWCString& newDatasetName);

            //-------------------------------------------------
            // If the dataset is open, inform the user to close
            // it.  If it is not open, delete all files in the
            // dataset directory, then delete the directory itself
            //-------------------------------------------------
  int DatasetDelete(const RWCString& groupName,
                    const RWCString& datasetName);

            //-------------------------------------------------
            //...Create dataset info file
            //
            // -Create a file with the following:
```

```
            //     Dataset Name, Description, date/time of creation,
            //          creating operator, existing scalar properties
            //-------------------------------------------------
 int  DatasetInfo(const RWCString& groupName,   const RWCString& datasetName,
                                                RWCString* browseFile);

          //-----------------------------------------
          // Archive a dataset (along with its group)
          // to external media
          // (description deferred until release 2)
          //-----------------------------------------
 int DatasetArchive(const RWCString& groupName,
                    const RWCString& datasetName);

          //-------------------------------------------------------
          // Retrieve an archived dataset(along with its group) from
          // external media (description deferred until release 2)
          //-------------------------------------------------------
 int  DatasetRetrieve(const RWCString& groupName,
                      const RWCString& datasetName,
                      CVDSDataset* retrievedDataset);

           //-------------------------------------------------
           // Insure valid group name. Make the directory for the
           // dataset. Copy a site.def file from the group directory,
           // a view.prm file from the system/etc directory, a
           // color.thresh file from the system/etc directory.
           // Instantiate a temporary dataset, set the operatorName,
           // creation date/time and a description, then have the
           // temporary dataset write its dataset.prm file.
           //-------------------------------------------------
  int  DatasetNew(const RWCString& groupName,
                        const RWCString& datasetName,
                        const RWCString& datasetDesc );

          //-------------------------------------------------------
          // Open a dataset.  If it doesn't exist, create it
          //-------------------------------------------------------

  CVDSDataset* DatasetOpenCreate(const RWCString& groupName,
                       const RWCString& datasetName,
                       const RWCString& datasetDesc);

 //-------------------------------------------------
 // Other methods needed by external applications
 //-------------------------------------------------



         //-----------------------------------------------
         // Open an existing dataset
         //-----------------------------------------------
CVDSDataset*  DatasetOpen(const RWCString& groupName,
                  const RWCString& datasetName);

         //-----------------------------------------------
         // Get the list of all group names and descriptions
         //-----------------------------------------------
 int GetListOfGroups(COrderedStringList* groupNames,
                     COrderedStringList* groupDescs);

         //-----------------------------------------------
         // Get the list of all dataset names in a group
         //-----------------------------------------------
int GetListOfDatasets(const RWCString& groupName,
                      COrderedStringList*  datasetNames);
```



```
         //-------------------------------------------------
         // Get the list of all open dataset names and
         // descriptions for a group
         // Call methods in CDatasetList to do this
         //-------------------------------------------------

  int GetListOfOpenDatasets(const RWCString& groupName,
                            COrderedStringList* datasetNames,
                            COrderedStringList* datasetDescs);

         //-------------------------------------------------
         // Get the list of all open dataset names and
         // descriptions and groups  (all groups
         //-------------------------------------------------
  int GetListOfAllOpenDatasets (COrderedStringList* datasetNames,
                            COrderedStringList* datasetDescs,
                            COrderedStringList* groups);

         //-------------------------------------------------
         // Set the operator name
         //-------------------------------------------------
  int  SetOperatorName(const RWCString& opName);

         //-------------------------------------------------
         // Start the VDS GUI
         //-------------------------------------------------
  int VDSStartGUI(const RWCString& displayName, CVDSServer* vdsServer,
                  CVDSUser* vdsUser);

 };

#endif
```

```
//*************************************************************************
//  File:               datasetviewer.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CDatasetViewer
//
//  Class Description:  CDatasetViewer encapsulates the viewing of a dataset.
//                      It contains those entities that are common to all
//                      views.  It also manages all view classes.
//
//  Inheritance:        CDatasetViewer
//
//  Required Class Libraries: Rogue Wave Math/Tools,
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/25/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined CDatasetViewer_H_
    #define  CDatasetViewer_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                           //C++ input/output streams
#include <standard.h>                           //MTI Standard Definitions

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>                         //RWCString
#include <rw/rstream.h>

#include <ictypes.h>                            //ICER common defs

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>

#include <view.h>



//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CVDSDataset;
class CViewList;
//class CView;
class CViewWindow;
class CRenderModel;
class C3dObjectSet;
class CSortedStringList;
class CObjectSelectWindow;
class CObjectEditWindow;
class C3dObject;
```



```
class C3dIndicator;
class MyColorEditor;
class SoXtTransformSliderSet;
class SoXtFullViewer;
class CPropertyValues;
class CObjectAttributeWindow;
class C3dPoint;
class CRegionSelectWindow;

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CDatasetViewer
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

//...Pointers to needed objects
CVDSDataset*     pDataset;          //pointer to parent dataset object

//...These are pointers because they are only created when the first view
//...is requested
CRenderModel*    pRenderModel;          //render model object
CViewList*       pViewList;             //list of views
C3dObjectSet*    pCutplaneSet;          //set of cutplanes
CSortedStringList*   pSelectedObjects;  //currently selected objects
CSortedStringList*   pIncludedObjects;  //included objects for region selection
CSortedStringList*   pExcludedObjects;  //excluded objects for region selection

//...General parameters
int              nextViewNumber;            //next view number
SoXtFullViewer          *currentViewer;             //current viewer
C3dIndicator            *currentIndicator;          //current data indicator
C3dObject               *currentObject;             //current geometric object

CObjectSelectWindow     *objectSelectWindowPointer;  //obj selection window point
---->er
CObjectEditWindow       *objectEditWindowPointer;    //obj edit window pointer
SoXtTransformSliderSet *transformEditWindowPointer; //transform edit window poin
---->ter
MyColorEditor           *colorEditWindowPointer;     //color edit window pointer
CPropertyValues         *propertyWindowPointer;      //property display window po
---->inter
CObjectAttributeWindow *objectAttributeWindowPointer; //obj attribute win ptr
CRegionSelectWindow     *regionSelectWindowPointer;  //region selection window po
---->inter

  //=======================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=======================================================================
public:
  CDatasetViewer(CVDSDataset* pDataset);
  ~CDatasetViewer(void);

//=======================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=======================================================================
private:
```

```
  //------------------------------------------------------------
  //...Get view parameters filename
  //------------------------------------------------------------
  RWCString GetViewParametersFilename(void)        const;

//=============================================================================
// OPERATORS
//=============================================================================
public:
    friend ostream& operator<<(ostream& os, const CDatasetViewer& P)
                                                  {P.Print(os); return os;}
    CDatasetViewer& operator=(const CDatasetViewer& other);

//=============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=============================================================================
public:
  CVDSDataset*        GetDatasetPointer(void)        const {return pDataset;}
  CSortedStringList*  GetSelectedObjects(void)       const {return pSelectedObje
---->cts;}
  CSortedStringList*  GetIncludedObjects(void)       const {return pIncludedObje
---->cts;}
  CSortedStringList*  GetExcludedObjects(void)       const {return pExcludedObje
---->cts;}
  C3dObjectSet*       GetSetOfCutplanes(void)        const {return pCutplaneSet
---->;}
  CViewList*          GetListOfViews(void)           const {return pViewList;}
  CRenderModel*       GetRenderModelPointer(void)    const {return pRenderModel
---->;}

  void                SetObjectSelectWindowPointer(CObjectSelectWindow* val)
                                  {objectSelectWindowPointer=val;}
  CObjectSelectWindow* GetObjectSelectWindowPointer(void) const
                                  {return objectSelectWindowPointer;}

  void                SetObjectEditWindowPointer(CObjectEditWindow* val)
                                  {objectEditWindowPointer=val;}
  CObjectEditWindow*  GetObjectEditWindowPointer(void) const
                                  {return objectEditWindowPointer;}

  void                SetTransformEditWindowPointer(SoXtTransformSliderSet* val
---->)
                                  {transformEditWindowPointer=val;}
  SoXtTransformSliderSet* GetTransformEditWindowPointer(void) const.
                                  {return transformEditWindowPointer;}

  void                SetColorEditWindowPointer(MyColorEditor* val)
                                  {colorEditWindowPointer=val;}
  MyColorEditor*      GetColorEditWindowPointer(void) const
                                  {return colorEditWindowPointer;}

  void                SetCurrentViewer(SoXtFullViewer* val)
                                  {currentViewer = val;}
  SoXtFullViewer*     GetCurrentViewer(void) const
                                  {return currentViewer;}

  void                SetPropertyWindowPointer(CPropertyValues* val)
                                  {propertyWindowPointer=val;}
  CPropertyValues*    GetPropertyWindowPointer(void) const
                                  {return propertyWindowPointer;}

  void                    SetObjectAttributeWindowPointer(
                                  CObjectAttributeWindow* val)
                                  {objectAttributeWindowPointer = val;}
  CObjectAttributeWindow*     GetObjectAttributeWindowPointer(void) const
                                  {return objectAttributeWindowPointer;}
```



```
  void                SetRegionSelectWindowPointer(CRegionSelectWindow* val)
                                  {regionSelectWindowPointer=val;}
  CRegionSelectWindow* GetRegionSelectWindowPointer(void) const
                                  {return regionSelectWindowPointer;}

//=============================================================================
// OTHER METHODS
//=============================================================================
public:
  void          Print(ostream& os=cout)  const;      //Print object data
  IcerClassType IsA(void)                const      {return _CDATASETVIEWER;}

  //------------------------------------------------------------
  //...Read and write shared  view parameters in view parameter file
  //...For now, this includes only cutplanes
  //------------------------------------------------------------
  int  ReadSharedViewParameters(void);
  int  WriteSharedViewParameters(void);

  //------------------------------------------------------------
  //...Read and write shared  view parameters in view parameter file
  //...For now, this includes only cutplanes
  //------------------------------------------------------------
  int  ReadSharedViewParameters(const RWCString& prmFile,
                                const RWCString& section);
  int  WriteSharedViewParameters(const RWCString& prmFile,
                                 const RWCString& section);

  //-----------------------------------------------------
  //...Get the current view window
  //   (ie the one which was last clicked on with the mouse)
  //-----------------------------------------------------
  CViewWindow*        GetCurrentViewWindow(void);

  //-----------------------------------------------------
  //...Get/Set the current data indicator
  //   (ie the one which was last clicked on with the mouse)
  //-----------------------------------------------------
  C3dIndicator*       GetCurrentIndicator(void) const
                                  {return currentIndicator;}
  void                SetCurrentIndicator(C3dIndicator* val)
                                  {currentIndicator=val;}

  //-----------------------------------------------------
  //...Get the current geometric object
  //   (ie the one which was last clicked on with the mouse)
  //-----------------------------------------------------
  C3dObject*          GetCurrentObject(void) const
                                  {return currentObject;}
  void                SetCurrentObject(C3dObject* object)
                                  {currentObject = object;}

  //-----------------------------------------------------
  //...Initialize rendering for the dataset
  //
  // The very first time a view is created, we will:
  //  -Create a Set of Cutplanes
  //  -Read the shared view parameters
  //  -Create a CRenderModel
  //  -Create a CViewList
  //-----------------------------------------------------
  int   InitializeRendering(void);

  //-----------------------------------------------------
  //...Create a view of parent dataset
```

```
// -Instantiate a view object
// -Add the view to the set of views
//
// The very first time a view is created, we will:
// -Initialize rendering
//-------------------------------------------------------
CView* CreateView(void);
CView* CreateViewForCopy(void);

//-------------------------------------------------------
//...Delete a particular view of parent dataset
//
// -Remove the view from the set of views
// -Delete the view
//-------------------------------------------------------
 int     RemoveView(CView *view);

//-------------------------------------------------------
//...Delete all views of parent dataset
//
// -Loop over all views in the set of views
// -Remove the view from the set of views
// -Delete the view
//-------------------------------------------------------
int      RemoveAllViews(void);

//-------------------------------------------------------
//...Copy a view to another view
//
// -Instantiate a new view with the copy constructor
// -Add the new view pointer to the set of views
//-------------------------------------------------------
int     CopyView(CView *view);

//-------------------------------------------------------
//...Close a view
//
// -Close a view
//-------------------------------------------------------
static void XTimerCallback(XtPointer cbs, XtIntervalId *id){
          if(id);
          CView *pView = ((CView *)cbs);
          CDatasetViewer *pDatasetViewer = pView->GetDatasetViewerPointer();
          pDatasetViewer->CloseView(pView);}

int CloseView(CView *view);

//-------------------------------------------------------
//..Refresh all views
//
// -Loop over all views in the set of views
// -Call CView's RenderAll method for each view in set
//  (RenderAll forces a complete update (ie loads pointset, etc)
//-------------------------------------------------------
int      UpdateAllViews(void);

//-------------------------------------------------------
//..Refresh all views
//
// -Loop over all views in the set of views
// -Call CView's Render method for each view in set
//  (Render does a partial update depending upon what has changed in the
//   view)
//-------------------------------------------------------
int      UpdateAllViewsNoForce(void);
```



```
//-------------------------------------------------------
//..Update all view's manipulator buttons to coincide with each other
//
// -Loop over all views in the set of views
// -Call CViewWindow's UpdateManipulatorButtons Render method for each
//  view in set

//-------------------------------------------------------
int      UpdateAllViewsManipulatorButtons(int button);

//-------------------------------------------------------
//..Force all colormaps to be recomputed, for all open views
//-------------------------------------------------------
void     UpdateAllColormapChangedFlags(void);

//-------------------------------------------------------
//..Update the object selection window
//-------------------------------------------------------
int      UpdateObjectSelectionWindow(void);

//-------------------------------------------------------------------
//..Display the property display window which contains all propeties
//..at the point represented by the data indicator.
//-------------------------------------------------------------------
int      IndicatorSelected(C3dIndicator *pObject);

//-------------------------------------------------------------------
//..Update the object selection window and the selected object list
//-------------------------------------------------------------------
int      ObjectSelected(C3dObject *object);
int      ObjectUnSelected(C3dObject *object);

//-------------------------------------------------------------------
//..Update the transform edit window because object's transform node has chang
---->ed
//-------------------------------------------------------------------
int      ObjectManipulated(C3dObject *pObject);
int      ObjectManipulated(void) {
           ObjectManipulated(currentObject);
           return(0);
}

//-------------------------------------------------------------------
//..Update the included and excluded object selection list
//-------------------------------------------------------------------
int      ObjectIncluded(C3dObject *object);
int      ObjectUnIncluded(C3dObject *object);
int      ObjectExcluded(C3dObject *object);
int      ObjectUnExcluded(C3dObject *object);

//-------------------------------------------------------------------
//..Synchronize the selected object list with the geometry renderer's
//..selected object list
//-------------------------------------------------------------------
int      UpdateGeometryRenderedObjects(void);

//-------------------------------------------------------------------------
//  A method to take a manipulator type
//  and set it in the render model for all views
//-------------------------------------------------------------------------
void   SetManipType(int type);

//-------------------------------------------------------------------
//..Check to see if the given object is already selected
//..(ie is already in the selected object list)
//-------------------------------------------------------------------
```

```
  boolean IsObjectSelected(C3dObject *pObject);

  //-------------------------------------------------------------------
  //..Update the colormap data for all views
  //-------------------------------------------------------------------
  void UpdateColormapData(void);


  //====================
  //                    //
  // 3dObject methods //
  //                    //
  //====================

  //-------------------------------------------------------------------
  //..Get the center of an object in world space
  //..Passes back a C3dPoint describing the center of the box
  //-------------------------------------------------------------------
  void GetObjectCenter(C3dObject* pObject,
                       CView *pView,
                       C3dPoint* pointCenter);

  //-------------------------------------------------------------------------
  //..Get the bounding box of an object in world space
  //..Passes back 2 C3dPoints describing the min and max vertex of the box
  //-------------------------------------------------------------------------
  void GetObjectBoundingBox(C3dObject* pObject,
                       CView *pView,
                       C3dPoint* pointMin,
                       C3dPoint* pointMax);


  //=========================
  //                        //
  // Status window methods //
  //                        //
  //=========================

  //-------------------------------------------------------------------
  //..Update the status window for all views
  //-------------------------------------------------------------------
  void UpdateStatusWindow(void);

  //-------------------------------------------------------------------
  //..Update the status window's status message line for all views
  //-------------------------------------------------------------------
  void UpdateDatasetStatusMessage(RWCString msg);

  //-------------------------------------------------------------------
  //..Update the status window's status message line for the given view
  //-------------------------------------------------------------------
  void UpdateViewStatusMessage(CView* view, RWCString msg);

  //-------------------------------------------------------------------
  //..Get the current status message line for the given view
  //-------------------------------------------------------------------
  RWCString GetCurrentStatusMessage(CView* pView);
};

#endif
```

```
//*************************************************************************
//  File:               datasetmanagerwin.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CDatasetManagerWindow
//
//  Class Description:  The CDatasetManagerWindow class encapsulates all
//                      behaviors of the dataset manager window in ICERVS III.
//                      The class builds a UIMX generated top level window
//                      and handles all the  window callbacks
//                      for the  menubar and the pop-up manus.
//
//  Inheritance:        CUserCallbackFacilityCDatasetManagerWindow
//
//  Required Class Libraries: Rogue Wave Tools, UNRAS
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/28/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined DatasetManagerWindow_H_
    #define  DatasetManagerWindow_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include  <Xm/Xm.h>                    //Motif include files

#include "ictypes.h"                   //Our own include files
#include "callback.h"
#include "csortstr.h"
#include "group.h"
#include "vdconfig.h"
#include "callback.h"

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CVDSServer;
class CVDSUser;
class CDatasetManager;
```



```
class CFileSelectionWindow;

class CVDSDataset;
class _UxCDatasetManagerMainWindow;
//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------

class CDatasetManagerWindow  :public CUserCallbackFacility

{
//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  int           objectDefined;                  //0=undefined  1=OK  -n=error

  //...Pointers to needed objects
  CDatasetManager       *pDatasetManager;     //pointer to our parent
  CFileSelectionWindow *fileSelectionWindow;
  CFileSelectionWindow *fileSelection2Window;

  //...Our window
  RWCString             windowName;             //our window name
  RWCString             windowTitle;            //our window title
  Widget                myWidget;               //our X widget
  _UxCDatasetManagerMainWindow*  myxWindow;     //our window object (UIMX)

  CVDSServer*           myServer;
  XtInputId             listenInputId;          //Listen socket input id

  //...The list of groups and datasets to be displayed
  CSortedStringList     myGroupList;
  CSortedStringList     myDatasetList;

  //...The currently selected group and dataset
  RWCString             currentGroupName;
  RWCString             currentDatasetName;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
      CDatasetManagerWindow(CDatasetManager *datasetManager,
                            const RWCString& name);
      ~CDatasetManagerWindow(void);


//=========================================================================
// PUBLIC METHODS
//=========================================================================
public:

  boolean       IsDefined(void)          const;

  //-----------------------------------------
  //...Menu Item Callback Methods for Group
  //-----------------------------------------

  //-----------------------------------------------------------------
  //...Quit
```

```
//
// -Call CDatasetManager method: Quit
//-------------------------------------------------------------------
void Quit(void);

//-------------------------------------------------------------------
//...Delete a Group
//
// -Call CDatasetManager method: GroupDelete with the selected group name
// as an argument.
// -Remove the group from the group list and rewrite the list of groups
// on the window, making the previous group the selected one.
//-------------------------------------------------------------------
void GroupDelete(void);


//-------------------------------------------------------------------
//...Edit the Default Site Parameters
//
// -Call the group's CSiteDefinition's method:
//  fname = GetSiteParameterFilename()
// -Call the CSiteDefinition's method: WriteSiteParameters(fname)
// -Invoke the editor object with the site filename as an argument
// -Call the CSiteDefinition's method: ReadSiteParameters(fname)
//-------------------------------------------------------------------
void GroupEditDefaults(void);


//-------------------------------------------------------------------
//...Edit the Group Log File
//
// -See Phase II code: IcerMain::SiteLogEdit
//-------------------------------------------------------------------
void GroupEditLog(void);


//-------------------------------------------------------------------
//...Create a new Group
//
// -Instantiate a CWGroupAdd window to get new group name, description
// -Call CDatasetManager method: GroupNew with the new group name, desc
//  as an argument.
// -Add the group to the group list
// -Rewrite the list of groups on the window, making the new group the
//  selected one.
//-------------------------------------------------------------------
void GroupNew(void);


//===========================================
//...Menu Item Callback Methods for Dataset
//===========================================

//-------------------------------------------------------------------
//...Create View of a dataset
//
// -Call CDatasetViewer method: CreateView
//-------------------------------------------------------------------
void DatasetView(void);


//-------------------------------------------------------------------
//...Make a copy of a dataset
//
// -Instantiate a CWDatasetAdd window to get new dataset name, description
// -Call the DatasetManager method: DatasetCopy with old group name,
```



```
//  old dataset name, new dataset name and description
// -Add the new dataset to the datasetlist
// -Rewrite the list of datasets on the window, making the new dataset the
//  selected one.
//-------------------------------------------------------------------
void DatasetCopy(void);


//-------------------------------------------------------------------
//...Delete a Dataset
//
// -Call CDatasetManager method: DatasetDelete with the selected group name
//  and dataset name as arguments.
// -Remove the dataset from the dataset list and rewrite the list of
//  datasets on the window, making the previous dataset the selected one.
//-------------------------------------------------------------------
void DatasetDelete(void);


//-------------------------------------------------------------------
//...Edit the dataset's site Parameters
//
// -Call the dataset's CSiteDefinition's method:
//  fname = GetSiteParameterFilename()
// -Call the CSiteDefinition's method: WriteSiteParameters(fname)
// -Invoke the editor object with the site filename as an argument
// -Call the CSiteDefinition's method: ReadSiteParameters(fname)
//-------------------------------------------------------------------
void DatasetEditParameters(void);


//-------------------------------------------------------------------
//...Browse the dataset's information
//
// -Call CDatasetManager method: DatasetInfo with the group name,
//  dataset name as an argument (returns filename of just created info file)
// -Invoke Browser Window with filename as argument
// -Delete browse file
//-------------------------------------------------------------------
void DatasetInfo(void);


//-------------------------------------------------------------------
//...Archive a dataset to external media
//
// -Call CDatasetManager method: DatasetArchive with the group name,
//  dataset name as an argument
//-------------------------------------------------------------------
void DatasetArchive(void);
void DatasetArchiveCompleted(UserCBF cbf);
void DatasetArchiveCanceled(UserCBF cbf);


//-------------------------------------------------------------------
//...Retrieve an archived dataset from external media
//
// -Call CDatasetManager method: DatasetRetrieve with the group name,
//  dataset name as an argument
//-------------------------------------------------------------------
void DatasetRetrieve(void);
void DatasetRetrieveCompleted(UserCBF cbf);
void DatasetRetrieveCanceled(UserCBF cbf);


//-------------------------------------------------------------------
//...Create a new Dataset
```

```
  //
  // -Instantiate a CWDatasetAdd window to get new dataset name, description
  // -Call CDatasetManager method: DatasetNew with the new dataset name, desc
  //  as an argument.
  // -Add the dataset to the dataset list
  // -Rewrite the list of datasets on the window, making the new dataset
  //  the current one
  //-------------------------------------------------------------------------
  void DatasetNew(void);

  void GeneralHelp(void);
  void DebugHelp(void);
  void AboutHelp(void);

  void GroupListSelection(XtPointer cbf);
  void DatasetListSelection(XtPointer cbf);
  void DatasetDoubleClickSelection(XtPointer cbf);

  //...Menubar methods
  int  EnableMenuButton(const RWCString& buttonName)     const;
  int  DisableMenuButton(const RWCString& buttonName)    const;
  int  RemoveMenuButton(const RWCString& buttonName)    const;

//============================================================================
// OPERATORS
//============================================================================
public:
    friend ostream& operator<<(ostream& os, const CDatasetManagerWindow& P)
                                                    {P.Print(os); return os;}
    CDatasetManagerWindow& operator=(const CDatasetManagerWindow& other);

//============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//============================================================================
public:


  const RWCString& GetWindowName(void)       const {return windowName;}
  const RWCString& GetWindowTitle(void)      const {return windowTitle;}
        Widget     GetWidget(void)           const {return myWidget;}

  RWCString      GetCurrentDatasetName(void) const {return currentDatasetName;}
  RWCString      GetCurrenGroupName(void)    const {return currentGroupName;}

  void  SetWindowName(const RWCString& name)          {windowName = name;}
  void  SetWindowTitle(const RWCString& name)        {windowTitle = name;}

//============================================================================
// OTHER METHODS
//============================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print object data
  IcerClassType  IsA(void)               const   {return _CDATASETMANAGERWIN;}
  void FillLists(void);


  //...Start the X main event loop for this window
//  int StartMainEventLoop(CVDSServer* server);

  //...Display the window
  int   Start(CVDSServer* vdsServer, CVDSUser* vdsUser);

private:

 void GroupNewCancelled(UserCBF cbf);
 void GroupNewCompleted(UserCBF cbf);
```



```
 void DatasetCopyCancelled(UserCBF cbf);
 void DatasetCopyCompleted(UserCBF cbf);

 void DatasetNewCancelled(UserCBF cbf);
 void DatasetNewCompleted(UserCBF cbf);
};

#endif
```

```
//*********************************************************************
//  File:               filenames.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:  The functions in this file implement the class
//                     CFilenames.
//
//  Class Description:   CFilenames is provided as a means to easily get
//  access to the full path names of all files used in ICERVS Phase III
//  To gain access to the filenames, instantiate a CFilenames object and
//  access data via methods.  Most of the methods are static and can therefore
//  be accessed without instantiating a CFilenames object provided that
//  one has been instantiated (and therefore initialized) somewhere else
//  (such as in the main routine).
//
//  Inheritance:   CFilenames
//
//  Required Class Libraries: Roque Wave
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/31/95    G. Hirschman    Created with inspiration from
//                                          Phase II CGlobalPDC class
//
//*********************************************************************
#if !defined CFilenames_H_
    #define  CFilenames_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                   //C++ input/output streams (ALWAYS)
#include "standard.h"                   //MTI Standard Definitions  (ALWAYS)
#include "ictypes.h"                    //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>
#include <rw/rstream.h>


//=====================================================================
// OTHER DECLARATIONS
//=====================================================================


//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CFilenames
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:
```



```
//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:
    static  boolean initialized ;

    static  RWCString       systemRootPath;
    static  RWCString       systemEtcPath;
    static  RWCString       systemBinPath;
    static  RWCString       systemDataPath;
    static  RWCString       systemLibPath;

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
    CFilenames(void);                                   //Default constructor
    ~CFilenames(void);                                  //Destructor

    CFilenames(const CFilenames& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CFilenames& P)
                                                        {P.Print(os); return os;}
    CFilenames& operator=(const CFilenames& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:
//-----------------------------------------
// The Gets -- Used To Obtain Access To Data
//-----------------------------------------


//-----------------------------------------
// Get Fundamental system paths (directories)
//-----------------------------------------
static RWCString& GetSystemRootPath(void)
        {return CFilenames::systemRootPath;}

static RWCString& GetSystemEtcPath(void)
        { return systemEtcPath; }

static RWCString& GetSystemBinPath(void)
        { return systemBinPath; }

static RWCString& GetSystemDataPath(void)
        { return systemDataPath;}

static RWCString& GetSystemLibPath(void)
        { return systemLibPath;}

//-----------------------------------------
// Get system level files
//-----------------------------------------
static RWCString GetVDSConfigFilename(void)
        { return GetSystemEtcPath() + "/config.vds";}
```

```
static RWCString GetSensorDefinitionFilename(const RWCString sensor)
          { return GetSystemEtcPath() + "/" + sensor + ".def";}

static RWCString GetDefaultThresholdFilename(void)
          { return GetSystemEtcPath() + "/default.thresh";}

static RWCString GetDefaultColormapFilename(void)
          { return GetSystemEtcPath() + "/default.cmap";}

static RWCString GetDefaultViewParametersFilename(void)
          { return GetSystemEtcPath() + "/view.prm";}

static RWCString GetDefaultSiteDefinitionFilename(void)
          { return GetSystemEtcPath() + "/site.def";}

//-----------------------------------------
// Get Group specific path and filenames
//-----------------------------------------
static RWCString GetGroupPath(const RWCString group)
          { return GetSystemDataPath() + "/" + group; }

static RWCString GetDefaultSiteDefinitionFilename(const RWCString group)
          { return GetGroupPath(group) + "/site.def";}

static RWCString GetSiteLogFilename(const RWCString group)
          { return GetGroupPath(group) + "/site.log";}


//-----------------------------------------
// Get Dataset specific path and filenames
//-----------------------------------------
static RWCString GetDatasetPath(const RWCString group,
                                const RWCString dataset)
    { return GetSystemDataPath() + "/" + group + "/" + dataset ; }

static RWCString GetDatasetParametersFilename(const RWCString group,
                                              const RWCString dataset)
    { return GetDatasetPath (group,dataset)  + "/dataset.prm" ; }

static RWCString GetViewParametersFilename(const RWCString group,
                                           const RWCString dataset)
    { return GetDatasetPath(group,dataset)  + "/view.prm" ; }

static RWCString GetObjectFilename(const RWCString group,
                                   const RWCString dataset)
    { return GetDatasetPath(group,dataset) + RWCString("/object.iv");}

static RWCString GetIndicatorFilename(const RWCString group,
                                      const RWCString dataset)
    { return GetDatasetPath(group,dataset) + RWCString("/indicator.iv");}

static RWCString GetSiteDefinitionFilename(const RWCString group,
                                           const RWCString dataset)
    { return GetDatasetPath(group,dataset) + RWCString("/site.def");}

static RWCString GetNonScalarPropertyPath(const RWCString group,
                                          const RWCString dataset)
    { return GetDatasetPath(group,dataset) + "/propertytmp";}

static RWCString GetPermanentNonScalarPropertyPath(const RWCString group,
                                                   const RWCString dataset)
    { return GetDatasetPath(group,dataset) + "/property";}

static RWCString GetTextureMapPath(const RWCString group,
                                   const RWCString dataset)
```



```
    { return GetDatasetPath(group,dataset) + "/texture";}

//------------------------------------------------------------------------
// There are no Sets -- everything is set automatically on instantiation
//------------------------------------------------------------------------


//========================================================================
// OTHER METHODS
//========================================================================
public:
    void    Print(ostream& os=cout)  const;          //Print object data
    IcerClassType IsA(void)          const          {return _CFILENAMES;}
};

#endif
```

```
//*************************************************************************
//  File:               group.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CGroup
//
//  Class Description:  CGroup encapsulates a group of datasets.  It contains
//                      the list of datasets that are part of the group.
//                      It contains the default site parameters for all
//                      datasets within the group.  It also manages the
//                      operator log which stores notes about the datasets
//                      contained within the group.
//
//  Inheritance: <RWCollectable>  <CNamedItem>  <CGroup>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/27/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined CGroup_H_
   #define  CGroup_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                           //C++ input/output streams
#include <standard.h>                           //MTI Standard Definitions
#include <ictypes.h>                            //MTI symbols for types
#include <grouplog.h>                           //CGroupLog
#include <sitedefinition.h>                     //CSiteDefinition
//#include "datasetlist.h>                      //CDatasetList
#include <csortstr.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CGroup     :public CNamedItem
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  RWCString             dataPath;               //path to our group
```



```
  CSiteDefinition       *defaultSite;           //the default site defs
  CGroupLog             groupLog;               //operator log
  COrderedStringList    datasetList;
//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
     CGroup(void);
     CGroup(const RWCString& name,              //Constructor
            const RWCString& description);
     ~CGroup(void);                             //Destructor

     CGroup(const CGroup& other);               //Copy constructor
//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

  //-----------------------------------------------------------------------
  //...Set our data path using group name as directory name
  //...This is called from our constructor
  //...Constructor will then instantiate a CSiteDefinition w/ the dataPath
  //...as an argument to it
  //-----------------------------------------------------------------------
  void SetDataPath(const RWCString& name);

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CGroup& P)
                                                {P.Print(os); return os;}
    CGroup& operator=(const CGroup& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

  const RWCString GetDataPath(void)             const  {return dataPath;}
  CSiteDefinition* GetDefaultSiteDefinition(void)  const  {return defaultSite;}
  CGroupLog       GetGroupLog(void)             const  {return groupLog;}

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
    void          Print(ostream& os=cout)  const;         //Print object data
    IcerClassType IsA(void)                const          {return _CGROUP;}
    int       Initialize(void);
    //...Get the log filename
    RWCString  GetGroupLogFilename(const RWCString* path=0)  const;
    CSortedStringList GetDatasetList(void)             const;
---->
//-------------------------------------------------------------------------
// A few convenience functions
//-------------------------------------------------------------------------
  boolean AppendToLog(const CGroupLogEntry& logEntry)
                    {return groupLog.AppendToLog(logEntry);}

  void   SetGroupId(long idNumber)              {SetIdNumber(idNumber);   }
  void   SetGroupId(const RWCString& id)        {SetTheIdString(id);      }
  void   SetGroupDescription(const RWCString& desc){SetTheDescription(desc); }

  const  RWCString& GetGroupName(void)       const{return GetTheName();     }
```

```
  const  RWCString& GetGroupId(void)         const{return GetTheIdString(); }
  const  RWCString& GetGroupDescription(void)const{return GetTheDescription();}

         RWBoolean isEqual(const RWCollectable *a) const;

};

#endif
```

```
//*************************************************************************
//  File:               groupset.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CGroupSet
//
//  Class Description:  CGroupSet encapsulates the collection of CGroup objects
//
//
//  Inheritance: <RWOrdered> <CSetOfNamedItems> <CGroupSet>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name          Remarks
//  ---------   --------    --------      -------
//  102:00:00   03/20/94    D. Smith      Created
//  103:00:00   04/27/95    L. Cowper     Modified for Phase III
//                                        Got rid of notion of directory
//                                        list file, now uses system services
//
//*************************************************************************/
#if !defined CGroupSet_H_
    #define  CGroupSet_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                       //C++ input/output streams
#include <standard.h>                       //MTI Standard Definitions
#include <ictypes.h>                        //MTI symbols for types

#include <rw/compiler.h>                    //RW compiler ident
#include <rw/cstring.h>                     //RWCString
#include <rw/rstream.h>

#include <csetof.h>                         //CSetOfNamedItems
#include <group.h>                          //CGroup

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CGroupSet :public CSetOfNamedItems
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
```



```
//=========================================================================
private:

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
  CGroupSet(const char* name=0,int initialSize=5);
  ~CGroupSet(void);

   CGroupSet(const CGroupSet& other);        //Copy constructor

//=========================================================================
// PROTECTED METHODS
//=========================================================================
protected:
  CGroup*       Find(int id) const      {return (CGroup *) FindItem(id);}

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CGroupSet& P)
                                                {P.Print(os); return os;}
    CGroupSet& operator=(const CGroupSet& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void          Print(ostream& os=cout)  const;         //Print object data
  IcerClassType IsA(void)                const          {return _CGROUPSET;}

  int           Add(const CGroup* group);

  boolean       Remove(const RWCString& groupName);
  boolean       Remove(CGroup* group);
  boolean       RemoveAll(void);

   boolean       RemoveById(const RWCString& id);

  int           GetListOfNames(CSortedStringList* list)
                        {return GetListOfItemNames(list); }

   int           GetListOfDescriptions(CSortedStringList* list)
                        {return GetListOfItemDescriptions(list); }

//-----------------------------------------
// These functions are defined for convenience
//-----------------------------------------

  CGroup*   Find(const RWCString& groupName) const
                                {return (CGroup *) FindItem(groupName);}

  CGroup*   FindByGroupId(const RWCString& id) const
                                {return (CGroup *) FindItemByIdString(id);}
```

```
  CGroup*   FindById(const RWCString& id) const
                          {return (CGroup *) FindItemByIdString(id);}

  CGroup*   FindByDescription(const RWCString& desc) const
                          {return (CGroup *) FindItemByDescription(desc);}

  CGroup*   GetFirst(void)          {return (CGroup *) GetFirstItem();}
  CGroup*   GetNext(void)           {return (CGroup *) GetNextItem();}
  CGroup*   GetPrevious(void)       {return (CGroup *) GetPreviousItem();}
  CGroup*   GetLast(void)           {return (CGroup *) GetLastItem();}


};

#endif
```

```
//*****************************************************************************/
//  File:               inventor.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CInventor
//
//  Class Description:  CInventor encapsulates all behaviors of the
//                      Open Inventor Tree used for rendering in
//                      ICERVS III.
//
//  Inheritance:        CInventor
//
//  Required Class Libraries:  Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   06/01/95    S. Bardsley     Original
//  103:00:00   08/01/95    L. Cowper       Updated to include object selection
//
//*****************************************************************************/
#if !defined CInventor_H_
  #define  CInventor_H_


//============================================================================
// REQUIRED INCLUDE FILES
//============================================================================
#include  <iostream.h>          //C++ input/output streams (ALWAYS)
#include  <stddef.h>            //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>

#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/actions/SoGetBoundingBoxAction.h>
#include <Inventor/manips/SoTransformBoxManip.h>
#include <Inventor/manips/SoTabBoxManip.h>
#include <Inventor/manips/SoTrackballManip.h>
#include <Inventor/manips/SoTransformManip.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/SoPickedPoint.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/events/SoEvent.h>
#include <Inventor/draggers/SoDragger.h>
#include <Inventor/draggers/SoTabBoxDragger.h>
#include <Inventor/Xt/viewers/SoXtViewer.h>
#include <Inventor/sensors/SoNodeSensor.h>

#include <standard.h>                   //MTI Standard Definitions  (ALWAYS)
#include <ictypes.h>                    //MTI symbols for types
```



```
#include <view.h>
#include <coord3manip.h>

//============================================================================
// OTHER DECLARATIONS
//============================================================================
//----------------------------------------
// Define some tree node types
// There are 3 FULL node types
// Node type 1 is used for EMPTY node type
// Node type 4 is used for UNKNOWN node type
//----------------------------------------
#define SPATIAL_DATA                        1
#define SELECTED_REGION_SPATIAL_DATA        2

#define SPATIAL_DATA_MASK                   0x0002
#define SELECTED_REGION_SPATIAL_DATA_MASK   0x0004
#define NODE_TYPE_3_MASK                    0x0008

#define ALL_SPATIAL_DATA_MASK               0x000E


class C3dObject;

//----------------------------------------------------------------------------
// CLASS DEFINITION
//----------------------------------------------------------------------------
class CInventor
{

//============================================================================
// DECLARATIONS (enums, defines, etc)
//============================================================================
public:

//============================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//============================================================================
private:

  //  Pointers to important objects
  CRenderModel    *pRenderModel;
  CVDSDataset     *pDataset;
  SoSeparator     *pRoot;               //Inventor tree root node
  SoSeparator     *pRoot2;              //Inventor tree selection root node
  SoPickStyle     *pointsetPickStyle;   //Pointset pick style for enabling
                                        //or disabling pointset picking.

  //  Render control variables
  SbBool          mouseReleaseFlag;

  //  Flag indicating if selected object is composite
  boolean   compositeObjectSelected;

  //  Flag indicating if deselected object is composite
  boolean   compositeObjectDeselected;

  // Cache for view parameters
  unsigned  highResolutionDisplayMode;
  unsigned  showObjects;
  unsigned  showIndicators;
  RWCString propertyType;
  unsigned  applyCutplanes;
  CColor    colorLowResolutionPoints;
  CColor    colorSelectedObject;
  CColor    colorSelectedRegion;
  int       antiAliasing;
```

```
  // Cache for colormap parameters
  CColor    primaryMinColor, primaryMaxColor;
  CColor    secondaryMinColor, secondaryMaxColor;
  CColor    outsideColor;
  float     primaryRangeMin, primaryRangeMax;
  float     secondaryRangeMin, secondaryRangeMax;
  int       secondaryRangeOption;
  CView     *currentView;

  // variables used for property coloring
  float* propertyValues;
  float* selectedPropertyValues;

  // The flushCacheFlag invalidates all entries in the cache which will
  // cause all values to be updated regardless of their current state.
  int flushCacheFlag;

  // variables needed for node selection and manipulators
  SoTabBoxManip       *myTabBox;
  SoTransformBoxManip *myTransformBox;
  SoTrackballManip    *myTrackball;
  Coordinate3Manip    *myReshape;
  SoTransformManip    *currentManip;
  SoPath              *manipPath;
  SoSelection         *selectionRoot;
  SoPath              *selectPath;
  boolean             performSelectionCallback;
  const SoEvent       *currentEvent;
  boolean             justChangedManipulatorType;
  int                 myManip;

  // this is needed to fully qualify the inventor scene in the scene database.
  RWCString datasetName;

  boolean             pickFilterFlag;

  // Number of data points
  // (both in the selected region and outside the selected region)
  unsigned int numberNormalPoints;
  unsigned int numberSelectedPoints;

//============================================================================
// CONSTRUCTORS AND DESTRUCTORS
//============================================================================
public:
  CInventor(CRenderModel *parent, CVDSDataset *dataset);
  ~CInventor(void);

  CInventor(const CInventor& other);        //Copy constructor


//============================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//============================================================================
private:


  //--------------------------------------------------------------------
  int CreatePointsetSubgraph(void);
  int CreateCutplaneSubgraph(void);
  int CreateObjectSubgraph(void);
  int CreateIndicatorSubgraph(void);
  int CreateUniverseCubeSubgraph(void);
  //
  //  These methods create the individual branches in the Inventor scene graph
```



```
  //  and is intended to be called only by GenerateScene().  The purpose of
  //  these methods is to create and attach a subgraph to the default scene
  //  graph.  Each of these subgraphs may be unique in some way, therefore
  //  these methods encapsulate that uniqueness.
  //--------------------------------------------------------------------


  //--------------------------------------------------------------------
  SoNode* GetNodeByName(const char *name, const SoType Id);
  //
  //  This is a wrapper around some Inventor calls to make it simpler and
  //  safer to get nodes from the scene graph by name.  The lookup is
  //  started from the root node (pRoot).  If/When the node is found it is
  //  verified to be of the correct type and then returned.  The value
  //  returned must be cast to the correct node type when returned. If the
  //  node is not found or is not of the specified type, a 0 is returned.
  //--------------------------------------------------------------------


  //--------------------------------------------------------------------
  int LoadPointset(const char *nodeName, CView* view);
  //
  //  This method is used to load the SoCoordinate3 node of a pointset.
  //  The point data is acquired from the spatialdata object.
  //--------------------------------------------------------------------


  //--------------------------------------------------------------------
  void ApplyColormap(const char *nodeName, CView* view);
  //
  //  This method is used to apply the colormap to the point data.  The
  //  colormap is specified by data in the CColormapData object, which
  //  is contained in the view.  The color is determined by the currently
  //  selected property.
  //--------------------------------------------------------------------


  //--------------------------------------------------------------------
  SbBool isTransformable(SoNode *aNode);
  SoPath *createTransformPath(SoPath *inputPath);
  //
  // These methods determine if a node is influenced by transforms, and
  // create an SoPath to transforms respectivly.  These are primarily during
  // object selection for attaching manipulators.
  //--------------------------------------------------------------------


//============================================================================
// OPERATORS
//============================================================================
public:
  friend ostream& operator<<(ostream& os, const CInventor& P){
    P.Print(os);
    return os;
  }

  CInventor& operator=(const CInventor& other);


//============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//============================================================================
public:
//  CRenderModel*   GetRenderModelPointer(void);
  SoNode*         GetRootNode(void) {return (SoNode*)pRoot;}
//  SoNode*         GetRootNode(void);
```

```
  SoSelection*      GetSelectionRoot(void)  {return selectionRoot;}


  SoTransformManip* GetCurrentManip(void)  {return currentManip;}

  void    SetPerformSelectionCallback(boolean flag)
                                    {performSelectionCallback = flag;}

  boolean GetPerformSelectionCallback(void) const
                                    {return performSelectionCallback;}

  boolean GetPickFilterFlag(void) const
                                    {return pickFilterFlag;}
//=============================================================================
// OTHER METHODS
//=============================================================================
public:
  void Print(ostream& os=cout)  const;        //Print object data

  IcerClassType IsA(void) const{
    return _CINVENTOR;
  }


  //-------------------------------------------------------------------------
  int Initialize(void);
  int Initialize(const RWCString& inventorFile);
  //
  //  Create the Inventor scene graph new or from an IV file.
  //  - Create root node callback, this must always be first
  //  - Build pointset nodes
  //  - Build Cutplane nodes
  //  - Build Geometric Object nodes
  //  - Build Data Indicator nodes
  //  - Create render timer callback, this must always be last
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int RegisterView(CView* view);
  //
  // Register the view with the scene.  The ViewWindow is connected to the
  // scene so that the view window knows where to send render area events.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  void Render(CView* view);
  //
  //  Apply the view pararmerters from the specified view to the Inventor
  //  scene graph in preparation for rendering.  This method does not really
  //  cause the rendering to occur, but is intended to be called from the
  //  CRenderModel::RenderCallback() method.  So technically rendering has
  //  already commenced, and Render() will cause the scene to render correctly
  //  for the specified CView object.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int Update(CView* view);
  //
  //  Update the view pararmerters from the specified view to the Inventor
  //  scene graph in preparation for rendering.  The difference
  //-------------------------------------------------------------------------
```



```
  //-------------------------------------------------------------------------
  int SaveScene(const RWCString &filename);
  int RestoreScene(const RWCString &filename);
  //
  //  Save the scene to the specified file.  The file saved, is an inventor
  //  ascii file.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddObjects(SoSeparator *node);
  int RemoveObjects(void);
  //
  //  Access geometric objects in Inventor tree
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddCutplanes(SoSeparator *node);
  int RemoveCutplanes(void);
  //
  //  Access cutplanes in Inventor tree
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddIndicators(SoSeparator* node);
  int RemoveIndicators(void);
  //
  //  Access data indicator objects in Inventor tree
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  SbXfBox3f GetSceneBBox(CView* view);
  //
  //  get the bounding box for the scene graph
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  void PointSetSelectionCB(SoPath *selectionPath);
  //
  //  Method called by selection callback when a pointset is selected
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  SoPath* FaceSetSelectionCB(SoPath *selectionPath);
  //
  //  Method called by selection callback when a pointset is selected or
  //  deselected
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  static SoPath* pickFilterCB(void *data, const SoPickedPoint *pick){
    return ((CInventor *)data)->pickFilterCB(pick);
  };
  SoPath* pickFilterCB(const SoPickedPoint *pick);
  //
  //  first thing called when a node in the scene graph gets picked
  //  (it is called before the selection or deselection callbacks)
  //-------------------------------------------------------------------------
```

```
//------------------------------------------------------------------------
static void finishCB(void *data, SoSelection *sel){
  ((CInventor *)data)->finishCB(sel);
};
void finishCB(SoSelection *sel);
//
//  last thing called when a node in the scene graph gets selected
//  or deselected
//  (it is called after the selection or deselection callbacks)
//------------------------------------------------------------------------


//------------------------------------------------------------------------
static void selectionCB(void *data, SoPath *selectionPath){
  ((CInventor *)data)->selectionCB(selectionPath);
};
void selectionCB(SoPath *selectionPath);
//
//  called when a node in the scene graph gets selected by the mouse
//------------------------------------------------------------------------


//------------------------------------------------------------------------
static void deselectionCB(void *data, SoPath *selectionPath){
  ((CInventor *)data)->deselectionCB(selectionPath);
};
void deselectionCB(SoPath *selectionPath);
//
//  called when a node in the scene graph gets deselected by the mouse
//------------------------------------------------------------------------


//------------------------------------------------------------------------
static void eventCB(void *data, SoEventCallback *eventNode){
  ((CInventor *)data)->eventCB(eventNode);
};
void eventCB(SoEventCallback *eventNode);
//
//  called when any inventor event occurs
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    forceCache(void) {flushCacheFlag=1;}
//
//  Force the cache flag to ensure rendering
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    SetManipType(int val);
//
//  Set the manipulator type
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    DetachManipulator(int forceFlag);
void    AttachManipulator(SoPath* selectionPath);
void    AttachManipulator(void);
//
// Detach the current manipulator from the current manipulator path.
// Attach the current manipulator to the current manipulator path.
//------------------------------------------------------------------------
```



```
//------------------------------------------------------------------------
void    DeSelectAllObjects(void);
//
//  Deselect all objects in the Inventor scenegraph
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    SelectAllObjects(void);
//
//  Select all objects in the Inventor scenegraph
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    ObjectSelection(C3dObject *pObject, int selectFlag);
//
// Select/Deselect an object in the Inventor scenegraph
//------------------------------------------------------------------------


//------------------------------------------------------------------------
boolean    ShapeNodeSelected(SoPath *selectionPath);
boolean    PointSetSelected(SoPath *selectionPath);
boolean    IndexedFaceSetSelected(SoPath *selectionPath);
boolean    IsReshapeManipulator(void);
boolean    IsDetachable(void);
boolean    IsManipulatorMode(void);
C3dObject* DataIndicatorSelected(SoPath *selectionPath,SoGroup* pObjectNode);
C3dObject* GeometricObjectSelected(SoGroup* pObjectNode);
SoGroup*   GetSelectedObjectTopNode(SoPath *selectionPath,boolean composite);
boolean    CompositeObjectSelected(SoPath *selectionPath);
//
// Check selected node for various types
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void       PrintPathDebugInfo(SoPath* path, const RWCString &desc);
void       PrintNodeDebugInfo(SoNode* pNode, const RWCString &desc);
//
// Print path and node debug info to standard error
//------------------------------------------------------------------------


//------------------------------------------------------------------------
static void CInventor::adjustScaleTabSizeCB(void*, SoDragger*);
static void CInventor::adjustScaleTabSizeCB(void*, SoXtViewer*);
//
// Callback to adjust the size of scale tabs.  Used only for SoTabBoxManip
// Added to dragger as a motion callback
//------------------------------------------------------------------------


//------------------------------------------------------------------------
SoSeparator* ReadFile(const RWCString& filename);
int WriteFile(SoSeparator* writeNode, const RWCString& filename);
//
//  Read/write an inventor IV file to/from the root node of the scene
//  graph.  ReadFile() returns a valid pointer on success and 0 on failure.
//------------------------------------------------------------------------


//------------------------------------------------------------------------
void    SetPointPicking(boolean enable);
//
//  Enable or disable pointset picking (vs. object picking)
//------------------------------------------------------------------------
```

```
  //------------------------------------------------------------------------
  void    ComputeCameraRange(float* min,float* max);
  //
  //  color by camera range
  //------------------------------------------------------------------------
};
#endif
```

```
//*****************************************************************************/
//  File:               nonscalardata.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CNonscalarData
//
//  Class Description:  CNonscalarData provides a link between the
//                      non-scalar data and the octree which stores a
//                      reference to the non-scalar data on disk
//
//  Inheritance:        CNonscalarData
//
//  Required Class Libraries:
//                      Rogue Wave Math.h++ v6.1
//                      Rogue Wave Tools.h++
//                      TrueSolid v2.3c
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   010/25/95    L. Cowper       Created
//
//*****************************************************************************/
#if !defined CNonscalarData_H_
    #define  CNonscalarData_H_

//===========================================================================
// REQUIRED INCLUDE FILES
//===========================================================================
#include <stddef.h>                     //Standard C include files
#include <iostream.h>                   //C++ input/output streams
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include <Ts/grf.h>
#include <3dpoint.h>

#include <standard.h>                   //Our own include files
#include <ictypes.h>
#include <spatialdata.h>
#include <vdsdataset.h>


//===========================================================================
// OTHER DECLARATIONS
//===========================================================================
static char* nonscalarPropertyName[] = {"non_scalar_data_index", NULL};


//---------------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------------
class CNonscalarData
```



```
{

//===========================================================================
// DECLARATIONS (enums, defines, etc)
//===========================================================================
public:


//===========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//===========================================================================
private:

  //  Pointers to needed objects
  CVDSDataset*      pDataset;       //pointer to parent
  CSpatialData*     pSpatialData;   //pointer to CSpatialData object
  int               nextIndex;      //next nonscalar data index number


  //  Truesolid Variables needed for dealing with the TrueSolid library
  ts_octree_ptr     tree ;              //octree pointer
  ts_setnode_varptr setstuff ;          //ptr to node stuff


//===========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//===========================================================================
public:

  CNonscalarData(CVDSDataset* pDataset);

  ~CNonscalarData(void);

  CNonscalarData(const CNonscalarData& other);        //Copy constructor

//===========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//===========================================================================
private:
int InitializeNonScalarDirectory(void);

//===========================================================================
// OPERATORS
//===========================================================================
public:
  friend ostream& operator<<(ostream& os, const CNonscalarData& P){
    P.Print(os);
    return os;
  }

  // the old copy constructor
  CNonscalarData& operator=(const CNonscalarData& other);

//===========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//===========================================================================
public:
  int GetNextIndex(void)   {return nextIndex;}

//===========================================================================
// OTHER METHODS
//===========================================================================
public:

  void Print(ostream& os=cout)  const;
  IcerClassType IsA(void) const {return _CNONSCALARDATA;}
```

```
  //..Initialize NonScalar data
  int Initialize(void);

  //..Flushes the NonScalar data directory
  int FlushNonScalarDirectory(void);

  //..Add some non-scalar data properties at a given spatial point
  int Add(C3dPoint* point);

  //..Remove all non-scalar data properties at a given spatial point
  int Remove(C3dPoint* point);

  //..Get non-scalar Index value at a given spatial point
  int GetIndex(C3dPoint* point);

  //..Check for existance of nonscalar data
  boolean NonscalarDataExists(C3dPoint* point);

};

#endif
```

```
//*********************************************************************/
//  File:               property.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CProperty
//
//  Class Description:  CProperty encapsulates a material property.
//                      A material property defines a spatial data property
//                      which is part of a dataset.
//
//
//  Inheritance: <RWCollectable> <CNamedItem> <CProperty)
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  102:00:00   03/20/94    D. Smith        Created
//  103:00:00   04/27/95    L. Cowper       Modified for Phase III
//
//*********************************************************************/
#if !defined CProperty_H_
    #define  CProperty_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                           //C++ input/output streams
#include <standard.h>                           //MTI Standard Definitions
#include <ictypes.h>                            //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/ordcltn.h>                         //RWOrdered class
#include <rw/rstream.h>

#include <cnamed.h>                             //CNamedItem

//=====================================================================
// OTHER DECLARATIONS
//=====================================================================
class CVDSDataset;

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CProperty  :public CNamedItem
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
```



```
//=====================================================================
private:
static  unsigned        nextPropertyNumber;     //Next property number for
                                                //defining a unique name

  CVDSDataset   *pDataset;                      //pointer to parent
  RWCString     propertyName;                   //Property name
  RWCString     propertyDesc;                   //Property description
  RWCString     propertyType;                   //Property type
  RWCString     unitsLabel;                     //Property units
  double        minimumRange;                   //Property range min value
  double        maximumRange;                   //Property range max value

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
  CProperty(const RWCString& name, const RWCString& descrip, long id);
  CProperty(CVDSDataset *parent,
            const RWCString& name, const RWCString& descrip, long id);
  CProperty(const RWCString& name, long id);
  CProperty(CNamedItem* owner);
  CProperty(void);
 ~CProperty(void);

    CProperty(const CProperty& other);          //Copy constructor
    CProperty(const CProperty& other, CNamedItem* owner);

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CProperty& P)
                                                {P.Print(os); return os;}
private:
    CProperty& operator=(const CProperty& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:
  void  SetUnitsLabel(const RWCString& units);
  void  SetPropertyType(const RWCString& type)      {propertyType=type;}
  void  SetMinimumRange(double min);
  void  SetMaximumRange(double max);

  const RWCString& GetUnitsLabel(void)      const   {return unitsLabel;}
  const RWCString& GetPropertyType(void)    const   {return propertyType;}
  double GetMinimumRange(void)              const   {return minimumRange;}
  double GetMaximumRange(void)              const   {return maximumRange;}

  void   GetDataRange(double* min,double* max)  const {
                                                  *min = minimumRange;
                                                  *max = maximumRange;
                                                 }

//=====================================================================
// OTHER METHODS
//=====================================================================
public:
  void          Print(ostream& os=cout)  const;         //Print object data
```

```
  IcerClassType IsA(void)                const          {return _CPROPERTY;};

  int           ReadParameters(long id,  const RWCString& prmFile);
  int           WriteParameters(long id, const RWCString& prmFile)      const;
  int           WriteParameters(long id, ostream& os)                   const;

  RWCString     GetNewPropertyId(void);


//-----------------------------------------------------------------------
// These functions are defined for convenience
//-----------------------------------------------------------------------
  void          SetPropertyId(const RWCString& id)    {SetTheIdString(id);}
  void          SetPropertyName(const RWCString& name);
  void          SetPropertyDescription(const RWCString& desc)
                                                  {SetTheDescription(desc);}

  const RWCString& GetPropertyId(void)       const {return GetTheIdString();}
  const RWCString& GetPropertyName(void)     const {return GetTheName();}

  const RWCString& GetPropertyDescription(void) const
                                                 {return GetTheDescription();}

};

#endif
```

```
//*********************************************************************/
//  File:               propertyset.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CPropertySet
//
//  Class Description:  CPropertySet encapsulates the set of properties.
//
//  Inheritance: <RWOrdered> <CSetOfNamedItems> <CPropertySet>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  102:00:00   03/20/94    D. Smith        Created
//  103:00:00   04/27/95    L. Cowper       Modified for Phase III
//
//*********************************************************************/
#if !defined CPropertySet_H_
    #define  CPropertySet_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                       //C++ input/output streams
#include <standard.h>                       //MTI Standard Definitions
#include <ictypes.h>                        //MTI symbols for types

#include <rw/compiler.h>                    //RW compiler ident
#include <rw/rstream.h>

#include <csetof.h>                         //CSetOfNamedItems
#include <property.h>                        //CProperty

//=====================================================================
// OTHER DECLARATIONS
//=====================================================================

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CPropertySet :public CSetOfNamedItems
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:

//=====================================================================
```



```
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
     CPropertySet(const char* setName,int initialSize=5);
    ~CPropertySet(void);                            //Destructor

    CPropertySet(const CPropertySet& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CPropertySet& P)
                                                {P.Print(os); return os;}
    CPropertySet& operator=(const CPropertySet& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:

//=====================================================================
// OTHER METHODS
//=====================================================================
public:
  void          Print(ostream& os=cout)  const;         //Print object data
  IcerClassType IsA(void)                const         {return _CPROPERTYSET;}

  CProperty*    FindAndDuplicate(const RWCString& name) const;

  int           Add(const CProperty* property);
  int           Add(const CProperty* property, boolean noDuplicates);

  boolean       Remove(const RWCString&      propertyName);
  boolean       Remove(CProperty* property);
  boolean       RemoveAll(void);

//---------------------------------------------------------------------
// These functions are defined for convenience
//---------------------------------------------------------------------
  int           GetListOfNames(CSortedStringList* aList)        const
                        {return GetListOfItemNames(aList); }

  int           GetListOfDescriptions(CSortedStringList* aList) const
                        {return GetListOfItemDescriptions(aList); }

  CProperty*    Find(const RWCString& propName) const
                        {return (CProperty *) FindItem(propName);}

  CProperty*  GetFirst(void)
                        {return (CProperty *) GetFirstItem();       }

  CProperty*  GetNext(void)
                        {return (CProperty *) GetNextItem();        }

  CProperty*  GetPrevious(void)
                        {return (CProperty *) GetPreviousItem();    }

  CProperty*  GetLast(void)
                        {return (CProperty *) GetLastItem();        }
```

```
};

#endif
```

```
//*************************************************************************/
//  File:                psuedomenu.h
//
//  Project Name:        ICERVS Phase III
//
//  File description:    This file contains the declaration for class
//                       CPsuedoMenu
//
//  Class Description:  The CPsuedoMenu class encapsulates all behaviors of
//  the view window psuedo menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on psuedo menu CB functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CPsuedoMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name           Remarks
//  ---------   --------    --------       -------
//  103:00:00   04/11/96    H.Belawski     Created
//
//*************************************************************************/
#if !defined PsuedoMenuCB_H_
    #define  PsuedoMenuCB_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include <standard.h>
#include <ictypes.h>
#include <viewwin.h>
#include <callback.h>



//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CPropertyValues;

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------

class CPsuedoMenu :public CUserCallbackFacility
```



```
{

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  //...Pointers to needed objects
  CViewWindow     *pViewWindow;       //pointer to our parent
  CView           *pView;          //pointer to view object

  CPropertyValues *propertyWin;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    CPsuedoMenu(CViewWindow *viewWindow);
    ~CPsuedoMenu(void);

        CPsuedoMenu(const CPsuedoMenu& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:


//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CPsuedoMenu& P)
                                                    {P.Print(os); return os;}
    CPsuedoMenu& operator=(const CPsuedoMenu& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  CViewWindow*      GetViewWindowPointer(void)     const {return pViewWindow;}
  CView*            GetViewPointer(void)           const {return pView;}
  CPropertyValues*  GetPropertyWindowPointer(void)
                                  const {return propertyWin;}

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType IsA(void)                const        {return _CDATASETMENU;}

  //-----------------------------------------------------------------------
  //...View Window Inventor Function: Property values
  //...List property values associated with a data point indicator
  //
  //..Display the property display window which contains all propeties
  //..at the point represented by the data indicator.
  //-----------------------------------------------------------------------
     int IndicatorSelected(C3dIndicator *pObject);


  // Button callback
     void IndicatorSelectedCancel(UserCBF cbf);
```

```
);

#endif
```

```
//*****************************************************************************/
//  File:                rendermodel.h
//
//  Project Name:        ICERVS Phase III
//
//  File description:    This file contains the declaration for class
//                       CRenderModel
//
//  Class Description:   CRenderModel encapsulates all behaviors of rendering
//                       in ICERVS III.   It serves as an interface between
//                       the VDS software and the actual rendering of the
//                       Truesolid octree and the rendering of the Open
//                       Inventor tree.
//
//
//  Inheritance:         CRenderModel
//
//  Required Class Libraries: Rogue Wave Math/Tools, Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name           Remarks
//  ---------   --------    --------       -------
//  103:00:00   04/26/95    L. Cowper      Created
//
//*****************************************************************************/
#if !defined CRenderModel_H_
    #define  CRenderModel_H_


//============================================================================
// REQUIRED INCLUDE FILES
//============================================================================
#include <iostream.h>                 //C++ input/output streams (ALWAYS)
#include  <stddef.h>                  //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>

#include <rw/compiler.h>                      //RW compiler ident
#include <rw/cstring.h>                       //RWCString
#include <rw/rstream.h>
#include <rw/rwdate.h>                        //RWDate class
#include <rw/rwtime.h>                        //RWTime class

#include <standard.h>                 //MTI Standard Definitions
#include <ictypes.h>                  //MTI symbols for types
#include <dictionary.h>
#include <vdsdataset.h>
#include <3dobject.h>
#include <cutplane.h>
#include <view.h>
#include <viewwin.h>


//============================================================================
// OTHER DECLARATIONS
```



```
//============================================================================
class CInventor;
class CTruesolidView;


//============================================================================
// CLASS DEFINITION
//============================================================================
class CRenderModel{


//============================================================================
// DECLARATIONS (enums, defines, etc)
//============================================================================
public:

//============================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//============================================================================
private:
  CDatasetViewer* pDatasetViewer;
  CVDSDataset*    pDataset;
  CInventor*      pGeometryRenderer;

  //  Look Up table objects
  CDictionary     ViewTable;              //render area to view lookup
  CDictionary     VolumeViewTable;        //view to Truesolid view lookup

  //  Variables for the StartTimer and StopTimer functions
  struct itimerval StartTime, StopTime;


//============================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//============================================================================
private:

  //---------------------------------------------------------------------------
  int RegisterView(CView* view);
  void unRegisterView(CView* view);
  int RegisterTsView(CTruesolidView* tsView);
  void unRegisterTsView(CTruesolidView* tsView);
  //
  //  Lookup table access (given render area, return view pointer)
  //---------------------------------------------------------------------------


//============================================================================
// CONSTRUCTORS AND DESTRUCTORS
//============================================================================
public:
  CRenderModel(CDatasetViewer *parent);
  ~CRenderModel(void);

  CRenderModel(const CRenderModel& other);      //Copy constructor


//============================================================================
// OPERATORS
//============================================================================
public:
  friend ostream& operator<<(ostream& os, const CRenderModel& P){
    P.Print(os);
    return os;
  }
```

```
  CRenderModel& operator=(const CRenderModel& other);


//============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//============================================================================
public:
  CDatasetViewer* GetDatasetViewerPointer(void) {return pDatasetViewer;}
  CInventor* GetInventorPointer(void) {return pGeometryRenderer;}
//  CVDSDataset* GetDatasetPointer(void) {return pDataset;}

  CDictionary* GetViewTable(void) {return &ViewTable;}


//============================================================================
// OTHER METHODS
//============================================================================
public:
  void Print(ostream& os=cout)  const;        //Print object data
  IcerClassType IsA(void) const {return _CRENDERMODEL;}


  //-------------------------------------------------------------------------
  int Initialize(void);
  int Initialize(const RWCString &filename);
  //
  //  Initialize the render model.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int SaveScene(CView *view, const RWCString &filename);
  int RestoreScene(CView *view, const RWCString &filename);
  //
  //  Save the render model's scene.  This includes both the geometry and
  //  volumetric scenes  (for now only the geometry info is saved).  The saved
  //  scene can be recalled with the Initalize(RWCString &filename) method.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddObjectSet(C3dObjectSet *objectSet);          //add object to tree
  int RemoveObjectSet(void);                          //remove object from tree
  //
  void SelectAllObjects(void);                        //select all objects
  void UnSelectAllObjects(void);                      //un-select all objects
  //
  void ObjectSelection(C3dObject *pObject, int selectFlag);//select or unselect
                                                           //an object
  //  Inventor geometric object interface methods
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddIndicatorSet(C3dObjectSet *indicatorSet);  //add indicators to tree
  int RemoveIndicatorSet(C3dObjectSet *indicatorSet); //remove indicators
  //
  //  Inventor data indicator interface methods
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int AddCutplaneSet(C3dObjectSet *cutplaneSet);    //add cutplanes to tree
  int RemoveCutplaneSet(void);   //remove cutplanes from tree
  //
  //  Inventor cutplane interface methods
```



```
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int CreatePSImage(const CViewWindow* view,
                    const RWCString& outputFile);
  //
  //  Create a Postscript file from GL buffer image
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int CreateDIBImage(const CViewWindow* view,
                     const RWCString& outputFile);

  //
  //  Create a DIB file from GL buffer image
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int CreateThreshold(const CViewWindow* view,
                      int nColors,
                      const RWCString& rgb,
                      const RWCString& threshold);
  //
  //  Create a spatial data colormap file from a set of RGB values
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int GetWorldPointCoordinates(const CViewWindow* view,
                               double xScreen, double yScreen,
                               double* x, double* y, double* z);
  //
  //  Get a spatial data point given a screen coordinate
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int LinkView(CView* view);
  //
  // Link a view to the render model
  // This must be called each time a new view is created by the CDatasetViewer
  //  -Register Examiner Viewer with the scene graph
  //  -call RegisterView method to add view pointer to lookup table with view
  //     render area.
  //  -instantiate a CTruesolidView object
  //  -call InitializeTS method to initialize CTruesolidView object
  //  -call RegisterTsView method to add view pointer to lookup table with
  //     CTruesolidView object pointer.
  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------
  int UnlinkView(CView* view);
  //
  //  UnLink a view from the render model
  //  This must be called each time a view is deleted by the CDatasetViewer
  //  -call UnregisterView method to remove entry from lookup table for CView
  //     object
  //  -get the CTruesolidView object pointer from the tsview lookup table
  //  -delete the CTruesolidView object
  //  -call UnRegisterTsView method to remove entry from lookup table for
  //     CTruesolidView object pointer
  //-------------------------------------------------------------------------
```

```
  //--------------------------------------------------------------------
  SoNode* GetSceneRoot(void) {return pGeometryRenderer->GetRootNode();}
  //
  //  Get the root node of the inventor scene graph.  This is a violation
  //  of the public interface.  It is needed by the view window for camera
  //  view transformations.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  SbXfBox3f GetBoundingBox(CView* view){
    return pGeometryRenderer->GetSceneBBox(view); }
  //
  //  get the bounding box for the scene in the specified view.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  int Render(CView* view);
  //
  //  This method causes the Render Model to update all pertinent view
  //  parameters in the Geometry Renderer, and the Volumetric Renderer.
  //  And if necessary the scene is re-rendered.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  int RenderAll(CView* view);
  //
  //  This method causes the Render Model to update all pertinent view
  //  parameters in the Geometry Renderer, and the Volumetric Renderer.
  //  And if necessary the scene is re-rendered. It also forces the cache
  //  flag in both renderers to force a recompute of the pointset in the
  //  geometry renderer and to force a truesolid recompute in the volumetric
  //  renderer.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  static void RenderCallback(void *data, SoAction *action){
    ((CRenderModel *)data)->RenderCallback(action);
  }
  void RenderCallback(SoAction *action);
  //
  //  The rendering callback will determine which view caused the render
  //  action, and then call Render() with the appropriate view pointer.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  static void StopTimerCallback(void *data, SoAction *action){
    ((CRenderModel *)data)->StopTimerCallback(action);
  }
  void StopTimerCallback(SoAction *action);
  //
  //  Rendering timer callback.
  //  During rendering a timer is started at beginning and stopped at end
  //  in order to get an image update rate
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  void StartTimer(void);
  float StopTimer(void);
  //
  //  These routines are used for starting and stopping the interval timer
  //  used for timing rendering.  StartTimer() is called by  Render().
  //  A callback node on the scene graph will call StopTimer() when scene
  //  graph traversal is complete.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
```



```
  void   SetManipType(int type);
  //
  //  A method to take the manipulator type selected in the view window,
  //  and set it in the geometric renderer.
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  void    DetachManipulator(int forceFlag) {
    pGeometryRenderer->DetachManipulator(forceFlag);}
  //
  //  Detach the current manipulator
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  void    SetPointPicking(boolean enable) {
    pGeometryRenderer->SetPointPicking(enable);}
  //
  //  Enable or disable pointset picking (vs. object picking)
  //--------------------------------------------------------------------

  //--------------------------------------------------------------------
  void    ComputeCameraRange(float* min, float* max) {
    pGeometryRenderer->ComputeCameraRange(min, max);}
  //
  //  color by camera range
  //--------------------------------------------------------------------
};
#endif
```

```
//*************************************************************************/
//  File:                sitedefinition.h
//
//  Project Name:        ICERVS Phase III
//
//  File description:    This file contains the declaration for class
//                       CSiteDefinition
//
//  Class Description:   CSiteDefinition encapsulates the definition of a site.
//                       A site defines the dimensional aspects of the world
//                       space being modeled.
//
//                       When a CSite is part of a CGroup, it represents the
//                       default site definitions.
//
//                       When a CSite is part of a CDataset, it represents the
//                       site definitions used for the dataset.
//
//  Inheritance: <RWCollectable>  <CNamedItem>  <CSiteDefinition>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name          Remarks
//  ---------   --------    --------      -------
//  102:00:00   03/20/94    D. Smith      Created
//  103:00:00   04/27/95    L. Cowper     Modified for Phase III
//                                        took out site directory list
//                                        in lieu of using system calls
//                                        to manage directorys
//
//*************************************************************************/
#if !defined CSiteDefinition_H_
    #define  CSiteDefinition_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                   //C++ input/output streams )
#include <standard.h>                   //MTI Standard Definitions
#include <ictypes.h>                    //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>                         //RWCString
#include <rw/rstream.h>

#include <cnamed.h>                             //CNamedItem
#include <standard.h>                            //MTI Standard Definitions
#include <ictypes.h>                             //MTI symbols for types

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CSiteDefinition :public CNamedItem
{
```



```
//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  RWCString             groupName;              //group name
  RWCString             datasetName;            //dataset name

  //...The following are read from the site definition file
  RWCString             siteName;               //Site name
  RWCString             siteDescription;        //Site description

  RWCString             spatialDataUnits;       //Default units for data
  double                dataResolution;         //Default data resolution in
                                                //        dataset units

  double                siteHeight;             //Height of site (z-axis)
  double                siteWidth;              //Width of site  (x-axis)
  double                siteLength;             //Length of site (y-axis)
  double                siteHeightOffset;       //Height Offset from origin
  double                siteWidthOffset;        //Width Offset from origin
  double                siteLengthOffset;       //Length offset from origin

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
     CSiteDefinition(const RWCString& group,
                     const RWCString& dataset);             //Constructor
     ~CSiteDefinition(void);                                //Destructor

     CSiteDefinition(const CSiteDefinition& other);         //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:
    //...Get the site definition filename (for both dataset and group specific)
    RWCString  GetSiteDefinitionFilename(void) const;

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CSiteDefinition& P)
                                                        {P.Print(os); return os;}
    CSiteDefinition& operator=(const CSiteDefinition& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

  //---------------------------------------------------------
  //...These definitions are set by the instatiating class
  //---------------------------------------------------------
  void  SetSiteName(const RWCString& name)          {siteName = name;}
  void  SetSiteDescritpion(const RWCString& desc)   {siteDescription = desc;}

  const RWCString&  GetSiteName(void)         const  {return siteName;}
  const RWCString&  GetSiteDescription(void)  const  {return siteDescription;}
```

```
  const RWCString&  GetSpatialDataUnits(void)  const  {return spatialDataUnits;}
  const double      GetDataResolution(void)    const  {return dataResolution;}
  const double      GetSiteHeight(void)        const  {return siteHeight;}
  const double      GetSiteWidth(void)         const  {return siteWidth;}
  const double      GetSiteLength(void)        const  {return siteLength;}
  const double      GetSiteHeightOffset(void)  const  {return siteHeightOffset;}
  const double      GetSiteWidthOffset(void)   const  {return siteWidthOffset;}
  const double      GetSiteLengthOffset(void)  const  {return siteLengthOffset;}

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
    void          Print(ostream& os=cout)  const;    //Print object data
    IcerClassType IsA(void)                const    {return _CSITEDEFINITION;}

    //...Read site definition file
    int ReadSiteDefinitions(const RWCString& prmFile);

    //...Write site definition file
    int WriteSiteDefinitions(const RWCString& prmFile) const;

    //...Read site definition file
    int ReadSiteDefinitions(void);

    //...Write site definition file
    int WriteSiteDefinitions(void) const;

    //...Print site definition file to file
    int PrintDefinitions(const RWCString& outFile) const;

    //...Print site definition file to stream
    int PrintDefinitions(ostream& os) const;

    //...Create a site definition file
    int CreateDefinitionFile(const RWCString& filename) const;

};

#endif
```

```
//*************************************************************************/
//  File:               spatialdata.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CSpatialData
//
//  Class Description:  CSpatialData encapsulates the interface between
//  the VDS and the truesolid octree library.  It handles all non rendering
//  functionality. (CTruesolidView handles the rendering aspects)
//
//  Inheritance: <CSpatialData>
//
//  Required Class Libraries: Rogue Wave Math/Tools, TrueSolid
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name          Remarks
//  ---------   --------    --------      -------
//  102:00:00   03/20/94    L. Cowper     Created
//  103:00:00   06/10/95    L. Cowper     Modified for Phase III
//
//*************************************************************************/
#if !defined CSpatialData_H_
   #define  CSpatialData_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                          //C++ input/output streams
#include <standard.h>                          //MTI Standard Definitions
#include <ictypes.h>                           //MTI symbols for types

#include <unistd.h>
#include <rw/compiler.h>                       //RW compiler ident
#include <rw/cstring.h>                        //RWCString class
#include <rw/rstream.h>

#include  <Ts/octree.h>
#include  <Ts/Csg_memprim.h>

#include <3dconvert.h>

class C3dPoint;
class C3dObjectSet;
class C3dObject;
class CAnalysis;
class COrderedStringList;
class CVDSDatasetSet;
class CVDSDataset;
class CView;

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
#define MAX_PROPERTIES          32
#define PROPERTY_DEFAULT_FLOAT  -32768.
#define PROPERTY_DEFAULT_INT    -32768
```



```
#define BOTTOM_TREE_LEVEL  20
#define MAX_NODE_INIT      3

//----------------------------------------
// Define some tree node types
// There are 3 FULL node types
// F1 nodes are normal nodes
// F2 are selected region nodes
// F3 are nodes that were just erased
// Node type 1 is used for EMPTY node type
// Node type 3 is used for UNKNOWN node type
//----------------------------------------
#define EMPTY_NODE          0
#define NODE_TYPE_1         1
#define NODE_TYPE_2         2
#define NODE_TYPE_3         3
#define UNKNOWN_NODE        3

#define EMPTY_NODE_MASK          0x0001     //node type 0
#define NODE_TYPE_1_MASK         0x0002     //node type 1
#define NODE_TYPE_2_MASK         0x0004     //node type 2
#define NODE_TYPE_3_MASK         0x0008     //node type 3
#define UNKNOWN_NODE_MASK        0x0008     //node type 3

#define FULL_NODE_MASK           0x0016     //nodes types 1,2,4

#define TS_MIN                   -1.
#define TS_MAX                   .9999
//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CSpatialData
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  int               objectDefined;              //object is defined flag
  boolean           nodesExist;                 //tree nodes exist flag
  C3dConvert        dataConvert;                //spatial data conversion
  CConvert          propertyConvert;            //property conversion

  RWCString         dataset;                    //associated dataset name
  RWCString         propertyID;                 //property being viewed
  RWCString         prop_names[MAX_PROPERTIES]; //array of property names

  CVDSDataset       *pDataset;                  //pointer to parent object
  CAnalysis         *pAnalysis;                 //pointer to analysis object

  double            tankSize_xmin,tankSize_xmax;   //Adjusted World Dimensions
  double            tankSize_ymin,tankSize_ymax;   //Set from outside
  double            tankSize_zmin,tankSize_zmax;

  double            propertyMin[MAX_PROPERTIES]; //array of property min ranges
  double            propertyMax[MAX_PROPERTIES]; //array of property max ranges

  //-------------------------------------------------------------------
  //...Variables needed by Truesolid (used to be old v structure)
  //...Made public so other objects can access elements same as old way
  //-------------------------------------------------------------------
```

```
public:
  ts_octree_ptr     tree ;                  //octree ptr
  ts_octree_ptr     treetemp ;              //temp octree ptr

  ts_Csg_varptr     csgstuff;               //TS Csg related things
  ts_setnode_varptr setstuff ;              //ptr to node stuff

  ts_int16          property[MAX_PROPERTIES];   //preallocated ptrs to props
  float             property2[MAX_PROPERTIES];  //preallocated ptrs to props
  int               num_Properties;         //number of properties

  FILE              *fp;                    //for ts_for_all_nodes io
  int               nodeCount;              //tree node count
  int               level;                  //tree level
  double            volume;                 //dumped nodes volume
  int               numDerivedProps;        //no. of derived props(1 or 2)
  int               derivedPropertyIndex[2];//array of indices of derived
                                            //properties in prop list
  int               currentNodeType;
  C3dPoint          *pDataPoints;           //array of data points in tree
  C3dPoint          *pSelectedDataPoints;   //array of data points in tree

  float             *pPropertyValues;       //array of prop values in tree
  float             *pSelectedPropertyValues;//array of prop values in tree

  RWCString         displayPropertyName;    //property name for display
  int               displayPropertyIndex;   //display property index

  float             lowerThresholdMin;      //min threshold value
  float             lowerThresholdMax;      //max threshold value
  float             upperThresholdMin;      //min threshold value
  float             upperThresholdMax;      //max threshold value

  long int          infoArray[32][16];      //array of tree info

//...Existing properties (ie those which were added to the dataset)
  int num_existing_properties;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
  CSpatialData(CVDSDataset *pDataset);
  ~CSpatialData(void);                      //Destructor

  CSpatialData(const CSpatialData& other);  //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:
  //...Needed to get rid of linker warning
  void      TsCopyright(void)    const                {if (ts_copyright);}

//-------------------------------------------------------------------------
//...Initialize all properties that should be known to Truesolid.  These
//...include the standard properties: (z-value, non-spatial property index),
//...and the existing properties: (those properties which are listed in the
//...dataset parameter file)
//-------------------------------------------------------------------------
  int       InitProperties(ts_octree_ptr tree);

//-------------------------------------------------------------------------
//...Syncronizes the property lists between the dataset and the octree.
//...If a property exists in the dataset that's not in the octree, it is
//...removed from the dataset list. It is assumed that the octree is always
```



```
//...right.
//-------------------------------------------------------------------------
  void      SyncProperties(ts_octree_ptr tree , CVDSDataset* pDataset);

//-------------------------------------------------------------------------
//...Changes all imbedded spaces in the property type to underscores.
//...The property type is used for the file extensions on the property files
//...and must not have any imbedded spaces in it.
//-------------------------------------------------------------------------
  void      FixPropertyType(RWCString& type);

//-------------------------------------------------------------------------
//...Sets all property's range values in the v structure so the truesolid
//...callback functions can access these values.
//-------------------------------------------------------------------------
  int       SetPropertyRange(void);

//-------------------------------------------------------------------------
//...Creates a dummy node in a new tree so that it won't be empty.  This will
//...also write a default property value to all added properties.
//...(Truesolid has problems with empty trees!)
//-------------------------------------------------------------------------
  void      CreateTopNode(void);

//-----------------------------------------
//...Writes an ASCII heading to an output report
//-----------------------------------------
  void      WriteASCIIHeading(FILE *fd);

//-----------------------------------
//...Get a list of node types in tree
//-----------------------------------
  int       GetNodeList(unsigned int* node_mask);

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CSpatialData& P)
                                                    {P.Print(os); return os;}
    CSpatialData& operator=(const CSpatialData& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  void      SetTankSize(float xmin, float xmax,
                        float ymin, float ymax,
                        float zmin, float zmax)    {tankSize_xmin = xmin;
                                                    tankSize_ymin = ymin;
                                                    tankSize_zmin = zmin;
                                                    tankSize_xmax = xmax;
                                                    tankSize_ymax = ymax;
                                                    tankSize_zmax = zmax; }

  int           GetNumProperties(void)      const    {return num_Properties;}
  ts_octree_ptr GetTree(void)       const            {return tree;}
  ts_Csg_varptr GetCsgstuff(void)   const            {return csgstuff;}
  ts_setnode_varptr GetSetstuff(void) const          {return setstuff;}
  CVDSDataset*  GetDatasetPointer(void)  const       {return pDataset;}
  CAnalysis*    GetAnalysisPointer(void)  const      {return pAnalysis;}
  C3dConvert    GetDataConvert(void) const           {return dataConvert;}

//=========================================================================
// OTHER METHODS
//=========================================================================
```

```
public:
  void          Print(ostream& os=cout)  const;      //Print object data
  IcerClassType IsA(void)                const       {return _CSPATIALDATA;}

  int           InitializeTree(void);                //Initialize the octree
  boolean       IsTreeChanged(void)   const;         //Has tree changed flag
  boolean       DoesTreeExist(void)   const;         //Does tree exist flag
  boolean       NodesExist(void)      const          //Do tree nodes exist flag
                                      {return nodesExist;}
  boolean       SpatialDataExists(void)  const;      //Does spatial data exist
  int           UpdateTreeInfo(void)     const;      //Update tree info

  //-------------------------------------------------------------
  //...Returns a pointer to an octree
  //... 0=Current Tree,  -1=temporary tree
  //-------------------------------------------------------------
  ts_octree_ptr GetOctree(int number=0)  const;  //Get pointer to octree (1-16)

//--------------------------------------
//...Gets a given property's range values
//--------------------------------------
  int        GetPropertyRange(const RWCString& propertyType, double *propMin ,
                                                             double *propMax);
  int        SetPropertyRange(const RWCString& propertyType,
                              double    propMin ,
                              double    propMax);

//-------------------------------------------------------------------------
//...These Callback methods used by Truesolid are called for each node
//...in the octree that is traversed
//-------------------------------------------------------------------------
  int    Real_dump_node_for_ASCII_file( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_Memory( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_property_Memory( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_Memory_with_property( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_property_display( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_thresholding( Ts_for_nodes_info_ptr info );
  int    Real_dump_node_for_dual_thresholding( Ts_for_nodes_info_ptr info );

  //---------------------------
  //...Creates a new empty tree
  //---------------------------
  int        NewTree(void);

  //-------------------------------------------------------------------
  //...Creates a tree from an existing octree file.  All properties that
  //...currently exist for the dataset will be read. The name of the octree
  //...is contained in the dataset class.
  //-------------------------------------------------------------------
  int        ExistingTree(void);

  //-----------------------------------------
  //...Reads an octree file from disk into memory
  //-----------------------------------------
  int        ReadTree(int treeflag,
                      const RWCString& treeFileName,
                      int nodeFlag);

  //-------------------------------------------------------------------
  //...Writes an octree file to disk from memory. Also writes all property
  //...files that Truesolid was told about.
  //-------------------------------------------------------------------
  int        SaveTree(const RWCString& fileNmae);

  //-----------------------------------------
  // Gets node information about the current tree
```



```
  //-----------------------------------------
  int        GetTreeInfo(const RWCString& filename);

  //-----------------------------------------------
  //...Gets node information about a particular tree
  //-----------------------------------------------
  int        GetTreeInfo(const RWCString& filename, ts_octree_ptr t);

  //-----------------------------------------------
  //...Gets the tree filename for the working dataset
  //-----------------------------------------------
  RWCString  GetTreeFilename(void);
  RWCString  GetTemporaryTreeFilename(void);

  //-------------------------------------------------------------------------
  //...Adds a point to the octree.  Also adds an array of properties.
  //...Also adds the 2 standard properties (z-value,non-scalar property index).
  //-------------------------------------------------------------------------
  int        AddTreePoint(int numProperties,
                          C3dPoint* volumetricPoint,
                      int propertyFlag,
                          double* properties,
                          COrderedStringList propertyNames,
                          int nodeType );

  //-------------------------------------------------------------------------
  //...Updates the internal property list in spatial data.
  //...It must be called before adding any data points with AddTreePoint.
  //-------------------------------------------------------------------------
  int        AddTreePointsStart(void);

  //-------------------------------------------------------------------------
  //...Updates the display field with the new property values just added.
  //...It must be called after adding the last data point with AddTreePoint.
  //-------------------------------------------------------------------------
  int        AddTreePointsDone(int numProperties,
                COrderedStringList propertyNames);

  //-----------------------------------------------
  //...Dumps the contents of a tree to an ASCII file
  //-----------------------------------------------
  int        TreeToASCII( const RWCString& fileName );

  //-----------------------------------------------
  //...Dumps the contents of a tree for a given node type to a memory array
  //-----------------------------------------------
  C3dPoint*  TreeToMemory(unsigned int nodeTypeMask);

  //-----------------------------------------------
  //...Erase all nodes in the octree with the
  //...property value within a given threshold
  //-----------------------------------------------
  int        EraseThreshold(const RWCString& propertyName,
                            float threshMin,
                            float threshMax);

  //-----------------------------------------------
  //...Erase all nodes in the octree with the
  //...property value within 2 given threshold2
  //-----------------------------------------------
  int        EraseThreshold(const RWCString& propertyName,
                            float lowerThreshMin,
                            float lowerThreshMax,
                            float upperThreshMin,
                            float upperThreshMax);
```

```
  //-----------------------------------------------------
  //...Deletes the data point and property memory arrays
  //-----------------------------------------------------
  void       SpatialDataArrayCleanUp(void);

  //-----------------------------------------------------
  //...Dumps the given scalar property values for a given node type
  //...to a memory array
  //-----------------------------------------------------
  float*     TreeToPropertyMemory( const RWCString& propertyName,
                                   unsigned int nodeTypeMask );

  //-----------------------------------------------------
  //...Dumps the contents of a tree for a given node type
  //...(both spatial data and and a given scalar property values)
  //...to memory arrays
  //-----------------------------------------------------
  void       TreeToMemoryWithProperty(const RWCString& propertyName,
                                      unsigned int nodeTypeMask,
                                      C3dPoint** spatialData,
                                      float** propertyData);

  //-----------------------------------------------------
  //...Erases a point (in external units) from the octree
  //... This is done by setting node type to unknown (0)
  //-----------------------------------------------------
  int        EraseTreePoint( C3dPoint* point, int level);

  //-----------------------------------------------------
  //...Erases a point (in tree units) from the octree
  //... This is done by setting node type to unknown (0)
  //-----------------------------------------------------
  int        EraseInternalTreePoint( C3dPoint* point, int level);

  //---------------
  //...Copy a tree
  //---------------
  ts_octree_ptr CopyTree(ts_octree_ptr oldtree);

  //--------------------------
  //...Data conversion methods
  //--------------------------
  C3dPoint*  ConvertExternalToTreeUnits(C3dPoint *pointExternal);
  C3dPoint*  ConvertTreeToExternalUnits(C3dPoint *pointTree,
                                        boolean  adjustPoint);

  //-----------------------------------------------------
  //...Gets the maximum resolution stored in the octree
  //...OR
  //...Get the maximum resolution stored in the octree for a given nodeType
  //-----------------------------------------------------
  int        GetMaximumTreeLevel(ts_octree_ptr tree);
  int        GetMaximumTreeLevel(ts_octree_ptr tree, int nodeType);

  //-----------------------------------------------------------------
  //...Calculates the resolution to be used when storing data in the octree
  //-----------------------------------------------------------------
  int        CalculateTreeLevel(void);

  //---------------------------------
  //...Gets the number of tree nodes
  //---------------------------------
  int        GetSpatialDataSize(int nodeType)  {return GetNodeCount(tree,
                                                                    nodeType);}

  //=============================
```



```
  //...New methods for Phase III
  //=============================

  //-----------------------------------------------------------------
  //...Gets the property value at a given spatial point for a given property
  //-----------------------------------------------------------------
  RWCString GetPropertyValue(C3dPoint* point,RWCString& propertyName);

  //-----------------------------------------------------------------
  //...Deletes the property value at a given spatial point for a given property
  //-----------------------------------------------------------------
  int DeletePropertyValue(C3dPoint* point,RWCString& propertyName);

  //=================================================
  //...Scalar Property Data Methods are defined below
  //=================================================

  //--------------------------
  //...Data conversion methods
  //--------------------------
  long       ConvertPropertyToTree(double valueExternal,
                                   const RWCString &propertyType);
  double     ConvertTreeToProperty(long valueInternal,
                                   const RWCString &propertyType);

  //-----------------------------------------------------------------------
  //...Gets a property's ID number stored in Truesolid as well as its
  //...display property ID. Also gets the property type (ie DENSITY)
  //-----------------------------------------------------------------------
  int        GetPropertyIndex(const RWCString& property_id,
                              int *id,
                              int *displayId,
                              RWCString *name);

  //-----------------------------------------------------------------------
  //...Updates TrueSolid property information to match dataset property
  //...information. (Should be called whenever new properties are added
  //...to the dataset.)
  //-----------------------------------------------------------------------
  int        UpdateProperties(void);

  //-----------------------------------------------
  //...Method to update a given property's range values
  //...with specified value (if necessary)
  //-----------------------------------------------
  int        UpdatePropertyRange(const RWCString& propertyType,
                                 double    propertyValue);

  //-----------------------------------------------------------------
  //...Method to set property range values in the
  //...dataset from the current spatialdata property range values
  //-----------------------------------------------------------------
  int        SetDatasetPropertyRanges(void);

  //-----------------------------------------------------------------
  //...Method to get initial property range
  //...values from the dataset and set them in spatial data
  //-----------------------------------------------------------------
  int        GetDatasetPropertyRanges(void);

  //-----------------------------------------------------------------------
  //...Dumps the contents of a specified property to its display property field
  //-----------------------------------------------------------------------
  int        ConvertPropertyForDisplay(const RWCString& property);

  //-----------------------------
```

```
  //...Get a count of nodes in tree
  //-------------------------------
  int        GetNodeCount(ts_octree_ptr tree,
                          int nodeType);

};

#endif
```

```
//*************************************************************************/
//  File:               statuswindow.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CStatusWindow
//
//  Class Description:  The CStatusWindow class encapsulates all behaviors of
//  the view window's status window in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the status window methods in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CStatusWindow>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined StatusWindow_H_
    #define  StatusWindow_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>              //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>      //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include  <Xm/Xm.h>               //Motif include files

#include <Inventor/sensors/SoNodeSensor.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>

#include <standard.h>             //Our own include files
#include <ictypes.h>
#include <view.h>
#include <vdsdataset.h>
#include <3dpoint.h>
#include <colorlegend.h>
```



```
//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CViewWindow;


//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------

class CStatusWindow

{

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  friend CViewWindow;

  int statusWindowSize;           //size of status window length in pixels
  int flushCacheFlag;

  //...Pointers to needed objects
  CViewWindow       *pViewWindow;         //pointer to our parent

  // Widgets for the status window
  Widget  statusWindowWidget;

  Widget  frame;
  Widget  scrolledWindow;
  Widget  form;
  Widget  labelData;
  Widget  labelDistance;
  Widget  labelProperty;
  Widget  labelAxisX;
  Widget  labelAxisY;
  Widget  labelAxisZ;
  Widget  labelDataset;
  Widget  labelCutplane;
  Widget  labelImageSize;
  Widget  valueData;
  Widget  valueDistance;
  Widget  valueProperty;
  Widget  valueAxisX;
  Widget  valueAxisY;
  Widget  valueAxisZ;
  Widget  valueDataset;
  Widget  valueCutplane;
  Widget  valueImageSize;
  Widget  statusLine;
  Widget  valuePropertyName;
  Widget  valuePropertyMin;
  Widget  valuePropertyMax;
  Widget  insetForm;

  ColorLegend    *colorLegend;

  // Variables for the inset viewer and scene graph
  SoXtExaminerViewer *insetViewer;
  SoSeparator        *insetRoot;
  SoTransform *insetTransform;
```

```
  //...Cached field values
  //-----------------------
  C3dPoint dataPoint;
  double   propertyValue;

  double   axisMinRangeX;
  double   axisMaxRangeX;
  double   axisMinRangeY;
  double   axisMaxRangeY;
  double   axisMinRangeZ;
  double   axisMaxRangeZ;

  boolean   dataset;
  int       cutplanes;

  int       imageSizeX;
  int       imageSizeY;

  RWCString propertyLabel;
  RWCString propertyUnits;
  double    propertyMinValue;
  double    propertyMaxValue;

  RWCString colorbar;

  RWCString currentMessage;


//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:

  CStatusWindow(CViewWindow *parent);
  ~CStatusWindow(void);

  CStatusWindow(const CStatusWindow& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:
  //...Create an empty status window (called from constructor)
  void           CreateStatusWindow(void);

 //-------------------------------------------------
 //...Lowest level of status window update
 //-------------------------------------------------

  //...Update Current DataPoint and property value on status window
  void  UpdateCurrentPoint(void);

  //...Updates the property label and its min, max values on the status window
  void  UpdatePropertyType(void);

  //...Update Current DataPoint on status window
  void  UpdateDataPoint(void);

  //...Update Distance between last 2 clicked Data Points on status window
  void  UpdateDistanceValue(void);

  //...Updates the property label and its min, max values on the status window
  //...Update current property value on status window
  void  UpdatePropertyValue(void);

 //...Update current property label on status window
```



```
  void  UpdatePropertyLabel(void);

  //...Update current property Min value on status window
  void  UpdatePropertyMinValue(void);

  //...Update current property Max value on status window
  void  UpdatePropertyMaxValue(void);

 //...Update Axis Ranges on status window
  void  UpdateAxisRanges(void);

 //...Update dataset change flag widget on status window
  void  UpdateDatasetChanged(void);

  //...Update cutplanes active flag widget on status window
  void  UpdateCutplaneChanged(void);

  //...Update image size on status window
  void  UpdateImageSize(void);

  //  This creates the inset scene graph from a file
  int CreateInsetSceneGraph(void);


//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CStatusWindow& P)
                                                    {P.Print(os); return os;}
    CStatusWindow& operator=(const CStatusWindow& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  void   SetFlushCacheFlag(int val)     {flushCacheFlag = val;};
  int    GetStatusWindowSize(void)      {return statusWindowSize;};
  Widget GetStatusWindowWidget(void)    {return statusWindowWidget;};

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void          Print(ostream& os=cout)  const;       //Print data
  IcerClassType IsA(void)                const        {return _CSTATUSWINDOW;}

 //-------------------------------------
 //...Turn on and off the status window
 //-------------------------------------
 //...Show/Hide the status window
 //...This realizes the window and calls UpdateStatusWindow method
     void  ToggleStatusWindow(int toggleflag);

 //-------------------------------------------------------------------------
 //...Highest level of status window update
 //...This method will always be called each time the status window is shown
 //-------------------------------------------------------------------------
 //...Update anything that has changed on the status window
 //...This calls each of the individual update methods below.
     void  UpdateStatusWindow(void);

 //...Update colorbar widget on status window
  void  UpdateColorBar(void);

 //...Update Status line on status window
```

```
  void  UpdateStatusLine(RWCString statusMsg);

 //...Get Status line on status window
  RWCString GetStatusLine(void)    {return currentMessage;}
};

#endif
```

```
//*****************************************************************************/
//  File:               tsview.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CTruesolidView
//
//  Class Description:  CTruesolidView encapsulates all behaviors of Truesolid
//                      rendering.  This class handles the rendering of the
//                      octree while the CSpatialData class handles the access
//                      of the octree.
//
//                      The Truesolid image generated in this class is written
//                      to the GL hardware buffers (color and depth)
//
//  Inheritance:        CTruesolidView
//
//  Required Class Libraries:
//                      Rogue Wave Math.h++ v6.1
//                      Rogue Wave Tools.h++
//                      TrueSolid v2.3c
//
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/25/95    L. Cowper       Created
//
//*****************************************************************************/
#if !defined CTruesolidView_H_
    #define  CTruesolidView_H_

//===========================================================================
// REQUIRED INCLUDE FILES
//===========================================================================
#include <stddef.h>                     //Standard C include files
#include <iostream.h>                   //C++ input/output streams
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include <Ts/tview.h>                 //Truesolid include files
#include <Ts/grf.h>
#include <Ts/Csg_memprim.h>
#include <Ts/threshold.h>
#include <Ts/shader.h>

//#include <X11/Xlib.h>               //Standard C include files
//#include <X11/Xutil.h>
//#include <X11/Intrinsic.h>

#include <GL/gl.h>
#include <GL/glu.h>
//#include <GL/glx.h>                   //Open GL Include files

#include <standard.h>               //Our own include files
```



```
#include <ictypes.h>
#include <3dconvert.h>
#include <vdsdataset.h>
#include <color.h>


//===========================================================================
// OTHER DECLARATIONS
//===========================================================================
class CRenderModel;


//---------------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------------
class CTruesolidView
{

//===========================================================================
// DECLARATIONS (enums, defines, etc)
//===========================================================================
public:

  enum RenderType {
    RENDER_STATE_NOOP    = 1,  // do nothing
    RENDER_STATE_DISPLAY = 2,  // write buffers only
    RENDER_STATE_COMPUTE = 4   // compute new color buffer and zbuffer
  };

//===========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//===========================================================================
private:

  //  Pointers to needed objects
  CRenderModel*      pRenderModel;      //pointer to parent
  CVDSDataset*       pDataset;
  CView*             pView;

  C3dConvert unitConvert;


  //  Truesolid Variables needed for dealing with the TrueSolid library
  ts_octree_ptr      tree ;             //octree pointer
  ts_memprim_info    meminfo ;          //memory primitive structure
  ts_Csg_ptr         csg ;              //CSG pointer
  ts_tview_ptr       tview ;            //tree view pointer
  int                den_num ;          //density number
  ts_grf_varptr      grfstuff ;         //graphics structure
  ts_grf_info        grf_info ;         //contains color and z buffers
  ts_Csg_varptr      csgstuff ;         //CSG structure
  ts_Xform           display_xform ;    //display transform
  ts_thresholdptr    thresh ;           //threshhold pointer
  ts_Real            near, far ;        // depth buffer near and far

  // temp pointers
  ts_IF        *tsDepthBuffer;
  ts_grf_ulong *tsColorBuffer;

  // buffers to be passed to openGL
  GLuint   *glDepthBuffer;
  GLushort *glColorBuffer;

  // Cache for view parameters. These will be updated whenever
  // Update() is called.  This call indicates that  one or more of the
  // variables has changed.  This is usually called by the View and/or
```

```
  // ViewWindow.
  int       renderAreaSizeX;
  int       renderAreaSizeY;
  float     antiAliasing;
  RWCString thresholdFile;
  unsigned  applyCutplanes;
  unsigned  highResolutionDisplayMode;
  RWCString propertyType;
  CColor    backgroundColor;

  // viewpoint cache, these come from the view's camera but they
  // are maintained as view parameters.
  C3dPoint vpPosition;
  double   vpOrientation[9];
  double   vpHeight;
  double   vpWidth;

  // The flushCacheFlag invalidates all entries in the cache which will
  // cause all values to be updated regardless of their current state.
  int flushCacheFlag;

  // rendering state bits
  unsigned int renderStateFlag;

  // Truesolid shader structure
  ts_shader_ptr      shader;
  ts_shader_info_ptr shader_data;



//=============================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=============================================================================
public:

  CTruesolidView(CRenderModel *renderModel,
                 CVDSDataset *dataset,
                 CView* view);

  ~CTruesolidView(void);

  CTruesolidView(const CTruesolidView& other);        //Copy constructor

//=============================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=============================================================================
private:
  CTruesolidView(void);

  void TsCopyright(void) const {if (ts_copyright);}

  boolean IsViewChanged(void) const;

  int ReadThresholdFile(RWCString &filename);
  int ResizeImage(void);
  void SetViewpoint(void);
  int ComputeImage(void);
  void DisplayImage(void);
  int ColorPlot(void);
  void CreateOctreeCsg(void);

  //  Cutplane methods in Truesolid
  //  Not needed for Phase III - Release A
  void CreateCutCsg(void);
  ts_Csg_ptr BuildSimpleCutPlane(int cutplaneNo);
  ts_Csg_ptr BuildSliceCutPlane(int cutplaneNo);
```



```
  ts_Csg_ptr BuildCornerCutPlane(int cutplaneNo);
  void GetCutPlaneTransformation(int number, ts_Xform* T);

//=============================================================================
// OPERATORS
//=============================================================================
public:
  friend ostream& operator<<(ostream& os, const CTruesolidView& P){
    P.Print(os);
    return os;
  }

  // the old copy constructor
  CTruesolidView& operator=(const CTruesolidView& other);

//=============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=============================================================================
public:
  CRenderModel* GetRenderModelPointer(void);
  void forceCache(void) {flushCacheFlag=1;}

//=============================================================================
// OTHER METHODS
//=============================================================================
public:

  void Print(ostream& os=cout)  const;
  IcerClassType IsA(void) const {return _CTRUESOLIDVIEW;}

  //  Initialize Truesolid view
  int Initialize(void);

  // Update() is called to indicate one or more of the variables
  // has changed.  This is usually called by the View and/or ViewWindow.
  int Update(void);

  //  Render an image
  int Render(void);


  //-------------------------------------------------------------------------
  int SaveScene(const RWCString &filename);
  int RestoreScene(const RWCString &filename);
  //
  //  Save the scene to the specified file.  The file saved is _____ type.
  //-------------------------------------------------------------------------


  //  Truesolid/Data mouse pick conversion
  //  returns world point (x,y,z) which is projected onto screen coordinates
  int GetWorldPointCoordinates(double xScreen, double yScreen,
                               double *x, double *y, double *z);

  //  Utility method to print GL diagnostics
  void QueryGL(void);

  //  Method to set the view transform
  //  Updates the view transform given a camera position in dataset units
  void SetViewpoint(C3dPoint position, double *orientation,
                    double width, double height);

  //  Create a threshold file from an RGB file
  int CreateThreshold(int nColors,
                      const RWCString& rgb,
                      const RWCString& threshold);
```

```
  //  Genereate image files (Postscript and DIB format)
  int WritePostscript(const RWCString& outputFile);
  int WriteDIB(const RWCString& outputFile);
};

#endif
```

```
//*********************************************************************/
//  File:               userdisplay.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:  The functions in this file implement the class
//                     CUserDisplay.
//
//  Class Description:   CUserDisplay is provided as a means to easily get
//                       access to the VDS user display data.
//
//
//  Inheritance:   CUserDisplay
//
//  Required Class Libraries: Roque Wave
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name        Remarks
//  ---------   --------    --------    -------
//  103:00:00   05/31/95    L. Cowper   Created
//
//*********************************************************************/
#if !defined CUserDisplay_H_
   #define  CUserDisplay_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                   //C++ input/output streams (ALWAYS)
#include "standard.h"                   //MTI Standard Definitions  (ALWAYS)
#include "ictypes.h"                    //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>
#include <rw/rstream.h>

#include  <X11/Intrinsic.h>

//=====================================================================
// OTHER DECLARATIONS
//=====================================================================


//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CUserDisplay
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:
```



```
  static Widget         topLevelWidget;         //the top level widget
  static Display*       display;                //the X display
  static Window         window;                 //the X window

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
    CUserDisplay(void);                         //Default constructor
    ~CUserDisplay(void);                        //Destructor


//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CUserDisplay& P)
                                                {P.Print(os); return os;}

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:
//-----------------------------------------------
// The Gets and Sets -- Used To Obtain Access To Data
//-----------------------------------------------
  static Widget     GetTopLevelWidget(void) {return topLevelWidget;}
  static void       SetTopLevelWidget(Widget widget) {topLevelWidget=widget;}

  static Display*   GetDisplay(void) {return display;}
  static Window     GetWindow(void) {return window;}


//=====================================================================
// OTHER METHODS
//=====================================================================
public:
    void          Print(ostream& os=cout)  const;    //Print object data
    IcerClassType IsA(void)                const    {return _CUSERDISPLAY;}

    void          Initialize(void);          //Initialize data members
};

#endif
```

```
//*********************************************************************/
//  File:               vdsdataset.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CVDSDataset
//
//  Class Description:  CVDSDataset encapsulates the behavior of a dataset,
//                      and is of primary importance in that most of ICERVS
//                      functionality is related to datasets.
//
//                      A dataset is the collection of data (volumetric,
//                      scalar property, non-scalar property, geometric)
//                      that characterizes a particular site at a discrete
//                      time or state. A dataset is stored on disk as a set
//                      of disk files.  When a dataset is first created,
//                      the disk files are read and stored in memory.
//                      It is represented in memory as a TrueSolid octree,
//                      a set of indexes to non-scalar property files, a
//                      reference to the geometric object node in the
//                      Inventor Tree (Open Inventor), a reference to the
//                      indicator object node in the Inventor Tree, and some
//                      memory resident dataset parameters.
//
//  Inheritance: <RWCollectable>  <CNamedItem>  <CVDSDataset>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name          Remarks
//  ---------   --------    --------      -------
//  102:00:00   03/20/94    D. Smith      Created
//  103:00:00   04/26/95    L. Cowper     Modified for Phase III
//
//*********************************************************************/
#if !defined CVDSDataset_H_
    #define  CVDSDataset_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                   //C++ input/output streams )
#include <standard.h>                   //MTI Standard Definitions
#include <ictypes.h>                    //MTI symbols for types

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>                         //RWCString
#include <rw/rstream.h>

#include <cnamed.h>                             //CNamedItem
#include <cstrlist.h>                           //CSortedStringList

#include <propertyset.h>                        //CPropertySet

#include <sitedefinition.h>                     //CSiteDefinition
#include <datasetviewer.h>                      //CDatasetViewer
```



```
#include <3dpoint.h>                    //C3dPoint
#include <3dpointset.h>                 //C3dPointSet
#include <3dobject.h>                   //C3dObject
#include <3dobjectset.h>                //C3dObjectSet
#include <3dcomposite.h>                //C3dComposite
#include <spatialdata.h>                //CSpatialData


// OTHER DECLARATIONS
//=====================================================================
class CDataPoint;
class CNonscalarData;
class CNonScalar;
class CDatasetViewer;
class CPropertySet;
class CSiteDefinition;
class CDatasetManager;
class COrderedStringList;
class CVDSDatasetSet;

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CVDSDataset  :public CNamedItem
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:
  RWCString             groupName;              //group name
  CSiteDefinition       *pSiteDefinition;       //the site defs for this dataset
  int                   useCount;               //use count for this dataset

      //-----------------------------------------
      //Adjusted site parameters for cubical space
      //-----------------------------------------
  double tankSizeXmin;      //adjusted minimum xaxis tank size
  double tankSizeXmax;      //adjusted maximum xaxis tank size
  double tankSizeYmin;      //adjusted minimum yaxis tank size
  double tankSizeYmax;      //adjusted maximum yaxis tank size
  double tankSizeZmin;      //adjusted minimum zaxis tank size
  double tankSizeZmax;      //adjusted maximum zaxis tank size
  double tankSizeExtent;    //extent of largest axis

      //--------------------
      //  Dataset parameters
      //--------------------
  RWCString             operatorName;       //Operator who created dataset
  RWCString             createDateTime;     //Date/time of creation
  RWCString             modifyDateTime;     //Date/time of last modify
  RWCString             octreeFilename;     //name of octree file
  int                   lastUsedNonscalarIndex;  //last used nonscalar index

  boolean           containsNonscalarData; //Dataset contains nonscalar data
  boolean           changeFlag;            //Dataset changed since last save
  boolean           noSensorDataFlag;      //Dataset is new (no sensor data)
  boolean           readFlag;              //Dataset has been read from disk
  boolean           combinedFlag;          //Combined dataset
  boolean           dataLoadingFlag;       //Sensor Data is being loaded
```

```
  COrderedStringList    combinedIdList;     //Ids of combined datasets
  COrderedStringList    combinedNameList;   //Names of combined datasets

  CPropertySet          *pProperties;       //Definitions of material
                                            //   properties in this dataset
  CDatasetViewer        *pDatasetViewer;    //The Dataset viewer object
  C3dObjectSet          *pObjectSet;        //All my geometric objects
  C3dObjectSet          *pIndicatorSet;     //All my data indicator objects
  CSpatialData          *pTree;             //Pointer to the spatial data
  CNonscalarData        *pNonscalar;        //Pointer to non-scalar properties
  CDatasetManager       *pDatasetManager;   //parent


//-------------------------------------------------------------------------
// Data Inherited From CNamedItem  (Access by method only)
//-------------------------------------------------------------------------
//    theName       theDescription      idNumber  idString  myOwner


//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    CVDSDataset(void);
    CVDSDataset(CDatasetManager  *pDatasetManager,   //Constructor
                const RWCString& groupName,
        const RWCString& datasetName);

    ~CVDSDataset(void);                             //Destructor

    CVDSDataset(const CVDSDataset& other);          //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

  //-----------------------------------------------
  //...Get the filename of the dataset parameter file
  //-----------------------------------------------
  RWCString GetDatasetParametersFilename(void)      const;

  //-----------------------------------------------
  //...Initialize a new dataset
  //
  // -Copy Default Site Parameters File from group directory
  // -Copy Default View Parameters File from /etc directory
  // -Create an empty octree and write to disk
  // -Create empty geometric object file
  // -Create a dataset parameter file
  // -Create empty non-scalar index files
  //
  // (See Phase II code: CVdsMainWindow::DatasetNew)
  //
  // return -1 if error
  // return NULL if dataset was not added due to error
  //-----------------------------------------------
  int  InitializeDataset(void);

  //...After our site definitions are established, we must adjust our
  //...tank space to be cubical (done in our constructor)
  void AdjustTankSpace(void);

  //...Get a new datasetId
  RWCString  GetNewDatasetId(void);
```



```
  //-------------------------
  //..Data Indicator methods
  //-------------------------
  //...AddIndicator will add Indicator to Render Model
  ////int  AddIndicator(C3dObject* indicator);

  //...RemoveIndicator will remove Indicator from Render Model
  ////int  RemoveIndicator(C3dObject* indicator);

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CVDSDataset& P)
                                                {P.Print(os); return os;}
    CVDSDataset& operator=(const CVDSDataset& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

  //-------------------------------------------------------------
  //...These dataset parameters all need to be recalculated whenever the
  //...dataset parameters are edited (ie site height, site offset, etc.)
  //
  //...SetTankSizeXmin is the method who will also update the status window
  //...when it is called
  //-------------------------------------------------------------
  void  SetTankSizeXmin(const double val);
  void  SetTankSizeXmax(const double val)           {tankSizeXmax = val;}
  void  SetTankSizeYmin(const double val)           {tankSizeYmin = val;}
  void  SetTankSizeYmax(const double val)           {tankSizeYmax = val;}
  void  SetTankSizeZmin(const double val)           {tankSizeZmin = val;}
  void  SetTankSizeZmax(const double val)           {tankSizeZmax = val;}
  void  SetTankSizeExtent(const double val)         {tankSizeExtent = val;}

  //-------------------------------------------------------------
  //...These dataset parameters are set by the instatiating class
  //-------------------------------------------------------------
  void  SetOperatorName(const RWCString& name)      {operatorName = name;}
  void  SetCreateDateTime(const RWCString& when)    {createDateTime = when;}
  void  SetModifyDateTime(const RWCString& when)    {modifyDateTime = when;}
  void  SetOctreeFilename(const RWCString& name)    {octreeFilename = name;}
  void  SetLastUsedNonscalarIndex(const int val)  {lastUsedNonscalarIndex=val;}

  //-------------------------------------------------------------
  //...Make sure these dataset parameters are updated at the appropriate times
  //-------------------------------------------------------------
  void  SetContainsNonscalarData(boolean state) {containsNonscalarData=state;}
  void  SetChangeFlag(boolean state);
  void  SetNoSensorDataFlag(boolean state)          {noSensorDataFlag = state;}
  void  SetReadFlag(boolean state)                  {readFlag = state;}
  void  SetCombinedFlag(boolean state)              {combinedFlag = state;}
  void  SetDataLoadingFlag(boolean state)           {dataLoadingFlag = state;}

  void  SetSpatialData(CSpatialData* val)           {pTree = val;}
  void  SetNonscalarData(CNonscalarData* val)       {pNonscalar = val;}


  RWCString        GetGroupName(void)         const {return groupName;}
  CSiteDefinition* GetSiteDefinition(void)    const {return pSiteDefinition;}
  int              GetUseCount(void)          const {return useCount;}

  const double     GetTankSizeXmin(void)      const  {return tankSizeXmin;}
  const double     GetTankSizeXmax(void)      const  {return tankSizeXmax;}
```

```
  const double      GetTankSizeYmin(void)      const  {return tankSizeYmin;}
  const double      GetTankSizeYmax(void)      const  {return tankSizeYmax;}
  const double      GetTankSizeZmin(void)      const  {return tankSizeZmin;}
  const double      GetTankSizeZmax(void)      const  {return tankSizeZmax;}
  const double      GetTankSizeExtent(void)    const  {return tankSizeExtent;}

  const RWCString&  GetOperatorName(void)      const  {return operatorName;}
  const RWCString&  GetCreateDateTime(void)    const  {return createDateTime;}
  const RWCString&  GetModifyDateTime(void)    const  {return modifyDateTime;}
  const RWCString&  GetOctreeFilename(void)    const  {return octreeFilename;}
  int               GetLastUsedNonscalarIndex(void) const
                                               {return lastUsedNonscalarIndex;}

  boolean           GetContainsNonscalarData(void) const
                                               {return containsNonscalarData;}
  boolean           GetChangeFlag(void)        const  {return changeFlag;}
  boolean           GetNoSensorDataFlag(void)  const  {return noSensorDataFlag;}
  boolean           GetReadFlag(void)          const  {return readFlag;}
  boolean           GetCombinedFlag(void)      const  {return combinedFlag;}
  boolean           GetDataLoadingFlag(void)   const  {return dataLoadingFlag;}

  COrderedStringList* GetCombinedIdList(void)      {return &combinedIdList;}
  COrderedStringList* GetCombinedNameList(void)    {return &combinedNameList;}

  CPropertySet*       GetProperties(void)       const {return pProperties;}
  CDatasetViewer*     GetDatasetViewer(void)    const {return pDatasetViewer;}
  C3dObjectSet*       GetObjectSet(void)        const {return pObjectSet;}
  C3dObjectSet*       GetIndicatorSet(void)     const {return pIndicatorSet;}
  CSpatialData*       GetSpatialData(void)      const {return pTree;}
  CNonscalarData*     GetNonscalarData(void)    const {return pNonscalar;}
  CDatasetManager*    GetDatasetManager(void)   const {return pDatasetManager;}

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void         Print(ostream& os=cout)  const;   //Print object data
  IcerClassType IsA(void)               const    {return _CVDSDATASET;}

  //-------------------------------------------------------------------------
  //...useCount is incremented when "users" attach.
  //    and decremented on detach.  Intended to be monitored for zero
  //    useCount indicating that this dataset object may be destroyed
  //-------------------------------------------------------------------------
  int          Attach(void)        {return ++useCount;}
  int          Detach(void)        {return --useCount;}

  //-----------------------------------------------------------------
  //...We  need these methods to handle the logic for opening and closing
  //...datasets based on the number of views open on a dataset and
  //...who opened them.
  //-----------------------------------------------------------------
  int DatasetOpen(void);

  //...Make sure we ask operator if dataset should be saved
  //...if it has changed since last save.
  int DatasetClose(void);

  //-----------------------------------------------------------------
  //...Save a dataset (Note - no view parameters are saved here)
  //
  // -Save dataset parameters
  // -Save site parameters
  // -Save objects
  // -Save volumetric data and scalar properties
  //
```



```
  // return -1 if error saving dataset
  //-----------------------------------------------------------------
  int       Save(void);

  //-----------------------------------------------------------------
  //...Load a dataset (Note - no view parameters are loaded here)
  //
  // -Load dataset parameters
  // -Load site parameters
  // -Load objects
  // -Load volumetric data and scalar properties
  //
  // return -1 if error loading dataset
  //-----------------------------------------------------------------
  int       Load(int newDatasetFlag);

  //-------------------------------------------------
  //...Add an ICERVS formatted file to the dataset
  //-------------------------------------------------
  int AddICERVSFile( const RWCString& filename, int* num, int* reject);

  //==================================
  //...General dataset methods
  //==================================
  int       ReadDatasetParameters(void);
  int       ReadDatasetParameters(const RWCString& prmFile);

  int       WriteDatasetParameters(void);
  int       WriteDatasetParameters(const RWCString& prmFile);

  void      GetWorldLimits(double* xmin, double* xmax,
                           double* ymin, double* ymax,
                           double* zmin, double* zmax) const;

  void      GetWorldLimits(double* limits) const {
    GetWorldLimits(&limits[0], &limits[1],
                   &limits[2], &limits[3],
                   &limits[4], &limits[5]);
  }

  //==================================
  //...Spatial data interface methods
  //==================================

  //-----------------------------------------------------------------
  //...Erase a region of spatial data points from the spatial data
  //...This region is defined as a single geometric object
  //-----------------------------------------------------------------
  int       EraseRegion(const COrderedStringList& region);

  //-----------------------------------------------------------------
  //...Erase a region of spatial data points from the spatial data
  //   This region is defined as:
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be included in being erase
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be excluded from being erased
  //-----------------------------------------------------------------
   int       EraseRegion(C3dObject* object);

  //-----------------------------------------------------------------
  //...Change a region of spatial data points to a new node type
  //...This region is defined as a single geometric object
  //-----------------------------------------------------------------
  int       ChangeRegion(C3dObject* object, int newNodeType);
```

```
  //-------------------------------------------------------------
  //...Change a region of spatial data points to a new node type
  //   This region is defined as:
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be included in being changed
  //        to a new node type
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be excluded from being changed
  //        to a new node type
  //-------------------------------------------------------------
  int       ChangeRegion(const COrderedStringList& includedRegion,
             const COrderedStringList& excludedRegion,
             int newNodeType);

  //-------------------------------------------------------------
  //...Highlight (change the color of) a region of spatial data points
  //   This region is defined as:
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be included in being highlighted
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be excluded from being highlighted
  //-------------------------------------------------------------
  int       HighLightRegion(const COrderedStringList& includedRegion,
              const COrderedStringList& excludedRegion);

  //-------------------------------------------------------------
  //...Retrieve a region of points from the spatial data
  //   This region is defined as:
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be included in being retrieved
  //       -a list of pointers to geometric objects which define an
  //        area of spatial data to be excluded from being retrieved
  //-------------------------------------------------------------
  C3dPointSet RetrieveRegionPoints(const COrderedStringList& includedRegion,
                      const COrderedStringList& excludedRegion);

  //-------------------------------------------------------------
  //...Add a single ICERVS sensor data point to the spatial data
  //   This also includes all associated scalar property data
  //-------------------------------------------------------------
  int       AddPoint(const CDataPoint& point);

  //-------------------------------------------------------------
  //...Add an ICERVS sensor data file to the spatial data
  //   This also includes all associated scalar property data
  //-------------------------------------------------------------
  int       AddFile(const RWCString& fname);

  //-------------------------------------------------------------
  //...Erase a single ICERVS sensor data point from the spatial data
  //   This also includes all associated scalar property data
  //-------------------------------------------------------------
  int       ErasePoint(const CDataPoint& point);

  //--------------------------------
  //...Spatial data I/O interface methods
  //--------------------------------
  int       GetSpatialDataSize(int nodeType)
                        { return(pTree->GetSpatialDataSize(nodeType)); }

  C3dPoint* GetSpatialDataArray(unsigned int nodeTypeMask)
                        {  return(pTree->TreeToMemory(nodeTypeMask)); }

  float*    GetSpatialDataPropertyArray(const RWCString& propertyname,
                                        unsigned int nodeTypeMask)
            {return(pTree->TreeToPropertyMemory(propertyname,
```



```
                                        nodeTypeMask));}

  void      GetSpatialDataAndPropertyArrays(const RWCString& propertyname,
                                        unsigned int nodeTypeMask,
                                        C3dPoint** spatialData,
                                        float** propertyData);


  int   DumpDatasetASCII(const RWCString& filename)
                                        {return(pTree->TreeToASCII(filename));}
  int   DebugDumpASCII(const RWCString& filename);

  int   DumpDatasetStatistics(const RWCString& filename)
                                        {return(pTree->GetTreeInfo(filename));}

  int   CreatePostcript (const RWCString& filename);
  int   CreateDIBFile (const RWCString& filename);
  int   DumpScreen(void);

  //-----------------------------------------
  //...Spatial data analysis interface methods
  //-----------------------------------------
  int   ObjectVolumetricDifference(C3dObject* object ,
                                        const RWCString& filename);

  int   ObjectConsistencyCheck(const RWCString& filename);

  int   TreeInterpolate( int numRegions,
                         int fill,
                         C3dObject* object[],
                         double spacingHor[],
                         double spacingVert[],
                         double radiusMin[],
                         double radiusMax[],
                         int surfaceType[]);

   int  TreeInterpolate ( const C3dObjectSet& region,
                          double spacingHor[],
                          double spacingVert[],
                          double radiusMin[],
                          double radiusMax[],
                          int surfaceType[]);

  int GetDefaultGridSpacingValues( double *xSpacing,
                                   double *ySpacing,
                                   double *zSpacing);

  int CombinedDataset(CVDSDatasetSet *datasets);

  int FindConnectivity(const C3dObjectSet& region, int cellSize,
                       C3dPoint seedpoint);

  int TracePipe(C3dObject* cylindricalObject, double offset);

  int GoodnessOfFit(const C3dObjectSet& region, C3dObject* cylindricalObject,
        double* meanDistance, double* sigma);

  int FitBestCylinder(const C3dObjectSet& region, boolean fixedDiameter,
                      double* diameter, double* rotationX, double* rotationY,
                      double* translationX, double* translationY,
                      int* iterations, double* meanDistanceArray,
                      double* sigmaArray, C3dObject* object);

  int FitSurfaceMesh(const C3dObjectSet& region, C3dObject* object);

  int DerivedProperty( int numProps,
```

```
                        const RWCString& propertyA,
                        const RWCString& propertyB,
                        const RWCString& propertyC,
                        double minA, double minB,
                        double maxA, double maxB,
                        double scaleA, double scaleB);

    //--------------------------
    //...Geometric Object methods
    //--------------------------
    //...Add object to set of objects and to Render Model, highlights the object
    int  AddObject(const C3dObject& object);

    //...Remove object from set of objects and from Render Model
    //...Also remove the object from the selected object list in the
    //...DatasetViewer
    int  RemoveObject(C3dObject* object);

    //---------------------------------
    //..Non-Scalar Property methods TBD
    //---------------------------------
    //...Add Non-scalar properties to dataset for a given spatial point
    int  AddNonScalarProperties(C3dPoint point,
                                CNonScalar* pNonscalarProperties,
                                const RWCString& nonScalarFilePath,
                                int sensorInputFlag);

    //...Remove Non-scalar properties from dataset for a given spatial point
    int  RemoveNonScalarProperties(C3dPoint point,
                                   CNonScalar* pNonscalarProperties);

    //...Stores a nonscalar property file to the
    //...ICERVS nonscalar property directory.
    //...If the file already exists in the target directory,
    //...it will be renamed, stored,  and the new name will be returned.
    //...If the file doesn't already exists in the target directory,
    //...it will be stored, and the same name will be returned.
    //...If the file could not be stored, a null filename will be returned.
    RWCString StoreNonScalarFile(const RWCString& fileName,
                                 int sensorInputFlag);

    //...Removes a nonscalar property file from the
    //...ICERVS nonscalar property directory.
    int RemoveNonScalarFile(const RWCString& fileName);

    //...Strips off a directory path from a filename string
    RWCString StripDirectoryPath(const RWCString& fileName);

    //...Breaks a filename into its directory path, name, and extension
    int BreakApartFilename(const RWCString& completeFileName,
                                        RWCString& filePath,
                                        RWCString& fileName,
                                        RWCString& fileExtension);

    //-----------------------------------------
    // These functions are defined for convenience
    //-----------------------------------------
    void  SetDatasetId(const RWCString& id)           {SetTheIdString(id);}
    void  SetDatasetGroup(const RWCString& group)     {groupName = group;}
    void  SetDatasetName(const RWCString& name)       {SetTheName(name);}
    void  SetDatasetDescription(const RWCString& desc) {SetTheDescription(desc);}


    const RWCString& GetDatasetId(void)          const {return GetTheIdString();}
    const RWCString& GetDatasetName(void)        const  {return GetTheName();}
    const RWCString& GetDatasetDescription(void) const
```



```
                                        {return GetTheDescription();}

        RWBoolean isEqual(const RWCollectable *a) const;
    //-------------------------------------------------------------------
    // Methods Inherited From CNamedItem
    //-------------------------------------------------------------------
    // SetTheName          GetTheName              SetOwner
    // SetTheDecsription   GetTheDescription       GetOwner
    // SetIdNumber         GetIdNumber             GetOwnerIdString
    // SetTheIdString      GetTheIdString

    };

    #endif
```

```
//*********************************************************************
//  File:               vdserver.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CVDSServer and class VdsCommand
//
//  Class Description:  CVDSServer encapsulates the server for
//                      VDS.  Commands are received from multiple clients.
//                      They are interpreted, an appropriate "doit" method
//                      is called on the CVDSUser(corresponding to a client)
//                      object. An appropriate reply is sent back to the client
//                      by the CVDSUser object.
//
//                      VdsCommand encapsulates the VDS server command
//                      interpretation.
//
//  Inheritance: CSockets CServer CVDSServer
//               CServerCommand CVdsCommand
//
//  Required Class Libraries: Rogue Wave Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date       Name           Remarks
//  ---------   --------   --------       -------
//  103:00:00   5/11/95    J. Hutchison   Rewritten for Phase III
//                                        by cloning server from FSS\
//
//*********************************************************************
#if !defined CVDSServer_H_
    #define  CVDSServer_H_

#include <rw/compiler.h>                    //RW compiler ident
#include <rw/cstring.h>                     //RWCString
#include <rw/rstream.h>

#include <ictypes.h>                        //ICER common defs
#include <cserver.h>                        //CServer class
#include <vdserror.h>                       //VDS Error Codes
#include <vuserset.h>                       //CSetOfVDSUsers
#include <vduser.h>

class VdsCommand;
class CVolumetricDataClient;
class CDatasetManager;
//=====================================================================
// CLASS DEFINITION
//=====================================================================
class CVDSServer     :public CServer
{
//---------------------------------------------------------------------
// ----- D A T A   M E M B E R S -----
//---------------------------------------------------------------------
private:

int             maxUsers;
```



```
int             maxCommands;
int             numberOfCommands;
VdsCommand*     commandTable;

CSetOfVDSUsers  myUsers;



CDatasetManager*  datasetManager;
//---------------------------------------------------------------------
// Data inherited from CServer
//---------------------------------------------------------------------
//   debugFlags
//   serviceName
```

```
//-------------------------------------------------------------------
// ----- M E T H O D S -----
//-------------------------------------------------------------------

public:
CVDSServer(CDatasetManager* dm);
                        // No copy constructor -- only one VDS server
                        // using this one's communications channels
~CVDSServer(void);

boolean       IsDefined(void)           const   {return TRUE;}
IcerClassType IsA(void)                 const   {return _CVDSSERVER;}
void          Print(ostream& os=cout)   const;

friend ostream& operator<<(ostream& os, const CVDSServer& server)
                                  {server.Print(os); return(os);}
void SetMaxUsers(int number)                      {maxUsers = number;      }

CSetOfVDSUsers* GetSetOfUsers(void)          {return &myUsers;        }
CDatasetManager* GetDatasetManager(void)     {return datasetManager;  }

int  StartVDSServer(void);

int  ProcessConnectionRequest(int clientNumber);
int  ProcessDisconnectionRequest(int clientNumber);
int  ProcessCommand(int clientNumber, CServerCommand* cmd);

int  ExecuteServerCommand(CServerCommand* cmd);
int CreateStandaloneUser(const RWCString& path, const RWCString& display);

//-------------------------------------------------------------------
// Methods inherited from CServer
//-------------------------------------------------------------------
//   SetServiceName         GetServiceName      InitializeServer
//   SetShutdownFlag        GetShutdownFlag     ShutdownServer
//   SetCmdMsgPrintFlag     GetCmdMsgPrintFlag  StartServerCommandInputLoop
//                                              CheckForCommandInputAndExecute
//                                              ReceiveCommand
//                                              SendCommandReply
//                                              ProcessMessage
//                                              ProcessCommand
```



```
//===================================================================
// VDS SERVER COMMANDS
//===================================================================
private:

//-----------------------------------------------------------
// SERVER GENERAL Command Methods called by ProcessCommand and
//   ExecuteServerCommand
//-----------------------------------------------------------
int VDS_DebugPrintObject(CServerCommand* cmd);
                        //-----------------------------------------
            // Prints either a server or user object
                        //-----------------------------------------
int VDS_AddFile(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and passes on arguments to the
            // CVDSUser method to add the contents of a
            // standard ICERVS file to a specified VDS dataset
                        //-----------------------------------------------
int VDS_GetListOfGroups(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and passes on arguments to the
            // CVDSUser method to get a list of all groups
                        //-----------------------------------------------
int VDS_GetListOfDatasets(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and passes on arguments to the
            // CVDSUser method to get a list of all datasets
                        //-----------------------------------------------
int VDS_GetListOfOpenDatasets(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and passes on arguments to the
            // CVDSUser method to get a list of all open datasets
                        // in a group specified within the command
                        //-----------------------------------------------
int VDS_GetListOfAllOpenDatasets(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and passes on arguments to the
            // CVDSUser method to get a list of all open datasets
                        // in all groups
                        //-----------------------------------------------
int VDS_SetOperatorName(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes and passes on arguments to the
            // CVDSUser method to set the operator name
                        //-----------------------------------------------
int VDS_Start_GUI(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes and passes on arguments to the
            // CVDSUser method to  start VDS main window
                        // (Dataset Manager Window)
                        //-----------------------------------------------

int VDS_Terminate(CServerCommand* cmd);
                        //-----------------------------------------------
            // Decomposes command and sends all CVDSUsers a
            // shutdown  request.  Then the server
            // sends the dataset manager a shutdown request.
            // The server then replies with a shutdown
            // complete message, sets the server's shutdown flag
                        // and returns to the main program which then
                        // terminates the VDS process.
                        //-----------------------------------------------
```

```
};
```



```
//=============================================================================
// CLASS DEFINITION  -- VdsCommand
//=============================================================================

typedef int (CVDSServer::*VdsCmdFunction)(CServerCommand* cmd);

class VdsCommand        :public CServerCommand
{
//-----------------------------------------------------------------------------
// ----- D A T A   M E M B E R S -----
//-----------------------------------------------------------------------------
private:
  VdsCmdFunction  cmdMethod;                    //Command execution method

//-----------------------------------------------------------------------------
// ----- M E T H O D S -----
//-----------------------------------------------------------------------------
public:
  VdsCommand(void) :CServerCommand()
                                  {cmdMethod=0;}

  VdsCommand(const VdsCommand& other) :CServerCommand(other)
                                  {cmdMethod = (VdsCmdFunction)other.cmdMethod;}

  VdsCommand& operator=(const VdsCommand& other)
                                  {CServerCommand::operator=(other);
                                   cmdMethod = (VdsCmdFunction)other.cmdMethod;
                                   return(*this); }

  ~VdsCommand(void)               {}

   VdsCmdFunction GetCmdFunction(void)
                                  {return cmdMethod;}

   void Define(const RWCString& name, int nargs, VdsCmdFunction func)
                        {CServerCommand::Define(name,nargs); cmdMethod=func;}

};

#endif
```

```
//**************************************************************************
//  File:               vduser.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CVDSUser
//
//
//  Class Description:  CVDSUser encapsulates the user class
//                      which contains methods to support CVDSServer
//                      commands initiated by external clients (at present,
//                      only APPL).  Therefore all user methods actually
//                      support requests originating from APPL.
//                      This user class must be instantiated by the
//                      VDS server for each external client wishing to make
//                      requests of the server.
//                      The server decomposes commands from the external client
//                      and passes appropriate arguments to methods of CVDSUser.
//                      The methods perform the necessary functions and send
//                      appropriate replies back to the requesting external
//                      client.
//
//
//
//  Inheritance:   CVDSUser
//
//  Required Class Libraries: Rogue Wave Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/11/95    J. Hutchison    Cloned from FSS User
//
//**************************************************************************
#if !defined CVDSUser_H_
    #define  CVDSUser_H_

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/cstring.h>                         //RWCString
#include <rw/rstream.h>

#include <ictypes.h>                            //ICER common defs
#include <cnamed.h>                             //CNamedItem
#include <cstrlist.h>

class CVDSDataset;
class CVDSServer;
class UserCommand;
//==========================================================================
// CLASS DEFINITION
//==========================================================================
class CVDSUser       :public CNamedItem
{
//--------------------------------------------------------------------
// -----  S T A T I C   D A T A   M E M B E R S -----
//--------------------------------------------------------------------
private:
```



```
static unsigned         nextUserNumber;


//--------------------------------------------------------------------
// -----  D A T A   M E M B E R S -----
//--------------------------------------------------------------------
protected:
CVDSServer*     myServer;


//--------------------------------------------------------------------
// -----  M E T H O D S -----
//--------------------------------------------------------------------
public:
CVDSUser(CVDSServer* server=0);
CVDSUser(const CVDSUser& other);
                                        // No copy constructor for users
                                        // which correspond to external clients
~CVDSUser(void);

boolean       IsDefined(void)           const   {return TRUE;}
IcerClassType IsA(void)                 const   {return _CVDSUSER;}
void          Print(ostream& os=cout)   const;

friend ostream& operator<<(ostream& os, const CVDSServer& server);
CVDSUser& operator=(const CVDSUser& other);

void  SetUserId(const RWCString& id)            {SetTheIdString(id);     }
void  SetUserName(const RWCString& name)        {SetTheName(name);       }
void  SetUserDescription(const RWCString& desc) {SetTheDescription(desc);}
void  SetClientNumber(int number)               {SetIdNumber(number);    }

CVDSServer* GetServer(void)    const   {return myServer;        }

const RWCString& GetUserId(void)          const   {return GetTheIdString();   }
const RWCString& GetUserName(void)        const   {return GetTheName();       }
const RWCString& GetUserDescription(void) const   {return GetTheDescription();}
      int        GetClientNumber(void)    const   {return GetIdNumber();      }

      RWCString  GetNewUserId(void);


//--------------------------------------------------------------------
// VDS Server Command Methods
//--------------------------------------------------------------------
int USER_AddFile(const RWCString& groupName, const RWCString& datasetName,
        const RWCString& filename);
                        //-----------------------------------------------
                // Requests the dataset manager to open the dataset
                // then passes the filename, the group and dataset
                // names to a private method AddICERVSFile
                // for adding the file data to the VDS dataset
                //-----------------------------------------------
int USER_GetListOfGroups(void);
                        //-----------------------------------------------
                // Requests a list of all groups from the dataset
                // manager.  Sends back a list of group names and
                // descriptions to the requestor.
                //-----------------------------------------------
int USER_GetListOfDatasets(const RWCString& groupName);
                        //-----------------------------------------------
                // Requests a list of datasets in the group.
                // Gets back a list of dataset names and descrip-
                // tions and sends them back to the requestor
                //-----------------------------------------------
int USER_GetListOfOpenDatasets(const RWCString& groupName);
```

```
                    //-----------------------------------------------
            // Requests a list of open datasets in the group.
            // Gets back a list of dataset names and descrip-
            // tions and sends them back to the requestor
            //-----------------------------------------------
int USER_GetListOfAllOpenDatasets();
                    //-----------------------------------------------
            // Requests a list of open datasets(all groups)
            // Gets back a list of dataset names and descrip-
            // tions and group names and sends them back to
                    // the requestor
            //-----------------------------------------------
int USER_SetOperatorName(const RWCString& operatorName);
---->           //-----------------------------------------------
                    // Calls a method on the dataset manager to set
            // the operator name (i.e. the logged-in user)
            //-----------------------------------------------
int USER_VDSStartGUI(const RWCString& displayName);
                    //-----------------------------------------------
            // Requests the dataset manager to start the
            // VDS user interface on the specified display
            //-----------------------------------------------
int USER_VDSTerminate(void);
                    //-----------------------------------------------
            // Checks to see if there are any open datasets
                    // which were changed (but not saved).  If there
                    // is one, the user is asked whether to continue
                    // to terminate or to abort the terminating.
            // Status reply is returned to requestor
            //-----------------------------------------------

};
#endif
```

```
//*************************************************************************/
//  File:               view.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CView
//
//  Class Description:  CView encapsulates a view of a dataset.  It contains
//                      data members that are created dynamically when a view
//                      is created and are used for viewing the data in a view
//                      window.  These data members are transient.
//                      In addition, it contains the view parameters which are
//                      permanently stored parameters used to re-create a view
//                      window.
//
//  Inheritance:   <CView>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   04/25/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined CView_H_
    #define  CView_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include <iostream.h>                       //C++ input/output streams
#include <standard.h>                       //MTI Standard Definitions
#include <rw/compiler.h>                    //RW compiler ident
#include <rw/cstring.h>                     //RWCString
#include <rw/rstream.h>
#include <rw/rwdate.h>                      //RWDate class
#include <rw/rwtime.h>                      //RWTime class

#include <cnamed.h>                         //CNamedItem
#include <ictypes.h>                        //ICER common defs
//#include <viewwin.h>                        //CViewWindow
#include <color.h>                          //CColor
#include <3dpoint.h>                        //CColor


#include <X11/Xlib.h>
#include <X11/Xutil.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CCamera;
class CDatasetViewer;
class CVDSDataset;
class CRenderModel;
class CViewWindow;
```



```
class CColormapData;

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
class CView :public CNamedItem
{

//=========================================================================
// DECLARATIONS (enums, defines, etc)
//=========================================================================
public:

typedef enum { UNKNOWN              = 0,
               ORTHOGRAPHIC         = 1,
               PERSPECTIVE          = 2}  CameraType;

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:
  int    viewNumber;

  //-------------------------------
  //...Pointers to needed objects
  //-------------------------------
  CDatasetViewer*    pDatasetViewer;   //pointer to datasetviewer object
  CVDSDataset*       pDataset;         //pointer to dataset object
  CRenderModel*      pRenderModel;     //pointer to rendermodel object

  //-------------------------------
  //...The ColormapData object
  //-------------------------------
  CColormapData*       pColormapData;      //pointer to colormap data object

  //--------------------
  //...The view window
  //--------------------
  CViewWindow*       pViewWindow;      //pointer to view window object

  //-------------------------------
  //...The view window's camera
  //-------------------------------
  CCamera*           pCamera;          //CCamera object

  //-------------------------------
  //...Changed flags
  //-------------------------------
  int        viewChangedFlag;          //flag indicating any change to view
  unsigned   refreshScreen;            //flag indicating screen refresh

  //-------------------------------
  //...General parameters
  //-------------------------------
  RWCString  thresholdPath;            //Complete Threshhold filename

  //-------------------------------
  //...Current point and property
  //-------------------------------
  C3dPoint currentPoint;
  C3dPoint lastPoint;
  double currentDistance;
  double currentProperty;

  //=========================================
  //...View Parameters from View Parameter File
```

```
  //=============================================

  //------------------
  //...Boolean Options
  //------------------
  unsigned      showObjects;               //0=OFF        1=ON
  unsigned      manualRefresh;             //0=OFF        1=ON
  unsigned      statusWindow;              //0=OFF        1=ON
  unsigned      highResolutionDisplayMode; //0=OFF        1=ON

  //---------------------
  //..Various Parameters
  //---------------------
  float         antiAliasing;    //Used for TrueSolid pixel resolution
  RWCString     propertyType;    //Type of property to be viewed (NONE=z-value)
  RWCString     thresholdFile;   //Threshhold filename
  CameraType    cameraType;      //type of camera (ORTHOGRAPHIC or PERSPECTIVE)

  //---------------------
  //..Window  Parameters
  //---------------------
  int           renderAreaSizeX;    //pixel size of render area in X direction
  int           renderAreaSizeY;    //pixel size of render area in Y direction
  int           windowPositionX;    //pixel lower x position of window
  int           windowPositionY;    //pixel lower y position of window

  //---------------------
  //...Color Parameters
  //---------------------
  CColor          colorBackgroundWindow;    //Background window color
  CColor          colorObjects;             //Object color
  CColor          colorSelectedRegion;      //Selected region color
  CColor          colorSelectedObject;      //Selected object color
  CColor          colorLowResolutionPoints; //Low resolution points color

  //---------------------------
  //..Data Indicator Parameters
  //---------------------------
  CColor        indicatorColor;        //Data indicator color
  float         indicatorSize;         //Data indicator size (0.0 to 1.0)
  float         maxIndicatorSize;      //Maximum data indicator threshold
  unsigned      showIndicators;        //indicator visibility flag (0=OFF 1=ON)

  //---------------------------
  //..Cutplane Parameters
  //---------------------------
  int           applyCutplanesFlag;    //apply cutplanes flag

  //...Color by range flag
  boolean                     colorByRangeFlag;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
  CView(CDatasetViewer*  parent,
        CVDSDataset *dataset,
        CRenderModel *rendermodel,
        int viewNumb);

 ~CView(void);

  CView(const CView& other,int viewNumb);     //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
```



```
//=========================================================================
private:

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CView& P)
                                                  {P.Print(os); return os;}
    CView& operator=(const CView& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  CDatasetViewer*  GetDatasetViewerPointer(void) const {return pDatasetViewer;}
  CVDSDataset*     GetDatasetPointer(void)       const {return pDataset;}
  CRenderModel*    GetRenderModelPointer(void)   const {return pRenderModel;}
  CViewWindow*     GetViewWindowPointer(void)    const {return pViewWindow;}
  CCamera*         GetCameraPointer(void)        const {return pCamera;};
  CColormapData*   GetColormapDataPointer(void)  const {return pColormapData;};

  int              GetViewNumber(void)       const  {return viewNumber;}
  int              GetViewChangedFlag(void)  const  {return viewChangedFlag;}
  const RWCString& GetThresholdPath(void)    const  {return thresholdPath;}
  unsigned         GetRefreshScreen(void)    const  {return refreshScreen;}
  C3dPoint         GetCurrentPoint(void)     const  {return currentPoint;}
  double           GetCurrentDistance(void)  const  {return currentDistance;}
  double           GetCurrentProperty(void)  const  {return currentProperty;}

  boolean          GetColorByRangeFlag(void)  const {return colorByRangeFlag;}
  void             SetColorByRangeFlag(boolean flag) {colorByRangeFlag = flag;}

  void             SetViewNumber(int value)      {viewNumber = value;}
  void             SetViewChangedFlag(int value) {viewChangedFlag = value;}
  void             SetThresholdPath(const RWCString& value)
                                                 {thresholdPath = value;}
  void             SetRefreshScreen(unsigned value) {refreshScreen = value;}
  void             SetCurrentPoint(C3dPoint value);

  //-----------------------------------
  //...Sets and gets for view parameters
  //-----------------------------------
  unsigned         GetShowObjects(void)   const {return showObjects;}
  unsigned         GetManualRefresh(void) const {return manualRefresh;}
  unsigned         GetStatusWindow(void)  const {return statusWindow;}
  unsigned         GetHighResolutionDisplayMode(void)  const
                                        {return highResolutionDisplayMode;}

  float            GetAntiAliasing(void)        const {return antiAliasing;}
  const RWCString& GetPropertyType(void)        const {return propertyType;}
  const RWCString& GetThresholdFile(void)       const {return thresholdFile;}
  CameraType       GetCameraType(void)          const {return cameraType;}

  int              GetRenderAreaSizeX(void) const {return renderAreaSizeX;}
  int              GetRenderAreaSizeY(void) const {return renderAreaSizeY;}
  int              GetWindowPositionX(void) const {return windowPositionX;}
  int              GetWindowPositionY(void) const {return windowPositionY;}

  CColor&          GetColorBackgroundWindow(void)
                                        {return colorBackgroundWindow;}
  CColor&          GetColorObjects(void)
                                        {return colorObjects;}
  CColor&          GetColorSelectedRegion(void)
                                        {return colorSelectedRegion;}
  CColor&          GetColorSelectedObject(void)
```

```
                                        {return colorSelectedObject;}
  CColor&          GetColorLowResolutionPoints(void)
                                        {return colorLowResolutionPoints;}

  CColor&          GetIndicatorColor(void)    {return indicatorColor;}
  float            GetIndicatorSize(void)     const {return indicatorSize;}
  float            GetMaxIndicatorSize(void)  const {return maxIndicatorSize;}
  unsigned         GetShowIndicators(void)    const {return showIndicators;}
  int              GetApplyCutplanesFlag(void) const {return applyCutplanesFlag
---->;}


  void  SetShowObjects(unsigned val)              {showObjects = val;}
  void  SetManualRefresh(unsigned val)            {manualRefresh = val;}
  void  SetStatusWindow(unsigned val)             {statusWindow = val;}
  void  SetHighResolutionDisplayMode(unsigned val) {highResolutionDisplayMode =
---->val;}

  void  SetAntiAliasing(float value)              {antiAliasing = value;}
  void  SetPropertyType(const RWCString& val);
  void  SetThresholdFile(const RWCString& val)    {thresholdFile = val;}
  void  SetCameraType(CameraType val)             {cameraType = val;}

  void  SetRenderAreaSizeX(int val);
  void  SetRenderAreaSizeY(int val)               {renderAreaSizeY  =val;}
  void  SetWindowPositionX(int val)               {windowPositionX = val;}
  void  SetWindowPositionY(int val)               {windowPositionY = val;}

  void  SetColorBackgroundWindow(float r,float g, float b)
                  {colorBackgroundWindow.SetRGB(r,g,b);}
  void  SetColorObjects(float r,float g, float b)
                  {colorObjects.SetRGB(r,g,b);}
  void  SetColorSelectedRegion(float r,float g, float b)
                  {colorSelectedRegion.SetRGB(r,g,b);}
  void  SetColorSelectedObject(float r,float g, float b)
                  {colorSelectedObject.SetRGB(r,g,b);}
  void  SetColorLowResolutionPoints(float r,float g, float b)
                  {colorLowResolutionPoints.SetRGB(r,g,b);}

  void  SetIndicatorColor(float r,float g, float b)
                  {indicatorColor.SetRGB(r,g,b);}
  void  SetIndicatorSize(float val)               {indicatorSize = val;}
  void  SetMaxIndicatorSize(float val)            {maxIndicatorSize = val;}
  void  SetShowIndicators(unsigned val)           {showIndicators = val;}
  void  SetApplyCutplanesFlag(int val);

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void            Print(ostream& os=cout)     const;    //Print object data
  IcerClassType   IsA(void)                   const   {return _CVIEW;}

   //--------------------------------
   //...View General methods
   //--------------------------------
  int             InitializeView(void);    //Completes view initialization
  int             StartView(void);         //Displays the view window
  int             StopView(void);          //Hides the view window
  int             ViewAll(void);           //Views all the data

  //...Apply the view parameters to the view window
  void            ApplyViewParameters(void);          //Render the view

  //-------------------------------------
  // Get view parameters filename
```



```
  //-------------------------------------
   RWCString GetViewParameterFile(void)       const;

  //-----------------------------------------------
  //...Dump View Parameters to a file for viewing
  //-----------------------------------------------
  int DumpViewParameters(const RWCString&);

  //-------------------------------------
  //...View Parameters File access methods
  //-------------------------------------
  int ReadViewParameters(void);
  int WriteViewParameters(void);

  int  ReadViewParameters(const RWCString&);
  int  WriteViewParameters(const RWCString&,int readOptionsOnly);

 //-----------------------------------------
  // These functions are defined for convenience
  //-----------------------------------------------
  void  SetViewId(const RWCString& id)            {SetTheIdString(id);}
  void  SetViewName(const RWCString& name)        {SetTheName(name);}
  void  SetViewDescription(const RWCString& desc) {SetTheDescription(desc);}

  const RWCString& GetViewId(void)          const {return GetTheIdString();}
  const RWCString& GetViewName(void)        const  {return GetTheName();}
  const RWCString& GetViewDescription(void) const
                                                   {return GetTheDescription();}

//-------------------------------------------------------------------------
// Methods Inherited From CNamedItem
//-------------------------------------------------------------------------
// SetTheName          GetTheName              SetOwner
// SetTheDecsription   GetTheDescription       GetOwner
// SetIdNumber         GetIdNumber             GetOwnerIdString
// SetTheIdString      GetTheIdString

};

#endif
```

```
//*********************************************************************/
//  File:               viewlist.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CViewList.
//
//  Class Description:  CViewList encapsulates the behavior for a
//                      collection of CView objects.
//
//  Add() and Remove() routines from CSetOfNamedItems have been overridden
//  to support dynamically allocated CView objects.//
//  Inheritance: <RWOrdereds>  <CSetOfNamedItems>  <CViewList>
//
//  Required Class Libraries: Rogue Wave Math/Tools
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  102:00:00   03/20/94    D. Smith        Created
//  103:00:00   04/25/95    L. Cowper       Modified for Phase III
//
//*********************************************************************/
#if !defined CViewList_H_
    #define  CViewList_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include <iostream.h>                          //C++ input/output streams
#include<standard.h>                           //MTI Standard Definitions

#include <rw/compiler.h>                        //RW compiler ident
#include <rw/rstream.h>

#include<ictypes.h>                            //ICER common defs
#include<csetof.h>                             //CSetOfNamedItems
#include<view.h>                               //CView

//=====================================================================
// OTHER DECLARATIONS
//=====================================================================

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CViewList :public CSetOfNamedItems
{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
```



```
private:

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
  CViewList(const char* listName,int initialSize=5);
  CViewList(void);
 ~CViewList(void);

    CViewList(const CViewList& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CViewList& P)
                                                {P.Print(os); return os;}
    CViewList& operator=(const CViewList& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:

//=====================================================================
// OTHER METHODS
//=====================================================================
public:
  void          Print(ostream& os=cout)  const;         //Print object data
  IcerClassType IsA(void)                const        {return _CVIEWLIST;}

  int           Add(CView* view);

  boolean       Remove(const RWCString& viewName);
  boolean       Remove(CView* view);
  boolean       RemoveAll(void);

  boolean       RemoveById(const RWCString& viewId);

//---------------------------------------------------------------------
// These functions are defined for convenience
//---------------------------------------------------------------------
  void          ClearViewSet(void)                                {ClearItems();}

  int           GetListOfIds(CSortedStringList* aList)          const
                                {return GetListOfItemIdStrings(aList); }

  int           GetListOfNames(CSortedStringList* aList)        const
                                {return GetListOfItemNames(aList); }

  int           GetListOfDescriptions(CSortedStringList* aList) const
                                {return GetListOfItemDescriptions(aList); }

  CView*        Find(const RWCString& viewName)                 const
                                {return (CView *) FindItem(viewName);}

  CView*        FindById(const RWCString& viewId)               const
                                {return (CView *) FindItemByIdString(viewId);
---->}
```

```
  CView*  GetFirst(void)    {return (CView *) GetFirstItem();    }
  CView*  GetNext(void)     {return (CView *) GetNextItem();     }
  CView*  GetPrevious(void) {return (CView *) GetPreviousItem(); }
  CView*  GetLast(void)     {return (CView *) GetLastItem();     }
);

#endif
```

```
//*********************************************************************/
//  File:               viewwin.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CViewWindow
//
//  Class Description:  The CViewWindow class encapsulates all behaviors of
//  the view window in ICERVS III.  The class builds an Open Inventor
//  Full Viewer and handles all the view window callbacks for the
//  menubar and the render area.
//
//  Inheritance: <SoXtComponent> <CViewWindow>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  --------    --------    --------        -------
//  103:00:00   04/25/95    L. Cowper       Created
//
//*********************************************************************/
#if !defined ViewWindow_H_
    #define  ViewWindow_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include  <Xm/Xm.h>                    //Motif include files
#include  <Xm/Form.h>                   //Motif include files

#include <Inventor/SoType.h>                              //Inventor incude files
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/Xt/viewers/SoXtWalkViewer.h>
#include <Inventor/Xt/viewers/SoXtFlyViewer.h>
#include <Inventor/Xt/viewers/SoXtFullViewer.h>
#include <Inventor/Xt/SoXtDirectionalLightEditor.h>
#include <Inventor/Xt/SoXtComponent.h>
#include <Inventor/nodes/SoCallback.h>
#include <Inventor/nodes/SoOrthographicCamera.h>
#include <Inventor/nodes/SoText3.h>
#include <Inventor/nodes/SoLight.h>
#include <Inventor/actions/SoBoxHighlightRenderAction.h>
```



```
#include <Inventor/sensors/SoNodeSensor.h>

#include <standard.h>                 //Our own include files
#include <ictypes.h>
#include <inventor.h>
#include <rendermodel.h>
#include <viewmenu.h>
#include <3dpoint.h>

#include <statuswindow.h>
#include <vmenuanalyze.h>
#include <vmenudataset.h>
#include <vmenudebug.h>
#include <vmenuinput.h>
#include <vmenuoutput.h>
#include <vmenuobject.h>
#include <vmenuview.h>
#include <psuedomenu.h>


//=====================================================================
// OTHER DECLARATIONS
//=====================================================================
struct SoSceneViewerData;

class CView;
class CDatasetViewer;
class CVDSDataset;
class CRenderModel;


//...Toggle button macros
#define TOGGLE_ON(BUTTON) \
        XmToggleButtonState((Widget) BUTTON, TRUE, FALSE)
#define TOGGLE_OFF(BUTTON) \
        XmToggleButtonState((Widget) BUTTON, FALSE, FALSE)

#define MAX_USER_BUTTONS 12

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
class CViewWindow : public SoXtComponent{

//=====================================================================
// DECLARATIONS (enums, defines, etc)
//=====================================================================
public:

  //...The Individual menu functions broken out by their own classes
  friend class CDatasetMenu;
  friend class CViewMenu;
  friend class CAnalyzeMenu;
  friend class CObjectMenu;
  friend class CInputMenu;
  friend class COutputMenu;
  friend class CDebugMenu;
  friend class CStatusWindow;
  friend class CPsuedoMenu;

  //...The HighlightRenderAction stuff
  enum                    VdsEHighlight {
                          VDS_SELECTED_OBJECTS = 0,
                          VDS_INCLUDED_OBJECTS,
                          VDS_EXCLUDED_OBJECTS,
                           };
```

```
//=============================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=============================================================================
private:
  Widget savedHeadlightShowButton;
  Widget savedCutplaneShowButton;
  Widget savedViewersExaminerButton;
  Widget savedViewersWalkButton;
  Widget savedViewersFlyButton;

  //...Pointers to needed objects
  CView                 *pView;             //pointer to our parent
  CDatasetViewer        *pDatasetViewer;    //pointer to dataset viewer object
  CRenderModel          *pRenderModel;      //pointer to render model object
  CVDSDataset           *pDataset;          //pointer to dataset object

  //...Some Inventor widgets
  Widget                mgrWidget;         //the form widget
  Widget                myWidget;          //the vender shell  widget

  //...Our view window number
  int                   viewNum;

  //...some menu stuff
  Widget                menuWidget;          //topbar menu widget
  SoSceneViewerData     *menuItems;          //list of menu items data

  //...The Viewer stuff
  enum                  VdsEViewer {
                        VDS_VWR_EXAMINER = 0,
                        VDS_VWR_FLY,
                        VDS_VWR_WALK,
                        VDS_NUM_VIEWERS,  // insert new viewers before this.
                        VDS_VWR_NONE,
                         };

  //list of available viewers
  SoXtFullViewer        *viewerList[VDS_NUM_VIEWERS];
  SoNodeSensor          *insetSensor[VDS_NUM_VIEWERS];  //camera node sensor
  VdsEViewer            whichViewer;         //current viewer number
  SoXtFullViewer        *myViewer;           //the current Viewer
  boolean               userButtonsDefined; //the examiner viewer user buttons

  //...Rendering message text
  SoText3               *renderingMessage;

  //...Our Full Viewer window abort renderingr flag
  boolean                abortRenderFlag;    //this flag determines if
                                             //rendering should be aborted


  CDatasetMenu          *pDatasetMenu;       //dataset menu functions
  CViewMenu             *pViewMenu;          //view menu functions
  CAnalyzeMenu          *pAnalyzeMenu;       //analyze menu functions
  CObjectMenu           *pObjectMenu;        //object menu functions
  CInputMenu            *pInputMenu;         //input menu functions
  COutputMenu           *pOutputMenu;        //output menu functions
  CDebugMenu            *pDebugMenu;         //debug menu functions
  CStatusWindow         *pStatusWindow;      //status window functions
  CPsuedoMenu           *pPsuedoMenu;        //psuedo menu callback windows

  Widget popupWidget;
  Widget userButtons[MAX_USER_BUTTONS];      //user button widgets
  int    currentManipulatorButton;           //current manipulator button

  //...The HighlightRenderAction stuff
  SoBoxHighlightRenderAction *highlightRenderAction;
```



```
  //...The Directional Headlight stuff
  SoXtDirectionalLightEditor *headLightEditor;
  SoLight                    *currentLight;
  boolean                    headlightShow;

//=============================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=============================================================================
public:
   CViewWindow(CView *parent, int viewNumber);
   ~CViewWindow(void);

   CViewWindow(const CViewWindow& other);        //Copy constructor

//=============================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=============================================================================
private:
  void buildAndLayoutIcons(Widget parent);
  void buildAndLayoutMenu(Widget parent);
  void buildAndLayoutViewer(SoXtFullViewer *vwr);
  void buildAndLayoutStatusWindow(void);
  void buildAndLayoutUserButtons(void);
  void AddUserButton(Widget button);

  RWCString GetUserButtonFilename(int index);
  RWCString GetUserButtonReverseVideoFilename(int index);

  void ShowMenu(SbBool flag);

  //...Callback to intercept X events and process those we are interested in
  static SbBool AnyEventCallback(void *data, XAnyEvent *anyEvent);
  SbBool AnyEventCallback(XAnyEvent *anyEvent);


  //...Callback for all menu buttons to perform action
   static void       processTopbarEvent(Widget,
                                        SoSceneViewerData *,
                                        XmAnyCallbackStruct *);

  //...Callback for all icon buttons to perform action
   static void       processIconEvent(Widget,
                                      SoSceneViewerData *,
                                      XmAnyCallbackStruct *);

  //...Callback when a menu is displayed
   static void       menuDisplay(Widget,
                                 SoSceneViewerData *,
                                 XtPointer);

  //...Called after objects are added/deleted or the selection changes
   void              updateCommandAvailability();

  //...Abort rendering callback
   static SoGLRenderAction::AbortCode AbortRenderCallback(void *data);
   SoGLRenderAction::AbortCode AbortRenderCallback(void);

public:
  static void CViewWindow::UserButtonCB1(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w;cbs=cbs;
    ((CViewWindow *)userData)->UserButton1();
  }
  void CViewWindow::UserButton1(void);
```

```
  static void CViewWindow::UserButtonCB2(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w;cbs=cbs;
    ((CViewWindow *)userData)->UserButton2();
  }
  void CViewWindow::UserButton2(void);


  static void CViewWindow::UserButtonCB3(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton3();
  }
  void CViewWindow::UserButton3(void);


  static void CViewWindow::UserButtonCB4(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton4();
  }
  void CViewWindow::UserButton4(void);


  static void CViewWindow::UserButtonCB5(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton5();
  }
  void CViewWindow::UserButton5(void);


  static void CViewWindow::UserButtonCB6(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w;cbs=cbs;
    ((CViewWindow *)userData)->UserButton6();
  }
  void CViewWindow::UserButton6(void);


  static void CViewWindow::UserButtonCB7(Widget w, void *userData,
                                XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton7();
  }
  void CViewWindow::UserButton7(void);


  static void CViewWindow::UserButtonCB8(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton8();
  }
  void CViewWindow::UserButton8(void);


  static void CViewWindow::UserButtonCB9(Widget w, void *userData,
                                         XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton9();
  }
  void CViewWindow::UserButton9(void);


  static void CViewWindow::UserButtonCB10(Widget w, void *userData,
```



```
                                          XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton10();
  }
  void CViewWindow::UserButton10(void);


  static void CViewWindow::UserButtonCB11(Widget w, void *userData,
                                          XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton11();
  }
  void CViewWindow::UserButton11(void);


  static void CViewWindow::UserButtonCB12(Widget w, void *userData,
                                          XmPushButtonCallbackStruct *cbs){
    w=w; cbs=cbs;
    ((CViewWindow *)userData)->UserButton12();
  }
  void CViewWindow::UserButton12(void);

  void CViewWindow::UpdateManipulatorButtons(int button);

  //...Callback to intercept resize events
  static void ViewResizeCallbackHelper(Widget widget, XtPointer cdp,
                                                      XtPointer cbs);
  void ViewResizeCallbackHelper2(XtPointer cbs);

private:
  //...Switch the viewer type
  int       SwitchToViewer(VdsEViewer newViewer);

  //...Headlight Editor
  void      HeadlightEditor(boolean showEditor, boolean showHeadlight);

  //-----------------------------------
  //...View window icon function methods
  //-----------------------------------

  //-----------------------------------------------------
  //...Position camera to a set orthoginal viewpoint
  //
  // -Use CCamera's set method: SetOrientationVector to
  // Position the camera to the desired viewpoint
  // -Invoke the Render(view*) method in the Render Model
  //-----------------------------------------------------
  void      IconTop(void);
  void      IconBot(void);
  void      IconRight(void);
  void      IconLeft(void);
  void      IconFront(void);
  void      Iconback(void);

  //-----------------------------------------------------
  //...Select a particular object manipulator
  //
  // Use Inventor methods to set a particular manipulator
  // (We must build the reshape manipulator ourselves)
  //-----------------------------------------------------
  void      IconManipTransformbox(void);
  void      IconManipTrackball(void);
  void      IconManipTabbox(void);
  void      IconManipReshape(void);

  //-----------------------------------------------------
```

```
  //...Refresh the view window
  //
  // Call CRenderModel method: Render(view*)
  //-----------------------------------------------------
  void      IconRefresh(void);

  //-----------------------------------------------------
  //...View Parameters I/O methods
  //-----------------------------------------------------
  ////RWCString GetViewParametersFilename(void)        const;

  int       ReadViewParameters(void);
  int       WriteViewParameters(void);

  int       ReadViewParameters(const RWCString& prmFile);
  int       WriteViewParameters(const RWCString& prmFile);

  //  This gets called when the camera changes, to update the inset xform.
  static void InsetCameraCB(void *data, SoSensor *sensor){
    // If the status window is not available yet then just return.
    if(((CViewWindow *)data)->pStatusWindow != 0){
      ((CViewWindow *)data)->InsetCameraCB(sensor);
    }
  };
  void InsetCameraCB(SoSensor *sensor);

//=============================================================================
// OPERATORS
//=============================================================================
public:
    friend ostream& operator<<(ostream& os, const CViewWindow& P)
                                                {P.Print(os); return os;}
    CViewWindow& operator=(const CViewWindow& other);

//=============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=============================================================================
public:

   CPsuedoMenu*      GetCPsuedoMenuPointer(void)  const {return pPsuedoMenu;}
   CStatusWindow*    GetStatusWindowPointer(void) const {return pStatusWindow;}
   Widget            GetMgrWidget(void)           const {return mgrWidget;}
   Widget            GetMyWidget(void)            const {return myWidget;}
   CView*            GetViewPointer(void)         const {return pView;}
   CVDSDataset*      GetDatasetPointer(void)      const {return pDataset;}
   CRenderModel*     GetRenderModelPointer(void)  const {return pRenderModel;}
   SoXtFullViewer*   GetViewer(void)              const {return myViewer;}
   int               GetViewNumber(void)          const {return viewNum;}
   CObjectMenu*      GetObjectMenuPointer(void)   const {return pObjectMenu;}
   CAnalyzeMenu*     GetAnalyzeMenuPointer(void)  const {return pAnalyzeMenu;}


   //-----------------------
   //..Set view window size
   //-----------------------
   void              SetViewWindowSize(void);
   void              UpdateViewWindowSize(void);
   void              GetWindowPosition(int* posx, int*posy);

//=============================================================================
// OTHER METHODS
//=============================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print object data
  IcerClassType IsA(void)                const       {return _CVIEWWINDOW;}
```



```
  //...Create the view window and all of its components
  int     CreateViewWindow(void);

  //...Display the view window
  int     StartViewWindow(void);

  //...Hide the view window
  int     StopViewWindow(void);

  //...Set the view window title
  void    SetViewWindowTitle(void);

  //...Close the view window
  static void       CloseViewCallback(Widget,
                                      XtPointer userData,
                                      XtPointer){
                    ((CViewWindow *)userData)->CloseView();}
  void     CloseView();

  //...Switch to the desired highlightRenderAction type
  int      SwitchToHighlightRenderAction(VdsEHighlight newHighlightType);
};

#endif
```

```
//*********************************************************************/
//  File:               vmenuanalyze.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CAnalyzeMenu
//
//  Class Description:  The CAnalyzeMenu class encapsulates all behaviors of
//  the view window analyze menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the dataset functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CAnalyzeMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*********************************************************************/
#if !defined AnalyzeMenu_H_
    #define  AnalyzeMenu_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <standard.h>                  //Our own include files
#include <ictypes.h>
#include <viewWin.h>
#include <vdsdataset.h>
#include <UxXt.h>
#include <callback.h>                            //callback definitions


//=====================================================================
// OTHER DECLARATIONS
//=====================================================================
class CRegionSelectWindow;
```



```
//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------

class CAnalyzeMenu :public CUserCallbackFacility

{

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:

  //...Pointers to needed objects
  CViewWindow       *pViewWindow;         //pointer to our parent
  CView             *pView;
  CVDSDataset           *ourDataset;          //CVDSDataset class
  CRegionSelectWindow    *regionSelectWindow;

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
    CAnalyzeMenu(CViewWindow *parent);
    ~CAnalyzeMenu(void);

        CAnalyzeMenu(const CAnalyzeMenu& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:


//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CAnalyzeMenu& P)
                                                        {P.Print(os); return os;}
    CAnalyzeMenu& operator=(const CAnalyzeMenu& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:
  CViewWindow*  GetViewWindowPointer(void) const {return pViewWindow;}
  CView*        GetViewPointer(void) const {return pView;}
  CVDSDataset*  GetDatasetPointer(void) const {return ourDataset;};

//=====================================================================
// OTHER METHODS
//=====================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType  IsA(void)               const         {return _CANALYZEMENU;}

  //-------------------------------------------------------
  //...The folowing analyze methods will be described later
  //-------------------------------------------------------
  int  AnalyzeSelectRegion(void);
  void SelectRegionCancel(UserCBF cbf);
  int AnalyzeEraseRegion(void);

  int AnalyzeEraseInvisiblePoints(void);
  int AnalyzeColorByRange(int onOffFlag);
```

```
  int AnalyzeConnectivitySet(void);
  int AnalyzeConnectivityClear(void);

  int AnalyzePipeTracing(void);
  int AnalyzeGoodnessOfFit(void);
  int AnalyzeBestFit(void);
  int AnalyzeSurfaceMesh(void);
  int AnalyzeConsistencyCheck(void);
  int AnalyzeVolumetricDifference(void);
  int AnalyzeDeriveProperty(void);
);

#endif
```

```
  //  (CWDatasetAdd)
  //
  // -If more than one view exists for this dataset, ask:
  //  "Update all views to new dataset?"   Save answer in updateViewFlag
  //
  // -Invoke method on CDatasetManager to create a new dataset directory
  //  (CreateDatasetDirectory)
  //
  // -Call CVDSDataset copy constructor to instantiate a new data from
  //  the current dataset:
  //  (newdataset = CVDSDataset(*oldDataset))
  //
  // -Set new dataset's name and description using CVDSDataset set methods
  //
  // -Call the CDatasetsetManager method to add the new dataset object to VDS
  //  (AddDatasetObject)
  //
  // -Invoke the CVDSDataset method on the new dataset to save itself
  //  (DatasetSave)
  //
  // -Add the view pointer to the new dataset's set of views
  //
  // -Remove the view pointer from the original dataset's set of views
  //
  // -If updateViewFlag = TRUE then we want to update all views of the
  //  original dataset to be views on the new dataset
  //     -Remove each view from the original dataset's set of views
  //     -Add each view to the new dataset's set of views
  //
  // (See Phase II CVolumetricDataCLient::VDS_SaveAsDataset in vdagent.cpp)
  //-------------------------------------------------------------------
  int DatasetSaveAs(void);

  // Button callback
  void DatasetSaveAsOk(UserCBF cbf);
  void DatasetSaveAsCancel(UserCBF cbf);

  //-----------------------------------------------------
  //...View Window Dataset Menu Function: Insert
  //...Insert another dataset into this dataset (later)
  //-----------------------------------------------------
  int DatasetInsert();

};

#endif
```

```
//*********************************************************************/
//  File:               vmenudebug.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CDebugMenu
//
//  Class Description:  The CDebugMenu class encapsulates all behaviors of
//  the view window debug menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the dataset functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CDebugMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  --------    --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*********************************************************************/
#if !defined DebugMenu_H_
    #define  DebugMenu_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-Windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <ictypes.h>                   //Our own include files
#include <standard.h>
#include <viewwin.h>
#include <vdsdataset.h>


//=====================================================================
// OTHER DECLARATIONS
//=====================================================================

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------
```



```
class CDebugMenu

{

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:

  //...Pointers to needed objects
  CViewWindow       *pViewWindow;        //pointer to our parent
  CVDSDataset           *ourDataset;         //CVDSDataset class

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
    CDebugMenu(CViewWindow *parent);
   ~CDebugMenu(void);

        CDebugMenu(const CDebugMenu& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:

//=====================================================================
// OPERATORS
//=====================================================================
public:
    friend ostream& operator<<(ostream& os, const CDebugMenu& P)
                                                    {P.Print(os); return os;}
    CDebugMenu& operator=(const CDebugMenu& other);

//=====================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=====================================================================
public:

//=====================================================================
// OTHER METHODS
//=====================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType  IsA(void)                const        {return _CDEBUGMENU;}

  //-------------------------------------------------------
  //...View Window Debug Menu Function: Tree Statistics
  //...Output the octree statistics to a browser window
  //
  // -Invoke the CVDSDataset method: DebugDatasetStatistics
  //-------------------------------------------------------
  int OutputDebugTreestats(void);

  //-------------------------------------------------------
  //...View Window Debug Menu Function: View Parameters
  //...Output the view parameters to a browser window
  //
  // -Invoke the CView method: PrintParameters
  // -Browse the resulting file in a Browser window
  // -Delete the file when browsing is finished
  //-------------------------------------------------------
  int OutputDebugViewparms(void);
```

```
  //-----------------------------------------------------------------
  //...View Window Debug Menu Function: Dump ASCII
  //...Output the dimensional data in the octree along with tree level
  //...and node type to a browser window
  //
  // -Invoke the CVDSDataset method: DebugDumpASCII
  //-----------------------------------------------------------------
  int OutputDebugDumpASCII(void);

  //---------------------------------------------------------
  //...View Window Debug Menu Function: Input Point
  //...Input a dimensional data point into the octree (later)
  //---------------------------------------------------------
  int OutputDebugInputPoint(void);

  //---------------------------------------------------------
  //...View Window Debug Menu Function: Erase Point
  //...Erase a dimensional data point from the octree (later)
  //---------------------------------------------------------
  int OutputDebugErasePoint(void);

  //---------------------------------------------------------
  //...View Window Debug Menu Function: Write IV File
  //...Writes the entire Inventor tree to an IV file
  //---------------------------------------------------------
  int OutputDebugWriteIv(void);

  //---------------------------------------------------------
  //...View Window Debug Menu Function: Read IV File
  //...Read the entire Inventor tree from an IV file
  //---------------------------------------------------------
  int OutputDebugReadIv(void);

};

#endif
```

```
//*************************************************************************/
//  File:               vmenuinput.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CInputMenu
//
//  Class Description:  The CInputMenu class encapsulates all behaviors of
//  the view window input menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the dataset functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CInputMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//  103:00:01   07/14/95    J. Hutchison    Implemented InputSensorData
//
//*************************************************************************/
#if !defined InputMenu_H_
    #define  InputMenu_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <standard.h>                  //Our own include files
#include <ictypes.h>
#include <viewwin.h>
#include <vdsdataset.h>
#include <callback.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CInputNonscalarText;
class CInputNonscalarFile;
class CFileSelectionWindow;
```



```
class CPropertyValues;
//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------

class CInputMenu :public CUserCallbackFacility

{

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  //...Pointers to needed objects
  CViewWindow        *pViewWindow;        //pointer to our parent
  CVDSDataset            *ourDataset;         //CVDSDataset class

  CFileSelectionWindow *fileSelectionWindow;
  CInputNonscalarText *inputWindow;
  CInputNonscalarFile *input2Window;
  CPropertyValues *propertyWindowPointer;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    CInputMenu(CViewWindow *parent);
    ~CInputMenu(void);

        CInputMenu(const CInputMenu& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:


//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CInputMenu& P)
                                                {P.Print(os); return os;}
    CInputMenu& operator=(const CInputMenu& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType  IsA(void)                const        {return _CINPUTMENU;}

  //-----------------------------------------------------------
  //...View Window Input Menu Function: Sensor Data
  //...Input Sensor Data from ICERVS input file
  //
  // -See Phase II VdClient method: VDS_AddStandardICERVSFile
  // -This is the same function that APPL requests via VdUser
  //-----------------------------------------------------------
  int InputSensorData(void);
```

```
  void InputSensorDataCompleted(UserCBF cbf);
  void InputSensorDataCanceled(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Input Menu Function: 2D Drawing
  //...Input 2D drawings (later)
  //-------------------------------------------------------------
  int InputCadData2dDrawing(void);

  //-------------------------------------------------------------
  //...View Window Input Menu Function: 3D Model
  //...Input 3D models (later)
  //-------------------------------------------------------------
  int InputCadData3dModel(void);

  //-------------------------------------------------------------
  //...View Window Input Menu Function: Non-scalar Text
  //...Input non-scalar text & button callbacks
  //-------------------------------------------------------------
  int InputNonScalarText(void);
  void InputNonscalarTextSave(UserCBF cbf);
  void InputNonscalarTextClose(UserCBF cbf);



  //-------------------------------------------------------------
  //...View Window Input Menu Function: Non-scalar from file
  //...Input non-scalar data from file (later)
  //-------------------------------------------------------------
  int InputNonScalarFromFile(void);
  void InputNonscalarFileSave(UserCBF cbf);
  void InputNonscalarFileClose(UserCBF cbf);

};

#endif
```

```
//*********************************************************************/
//  File:               vmenuobject.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CObjectMenu
//
//  Class Description:  The CObjectMenu class encapsulates all behaviors of
//  the view window object menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the dataset functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CObjectMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*********************************************************************/
#if !defined ObjectMenu_H_
    #define  ObjectMenu_H_

//=====================================================================
// REQUIRED INCLUDE FILES
//=====================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-Windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <ictypes.h>                   //Our own include files
#include <standard.h>
#include <viewwin.h>
#include <vdsdataset.h>
#include <datasetmanager.h>
#include <datasetviewer.h>
#include <rendermodel.h>
#include <UxXt.h>
#include <callback.h>                          //callback definitions
#include <MyColorEditor.h>

#include <Inventor/Xt/SoXtTransformSliderSet.h>        //Inventor incude files
#include <Inventor/SoType.h>
```



```
#include <Inventor/nodes/SoCallback.h>
#include <Inventor/actions/SoAction.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/sensors/SoNodeSensor.h>
#include <Inventor/sensors/SoSensor.h>
#include <Inventor/Xt/SoXtComponent.h>

//=====================================================================
// OTHER DECLARATIONS
//=====================================================================
class CViewWindow;
class CView;
class C3dObject;
class CObjectSelectWindow;
class CObjectAttributeWindow;
class CObjectCreateWindow;
class CObjectEditWindow;
class CObjectLibrary;
class CFileSelectionWindow;
class CFileSelection2Window;
class CRequestWindow;
class CBrowserWindow;

//---------------------------------------------------------------------
// CLASS DEFINITION
//---------------------------------------------------------------------

class CObjectMenu :public CUserCallbackFacility

{

//=====================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=====================================================================
private:

  //...Pointers to needed objects
  CViewWindow    *pViewWindow;          //pointer to view window object
  CView          *pView;              //pointer to view object
  CDatasetViewer *pDatasetViewer;     //pointer to dataset viewer object
  C3dObject   *currentObject;          //pointer to current object

  CObjectSelectWindow *selectionWindow; //pointer to selection window
  CObjectAttributeWindow *attributeWindow;
  CObjectCreateWindow *createWindow;
  CObjectEditWindow *editWindow;
  CObjectLibrary *libraryWindow;
  CFileSelectionWindow *fileSelectionWindow;
  CFileSelectionWindow *fileSelection2Window;
  CRequestWindow* compositeBuildWindow;
  CBrowserWindow* browserWindow;

//=====================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=====================================================================
public:
    CObjectMenu(CViewWindow *parent);
    ~CObjectMenu(void);

        CObjectMenu(const CObjectMenu& other);        //Copy constructor

//=====================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=====================================================================
private:
```

```
//=============================================================================
// OPERATORS
//=============================================================================
public:
    friend ostream& operator<<(ostream& os, const CObjectMenu& P)
                                                    {P.Print(os); return os;}
    CObjectMenu& operator=(const CObjectMenu& other);

//=============================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=============================================================================
public:
  CViewWindow*  GetViewWindowPointer(void) const {return pViewWindow;}
  CView*        GetViewPointer(void)       const {return pView;}

//=============================================================================
// OTHER METHODS
//=============================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType IsA(void)                const        {return _COBJECTMENU;}

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Select
  //...Select an object from a list of all objects
  //
  // -Instantiate a window to select object (CWObjectSelect)
  //    -SelectAnItemCallback:
  //         -Get selected object name from window
  //         -Highlight the object in the view by invoking the CRenderModel
  //          method: HighlightRenderObject
  //         -Add selected object to list of selected objects in
  //          CDatasetViwer
  //         -Update status window
  //    -UnSelectAnItemCallback:
  //         -Get unselected object name from window
  //         -UnHighlight the object in the view by invoking the CRenderModel
  //          method: UnHighlightRenderObject
  //         -Remove unselected object from list of selected objects in
  //          CDatasetViwer
  //    -SelectAllItemsCallback:
  //         -Loop over all objects in the set of objects
  //            -Highlight each object in the view by invoking the
  //             CRenderModel  method: HighlightRenderObject
  //            -Add each object to list of selected objects in
  //             CDatasetViwer (no duplicates allowed)
  //         -Update status window
  //    -UnSelectAllItemsCallback:
  //         -Loop over all objects in the set of objects
  //            -UnHighlight each object in the view by invoking the
  //             CRenderModel method: UnHighlightRenderObject
  //            -Remove all objects from list of selected objects in
  //             CDatasetViwer
  //-----------------------------------------------------------------------
  int ObjectSelect(void);

  //Button callbacks
  void ObjectSelectCompleted(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Select
  //...Edit view settings
  //
  // -Instantiate a window to change an object's attributes
  //  (CWObjectAttribute)
```



```
  //    -OKCallback:
  //       -Get the display type, export flag, complexity flag from window
  //       -Loop over all object in the list of selected objects
  //          -For each object in list, update that object's three variables
  //-----------------------------------------------------------------------
  int ObjectAttributes(void);
  void ObjectAttributesClosed(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Create
  //...Create an object
  //
  // -Instantiate a window to Create object (CWObjectCreate)
  //    -CreateCallback:
  //       -Get the Object type, name, desc, export flag, and parameters
  //        from window
  //       -Instantiate the appropriate type of object
  //       -Set the object's parameters, name, desc, etc
  //       -Add object to dataset with CVDSDataset method: AddObject
  //-----------------------------------------------------------------------
  int ObjectCreate(void);
  void ObjectCreateCompleted(UserCBF cbf);
  void ObjectCreateCancelled(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Delete
  //...Delete an object
  //
  // -Get the objects to be deleted from the list of selected objects
  //
  // -Instantiate a verify message window to get operator OK
  //
  // -Remove object from dataset with CVDSDataset method: RemoveObject
  //-----------------------------------------------------------------------
  int ObjectDelete(void);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Copy
  //...Copy an object
  //
  // -Get the objects to be deleted from the list of selected objects
  //
  // -Make a copy of the object
  //
  // -Give the new object a different name
  //
  // -Add new object to dataset with CVDSDataset method: AddObject
  //-----------------------------------------------------------------------
  int ObjectCopy(void);

  // -Snap an object to the current point
  //-----------------------------------------------------------------------
  int ObjectSnap(void);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Composite Build
  //...Build a composite object from selected objects (later)
  //-----------------------------------------------------------------------
  int ObjectCompositeBuild(void);
  void BuildCancelled(UserCBF cbf);
  void BuildCompleted(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window Object Menu Function: Composite UnBuild
  //...UnBuild selected composite object (later)
  //-----------------------------------------------------------------------
```

```
  int ObjectCompositeUnBuild(void);
  void UnBuildCancelled(UserCBF cbf);
  void UnBuildCompleted(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: IGRIP Import
  //...Import an IGRIP file (later)
  //-------------------------------------------------------------
  int ObjectIgripImport(void);
  void ImportIGRIPCompleted(UserCBF cbf);
  void ImportIGRIPCanceled(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: IGRIP Export
  //...Export an IGRIP file (later)
  //-------------------------------------------------------------
  int ObjectIgripExport(void);
  void ExportIGRIPCompleted(UserCBF cbf);
  void ExportIGRIPCanceled(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Edit Parametric
  //...Edit a selected object parametrically
  //
  // -Instantiate a Window to Edit object (CWObjectCreate)
  // -In window:
  //    -set object type radio button to type of object being edited
  //    -grey out object type
  //    -fill in fields for object: name,desc, export flag, parameters
  //    -SaveCallback:
  //       -Remove object from dataset with CVDSDataset method: RemoveObject
  //       -Get the Object type, name, desc, export flag, and parameters
  //        from window
  //       -Instantiate the appropriate type of object
  //       -Set the object's parameters, name, desc, etc
  //       -Add object to dataset with CVDSDataset method: AddObject
  //-------------------------------------------------------------
  int ObjectEditParametric(void);
  void ObjectEditCompleted(UserCBF cbf);
  void ObjectEditCancelled(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Edit IGRIP
  //...Edit an object via IGRIP editor (later)
  //-------------------------------------------------------------
  int ObjectEditIgrip(void);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Transform
  //...Transform a selected object via Inventor's transform editor
  // (see View Transform for related functionality)
  // (also see ExaminerViewer source code for example)
  //-------------------------------------------------------------
  int ObjectEditTransform(void);

  //...Object transform slider
  SbBool                  ignoreVPCallback;
  const SbBool            GetIgnoreVPCallback(void) const {
    return ignoreVPCallback;
  }
  SoXtTransformSliderSet *objectXformSlider;
  SoTransform            *objectXformNode;
  SoNodeSensor           *xformNodeSensor;

  void              SetTransformSliderObject(C3dObject* object);
```



```
  static void       xformSliderCallback(void *userData, SoSensor *sensor){
                    ((CObjectMenu *)userData)->xformSliderCallback(sensor);}
  void              xformSliderCallback(SoSensor *sensor);

  static void       TransformSliderCloseCallback(void *userData,
                                                 SoXtComponent *cb){
                                    ((CObjectMenu *)userData)->
                                    SetTransformSliderCloseCallback(cb);}
  void              SetTransformSliderCloseCallback(SoXtComponent *cb);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Library
  //...Store/Retrieve  a selected object in the object library
  //-------------------------------------------------------------
  int ObjectLibrary(void);
  void ObjectLibraryCancel(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Color
  //...Color the selected objects via Inventor's color widget
  //
  // -Invoke an Inventor color editor on the selected object
  // (see ExaminerViewer source code for example)
  //-------------------------------------------------------------
  int ObjectColor(void);

 //...Object color stuff
  SbBool          ignoreCallback;
  const SbBool    GetIgnoreCallback(void)     const {return ignoreCallback;}
  MyColorEditor   *objectColorEditor;
  static void     ObjectColorCallback(void *userData,
                                      const SbColor *c);

  void            SetCurrentObject(C3dObject* object);

  static void     ColorCloseCallback(void *userData,
                                     SoXtComponent *cb){
                  ((CObjectMenu *)userData)->
                  SetColorCloseCallback(cb);}
  void            SetColorCloseCallback(SoXtComponent *cb);


  //-------------------------------------------------------------------
  //...View Window Object Menu Function: Info
  //...Display object information for the selected object
  //
  // -Open temp file for writing
  // -Loop over all objects in the list of selected objects
  //    -For each object, write its info to the temp file
  // -Close temp file
  // -Display temp files in Browser window
  // -Delete temp file when done browsing
  //-------------------------------------------------------------------
  int ObjectInfo(void);
  int BuildObjectInfoFile(C3dObject *object,const RWCString& filename);
  void ObjectInfoCompleted(UserCBF cbf);

  //-------------------------------------------------------------
  //...View Window Object Menu Function: Texture Map
  //...Texture map an image onto a selected object (later)
  //-------------------------------------------------------------
  int ObjectTextureMap(void);

};

#endif
```

```
//*************************************************************************/
//  File:               vmenuoutput.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      COutputMenu
//
//  Class Description:  The COutputMenu class encapsulates all behaviors of
//  the view window dataset menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the dataset functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <COutputMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined OutputMenu_H_
   #define  OutputMenu_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>                  //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>          //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <ictypes.h>                  //Our own include files
#include <standard.h>
#include <viewwin.h>
#include <vdsdataset.h>


//=========================================================================
// OTHER DECLARATIONS
//=========================================================================

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------
```



```
class COutputMenu

{

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  //...Pointers to needed objects
  CViewWindow       *pViewWindow;       //pointer to our parent
  CVDSDataset           *ourDataset;        //CVDSDataset class

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
    COutputMenu(CViewWindow *parent);
    ~COutputMenu(void);

        COutputMenu(const COutputMenu& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:


//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const COutputMenu& P)
                                                    {P.Print(os); return os;}
    COutputMenu& operator=(const COutputMenu& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:

//=========================================================================
// OTHER METHODS
//=========================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType  IsA(void)               const        {return _COUTPUTMENU;}


  //-----------------------------------------------------------------------
  //...View Window Output Menu Function: Points
  //...Output dimensional data and their scalar properties to an ASCII file
  //
  // -Invoke the CVDSDataset method: DumpDatasetASCII
  // -See Phase II VdClient method: VDS_DumpDatasetASCII for example
  //-----------------------------------------------------------------------
  int OutputPoints(void);

  //-------------------------------------------------------
  //...View Window Output Menu Function: Image Postscript
  //...Output a postscript image of data to file
  //
  // -Invoke the CVDSDataset method: CreatePostcript
  // -See Phase II VdClient method: VDS_DumpViewPostscript for example
  //-------------------------------------------------------
```

```
  int OutputImagePS(void);

  //-------------------------------------------------------------
  //...View Window Output Menu Function: Image DIB
  //...Output a DIB image of data to file
  //
  // -Invoke the CVDSDataset method: CreateDIBFile
  //-------------------------------------------------------------
  int OutputImageDIB(void);

  //-------------------------------------------------------------
  //...View Window Output Menu Function: Screen
  //...Dump the screen to the printer
  //
  // -Invoke the CVDSDataset method: DumpScreen
  //-------------------------------------------------------------
  int OutputScreen(void);

};

#endif
```

```
//*************************************************************************/
//  File:               vmenuview.h
//
//  Project Name:       ICERVS Phase III
//
//  File description:   This file contains the declaration for class
//                      CViewMenu
//
//  Class Description:  The CViewMenu class encapsulates all behaviors of
//  the view window view menu functions in ICERVS III.
//
//  This class is a convenience for software development.  It allows the
//  developer to work on the view functions in the view window without
//  getting in anyone's way. It is considered a friend class to the view
//  window class CViewWindow.
//
//  Inheritance:  <CViewMenu>
//
//  Required Class Libraries: Inventor
//
//
//  Mechanical Technology Inc. (MTI)
//  968 Albany-Shaker Road
//  Latham, NY 12210-0709
//
//  Proprietary Licensed Information.
//  Not to be duplicated, used, or disclosed,
//  except under terms of the license.
//
//  Revision    Date        Name            Remarks
//  ---------   --------    --------        -------
//  103:00:00   05/19/95    L. Cowper       Created
//
//*************************************************************************/
#if !defined ViewMenu_H_
   #define  ViewMenu_H_

//=========================================================================
// REQUIRED INCLUDE FILES
//=========================================================================
#include  <stddef.h>                    //Standard C include files
#include  <unistd.h>
#include  <string.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <iostream.h>

#include  <X11/StringDefs.h>           //X-windows include files
#include  <X11/Intrinsic.h>
#include  <X11/Xlib.h>
#include  <X11/Xutil.h>

#include <Inventor/Xt/SoXtTransformSliderSet.h>       //Our class definition
#include <Inventor/SoType.h>                          //Inventor incude files
#include <Inventor/nodes/SoCallback.h>
#include <Inventor/actions/SoAction.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/sensors/SoNodeSensor.h>
#include <Inventor/sensors/SoSensor.h>
#include <Inventor/SbBox.h>

#include <viewwin.h>
#include <standard.h>
#include <ictypes.h>
#include <UxXt.h>
```



```
#include <callback.h>

//=========================================================================
// OTHER DECLARATIONS
//=========================================================================
class CView;
class CViewIndicators;
class CViewSettingsWindow;
class CColormap;
class CViewCutplane;
class CICFileSelectionListBox;

//-------------------------------------------------------------------------
// CLASS DEFINITION
//-------------------------------------------------------------------------

class CViewMenu :public CUserCallbackFacility
{

//=========================================================================
// DATA MEMBERS (ALWAYS PRIVATE)
//=========================================================================
private:

  //...Pointers to needed objects
  CViewWindow *pViewWindow;    //pointer to view window object
  CView       *pView;          //pointer to view object

  CViewIndicators *indicatorWindow;
  CViewSettingsWindow *settingsWindow;
  CColormap *colorWindow;
  CViewCutplane *cutplaneWindow;
  CICFileSelectionListBox *fbox;
  CICFileSelectionListBox *fbox2;

  SoTransform *tmpTransform;

//=========================================================================
// CONSTRUCTORS AND DESTRUCTORS
//=========================================================================
public:
  CViewMenu(CViewWindow *viewWindow);
  ~CViewMenu(void);

  CViewMenu(const CViewMenu& other);        //Copy constructor

//=========================================================================
// PRIVATE METHODS (USED INTERNALLY BY CLASS)
//=========================================================================
private:

//=========================================================================
// OPERATORS
//=========================================================================
public:
    friend ostream& operator<<(ostream& os, const CViewMenu& P)
                                                  {P.Print(os); return os;}
    CViewMenu& operator=(const CViewMenu& other);

//=========================================================================
// METHODS FOR DATA ACCESS (SETS AND GETS) (usually inlined)
//=========================================================================
public:
  CViewWindow*  GetViewWindowPointer(void)    const {return pViewWindow;}
  CView*        GetViewPointer(void)    const {return pView;}
```

```
//=============================================================================
// OTHER METHODS
//=============================================================================
public:
  void          Print(ostream& os=cout)  const;        //Print data
  IcerClassType IsA(void)                const        {return _CVIEWMENU;}

  //-----------------------------------------------------------------------
  //...View Window - View Menu Function: Settings
  //...Edit some view parameter settings
  //
  // -Instantiate a window to allow editing (CWViewSettings)
  // -Apply Button Callback:
  //    -Get all the new view settings from window
  //    -Update view settings in CView's view parameters with Set methods
  //    -Update generalChangeFlag in CView's view parameters with Set method
  //    -Invoke the Render(view*) method in the Render Model
  // -Save As Default Button Callback:
  //    -Get all the new view settings from window
  //    -Get default view parameter filename from CView
  //     (GetDefaultViewParameterFilename)
  //    -Invoke CView's WriteParameters method with filename as argument
  // -Edit Default Color Button Callback:
  //    -Instantiate an Inventor Background Color Widget
  //    -Get background color and set in view parameters
  //-----------------------------------------------------------------------
  int           ViewSettings(void);

  //Button callbacks
  void          ViewSettingsCompleted(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window - View Menu Function: Colormap
  //...Get a colormap to use
  //-----------------------------------------------------------------------
  int ViewColormap(void);
  void CloseViewColormap(void);

  //-------------------------------------------------------
  //...View Window - View Menu Function: Cutplane
  //...Edit cutplane parameters (later)
  //-------------------------------------------------------
  int ViewCutplane(void);
  void ViewCutplaneCancel(UserCBF cbf);

  //-------------------------------------------------------
  //...View Window - View Menu Function: Transform
  //...Display Inventor transform editor for viewpoint control
  //-------------------------------------------------------
  int ViewTransform(void);

  //...Viewpoint transform slider
  SbBool                ignoreVPCallback;
  const SbBool          GetIgnoreVPCallback(void) const {
    return ignoreVPCallback;
  }
  SoXtTransformSliderSet *viewpointXformSlider;
  SoTransform           *viewpointXformNode;
  SoNodeSensor          *xformNodeSensor;
  static void           xformSliderCallback(void *userData, SoSensor *sensor){
    ((CViewMenu *)userData)->xformSliderCallback(sensor);
  };
  void                  xformSliderCallback(SoSensor *sensor);
  SbXfBox3f             GetSceneBBox( void );

  //-------------------------------------------------------
```



```
  //...View Window - View Menu Function: Indicators
  //...Edit the Indicator parameters
  //-------------------------------------------------------
  int ViewIndicators(void);

  //Button callbacks
  void          ViewIndicatorsCompleted(UserCBF cbf);

  //-------------------------------------------------------
  //...View Window - View Menu Function: Properties
  //...Select multiple properties for viewing (later)
  //-------------------------------------------------------
  int ViewProperties(void);

  //-----------------------------------------------------------------------
  //...View Window - View Menu Function: Copy
  //...Copy this view
  //
  // -Invoke the CDatasetViewer method: CopyView with a pointer to ourselves
  //
  // -Render the new view by invoking the CRenderModel's method: Render
  //
  //-----------------------------------------------------------------------
  int ViewCopy(void);

  //-----------------------------------------------------------------------
  //...View Window - View Menu Function: Save
  //...Save the view to a saved view file
  //
  // -Instantiate a window to input filename of saved view (CWViewSave)
  // -Ok Button Callback:
  //    -Get filename from window
  //    -Get path name of dataset and build total filename
  //    -Invoke CView's WriteParameters method with filename as argument
  //-----------------------------------------------------------------------
  int ViewSave(void);
  void ViewSaveCompleted(UserCBF cbf);
  void ViewSaveCanceled(UserCBF cbf);

  //-----------------------------------------------------------------------
  //...View Window - View Menu Function: Restore
  //...Restore the view from a saved view file
  //
  // -Instantiate a window to input filename of restored view (CWViewRestore)
  // -Ok Button Callback:
  //    -Get filename from window
  //    -Get path name of dataset and build total filename
  //    -Invoke CView's ReadParameters method with filename as argument
  //    -Update generalChangeFlag in CView's view parameters with Set method
  //    -Invoke the UpdateAllViews method in CDatasetViewer
  //     This will allow cutplanes to match in all views
  //-----------------------------------------------------------------------
  int ViewRestore(void);
  void ViewRestoreCompleted(UserCBF cbf);
  void ViewRestoreCanceled(UserCBF cbf);

  //-------------------------------------------------
  //...View Window - View Menu Function: Close
  //..Close a view
  //
  // -Call CDatasetViewer's RemoveView method
  //-------------------------------------------------
  int ViewClose(void);

  //-----------------------------------------------------------------------
  static void CloseViewCallback(Widget widget, XtPointer data, XtPointer cbs )
```

```
  {
    if(widget);
    if(cbs);
    ((CViewMenu *)data)->ViewClose();
  }
  //
  //  The CloseViewCallback will be called when the system detects that
  //  the view window is being closed and will then call ViewClose().
  //-------------------------------------------------------------------------
};

#endif
```