1992

Object-oriented interface to SECS-II

Don Breisch
Lehigh University

OBJECT-ORIENTED INTERFACE TO SECS-II

by

Don Breisch

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

December 1991

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

Dec 4 1997

Date

M Spezialetti

Professor M. Spezialetti
Thesis Advisor

Kenneth [illegible]

Chairman of Department

CONTENTS

LIST OF FIGURES

ABSTRACT

Software development can become more efficient and less error-prone as programmers become accustomed to the programming environment they are working in, especially if that environment has been well designed. In the Semiconductor industry, the software interface between manufacturing equipment and computers has become established to the point that programmers now have a stable environment in which to learn and grow. The industry standard SECS-II[1] details the content of messages exchanged between "smart" Semiconductor manufacturing equipment and host computers. The purpose is to ease the development of Data Automation Software Applications (DASA's) in the Semiconductor environment by providing a consistent interface between equipment and host computer.

A custom DASA must be developed for each type of equipment to support the unique capabilities of the equipment. Unfortunately, the complexity of a C language based interface to SECS-II has made it all but impossible for process engineers to develop these DASA's. Additionally, the flexibility of SECS-II data types, combined with the rigidity of C data types, has made each new DASA a "start from scratch" proposition.

This thesis highlights a SECS-II software package developed in **Borland C++**. This C++ interface to SECS-II eases DASA development by providing the programmer with a view of the SECS-II messages that may be easily grasped. In the SEMI Specification SECS-II messages are documented implicitly in a manner similar to tree data structures.

[1] SEMI Equipment Communication Standard 2

The C++ interface builds SECS-II message-handling member functions (subroutines) by prompting the programmer though a tree in a manner that corresponds to the SECS-II message standards in the SEMI Specification. If the programmer chooses to build the member functions directly, the tree class provides simple member functions that allow movement within the tree. The data that the programmer needs to access are simply nodes on the tree.

1. OVERVIEW

The last decade has seen a great effort within the Semiconductor industry to automate data flow to and from the equipment used in Semiconductor manufacture. This effort has been driven by the increased control and resulting product yield improvement that this automation can deliver. Two illustrative examples are:

- **The selection of recipe (processing instructions) for the equipment to process a particular manufacturing lot.** The chance of running a lot through the wrong recipe is much greater when this selection is done manually as compared to when it is done through data flow automation. Lots that are processed using the wrong recipe usually must be scrapped, resulting in the loss of thousands of dollars.

- **Data collection during processing and testing.** Automated data collection allows more data to be collected, with transcription errors eliminated. Better data enhances the ability of processing engineers to analyze and solve production problems.

Figure 1 shows a typical computer configuration for a cleanroom. Microcomputers control the data flow automation for one or several pieces of equipment. A Shop Information System (SIS), running on a minicomputer, oversees the movement of product throughout the cleanroom. The microcomputers are referred to as Automation Cell Computers. The term *Automation Cell* is used to refer to a single micro and the equipment it controls. All microcomputers are connected via TCP/IP to this Shop Information System. The micros often depend on the SIS for guidance. Equipment is connected to its controlling micro using RS-232.

TCP/IP, which stands for *Transmission Control Protocol / Internet Protocol*, is a set of computer networking protocols which allows two or more computers to communicate.

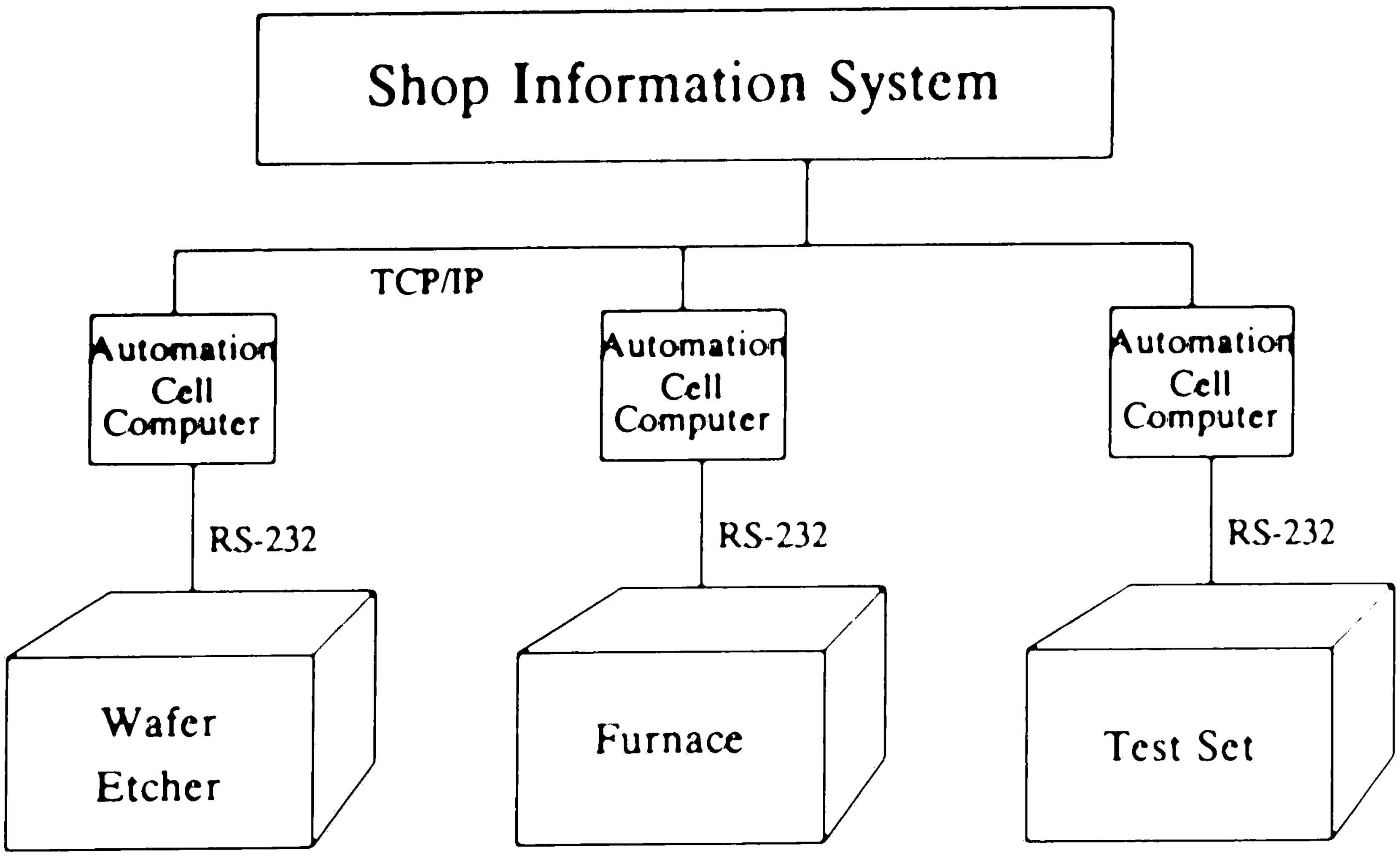
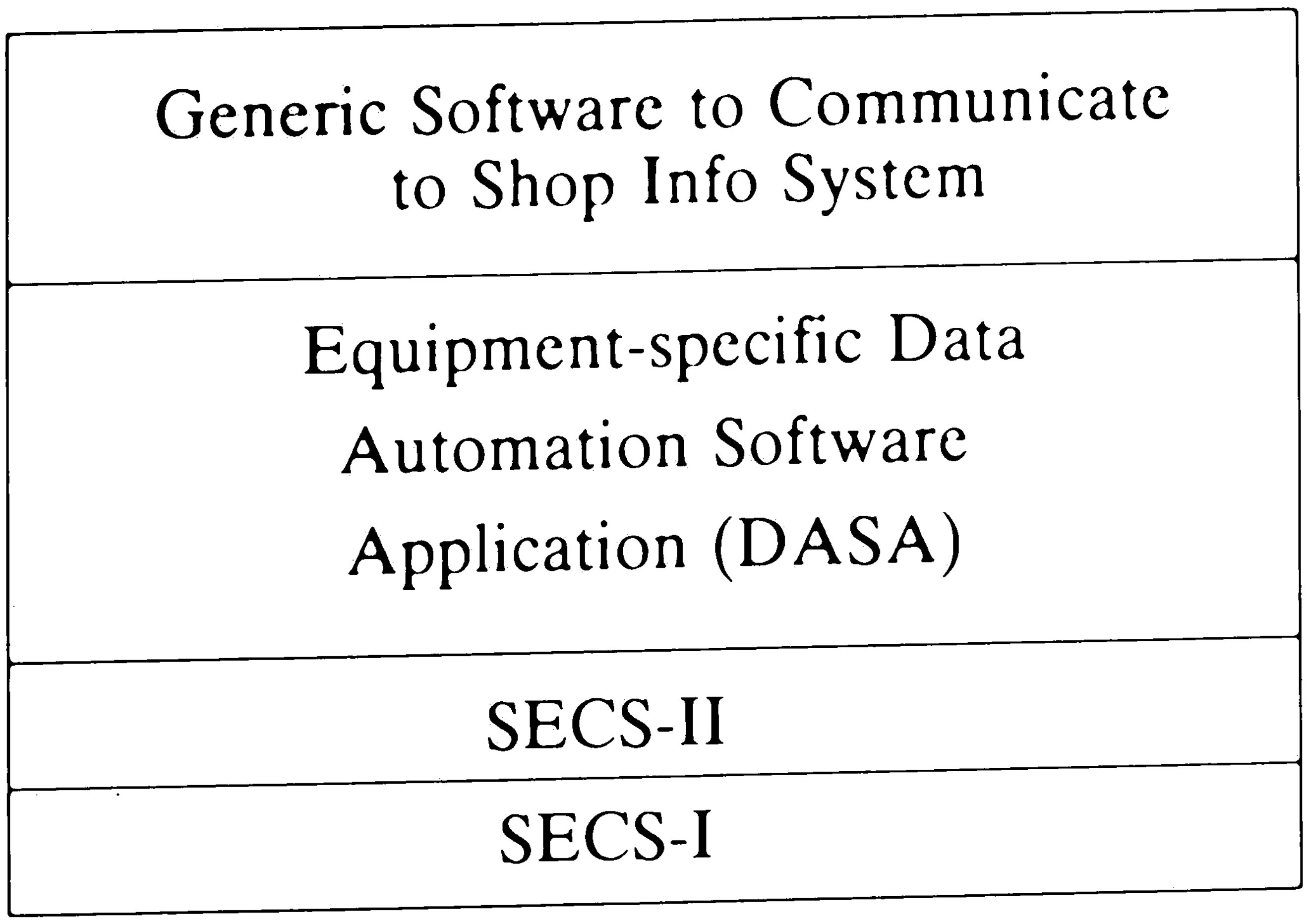


Figure 1. Semiconductor Cleanroom Computer System

1.1 The Need for SECS-I and SECS-II

A large barrier to the data automation effort was the dissimilar interfaces presented by the equipment. From the low-level communication protocol to the high-level structure of the data sent, the Data Automation Software Application (DASA) programmer could expect each new type of equipment to require a completely different software interface. SEMI, a consortium of Semiconductor manufacturers, sought to alleviate this by designing the SECS-I and SECS-II standards to make the software interface more consistent across all Semiconductor equipment. SECS stands for SEMI Equipment Communication Standard. SECS-I defines the protocol for exchange of messages between Semiconductor processing equipment and a host computer. SECS-II details the message content.

Many Semiconductor manufacturers developed their own software packages to handle the SECS standards. Data Automation Software Applications (DASA's), customized to handle the SECS-II messages for a particular machine, are built on top of these home-grown SECS interfaces. **Figure** 2 highlights the software layers found on a typical Automation Cell Computer.



Generic Software to Communicate to Shop Info System
Equipment-specific Data Automation Software Application (DASA)
SECS-II
SECS-I

Figure 2. Software Layers on Automation Cell Computer

The SECS-I, SECS-II, and SIS layers are generic, i.e., they can be used by any DASA. Typically, there is one daemon running to handle each of these tasks on a given Automation Cell. A *daemon* is a computer-wide process that is always running when the computer is up. Several different DASA's may be running on the same Automation Cell; each DASA would depend on the three daemons to provide communication to its Equipment and to the Shop Information System. The SECS-I software layer handles the low-level communication with all Equipment attached to the Automation Cell

Computer. The SECS-II layer routes incoming SECS-II messages to the correct DASA and outgoing SECS-II messages to the correct Equipment. The DASA software is designed to cull information from SECS-II messages received from its Equipment and to control its Equipment by sending SECS-II messages. The SIS layer allows a DASA to query the Shop Information System for information, such as the recipe that the manufacturing lot should use at the Equipment. The SIS layers also permits the DASA to update information on the Shop Information System, for example that the manufacturing lot is being moved from the Equipment to another piece of Equipment.

1.2 DASA Development

Due to the software layering described above, only the DASA layer must be developed for each new type of Equipment to be automated. The DASA programmer must learn the detail of the SECS-II messages the Equipment sends and expects to receive, must determine what data to cull from the messages received, must choose when and how to interact with the Equipment operator through the Equipment's display(s), and must decide how to control the Equipment through the manufacturing process. The DASA is the computer program that embodies these design decisions.

One of the most time-consuming tasks of DASA development is determining the exact structure of the SECS-II messages used by the Equipment. The exact structure must be known because the software must parse incoming SECS-II messages and build outgoing SECS-II messages. Absolute precision is required. There are two distinct phases to accomplishing this task:

1. Experiment with sending and receiving each type of SECS-II messages that the Equipment uses. Learn the structure of the messages in detail. The *SECS-II Internals* section below explains how there can be different types of SECS-II

messages.

2. Using the experimental data gathered in step 1, build subroutines to parse messages received from the Equipment and build messages to be sent to the Equipment.

The subroutines are packaged into a DASA. When the DASA has been debugged, it will run as a daemon process on the Automation Cell Computer, requiring little attention. It automatically carries out the tasks it was programmed to do: interacting with the Equipment operator in the cleanroom, storing data culled from incoming SECS-II messages, controlling Equipment activity by sending SECS-II messages, communicating with the Shop Information System.

1.3 SECS-II Internals

To understand the requirements of a SECS-II interface, some detail about the SECS-II standard itself must be understood.

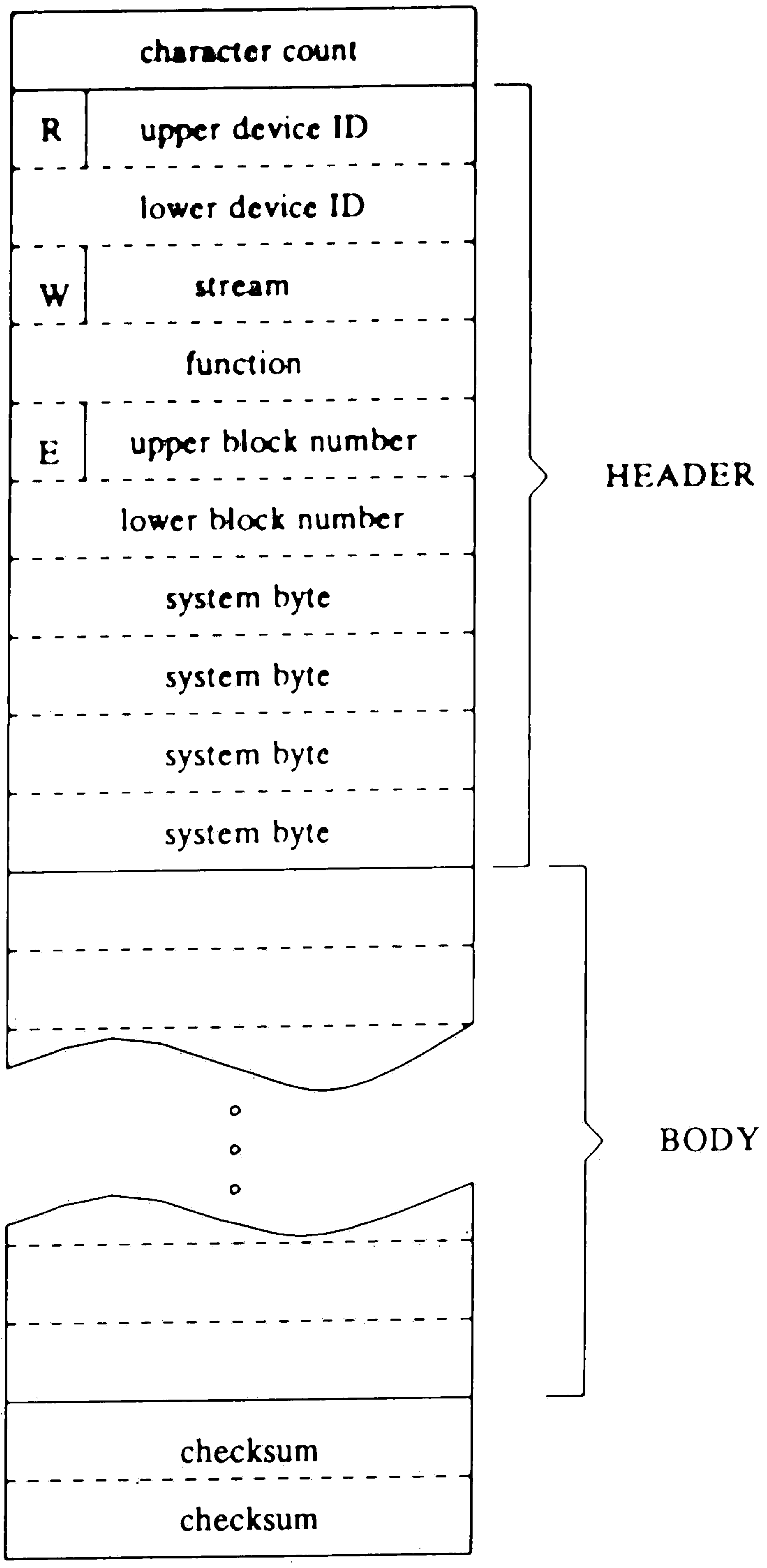


Figure 3. Byte Detail in One Block of a SECS-II Message

SECS-II messages are sent in blocks that may range in size from 13 to 256 bytes. The messages may be single-block or multi-block. **Figure 3** shows the byte structure of a typical SECS-II block, which consists of four parts:

1. **Character count**. This one-byte field contains the sum of the number of bytes in the Header and the Body. Since a SECS-II block may be variable in size, the character count is required to specify the number of bytes.

2. **Header**. This ten-byte field contains high-level information about the particular SECS-II message and block. The information within the Header is used by the SECS-II software layer to determine whether to send the block to Equipment or DASA, which particular Equipment or DASA to send it to, whether or not to set a timer to wait for a reply to the message and if the block is in the proper sequence for the SECS-II message of which it is a part.

   The information in the Header is also important to the DASA receiving the block. In particular, the Stream-Function tells the DASA what type of message it has received. The Header information includes:

   - **Reverse-Bit** (R-Bit). Indicates if the message is being sent from the Equipment or from the Host.
   - **Device ID**. A unique identifier for the Equipment receiving or sending the message.
   - **Wait-Bit (W-Bit)**. Indicates whether or not a reply to the message is expected.
   - **Stream-Function**. Categorizes information contained in the message.
   - **End-Bit (E-Bit)**. Set for only the last block in a given SECS-II message.
   - **Block Number**. SECS-II messages are sent in 256-byte blocks, with the Block Number starting at 1 and increasing by 1 for each subsequent block.

- **System Bytes**. Uniquely identify the message for a particular piece of Equipment.

3. **Body**. This field may be from 0 to 243 bytes in length. The Body is the actual data that is being sent to or from the Equipment.

4. **checksum**. This two-byte field is used to check the integrity of the transmitted SECS-II block.

SECS-II Messages are categorized by similar activities. The categories are called Streams. **Figure 4** cross-references the Stream numbers and their activities.

STREAM	ACTIVITY
Stream 1	Equipment Status
Stream 2	Equipment Control and Diagnostics
Stream 3	Material Status
Stream 4	Material Control
Stream 5	Exception Reporting
Stream 6	Data Collection
Stream 7	Process Program Management
Stream 8	Control Program Transfer
Stream 9	System Errors
Stream 10	Terminal Services
Stream 11	deleted
Stream 12	Wafer Mapping
Stream 13	Unformatted Data Set Transfers

Figure 4. SECS-II Streams

Within each Stream, Functions are defined for particular activities. With the exception of Function 0, which in every Stream closes an open conversation, like-numbered Functions specify unrelated activities from Stream to Stream.

A *Transaction* is defined as a complete interaction (or conversation) between the

Equipment and the Host computer. It may consist of one SECS-II message that does not require a reply, or a SECS-II message and the resulting reply.

A SECS-II message that initiates a conversation is called a *Primary* message. A Primary message is always an odd-numbered Function. A *Secondary* message is sent only in response to a received Primary message. Secondary messages are always even-numbered Functions. The Equipment and the Host both can send Primary and Secondary messages.

A reply to a Primary message is expected only if the Wait-Bit is set in the header(s) for the message. A Transaction can now be defined more precisely as the pairing of a Primary message with the Wait-Bit set and the returned Secondary message, or a Primary message with the Wait-Bit not set.

The sender of a Primary message with the Wait-Bit set waits a certain length of time for the corresponding Secondary message. If it does not receive a reply within the timeout period, it aborts the conversation.

The Body of a SECS-II message is a self-defining structure consisting of *Items* and *Lists*. Items are the data elements. Lists are groupings of Items which may include more Lists.

The SECS-II message structure parallels a data structure, well-known within the Computer Science community, called a *tree*. A tree is simply a grouping of elements, called *nodes*, into parent-child relationships. Nodes without children are called *leaf* nodes. As shown in **Figure 5**, a tree data structure looks much like a human's family tree, except that a child may have only one parent.

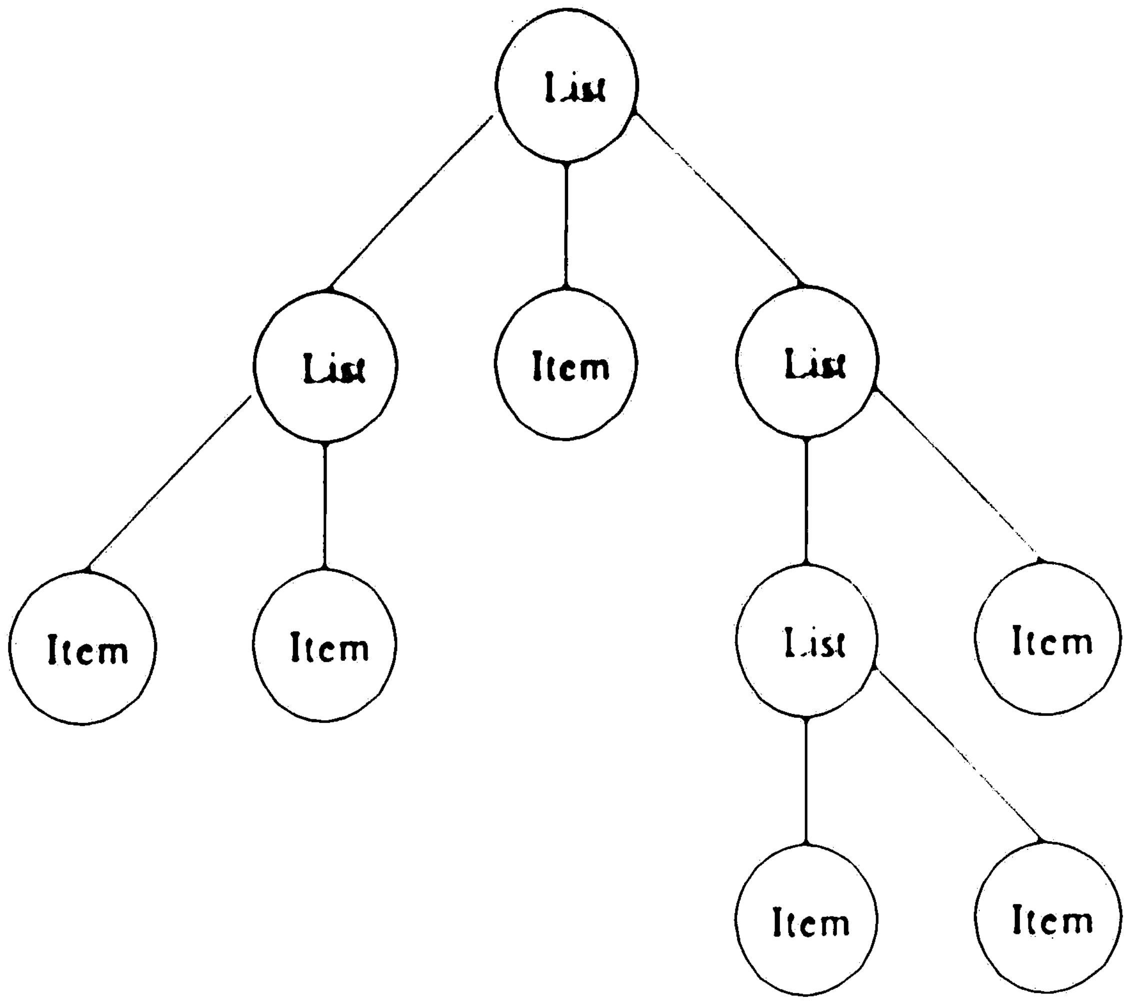


Figure 5. SECS-II Message as a Tree Data Structure

All SECS-II Items are leaf nodes on the tree. The parent nodes are all SECS-II Lists. One Stream-Function is differentiated from another by the configuration of Items and Lists in its Tree, the data type (see **Figure 6**) of each Item, and the entity which each Item represents (e.g., recipe or lot number). The SEMI SECS-II Specification details what the SECS-II Tree should look look like for over 200 Stream-Functions. A given SECS-II message may contain none of, some of, or all of the different data types shown in **Figure 6**.

SECS-II Data Type	Definition
ASCII	printable ASCII characters
BINARY	all possible 1-byte values
BOOLEAN	0 is false, != 0 is true
FLOAT4	4-byte floating point (IEEE 754)
FLOAT8	8-byte floating point (IEEE 754)
INT1	1-byte signed integer (2's complement)
INT2	2-byte signed integer (2's complement)
INT4	4-byte signed integer (2's complement)
INT8	8-byte signed integer (2's complement)
LIST	number of items in List
UINT1	1-byte unsigned integer
UINT2	2-byte unsigned integer
UINT4	4-byte unsigned integer
UINT8	8-byte unsigned integer

Figure 6. SECS-II Data Types

Integers are numbers with no fractional component, whereas floating point numbers may have fractional components. For integers in **Figure 6**, the most significant byte is sent first. For floating point, the sign bit is sent first.

As stated before, the SEMI SECS-II Specification details over 200 Stream-Functions. The messages of any piece of Equipment claiming to adhere to the SECS-II standard must conform to this SEMI Specification.

There is some flexibility in the SEMI Specification of any SECS-II Stream-Function because the Items in the specification, rather than being one of the fourteen data types listed in **Figure 6**, are members of a SECS-II Data Dictionary. This Data Dictionary, which is also part of the SEMI SECS-II Specification, specifies the acceptable SECS-II data types for 146 different Items. Lists can also provide flexibility in Stream-Function specification; variable-length lists are used when it is not known how many of a particular Item will appear in the List.

The SECS-II Specification for any particular piece of Equipment will specify the exact SECS-II data type for all SECS-II messages used by the Equipment.

1.4 SECS-II Message Example

As an example, consider Stream 6 Function 9. Known as a Formatted Variable Send, it is one of the most commonly used Stream-Functions. The structure of this message is specified in the SEMI Specification as:

```
L,4
    1. < PFCD >
    2. < DATAID >
    3. < CEID >
    4. L,n
        1. L,2
            1. < DSID(1) >
            2. L,m
                1. < DVVAL(1) >
                •
                •
                •
                m. < DVVAL(m) >
        2. L,2
            1. < DSID(2) >
            •
            •
            •
        •
        •
        •
        n. L,2
            1. < DSID(n) >
            •
            •
            •
```

Figure 7. Stream 6 Function 9 SEMI Specification

The specification in **Figure 7** can be transformed to the tree shown in **Figure 8**. The LIST at the top level of the tree (level 0) is a fixed LIST because it contains different

Items. However, the LISTs at levels 1 and 3 are variable lists because they can contain any number of like Items. The Items directly below level 1 are themselves LISTs, while the Items directly below level 3 are DVVAL. DVVAL is contained in the SEMI Specification Data Dictionary and is explained below.

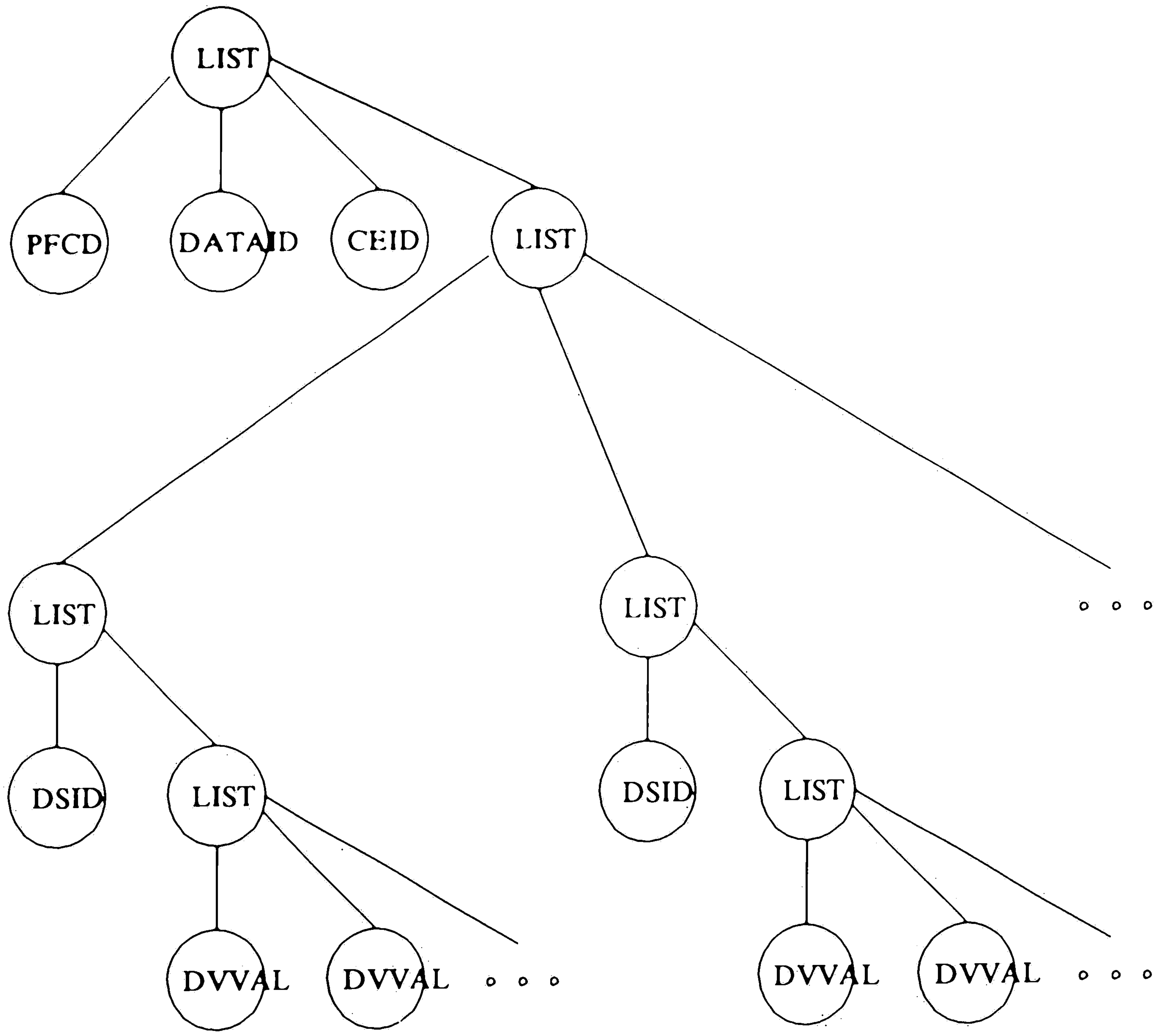


Figure 8. Stream 6 Function 9 SEMI Specification as a Tree

The Data Dictionary items in **Figure 7** and **Figure 8** are defined as follows:

- **PFCD - Predefined format code.** Used by the Equipment manufacturer to specify particular subsets of the structure of the Stream 6 Function 9 message. Valid type is BINARY.

- **DATAID - Data ID.** Valid types are INT*, UINT*.

- **CEID - Collection event ID.** Specifies the event on the Equipment that spawned this message. Valid types are INT*, UINT*.

- **DSID - Data Set ID.** Valid types are INT*, UINT*.

- **DVVAL - Data Value.** Valid types are BINARY, BOOLEAN, INT*, FLOAT*, UINT*.

1.5 The Importance of Programming Ease in a SECS-II Interface.

The flexibility of the SECS-II Stream-Function specification reduces the reusability of software designed to build and parse SECS-II messages. As an example, consider a DASA designed for a piece of Equipment which passes DVVAL in Stream 6 - Function 9 as a FLOAT4. A C language implementation would use a **float** type to store DVVAL. However, this implementation will fail for a piece of Equipment that sends DVVAL in Stream 6 - Function 9 as an INT2. Making matters more difficult is the fact that a single piece of Equipment may pass DVVAL as INT2 in one SECS-II message and as FLOAT4 in another.

The result of this flexibility is that each type of Equipment requires a custom DASA. It follows that an important goal of a SECS-II interface should be to make the programming of a DASA as easy as possible. It would be especially advantageous for a SECS-II interface to be programmable by non-programming professionals; in practice there are often many more DASA's desired than there are full-time C programmers to develop them.

The SECS-II interface currently used by AT&T requires an experienced C programmer to develop a DASA because it uses complex C data structures to represent the Lists and Items in SECS-II messages. This paper will describe a SECS-II interface which makes DASA development easier by letting the programmer view SECS-II messages as trees, providing the programmer with simple methods to move up, down, left, and right to nodes in the tree, and providing methods to easily allow the programmer to extract the value from or set the value of an Item node. Alternatively, the interface to be described can combine the heretofore distinct activities of defining the exact structure of SECS-II messages for a piece of Equipment and exploiting those definitions in software.

2. DESIGN GOALS

The overall goal of this SECS-II interface is to streamline the process that the DASA programmer must follow to produce software to interpret and build SECS-II messages for a particular piece of Equipment. This goal can be achieved by building the source code for the DASA programmer as the programmer is working to pinpoint the Equipment's implementation of the SECS-II messages.

2.1 GOAL 1: Reduce the Number of Interface Data Types

Figure 9 shows one way that the fourteen SECS-II data types could map into eleven C data types. The justification for the mapping is that the number of bytes and the interpretation of the bytes in the C types exactly match the corresponding SECS-II types. If the DASA prograr imer wants to use a database in his DASA, he may find that some of the eleven types are not supported by his database software. Even if all eleven types are supported, he may find the same data coming from two different Stream-Functions as different types, forcing him to do convert one of the types. It would be of benefit to the DASA programmer to further reduce the number of potential data types.

SECS-II Data Type	C Data Type
ASCII	char *
BINARY	unsigned char
BOOLEAN	unsigned char
LIST	n/a
INT1	char
UINT1	unsigned char
INT2	short
UINT2	unsigned short
INT4	long
UINT4	unsigned long
INT8	long[2]
UINT8	unsigned long[2]
FLOAT4	float
FLOAT8	double

Figure 9. Mapping SECS-II Data Types to C Data Types

2.2 GOAL 2: Streamline the Process of Agreeing with Equipment on Message Form

Most of the effort of developing a DASA for a new piece of Equipment involves creating subroutines that either build or parse SECS-II messages. There is a multi-step procedure that the programmer typically follows to accomplish this task:

1. Use *secsim*© or a similar program to build Primary and Secondary SECS-II messages to be sent to the Equipment. This construction is done off-line, guided by the SECS Specification that is provided with the Equipment.

2. Connect Host computer to the Equipment; for *secsim*©, the Host is a PC running DOS. To test the hand-built Secondary messages, run the Equipment to cause it to send Primary SECS-II messages to the Host. Likewise, request that *secsim*© send the customized Primary messages to the Equipment.

3. Because the SECS Specification for a given piece of Equipment may be vague, out-of-date, or even incorrect, many of the hand-built messages go through several iterations before the Equipment deems them acceptable. This process often takes several days.

4. The details of Primary and Secondary messages received from the Equipment are stored during testing of the customized messages. These must be studied to determine how to cull the desired information from the messages.

5. Subroutines can now be coded to build (or parse) the messages sent to (or received from) the Equipment. As a result of the experimentation, these subroutines should now "agree" with the Equipment.

The new SECS-II interface should strive to make the process of coming to terms with the Equipment more efficient.

2.3 GOAL 3: Make It Easier to Build SECS Messages to Send to the Equipment

Consider the C language example of building a SECS-II message, shown in **Figure 10**: it contains statements that can make C daunting for non-professionals. The message is designed to update several "equipment constants". The equipment constant ID's are stored in one array (*ecID*) and the equipment constant values are stored in another (*ecVal*).

NOTE 1 in **Figure 10** marks the declaration of the *SECSarray*, where the SECS-II message is built. It is an array of pointers to C type **char**, meaning that every piece of data stored in *SECSarray* must be a pointer to C type **char**. A pointer is a piece of data that "points to" another piece of data in the computer's memory; pointers are useful for strings of **char's** because unlike **long**, which always uses 4 bytes of computer memory, strings are of variable length. **NOTE 2** marks the definition of the top LIST in the SECS-II tree. This tree will have *numberOfConstants* LISTs below the top LIST, each containing one INT2 (equipment constant ID) and one INT4 (equipment constant value).

Since the code to initialize any of the lower LISTs is the same, it is put in a **for** loop. The statement at **NOTE 3** takes the address of the appropriate element in the *ecID* array, casts it as a **char** pointer, and puts it in the appropriate slot of the *SECSarray*. The statement at **NOTE 4** takes similar action for the *ecVal* array.

```
char                    *SECSarray[3 + (8 * NUMBER_OF_CONSTANTS)];/*** NOTE 1 ***/
short                   ecID[NUMBER_OF_CONSTANTS];
long                    ecVal[NUMBER_OF_CONSTANTS];

 .
 .
 .

/*                      Fill ecID[] & ecVal[] arrays from database*/

 .
 .
 .

SECSarray[0] = "LIST";                  /*** NOTE 2 ***/
SECSarray[1] = numberOfConstants;

for ( i = 0; i < NUMBER_OF_CONSTANTS; i++ )
                        {
                        j = i * 8;
                        SECSarray[2 + j] = "LIST";
                        SECSarray[3 + j] = "2";
                        SECSarray[4 + j] = "AINT2";
                        SECSarray[5 + j] = "1"
                        SECSarray[6 + j] = (char *) &ecID[i];/*** NOTE 3 ***/
                        SECSarray[7 + j] = "AINT4";
                        SECSarray[8 + j] = "1";
                        SECSarray[9 + j] = (char *) &ecVal[i];/*** NOTE 4 ***/
                        }

SECSarray[2 + (8 * NUMBER_OF_CONSTANTS)] = (char *) NULL:
```

Figure 10. C Code to Build SECS-II Message

The new SECS-II interface should strive to make programming SECS-II message building less complex.

2.4 GOAL 4: Make It Easier to Parse SECS Messages to Received from the Equipment

The C language SECS-II interface uses pointers directly into the SECS-II message to access items in the message. **Figure 11** shows the tree for Stream 6 Function 3. To access the targeted node in **Figure 11** would require a C statement such as shown in **Figure 12**. **Figure 11** also shows the path of the pointers which would be traversed to reach the targeted node. The C interface adds to the confusion by requiring an extra piece of data to access a leaf node: to get from the bottom-most LIST in **Figure 11** to the targeted node requires *un.dptr[1].un.bptr*.

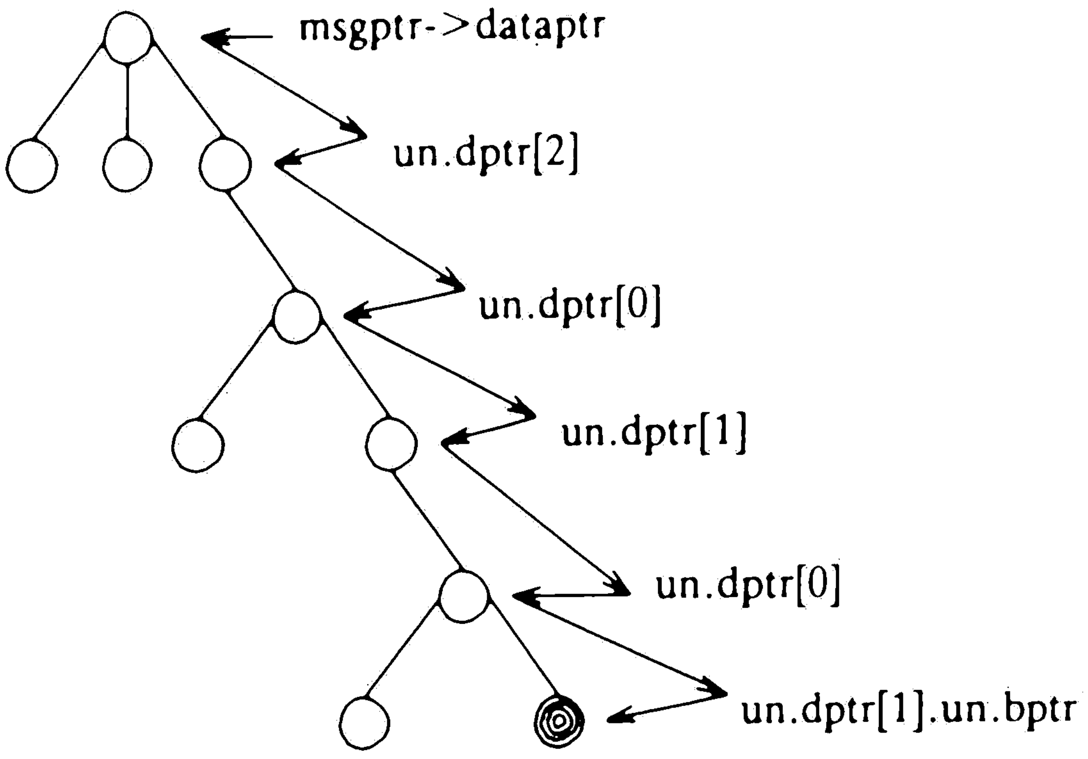


Figure 11. Tree Representation of Stream 6 Function 3 Message

```
dataValue = *msgptr->dataptr->un.dptr[2].un.dptr[0].un.dptr[1].un.dptr[0].un.dptr[1].un.bptr;
```

Figure 12. C Code to Access Target Node in Figure 11

A typical DASA must cull several pieces of data from each of over one-hundred messages that it might receive from the Equipment. Even if the DASA programmer creates tree diagrams such as shown in **Figure 11** for each of his Equipment's SECS-II messages, it is still difficult and error-prone to create the C statements to get to the desired data; the simple concept of moving from node to node in a tree does not easily translate to C statement such as that in **Figure 12**.

The new SECS-II interface will have a representation of the tree and movement within it that fits more closely to the simple concept of moving from node to node on a diagram. The extraction of data from SECS-II messages should be more understandable than described in the preceding example.

2.5 GOAL 5: Isolate Non-portable Code within SECS-II Data Types

To facilitate the porting of the SECS-II interface to different hardware platforms, it would be useful to segregate non-portable code. This provides the programmers porting the code with a small subset of the SECS-II interface that they must consider for change on the new Host. This goal benefits the DASA programmer in that it would make the SECS-II interface available more quickly to him on different Host computers.

There are two types of code within the interface that may be non-portable:

1. bit-level operations.

2. movement of data from incoming SECS-II message to native data types, and vice versa. Data is in high-order-byte to low-order-byte order in SECS-II messages. If the Host computer does not support this ordering in the data types the interface must use, there must be byte movement to translate between the two.

2.6 Summary

The new SECS-II interface strives to make the DASA programmer's job easier by presenting him with data representations that closely parallel the human view of SECS-II structures. Additionally, the interface should provide tools that make learning the Equipment and building the DASA more efficient.

3. OBJECT-ORIENTED PROGRAMMING PARADIGM

Many books have been written regarding the object-oriented approach to programming. While a comprehensive discussion is beyond the scope of this work, the following brief introduction will serve to illustrate the benefits of incorporating the paradigm into the SECS-II interface.

Object-oriented programming basically involves the packaging of data and the functions that can act on that data into bundles called *classes*. In C++, the data for a class is called the *member data* and the functions that can operate on the member data are called the *member functions*. Any particular instance of a class is called an *object*.

3.1 Example of a Class

To clarify these concepts, consider a class called DOG. Suppose that the DOG class had member data *breed*, *name*, and *dogID*, and member functions *DOG()*, *~DOG()*, *changeBreed()*, and *changeName()*. *DOG()* is a special member function, called a *constructor*, used to create a DOG object. Conversely, *~DOG()* is *destructor*, a special member function to delete a DOG object. A programmer wanting to use class DOG will be called a *client*. The DOG client would create an object of type DOG in C++ as follows:

```
DOG                    aDog( "beagle", "Peaches" );
```

The constructor for the DOG class sets *breed* equal to the first argument it receives (in this case, "beagle") and sets *name* equal to the second argument it receives ("Peaches"). Additionally, the DOG constructor sets *dogID* based on some internal algorithm for keeping a unique number for all dogs in the world.

The client that declared the DOG object *aDog* can change the object only by using the member functions provided. To change the *name*, the client would do something like:

```
aDog.changeName( "Satin" );
```

This changes the *name* to "Satin". The client has no means of changing *dogID* because he was not provided with a member function to do so; the bundling of data and functions into classes allows the class programmer to control how the client may use the class. Classes also permit the class programmer to hide the complexity of the class implementation from the client. The class programmer can in this manner provide the client with a simpler view of the class.

3.2 Class Inheritance

Another feature of Object-oriented programming is *inheritance*. A class that inherits from second class is said to be a *subclass*, or a *derived class*, of the second class. The second class may be referred to as the *superclass*. A subclass is exactly like its superclass, except that it typically has additional member data and/or member functions. Inheritance allows the building of class hierarchies in which common functionality is not duplicated, but instead serves as the basis from which subclasses are derived.

Continuing the example above, the DOG class is a subclass of the class PET. The member data *breed* and *name* might actually be declared in the PET class, and are a part of DOG simply because it is a subclass of PET. DOG adds the member data *dogID*; this member data and however DOG initializes it in its *DOG()* constructor are the only ways in which DOG differs from PET.

3.3 Object-oriented Programming and the SECS-II Interface

Object-oriented programming seemed to provide ways to achieve the goals of providing an easier-to-use interface to SECS-II. A TREE class would hide the complex details of moving about a tree from the client. A SECSELEMENT class could serve as a superclass for fourteen classes to represent the fourteen SECS-II data types; because the classes are in control of their clients usage, they could use fewer data types in their client interaction. Classes to represent each of the SECS-II messages in the SEMI Specification could be created, with each containing a tree Model of its specification; these classes would derive from a STREAMFUNCTION class that would bundle data and actions common to all Stream-Functions.

These possibilities resulted in the conception of the Object-oriented SECS-II Interface (OOSI). The next chapter discusses the class organization for the OOSI.

4. CLASS ORGANIZATION

Figure 13 highlights the class layout of the OOSI. The double circles are **Borland** C++ classes, and the single circles are classes created for the OOSI. A dashed line means *is a client of*, while a solid line translates to *is a subclass of*.

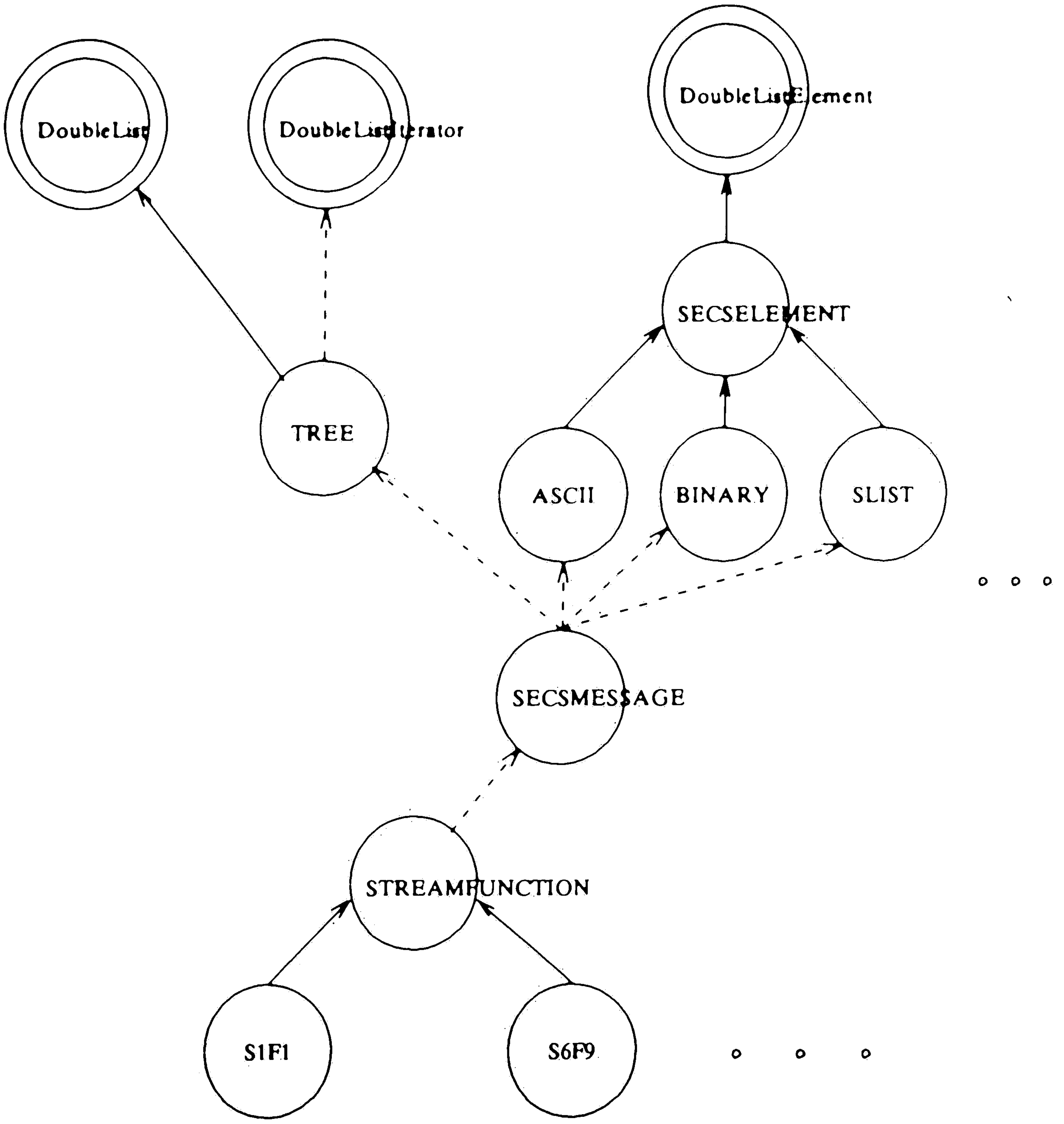


Figure 13. Object-Oriented SECS-II Interface Class Organization

Borland C++ comes with a small class library, as shown in Figure 14. The SECS-II interface was implemented to use as many of the provided classes as was feasible.

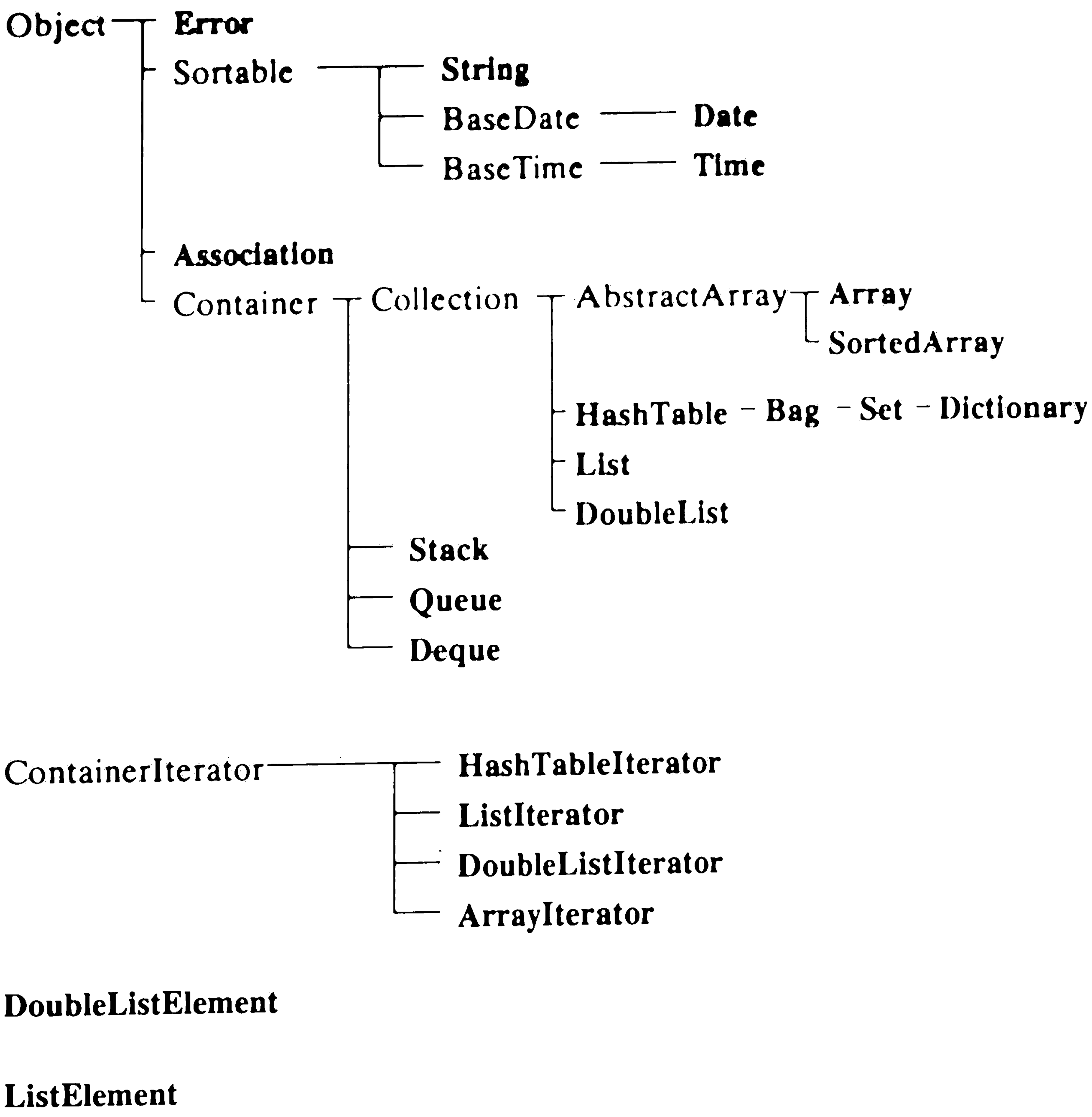


Figure 14. Borland C++ Class Library

4.1 TREE Class

A SECS-II message is classic example of a tree data structure. Because all programming activity involved with building, parsing, and moving around within SECS-II messages will involve the TREE class, it is the most important class required by the OOSI.

The TREE class is a generic class in that it would be useful to any application requiring a TREE, not just this OOSI application. An object of type TREE actually consists of just the parent node and its children nodes. The children, which are also TREE objects, are stored on a **Borland** DoubleList object.

Multi-level super-TREE's are formed by adding a TREE with children as a child to another TREE. Most SECS-II messages are represented as super-TREE's.

4.2 SECSELEMENT Classes

The SECSELEMENT class is an abstract class, meaning that no objects of type SECSELEMENT may be declared. The SECSELEMENT class serves as a base class from which fourteen classes, one for each SECS-II item type, are derived.

Thirteen of the fourteen SECSELEMENT subclasses are named identically to the SECS-II item types that they represent; the class for the LIST item type are named SLIST (SECS LIST) to avoid conflict with the **Borland C++ Class Library**.

Having a separate class to represent each of the SECS-II item types allows encapsulation of the storage details and conversion methods within these classes. This achieves the design goal of the isolation of non-portable code.

4.3 SECSMESSAGE Class

The SECSMESSAGE is a client of both the TREE class and the SECSELEMENT classes. A class is a client of second class when it declares and uses objects of the second classes type. A SECSMESSAGE object is created in one of two ways:

- a SECS-II message is received from the Equipment, and the byte stream is passed to a SECSMESSAGE constructor. The constructor gleans the header information from the SECS-II message and builds a super-TREE with the appropriate SECSELEMENT's at the nodes.

- a DASA subroutine has built a super-TREE to be sent to the Equipment as a SECS-II message. The subroutine requests the creation of a SECSMESSAGE; the appropriate SECSMESSAGE constructor converts the super-TREE into a byte stream, complete with SECS-II headers, ready to be sent to the Equipment.

4.4 STREAM-FUNCTION Classes

The STREAMFUNCTION class is the base class for classes designed to handle specific Stream-Function SECS-II messages. However, STREAMFUNCTION is not an abstract class; objects of type STREAMFUNCTION may be declared.

There are five basic functions of any object of type STREAMFUNCTION, or derived from STREAMFUNCTION:

- update the DASA based on a Primary SECS-II message received from the Equipment.

- update the DASA based on a Secondary SECS-II message received from the Equipment.

- send a Primary SECS-II message to the Equipment.

- send a Secondary SECS-II message to the Equipment.

- assist the user to build the Member Functions to accomplish any of the above.

Specific Stream-Function classes, e.g., S6F9 class, exist only for Primary messages. Activities for a Secondary message are handled by the class named for the Primary message. For example, a received Stream 6 Function 10 SECS-II message would be handled by the S6F9 class.

A production version of the OOSI would require over one hundred specific Stream-Function classes. This thesis includes a selected subset of these for illustration purposes. When the user is building an Equipment specific DASA, he will be building custom Stream-Function classes such as the S6F9C class.

5. INTERACTIVE MEMBER FUNCTION BUILDING WITH THE OOSI

This chapter illustrates how the OOSI allows the DASA programmer to combine the previously distinct activities of learning the SECS-II message structure specific to the Equipment and using this knowledge to construct software routines that will parse or build these SECS-II messages during manufacturing. Three design goals are achieved by the OOSI / DASA programmer interaction that will be discussed.

Member Functions to parse SECS-II messages from the Equipment are built by prompting the user through messages as they are received from the Equipment. Since the Member Function is built directly from the Equipment's version of any SECS-II message, agreement on the message structure is achieved immediately.

This direct building technique is not possible for messages that will be sent to the Equipment. However, to make the offline process easier, the user is prompted through the SEMI Specification for a given message and asked to tailor it to Equipment's version of the message. It is envisioned that the user would have the Equipment's SECS-II specification at hand while OOSI prompts them through the SEMI message. The end result would be Member Functions designed to build and send SECS-II messages to the Equipment. In cases where the Equipment's specification was incorrect, the user would have to repeat the build procedure until the Equipment accepts the message built by the Member Function.

5.1 Customizing Member Functions to Parse Messages from Equipment

The interaction in **Figure 15** would occur when the DASA programmer wants to see the detail of a Stream 6 Function 9 sent from the Equipment to the Host. A PC running the OOSI would be communicating with the Equipment. When the DASA programmer hits the RUN button on the Equipment, a Stream 6 Function 9 message is sent to the PC and the session in Figure 15 begins. The DASA programmer's input is in **bold**.

The interaction occurs on a line-by-line basis such that the DASA programmer sees only to the most recent prompt (••>). When the programmer enters the requested information, the next prompt is displayed. Note that the style of the left side of the prompts look very similar to the style of the SEMI Specification; it is also similar to the SECS-II specification for the Equipment. Inside the brackets are the SECSELEMENT object type (SECS-II data type) and the value(s) for the object. *<LIST 4>* indicates a LIST with 4 Items. *<BINARY 6>* is a BINARY object with a single value of 6. Note the indentation as the super-TREE is descended.

The two numbers separated by a colon (e.g. *4:1*) indicate the level of the object within the super-TREE and the child number within its level, respectively. *0:0* indicates the root of the super-TREE. The variable names entered by the DASA programmer become the variable names to store data values in the Member Function being built. The interactive session to build a Member Function to parse a Stream 6 Function 9 message from the Equipment would appear as:

```
BUILDING Member Function to update DASA based on
                              Stream 6 - Function 9 message received from Equipment.


When prompted by '==> ' :
                              - enter Variable Name to save data, OR
                              - enter 'n' to NOT save data.


<LIST 4>                      0:0
 <BINARY 6>                   1:1 ==> formatCode
 <INT2 1>                     1:2 ==> dataID
 <INT2 2>                     1:3 ==> collectionID
 <LIST 1>                     1:4
  <LIST 2>                    2:1
   <UINT1 0>                  3:1 ==> n
   <LIST 1>                   3:2
                              <ASCII 'RUN      '>4:1 ==> buttonName
```

Figure 15. Sample Session to Build S6-F9 Member Function

The tree diagram for a Stream 6 Function 9 message is shown in **Figure 16.**

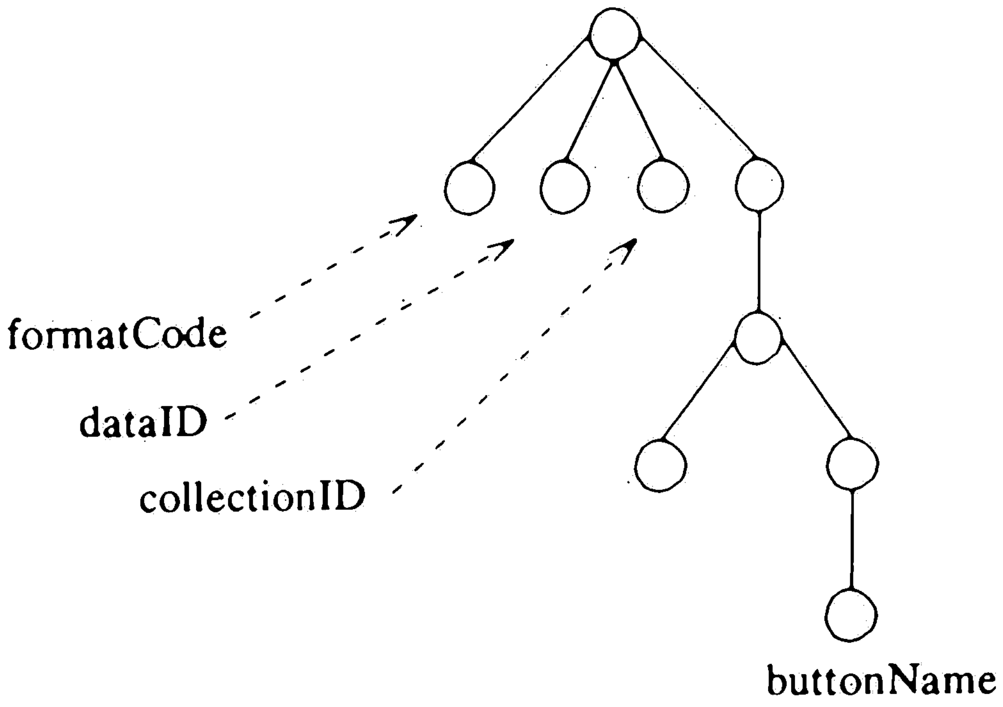


Figure 16. Stream 6 Function 9 Tree with Variable Names

Figure 17 shows the code generated by the **Figure** 15 session with the DASA programmer. Note that the programmer-specified variable names of *formatCode* and *dataID* are declared as long types, even though they were BINARY and INT2, respectively, in the received SECS-II message. The SECSELEMENT subclasses allow reduction in the number of data types that the DASA programmer must deal with.

The *TREE_PTR* = *primary->returnTree()* statement returns the root of the Primary message super-TREE. *MOVE_DOWN* is a macro to move the current node to the first (left-most) child. The current node is now the place within the Stream 6 Function 9 message where *formatCode* resides, so the value is extracted with *SEPTR->returnLong()*.

The remaining statements visit every node in the Stream 6 Function 9 TREE, extracting data where the DASA programmer requested, and finally, moving the current node back up to the root of the super-TREE. The code is indented to reflect movement down and up the super-TREE. The code generated would be:

```
//                     primaryUpdate() DECLARATION
                       void                 primaryUpdate();

//                     primaryUpdate() DEFINITION
void S6F9C::primaryUpdate()                 {

                       TREE                 *TREE_PTR;
                       long                 formatCode;
                       long                 dataID;
                       long                 collectionID;
                       char                 buttonName[11];


                       TREE_PTR = primary->returnTree();

                       MOVE_DOWN;
                        formatCode = SEPTR->returnLong();
                        MOVE_RIGHT( 1 );
                        dataID = SEPTR->returnLong();
                        MOVE_RIGHT( 1 );
                        collectionID = SEPTR->returnLong();
                        MOVE_RIGHT( 1 );
                        MOVE_DOWN;
                         MOVE_DOWN;
                          MOVE_RIGHT( 1 );
                          MOVE_DOWN;
                           strncpy( buttonName, SEPTR->returnString(), 10 );
                          MOVE_UP;
                         MOVE_UP;
                        MOVE_UP;
                       MOVE_UP;
};
```

Figure 17. Code Generated by S6-F9 Session

5.2 Customizing Member Functions to Send Messages to Equipment

Stream-Function classes of the OOSI (e.g., S2F23) provide Member Functions to build Member Functions for their customized subclasses (e.g., S2F23C). For example, the *buildSendPrimary()* Member Function of the S2F23 class will build a customized *sendPrimary()* Member Function for the S2F23C class. Each Stream-Function class has built-in TREE Models of the SEMI Specification for the Primary message and the Secondary message, keeping in mind that a class named for a Primary message is also responsible for the associated Secondary message (S2F23 class would update the DASA based on a Stream 2 Function 24 message received from the Equipment). The Stream-Function classes allow only valid Member Functions to be built, e.g., a Stream 6 Function 9 cannot be sent from the Host to the Equipment, so class S6F9 will generate an error message rather than build the *sendPrimary()* Member Function for the S6F9C class.

By definition, a given SECS-II message for the Equipment must be a subset of the same message in the SEMI Specification. **Figure 18** shows how the DASA programmer is prompted through the SEMI Stream 2 Function 23 tree to tailor the *sendPrimary()* Member Function to his Equipment's Stream 2 Function 23. The programmer's input is in **bold**. As before, the DASA programmer sees only to the most recent prompt (==>); the next prompt appears after the programmer has responded to the current prompt. The left side of the prompt uses the same basic style as the SEMI Specification, but there are differences owing to the fact that the Model Tree is based on the flexible SEMI Data Dictionary. In the second prompt in **Figure 18** , the *<long>* represents the TRID from the Data Dictionary. TRID may be of SECS-II type INT1, INT2, INT4, UINT1, UINT2, or UINT4. The SECSELEMENT classes for each of these types use a **long** for client access or update. The *<long>* prompt reminds the programmer that the SECS-II type entered must be a type that "trades in" **long**. The SECS-II Specification for the Equipment will specify precisely which SECS-II type the Equipment expects.

The second entry for non-LIST prompts is the number of occurrences of the SECS-II data type that are required at the current node. Again, the exact number required is given in the Equipment's SECS-II Specification. The last programmer entry for non-LIST prompts is the variable name that the programmer wishes to use to represent the current node in the Member Function being built.

The LIST prompts deserve special attention; there are two varieties, and each expects a single reply. In **Figure 18** , the *<LIST 5>* prompt signifies a fixed LIST, which means that for all Equipment this LIST will have 5 children, with the exception that for some Equipment the tree can be pruned at this point by having 0 children. The *<list>* prompt is for a variable LIST; the children of such a LIST will all be of the same SECS-II type. The DASA programmer must simply respond by indicating how many children will be in the LIST.

```
BUILDING Member Function to send
                        Stream 2 - Function 23 message to Equipment.


When prompted by '==> ' :
                        - for LIST, enter number of Items in LIST,
                        - for Items, enter <type number variableName>,
                          e.g., INT2 1 waferID.


<LIST 5>                0:0 Items in LIST ==> 5
  <long>                1:1 ==> INT2 1 traceID
  <string>              1:2 ==> ASCII 6 samplePeriod
  <long>                1:3 ==> INT4 1 totalSamples
  <long>                1:4 ==> INT2 1 reportGrpSize
  <list>                1:5 Items in LIST ==> 8
    <long>              2:1 ==> INT2 1 statusVarID
```

Figure 18. Sample Session to Build S2-F23 Member Function

The tree diagram for a Stream 2 Function 23 message is shown in **Figure 19.**

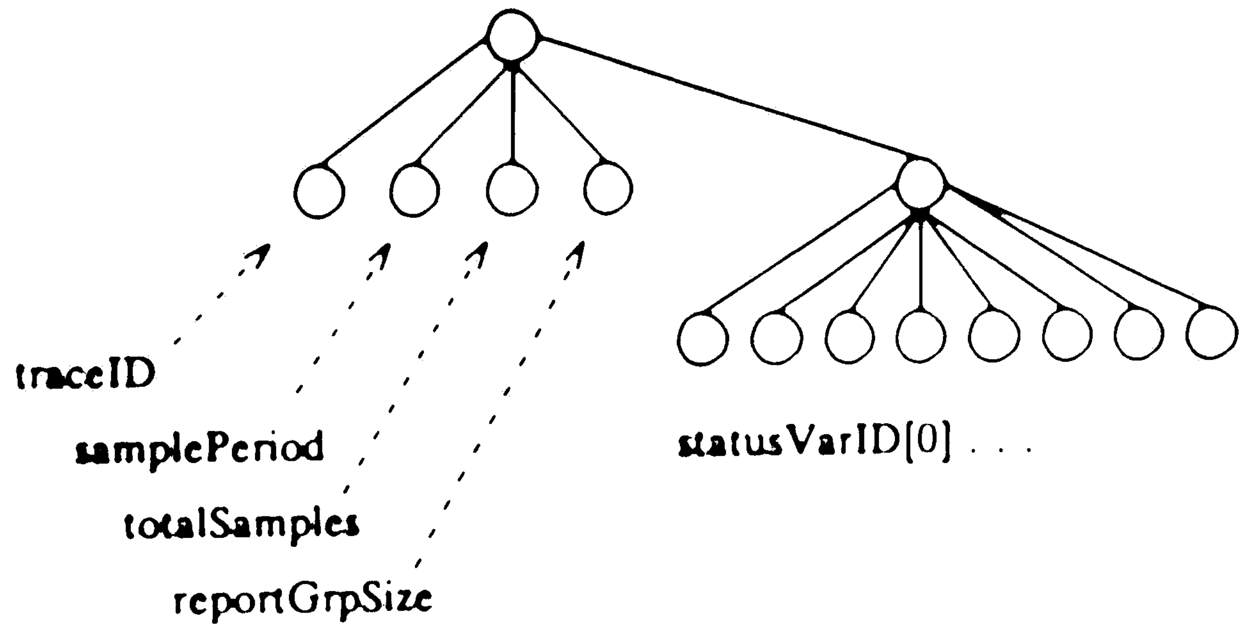


Figure 19. Stream 2 Function 23 Tree with Variable Names

Figure 20 shows the code generated from the session in **Figure 18.** Note that the variable names match the names provided by the DASA programmer. The statement *TREE_ROOT = TREE_PTR = new TREE* creates a TREE onto which the outgoing message will be built. *TREE_ROOT* will remain at the root node of the super-TREE, while *TREE_PTR* will be current node, moving around the super-TREE as it is built. *ADD_LIST_AND_MOVEDOWN(5L)* adds the LIST with 5 children at the root node and moves the current node to this LIST. Whenever a LIST is created, it becomes the current node. *ADD_INT2(1L, traceID)* adds an INT2 object, initialized with *traceID,* as the first (left-most) child of the LIST above. This new node does not become the current node, only LISTs will be current nodes when a TREE is being built.

The statement *TREE *TREE_MARK = TREE_PTR* saves the current node prior to entering a **for** loop that initializes the children of the variable LIST. Immediately inside this **for** loop, *TREE_PTR* is restored. In the example of **Figure 20,** this serves no purpose, but if there were nested variable LISTs it would ensure that all LISTs were built at the correct level in the TREE. Each of the eight INT2 objects are initialized and added to the super-TREE within the **for** loop.

The statement *primary = new SECSMESSAGE(...)* creates a SECSMESSAGE object from the super-TREE that was built. *primary->sendMessage()* sends the message to the Equipment.

```
//                      sendPrimary() DECLARATION
                        void                  sendPrimary();

//                      sendPrimary() DEFINITION
void S2F23C::sendPrimary()                    {

                        TREE                  *TREE_ROOT, *TREE_PTR;
                        long                  traceID[1];
                        char                  samplePeriod[7];
                        long                  totalSamples[1];
                        long                  reportGrpSize[1];
                        long                  statusVarID[8];


                        TREE_ROOT = TREE_PTR = new TREE;

                        ADD_LIST_AND_MOVEDOWN( 5L );
                         ADD_INT2( 1L, traceID );
                         ADD_ASCII( 6L, samplePeriod );
                         ADD_INT4( 1L, totalSamples );
                         ADD_INT2( 1L, reportGrpSize );
                         ADD_LIST_AND_MOVEDOWN( 8L );
                         TREE                 *TREE_MARK = TREE_PTR;
                         for ( int i = 0; i < 8; i++ ){
                         TREE_PTR = TREE_MARK;
                          ADD_INT2( 1L, &statusVarID[i] );
                         }

                        primary = new SECSMESSAGE( 2, 23, REPLY, TREE_ROOT );

                        primary->sendMessage();
};
```

Figure 20. Code Generated by S2-F23 Session

6. TREE CLASS

A TREE object is actually just a two-level tree, with a parent at level zero and the children at level one. Clients of TREE will often be dealing with super-TREE's, which starts with a TREE that has at least one child that is also a TREE, etc. The number of levels of a super-TREE is limited only by available memory. A super-TREE has one root node, which is the only node of the super-TREE with no parent. All nodes of a super-TREE are occupied by TREE's; leaf nodes simply have no children. To allow movement from node to node in the super-TREE, the client will use a TREE pointer instead of a TREE.

6.1 Member Data

TREE is a subclass of **Borland's** DoubleList class; the children of a TREE will be stored on this DoubleList. All children of a TREE must themselves be TREE's. Each TREE has a pointer to **Borland's** Object type. Object is the base class for all of **Borland's** C++ **Class Library**. This is where the *node* for the TREE is stored. A TREE also has a pointer to another TREE, which points to the *parent* of the TREE; in the case of the root node of a super-TREE it will be NULL.

The *childCnt* of a TREE is the number of children the TREE has. *childCursor* is child that is currently being added or examined. When a TREE is being built, *childCnt* is typically initialized to the number of children expected. As children are added, *childCursor* is increased and compared to *childCnt* to determine if the TREE has been completed. *level* is set to the TREE's level in a super-TREE, with zero being the root level.

6.2 Constructors

TREE has two constructors: *TREE()* to create an empty TREE and *TREE(Object& Node)* to create a TREE with the passed Object.

The creation of an empty TREE is necessary because the *addChild()* Member Function is the only way to add a node to the TREE but unfortunately, a TREE Member Function cannot be employed until a TREE object already exists.

6.3 Node-Adding Member Functions

The *addChild()* Member Function is the only way to add nodes to a TREE. If *addChild* notices that the root of the TREE is empty, it grafts the TREE passed to it to the root. A TREE pointer is returned by *addChild()*. If the child added is a potential parent itself, a pointer to this child is returned, otherwise a pointer to the "current" TREE is returned. This method of construction results in a super-TREE being built in prefix order.

6.4 Movement Member Functions

Four Member Functions are provided to allow movement within a super-TREE: *moveDown()*, *moveUp()*, *moveRight(X)*, and *moveLeft(X)*.

If the current node of the super-TREE has children, *moveDown()* will move down to (move the current node to) the left-most child, otherwise the current node will not change. Moving the current node to the left-most child is consistent with a prefix-order traversal of a super-TREE.

If the current node of the super-TREE is not the root node, *moveUp()* will move up to (move the current node to) the parent, otherwise the current node will not change.

moveRight(X) will move to the right (move the current node to the right) *X* nodes if possible. If the request is not possible, the current node remains unchanged.

moveLeft(X) functions in a manner similar to *moveRight(X)*, except that the movement is to the left.

6.5 Boolean Member Functions

atRoot(), *mayBeParent()*, *hasChildren()*, *hasRightSibling()*, and *hasLeftSibling()* are Member Functions that answer TRUE-or-FALSE questions about the current node of a TREE or super-TREE.

atRoot() determines if the current node is the root of a TREE or a super-TREE. It is used to determine whether or not *moveUp()* is possible from the current node.

mayBeParent() determines if the current node is a potential parent. This is set by clients of TREE by their use of the *addParentType()* Member Function. *mayBeParent()* is used so that a TREE can differentiate between objects being added to nodes that may have children and those that may not.

hasChildren() returns true if the node has children. It is used to determine whether or not a *moveDown()* will be possible.

hasRightSibling() returns TRUE if the current node has a sibling to the right. Similarly, *hasLeftSibling()* returns TRUE if the current node has a sibling to the left. These Member Functions are used to check the validity of a right or left move within a TREE before attempting the move.

7. SECSELEMENT CLASSES

Objects to be stored on a TREE are stored on a DoubleList, of which TREE is a subclass. An object must be a subclass of **Borland's** type Object to be stored on **Borland's** type DoubleList. Since SECSELEMENT objects will be stored on a TREE, it follows that the class SECSELEMENT must be a subclass of **Borland's** Object class. The SECSELEMENT class itself is an *abstract class*; no objects of this class can exist, instead the class serves as a common base for its subclasses. Some Member Functions are generic enough that they can be defined in SECSELEMENT. In other cases *pure virtual functions* are declared in SECSELEMENT, which are Member Functions that cannot be defined in the abstract class, but must be declared in its subclasses.

One of the major accomplishments of the SECSELEMENT classes is to reduce the total number of C types that the DASA programmer must handle to three: **double**, **long**, and (**char** *). The manner in which the SECS-II data types are mapped, or "flattened", into these three types is shown in **Figure 21**. A SECSELEMENT subclass converts to or from its flattened type only when sending data to or receiving data from a client. The representation within the class is consistent with the SECS-II data type, e.g., an INT2 object stores its data as **short**, not **long**.

SECS-II Data Type	C Data Type	Flattened C Type
ASCII	char *	char *
BINARY	unsigned char	long
BOOLEAN	unsigned char	long
LIST	n/a	long
INT1	char	long
UINT1	unsigned char	long
INT2	short	long
UINT2	unsigned short	long
INT4	long	long
UINT4	unsigned long	double
INT8	long[2]	double
UINT8	unsigned long[2]	double
FLOAT4	float	double
FLOAT8	double	double

Figure 21. Mapping SECS-II Data Types to OOSI Types

7.1 SECSELEMENT Member Data

className will be the same as the corresponding SECS-II data type, e.g., the INT2 class will house SECS-II INT2's. *classTypeValue* is an internal identifier for the SECSELEMENT classes, used as a means of determining if two SECSELEMENT objects are of the same type. *itemCode* is given in the SECS-II Specification. Each Item in the Body of a SECS-II message must begin with an Item Code to identify what type of Item it is. The number of bytes in the Item is stored as *length,* and *remainingBytes* are the number of bytes in the Item that the client has not yet accessed, given that the object is being accessed.

7.2 SECSELEMENT Constructor

The single *SECSELEMENT()* constructor simply sets the *className* and *classTypeValue*. Every SECSELEMENT subclass constructor calls the SECSELEMENT constructor after its own to initialize these values.

7.3 Identification Member Functions

isA() and *nameOf()* are the two class identification Member Functions.

isA() returns the *classTypeValue* so that two SECSELEMENT objects may be compared to see if they are of the same type. *nameOf()* returns a string containing the *className*. The name of the class is the same as the *className*, except for class SLIST, which has the *className* of LIST.

7.4 Member Functions to Build SECS-II Message

getItemCode(), *getItemHeader()*, *lengthInMessage()*, and *stringForMessage()* are Member Functions used only by SECSELEMENT and its subclasses to build SECS-II messages.

getItemCode() returns the SECS-II *itemCode* for the object. The object uses this internally to build the byte string to be put into an under-construction SECS-II message. *getItemHeader()* builds and returns just the header portion of the byte string for the object. This header portion consists of the *itemCode* and length bytes for the object. *lengthInMessage()* returns a count of the number of bytes that the byte string will require in the under-construction SECS-II message. *stringForMessage()* returns the byte string for the object, destined for a SECS-II message.

7.5 Boolean Member Functions

hasDouble(), *hasLong()*, and *hasString()* return a true or false depending on whether or not the SECSELEMENT object can return the type requested. These are useful for cases where it is not clear what type of object may be at a node of a SECS-II message tree when the DASA programmer is building Member Functions manually.

7.6 Data Access Member Functions

returnDouble(), *returnLong()*, or *returnString()* return the "next" data value for the object. For a given SECSELEMENT subclass only one of these will actually return a value, e.g., *returnDouble()* will return a value for an FLOAT4 object, but *returnLong()* and *returnString()* will not. Each object can have zero, one, or more occurrences of data within it; there is a pointer that starts at the first occurrence and is moved to the next occurrence each time one of these Member Functions is called. *itemsRemaining()* returns the number of the data occurrences remaining in the object. *bytesPerItem()* returns the number of bytes used by each occurrence of SECS-II data.

7.7 SECSELEMENT Subclasses

The fourteen subclasses of SECSELEMENT are: ASCII, BINARY, BOOLEAN, FLOAT4, FLOAT8, INT1, INT2, INT4, INT8, SLIST, UINT1, UINT2, UINT4, and UINT8. All subclasses have a constructor to create an object from an SECS-II byte stream from the Equipment, for example *INT1(unsigned long Length, unsigned char *Int1)*. Except for ASCII and SLIST, all subclasses have an additional constructor to create an object from a user-defined array, for example *INT1(unsigned long Length, long DataArray[])*. The single ASCII and SLIST constructors handle both types of object creation.

With the exception of ASCII and SLIST, the SECSELEMENT subclasses each have two additional pieces of Member Data: *data* and *ptr*. Each is a pointer to a C language type of size appropriate to store the Item type from a SECS-II message, for example *data* and *ptr* would be of type **(char *)** for the INT1 class. The *data* pointer will always point to the beginning of the object's Items in the computer's memory, while the *ptr* pointer will point to the object's "current" Item. As an example, consider a INT1 object with 5

Items; if *hasLong()* has been called twice, the "current" Item that *ptr* points to will be the third Item.

Each of the SECSELEMENT subclasses override the *bytesPerItem()* and *stringForMessage()* Member Functions of the SECSELEMENT class. Additionally, each of the subclasses will override one of the *hasDouble()* / *returnDouble()*, *hasLong()* / *returnLong()*, or *hasString()* / *returnString()* Member Function pairs, depending on the appropriate "flattened" type for the subclass.

8. SECSMESSAGE CLASS

SECS-II messages from the Equipment or to the Equipment are represented as SECSMESSAGE objects.

8.1 Member Data

messageFlat is a pointer to the SECS-II message as it would appear upon receipt from the Equipment or just prior to delivery to the Equipment. *messageLength* is the number of bytes in *messageFlat*. *messageTree* is a pointer to the TREE version of *messageFlat*. SECS-II messages received from the Equipment will be stored initially as *messageFlat*, then converted to *messageTree* for parsing. SECS-II messages to be sent to the Equipment will be built as *messageTree*, then converted to *messageFlat* to be sent. There is also Member Data to store each piece of data for the SECS-II Header, such as Stream, Function, System Bytes, etc.

8.2 Constructors

There are three constructors for the SECSMESSAGE class: *SECSMESSAGE(unsigned char *MessageIn, short Length)* and *SECSMESSAGE(char *FileName)* to create a SECSMESSAGE object from an incoming SECS-II message, and *SECSMESSAGE(short Stream, short Function, int ReplyRequired, TREE *TreePtr)* to create a SECSMESSAGE object to send out to the Equipment.

The *SECSMESSAGE(unsigned char *MessageIn, short Length)* constructor expects to receive a byte stream from the SECS-I interface. The *SECSMESSAGE(char *FileName)* constructor expects the name of a file that contains a binary copy of an actual SECS-II message. The OOSI has never communicated to piece of Equipment; the latter constructor was used for all incoming SECS-II messages. *parseMessage()* is called by both incoming SECS-II message constructors to transform the incoming byte stream at *messageFlat* to a TREE at *messageTree*.

The *SECSMESSAGE(short Stream, short Function, int ReplyRequired, TREE *TreePtr)* constructor for outgoing SECS-II messages expects to receive Stream, Function, R-Bit, and a populated super-TREE.

8.3 Member Functions for Internal Use by SECSMESSAGE

The *buildMessage()*, *addItemToMessage()*, *expandMessage()*, and *addSECSheader()* Member Functions are used only by the SECSMESSAGE class, to build SECS-II messages.

buildMessage() transforms the TREE at *messageTree* to a byte stream at *messageFlat*, in preparation for sending a SECS-II message to the Equipment. *addItemToMessage()* adds a single SECSELEMENT Item to an under-construction SECS-II message. *expandMessage()* dynamically allocates memory as required for an under-construction SECS-II message. *addSECSHeader()* breaks the Header-less SECS-II message into 256-byte blocks, putting the Header on each block.

8.4 Member Functions for SECSMESSAGE Client Use

The *sendMessage()*, *getSECSstream()*, *getSECSfunction()*, and *isPrimary()* Member Functions are available for use by clients of the SECSMESSAGE class.

sendMessage() calls *buildMessage()*, then sends the SECS-II message to the Equipment. For this thesis, however, the message was written to a file. *returnTree()* returns a pointer to *messageTree* so that clients of SECSMESSAGE may traverse the TREE, accessing and updating nodes as desired. *getSECSstream()* returns the Stream of the SECSMESSAGE object. *getSECSfunction()* returns the Function of the

SECSMESSAGE object. *isPrimary()* returns true if the SECSMESSAGE object is a Primary message, and returns false otherwise.

9. STREAM-FUNCTION CLASSES

The STREAMFUNCTION class serves as a base for all of the specific Stream-Function classes, such as S1F1, S2F23, S6F9, etc. A specific Stream-Function class (such as S1F1) may serve as a base for a customized Stream-Function class (such as S1F1C) should the DASA programmer choose to build the customized class. Figure 22 is an example showing which Member Functions within classes STREAMFUNCTION and S1F1 would be used to build the customized S1F1C class.

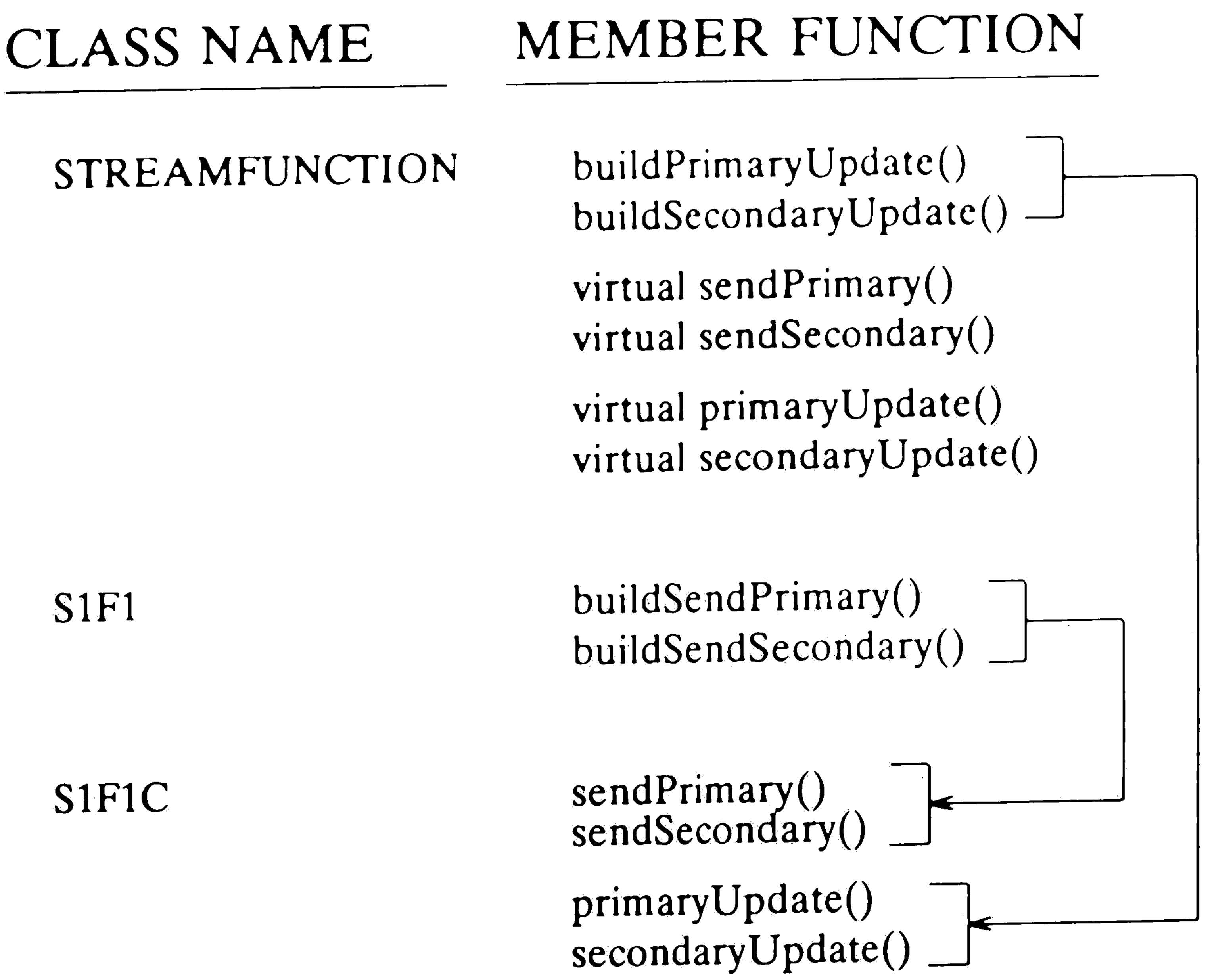


Figure 22. Member Functions that Build Customized S1F1C Class

Member Functions *sendPrimary()*, *sendSecondary()*, *primaryUpdate()*, and *secondaryUpdate()* are declared in STREAMFUNCTION as *virtual*, meaning that they must be defined in STREAMFUNCTION and they may be *overridden* in subclasses of STREAMFUNCTION. A subclass overrides a Member Function of a superclass by redefining what the Member Function does; the purpose of the redefinition is to more closely tailor the Member Function to the needs of the subclass.

In a customized Stream-Function class, these four Member Functions carry out the following activities:

- *sendPrimary()* - build a Primary SECS-II message and send it to the Equipment.
- *sendSecondary()* - in response to a Primary message received from the Equipment, build a Secondary SECS-II message and send it to the Equipment.
- *primaryUpdate()* - cull information from a Primary message received from the Equipment, update the DASA, and if a reply is required, call the *sendSecondary()* Member Function.
- *secondaryUpdate()* - cull information from a Secondary message received from the Equipment and update the DASA.

Because the activities will vary depending on the specific Stream-Function, these four Member Functions are defined to do nothing in STREAMFUNCTION, except that *sendSecondary()* builds and returns a Function 0 of the appropriate Stream to terminate the conversation with the Equipment.

The *buildPrimaryUpdate()* and *buildSecondaryUpdate()* Member Functions are used to build the *primaryUpdate()* and *secondaryUpdate()* Member Functions for a customized Stream-Function class. *buildPrimaryUpdate()* and *buildSecondaryUpdate()* may be defined at the general STREAMFUNCTION class level because they are driven by

SECS-II messages received from the Equipment; the specific structure detail is contained in those messages.

The *buildSendPrimary()* and *buildSendSecondary()* Member Functions are used to build the *sendPrimary()* and *sendSecondary()* Member Functions for a customized Stream-Function class. *buildSendPrimary()* and *buildSendSecondary()* must be defined at the specific Stream-Function class level because the user-prompting is based upon a TREE Model that differs from one Stream-Function to another.

9.1 STREAMFUNCTION Member Data

primary is a pointer to a SECMESSAGE; it is where the Primary SECS-II message will be stored. *secondary* is where the SECS-II Secondary message response to the Primary will be saved. *secondary* is also a pointer to SECSMESSAGE.

Indent is a **char** string that is globally available within the STREAMFUNCTION class for indentation of SECSELEMENT prompts during interaction with the DASA programmer. To keep the same "look and feel" as within the SEMI Specification, the prompt indents are increased when going down a TREE and decreased when going up a TREE. *IndentCnt* and *IndentInc* are used to keep track of the current indentation and the amount to increase indentation when going up or down a TREE level, respectively.

classFile, *declFile*, and *codeFile* are pointers to files where the customized Member Functions are built and stored.

9.2 STREAMFUNCTION Constructors

STREAMFUNCTION() creates an object with *primary* and *secondary* pointing to NULL. *STREAMFUNCTION(SECSMESSAGE *SECSmess)* creates an object with either *primary* or *secondary* pointing to the passed SECSMESSAGE, depending on whether the SECSMESSAGE is a Primary or Secondary.

9.3 Member Functions for Internal Use by STREAMFUNCTION

The *buildUpdate()*, *updateTreeWalk()*, *updatePromptUser()*, *buildSend()*, *sendTreeWalk()*, and *sendPromptUser()*, *indentMore()*, and *indentLess()* Member Functions are used only by STREAMFUNCTION objects.

buildUpdate() is called by *buildPrimaryUpdate()* and *buildSecondaryUpdate()*. *buildUpdate()* calls *updateTreeWalk()*, passing a pointer to the super-TREE version of the received SECS-II message as an argument. *updateTreeWalk()* is a *recursive* Member Function that visits each node on the super-TREE. *updateTreeWalk()* calls *updatePromptUser()* at each super-TREE node to prompt the DASA programmer for action.

In a fashion similar to the above, *buildSendPrimary()* and *buildSendSecondary()* call *buildSend()*, which calls *sendTreeWalk()*, which calls *sendPromptUser()*. The result is that the DASA programmer is prompted through a Model TREE (defined at and passed as an argument from a specific Stream-Function class), and a *sendPrimary()* or *sendSecondary()* Member Function is built for a custom Stream-Function class.

indentMore() and *indentLess()* are used to change the indenting level of the prompts to the DASA programmer as the super-TREE is descended or ascended.

9.4 Specific Stream-Function Classes

For the sake of illustration, the OOSI contains three specific Stream-Function classes: S1F1, S2F23, and S6F9. A marketable OOSI would have to include specific Stream-Function classes for all possible Stream-Functions.

9.5 Customized Stream-Function Classes

The three customized Stream-Function classes were built for the OOSI: S1F1C, S2F23C, and S6F9C. No customized Stream-Function classes would be delivered with a real-world OOSI, these classes are to be built by the DASA programmer.

10. CONCLUSION

The purpose of the Object-Oriented SECS-II Interface (OOSI) presented in this work is to ease the development of software to interpret and build SECS-II messages for a given piece of semiconductor Equipment.

The OOSI supplies a framework on which to build Equipment-specific Data Automation Software Applications (DASA's), and also provides tools that build Member Functions tailored to handle the Equipment's SECS-II messages. The tailoring process is accomplished by leading the DASA programmer through every SECS-II message that the Equipment may send or receive, and prompting for specifics regarding each part of the message.

Future enhancements of the OOSI might include: further simplification of the message-tailoring process by use of a graphical user interface and a mouse, integration of database commands into the tailored Member Functions as a means for storage and retrieval of the SECS-II data, and a high-level State-Event class as an alternative to the current *if-else* or *case* method of DASA design.

BIBLIOGRAPHY

Public Documents

SEMI Equipment Automation / Software Standards. Semiconductor Equipment and Materials International, 1990.

APPENDIX A: Source Code for the Object-oriented SECS-II Interface

This appendix contains printouts of the source code for the Object-oriented SECS-II Interface (OOSI). The printouts are ordered as follows:

- **Example mains**. Named *main.cpp*, *main1.cpp*, *main2.cpp*, and *main3.cpp*, these are short example programs that use the OOSI classes. The comments withing each program explain the function of the program.

- **General header files**. Files *custom.h*, *dasamac.h*, *messages.h*, *secstype.h* are general header files used by many of the OOSI classes.

- **Class files**. Each class consists of two files: a declaration file (e.g., *ascii.h*), and a definition file (e.g., *ascii.cpp*). The class files are alphabetized by class name.

```
/*
**              This version of main is a mini-DASA.
**
**              Two type of Primary messages are expected from the
**              Equipment: S1-F1 and S6-F9. Specific Secondary
**              messages are sent in response to these. Generic S?-F0
**              Secondaries are sent in response to all other Primary
**              messages.
**
**              This DASA sends a S2-F23 to the Equipment when it
**              gets a S1-F1. This STREAMFUNCTION is put on a LIST,
**              awaiting the Secondary reply from the Equipment. If
**              the reply comes, the S2F23C object is removed from
**              the LIST. When the user quits out of this DASA, all
**              "orphaned" Primary messages sent to the Equipment are
**              reported.
**
**              @(#)main.cpp    13.15   11/24/91
*/

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <process.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secselem.h>
#include <messages.h>
#include <list.h>
```

```
/*
**              This version of main is a mini-DASA.
**
**              Two type of Primary messages are expected from the
**              Equipment: S1-F1 and S6-F9. Specific Secondary
**              messages are sent in response to these. Generic S?-F0
**              Secondaries are sent in response to all other Primary
**              messages.
**
**              This DASA sends a S2-F23 to the Equipment when it
**              gets a S1-F1. This STREAMFUNCTION is put on a LIST,
**              awaiting the Secondary reply from the Equipment. If
**              the reply comes, the S2F23C object is removed from
**              the LIST. When the user quits out of this DASA, all
**              "orphaned" Primary messages sent to the Equipment are
**              reported.
**
**              @(#)main.cpp    13.15   11/24/91
*/

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <process.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secselem.h>
#include <messages.h>
#include <list.h>
```

```
#include <lstelem.h>


void
main()
{
        SECSMESSAGE     *Incoming, *Outgoing;
        STREAMFUNCTION  *streamFunction, *sfSaved;

        /*
        **      LIST to store Primary messages sent to Equipment.
        **      Secondary messages from the Equipment will be compared
        **      to this LIST: if their Primary is not on the LIST they
        **      will be thrown away, if their Primary is on the LIST
        **      they will update the DASA & the STREAMFUNCTION will be
        **      removed from the LIST.
        */
        List            sfList;
        ListIterator    sfIterator = (ListIterator &) sfList.initIterator();
        short           stream, function;
        char            fileName[80];


        while( 1 )      {
            /*
            **  Prompt user for incoming SECS-II message.
            */
            cout << "\nEnter filename for incoming SECS-II message" <<
                " ('q' to quit) ==> " << flush;
            cin >> fileName;

            if ( ( strlen( fileName ) == 1 &&
                ( strncmp( "q", fileName, 1 ) == 0 ) ) )        {
                    break;
        }
        else  {
          /*
          **    If the user did not type 'q' for QUIT,
          **    deal with incoming SECS-II message in a
          **    manner depending on what Stream-Function it is.
          */
          Incoming = new SECSMESSAGE( fileName );

          stream = Incoming->getSECSstream();
          function = Incoming->getSECSfunction();

          if ( Incoming->isPrimary() )    {
            /*
            **  Primary messages received from Equipment.
            */
            if ( stream == 1 && function == 1 ) {
              /*
              **    Create S1F1C object, send the
              **    Secondary message, delete the
              **    STREAMFUNCTION.
              */
              streamFunction = new S1F1C( Incoming );
              streamFunction->sendSecondary();
              delete streamFunction;
              /*
              **    Create S2F23C object, send it to the
              **    Equipment, and add it to the LIST of
              **    outstanding Primary messages.
              */
              Outgoing = new SECSMESSAGE( "_s2f23" );
              streamFunction = new S2F23C( Outgoing );
              streamFunction->sendPrimary();
              sfList.add( (Object &) *streamFunction );
            }
            else if ( stream == 6 && function == 9 )     {
                /*
                **      Create S6F9C object, send the
                **      Secondary message, delete the
                **      STREAMFUNCTION.
                */
                streamFunction = new S6F9C( Incoming );
                streamFunction->sendSecondary();
                delete streamFunction;
            }
            else        {
                /*
                **      Create generic STREAMFUNCTION object,
                **      send the Secondary message, delete the
                **      STREAMFUNCTION.
                */
                streamFunction = new STREAMFUNCTION( Incoming );
                streamFunction->sendSecondary();
                delete streamFunction;
            }
        }
        else    {
            /*
            **  Secondary messages received from Equipment.
            **
            **  Look for the matching Primary message on the
            **  LIST.
            */
            sfIterator = (ListIterator &)sfList.initIterator();
            while( sfIterator ) {
                sfSaved = (STREAMFUNCTION *)&((Object &)(sfIterator));
                if ( sfSaved->getStream() == stream &&
                    sfSaved->getSecondaryFunction() == function )       {
```

```
                                /*
                                **      The matching Primary is found;
                                **      update the DASA & remove the
                                **      STREAMFUNCTION from the LIST.
                                */
                                sfSaved->secondaryUpdate();
                                sfList.destroy( (Object &) *sfSaved );
                                break;
                        }
                        else    {
                            /*
                            **  This STREAMFUNCTION did not match;
                            **  go to the next item on the LIST.
                            */
                            sfIterator++;
                        }
                    }
                }
            }
        }

        /*
        **      Print out "orphaned" Primary messages still on the
        **      List of SECS-II messages:
        */
        sfIterator = (ListIterator &)sfList.initIterator();
        while( sfIterator )     {
            sfSaved = (STREAMFUNCTION *)&((Object &)(sfIterator));
            sfSaved->printOn( cout );
            sfIterator++;
        }

        exit( 0 );
}
```

```
/*
**              This version of main simply accepts incoming SECS-II
**              messages and prints out their structure and contents.
**
**              @(#)main1.cpp   13.1    11/24/91
*/

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <process.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secselem.h>


void
main()
{
        char            fileName[80];
        SECSMESSAGE     *Incoming;


        while( 1 )      {
            /*
            **  Prompt user for incoming SECS-II message.
            */
            cout << "\nEnter filename for incoming SECS-II message" <<
                " ('q' to quit) ==> " << flush;
            cin >> fileName;

            if ( ( strlen( fileName ) == 1 &&
                ( strncmp( "q", fileName, 1 ) == 0 ) ) )        {
                    break;
            }
            else        {
                /*
                **      If the user did not type 'q' for QUIT,
                **      create a new SECSMESSAGE object, print it
                **      to COUT, then delete the object.
                */
                Incoming = new SECSMESSAGE( fileName );
                Incoming->printOn( cout );
                delete Incoming;
            }
        }

        exit( 0 );

}
```

```
/*
**              This version of main accepts incoming SECS-II messages
**              and prompts the user to build the appropriate update
**              Member Function.
**
**              @(#)main2.cpp   13.1    11/24/91
*/

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <process.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secselem.h>
#include <strfunc.h>


void
main()
{
        char            fileName[80];
        SECSMESSAGE     *Incoming;
        STREAMFUNCTION  *streamFunction;


        while( 1 )      {
            /*
            **  Prompt user for incoming SECS-II message.
```

```
*/
cout << "\nEnter filename for incoming SECS-II message" <<
    " ('q' to quit) ==> " << flush;
cin >> fileName;

if ( ( strlen( fileName ) == 1 &&
    ( strncmp( "q", fileName, 1 ) == 0 ) ) )        {
        break;
}
else        {
    /*
    **      If the user did not type 'q' for QUIT,
    **      create a new SECSMESSAGE object, create a
    **      STREAMFUNCTION object from it, then call
    **      the appropriate "build*Update() Member
    **      Function to prompt the user to build a
    **      Member Function.
    */
    Incoming = new SECSMESSAGE( fileName );
    streamFunction = new STREAMFUNCTION( Incoming );

    if ( Incoming->isPrimary() )    {
        /*
        **  Send S?-F0 so to end conversation with
        **  the Equipment.
        */
        streamFunction->sendSecondary();
        streamFunction->buildPrimaryUpdate();
    }
    else    {
        streamFunction->buildSecondaryUpdate();
    }
    delete streamFunction;
}
```

```
        }

        exit( 0 );

}
```

```
/*
**              This version of main prompts the user to:
**                      - buildSendPrimary() for S1F1 & S2F23, and
**                      - buildSendSecondary() for S1F1 & S6F9.
**
**              For a full-fledged production system, the user
**              would have to be prompted for all Stream-Functions
**              that the Host could possibly send to the Equipment.
**
**              @(#)main3.cpp   13.1    11/24/91
*/

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <process.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secselem.h>
#include <messages.h>


void
main()
{
        STREAMFUNCTION  *streamFunction;


        streamFunction = new S2F23();
        streamFunction->buildSendPrimary();
        delete streamFunction;

        streamFunction = new S1F1();
        streamFunction->buildSendPrimary();
        streamFunction->buildSendSecondary();
        delete streamFunction;

        streamFunction = new S6F9();
        streamFunction->buildSendSecondary();
        delete streamFunction;

        exit( 0 );

}
```

```
/*
**              @(#)custom.h    13.5    9/19/91
*/

#include <clstypes.h>

//#define                secsElemClass   _firstUserClass
#define         treeClass       255
#define         secsElemClass   (treeClass+1)
#define         asciiClass      (secsElemClass+1)
#define         binaryClass     (asciiClass+1)
#define         booleanClass    (binaryClass+1)
#define         float4Class     (booleanClass+1)
#define         float8Class     (float4Class+1)
#define         int1Class       (float8Class+1)
#define         int2Class       (int1Class+1)
#define         int4Class       (int2Class+1)
#define         int8Class       (int4Class+1)
#define         slistClass      (int8Class+1)
#define         uint1Class      (slistClass+1)
#define         uint2Class      (uint1Class+1)
#define         uint4Class      (uint2Class+1)
#define         uint8Class      (uint4Class+1)

#define SIZEFILE        80
#define SECSTESTDIR     "e:\ame\secstest\"

enum    truthFlag       { FALSE, TRUE };

#define ASCII_ITEM      020
#define BINARY_ITEM     010
#define BOOLEAN_ITEM    011
#define INT1_ITEM       031
#define INT2_ITEM       032
```

```
#define INT4_ITEM       034
#define INT8_ITEM       030
#define FLOAT4_ITEM     044
#define FLOAT8_ITEM     040
#define LIST_ITEM       000
#define UINT1_ITEM      051
#define UINT2_ITEM      052
#define UINT4_ITEM      054
#define UINT8_ITEM      050

#define ASCII_BYTES     1
#define BINARY_BYTES    1
#define BOOLEAN_BYTES   1
#define INT1_BYTES      1
#define INT2_BYTES      2
#define INT4_BYTES      4
#define INT8_BYTES      8
#define FLOAT4_BYTES    4
#define FLOAT8_BYTES    8
#define LIST_BYTES      0
#define UINT1_BYTES     1
#define UINT2_BYTES     2
#define UINT4_BYTES     4
#define UINT8_BYTES     8
```

```
/*
**      Macros needed by customized S-F Classes.
**
**      @(#)dasamac.h   13.2    11/12/91
*/


#define TREE_PTR          treePtr
#define TREE_ROOT         treeRoot
#define TREE_MARK         treeMark



#define SEPTR   ((SECSELEMENT *) TREE_PTR->nodeAddr())



#define MOVE_DOWN         TREE_PTR = TREE_PTR->moveDown()
#define MOVE_UP           TREE_PTR = TREE_PTR->moveUp()
#define MOVE_RIGHT(x)     TREE_PTR = TREE_PTR->moveRight(x)
#define MOVE_LEFT(x)      TREE_PTR = TREE_PTR->moveLeft(x)



#define ADD_CHILD(x)      TREE_PTR = TREE_PTR->addChild(x)
#define ADD_NODE(x)       *new TREE(x)

#define ADD_ASCIIX(x,y) *new ASCII(x,y)
#define ADD_ASCII(x,y)  ADD_CHILD( ADD_NODE( ADD_ASCIIX(x,y)))

#define ADD_BINARYX(x,y)        *new BINARY(x,y)
#define ADD_BINARY(x,y) ADD_CHILD( ADD_NODE( ADD_BINARYX(x,y)))

#define ADD_BOOLEANX(x,y)       *new BOOLEAN(x,y)
```

```
#define ADD_BOOLEAN(x,y)        ADD_CHILD( ADD_NODE( ADD_BOOLEANX(x,y)))

#define ADD_FLOAT4X(x,y)        *new FLOAT4(x,y)
#define ADD_FLOAT4(x,y) ADD_CHILD( ADD_NODE( ADD_FLOAT4X(x,y)))

#define ADD_FLOAT8X(x,y)        *new FLOAT8(x,y)
#define ADD_FLOAT8(x,y) ADD_CHILD( ADD_NODE( ADD_FLOAT8X(x,y)))

#define ADD_INT1X(x,y)  *new INT1(x,y)
#define ADD_INT1(x,y)   ADD_CHILD( ADD_NODE( ADD_INT1X(x,y)))

#define ADD_INT2X(x,y)  *new INT2(x,y)
#define ADD_INT2(x,y)   ADD_CHILD( ADD_NODE( ADD_INT2X(x,y)))

#define ADD_INT4X(x,y)  *new INT4(x,y)
#define ADD_INT4(x,y)   ADD_CHILD( ADD_NODE( ADD_INT4X(x,y)))

#define ADD_INT8X(x,y)  *new INT8(x,y)
#define ADD_INT8(x,y)   ADD_CHILD( ADD_NODE( ADD_INT8X(x,y)))

#define ADD_UINT1X(x,y) *new UINT1(x,y)
#define ADD_UINT1(x,y)  ADD_CHILD( ADD_NODE( ADD_UINT1X(x,y)))

#define ADD_UINT2X(x,y) *new UINT2(x,y)
#define ADD_UINT2(x,y)  ADD_CHILD( ADD_NODE( ADD_UINT2X(x,y)))

#define ADD_UINT4X(x,y) *new UINT4(x,y)
#define ADD_UINT4(x,y)  ADD_CHILD( ADD_NODE( ADD_UINT4X(x,y)))

#define ADD_UINT8X(x,y) *new UINT8(x,y)
#define ADD_UINT8(x,y)  ADD_CHILD( ADD_NODE( ADD_UINT8X(x,y)))

#define ADD_LIST(x)     *new SLIST(x)
#define SET_CNT(x)      TREE_PTR->setChildCnt(x)
```

```
#define ADD_LIST_AND_MOVEDOWN(x)        ADD_CHILD( ADD_NODE( ADD_LIST(x)));SET_CNT(x)
```

```
/*
**      Contains all includes necessary for SECSMESSAGE client Classes.
**
**      @(#)messages.h  13.6    11/24/91
*/
#include <strfunc.h>
#include <s1f1.h>
#include <s1f1c.h>
#include <s2f23.h>
#include <s2f23c.h>
#include <s6f9.h>
#include <s6f9c.h>
```

```
/*
**      Contains all includes necessary for SECSELEMENT Class and its
**      subclasses.
**
**      @(#)secstype.h  13.1    8/26/91
*/
#include <secselem.h>
#include <ascii.h>
#include <binary.h>
#include <boolean.h>
#include <float4.h>
#include <float8.h>
#include <int1.h>
#include <int2.h>
#include <int4.h>
#include <int8.h>
#include <slist.h>
#include <uint1.h>
#include <uint2.h>
#include <uint4.h>
#include <uint8.h>
```

```
/*
**      class:  ASCII   -       DECLARATIONS
**
**              @(#)ascii.h     13.4    11/12/91
*/

class ASCII : public SECSELEMENT        {
protected:
        char    *data;
public:
        ASCII( unsigned long Length, char *Ascii );
        ~ASCII();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasString();
        char            *returnString();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  ASCII   -       MEMBER FUNCTIONS
**
**              @(#)ascii.cpp   13.6    11/12/91
*/

#include <iostream.h>
#include <string.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <ascii.h>

ASCII::ASCII( unsigned long Length, char *Ascii ) :
    SECSELEMENT( "ASCII", asciiClass )  {
        length = remainingBytes = Length;
        data = new char[length+1];
        strncpy( data, Ascii, length );
        data[length] = '\0';
        itemCode = ASCII_ITEM;
};

ASCII::~ASCII() {
        delete data;
};

short ASCII::bytesPerItem()     {
        return ASCII_BYTES;
};

unsigned char *ASCII::stringForMessage()         {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        strncpy( (char *) ptrOut, data, length );

        return ptrSave;
};

int ASCII::hasString()  {
        return 1;
};

char *ASCII::returnString()     {
        return data;
};

void ASCII::printOn( ostream& outputStream ) const      {
        outputStream << '<' << this->nameOf() << " '" << data << "'>" << flush;
};
```

```
/*
**      class:  BINARY  -       DECLARATIONS
**
**              @(#)binary.h    13.6    11/12/91
*/

class BINARY : public SECSELEMENT       {
protected:
        unsigned char   *data;
        unsigned char   *ptr;
public:
        BINARY( unsigned long Length, unsigned char *Binary );
        BINARY( unsigned long Items, long DataArray[] );
        ~BINARY();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  BINARY  -       MEMBER FUNCTIONS
**
**              @(#)binary.cpp  13.8    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <binary.h>

BINARY::BINARY( unsigned long Length, unsigned char *Binary ) :
    SECSELEMENT( "BINARY", binaryClass )        {
        length = remainingBytes = Length;
        data = ptr = new unsigned char[length];
        itemCode = BINARY_ITEM;

        memcpy( data, Binary, length );
};

BINARY::BINARY( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "BINARY", binaryClass )        {
        int     i;

        length = remainingBytes = Items * BINARY_BYTES;
        data = ptr = new unsigned char[length];
        itemCode = BINARY_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = (unsigned char) DataArray[i];
        }
};

BINARY::~BINARY()       {
        delete data;
};

short BINARY::bytesPerItem()    {
        return BINARY_BYTES;
};

unsigned char *BINARY::stringForMessage()       {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        for ( i = 0; i < length; i++ )  {
            *ptrOut++ = data[i];
        }

        return ptrSave;
};

int BINARY::hasLong()   {
        remainingBytes = length;
        ptr = data;
        return 1;
};

long BINARY::returnLong()       {
        if ( remainingBytes <= 0 )      {
            cerr << "BINARY: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            long        tmp = (long) *ptr;
            ptr += BINARY_BYTES;
            remainingBytes -= BINARY_BYTES;
            return tmp;
        }
};

void BINARY::printOn( ostream& outputStream ) const     {
        int     i, perLine = 0;

        outputStream << '<' << this->nameOf();
        for ( i = 0; i < length; i++, perLine++ )       {
                outputStream.width( 3 );
                outputStream.fill( ' ' );
                outputStream.setf( ios::right, ios::adjustfield );
                outputStream << hex << (int) data[i];
                if ( perLine >= 16 )    {
                    perLine = 0;
                    outputStream << endl;
                }
        }
        outputStream << ">" << flush;
};
```

⌘

```
/*
**      class:  BOOLEAN -       DECLARATIONS
**
**              @(#)boolean.h   13.4    11/12/91
*/

class BOOLEAN : public SECSELEMENT      {
protected:
        unsigned char   *data;
        unsigned char   *ptr;
public:
        BOOLEAN( unsigned long Length, unsigned char *Boolean );
        BOOLEAN( unsigned long Items, long DataArray[] );
        ~BOOLEAN();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  BOOLEAN -       MEMBER FUNCTIONS
**
**              @(#)boolean.cpp 13.6    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <boolean.h>

BOOLEAN::BOOLEAN( unsigned long Length, unsigned char *Boolean ) :
    SECSELEMENT( "BOOLEAN", booleanClass )      {
        int     i;

        length = remainingBytes = Length;
        data = ptr = new unsigned char[length];
        itemCode = BOOLEAN_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = *Boolean++;
        }
};

BOOLEAN::BOOLEAN( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "BOOLEAN", booleanClass )      {
        int     i;

        length = remainingBytes = Items * BOOLEAN_BYTES;
        data = ptr = new unsigned char[length];
        itemCode = BOOLEAN_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = (unsigned char) DataArray[i];
        }
};

BOOLEAN::~BOOLEAN()     {
        delete data;
};

short BOOLEAN::bytesPerItem()   {
        return BOOLEAN_BYTES;
};

unsigned char *BOOLEAN::stringForMessage()      {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        for ( i = 0; i < length; i++ )  {
            *ptrOut++ = data[i];
        }

        return ptrSave;
};

int BOOLEAN::hasLong()  {
        remainingBytes = length;
        ptr = data;
        return 1;
};

long BOOLEAN::returnLong()      {
        if ( remainingBytes <= 0 )      {
            cerr << "BOOLEAN: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            long        tmp = (long) *ptr;
            ptr += BOOLEAN_BYTES;
            remainingBytes -= BOOLEAN_BYTES;
            return tmp;
        }
};

void BOOLEAN::printOn( ostream& outputStream ) const    {
        int     i, perLine = 0;
        char    booleanValue;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < length; i++, perLine++ )       {
            booleanValue = ( (int) data[i] == 0 ) ? 'F' : 'T';
            outputStream << " " << booleanValue;
            if ( perLine >= 30 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  FLOAT4  -       DECLARATIONS
**
**              @(#)float4.h    13.5    11/12/91
*/

class FLOAT4 : public SECSELEMENT       {
protected:
        float   *data;
        float   *ptr;
public:
        FLOAT4( unsigned long Length, unsigned char *Float4 );
        FLOAT4( unsigned long Items, double DataArray[] );
        ~FLOAT4();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasDouble();
        double          returnDouble();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  FLOAT4  -       MEMBER FUNCTIONS
**
**              @(#)float4.cpp  13.8    11/12/91
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <float4.h>

FLOAT4::FLOAT4( unsigned long Length, unsigned char *Float4 ) :
    SECSELEMENT( "FLOAT4", float4Class )        {
        short   i, j, k, items = Length / FLOAT4_BYTES;
        unsigned char   *reverseFloat;

        length = remainingBytes = Length;
        data = ptr = new float[items];
        itemCode = FLOAT4_ITEM;

        for ( i = 0; i < items; i++ )   {
            /*
            **  Point to byte after end of current FLOAT4 in
            **  input byte stream. We'll get bytes in reverse
            **  order from the byte stream.
            */
            Float4 += sizeof(float);
            reverseFloat = (unsigned char *) &data[i];
            for ( j = 0; j < sizeof(float); j++ )       {
                *reverseFloat++ = *--Float4;
            }
            /*
            **  Point to byte after end of current FLOAT4 in
            **  input byte stream.
            */
            Float4 += sizeof(float);
        }
};

FLOAT4::FLOAT4( unsigned long Items, double DataArray[] ) :
    SECSELEMENT( "FLOAT4", float4Class )        {
        short   i;

        length = remainingBytes = Items * FLOAT4_BYTES;
        data = ptr = new float[Items];
        itemCode = FLOAT4_ITEM;

        for ( i = 0; i < Items; i++ )    {
            data[i] = (float) DataArray[i];
        }
};

FLOAT4::~FLOAT4()       {
        delete data;
};

short FLOAT4::bytesPerItem()    {
        return FLOAT4_BYTES;
};

unsigned char *FLOAT4::stringForMessage()       {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader, *reverseFloat;
```

```
long    i, j, items, itemLength;
short   headerLength;

/*
**      Create space to return the "SECS Message ready"
**      version of this item.
*/
itemLength = this->lengthInMessage();
ptrOut = new unsigned char[itemLength];
ptrSave = ptrOut;

/*
**      Get the Item Header and put it in the
**      return string.
*/
itemHeader = this->getItemHeader( &headerLength, itemHeader );
memcpy( ptrOut, itemHeader, headerLength );
delete( itemHeader );

/*
**      Point to first byte of actual data to be returned,
**      then copy the data in.
*/
ptrOut += headerLength;
items = length / FLOAT4_BYTES;
for ( i = 0; i < items; i++ )   {
    /*
    **  Point to byte after end of current FLOAT4 in
    **  output byte stream.
    */
    ptrOut += sizeof(float);
    reverseFloat = (unsigned char *) &data[i];
    for ( j = 0; j < sizeof(float); j++ )       {
        *--ptrOut = *reverseFloat++;
```

```
            }
            /*
            **  Point to byte after end of current FLOAT4 in
            **  output byte stream.
            */
            ptrOut += sizeof(float);
        }

        return ptrSave;
};

int FLOAT4::hasDouble() {
        remainingBytes = length;
        ptr = data;
        return 1;
};

double FLOAT4::returnDouble()   {
        if ( remainingBytes <= 0 )      {
            cerr << "FLOAT4: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            double      tmp = (double) *ptr;
            ptr += FLOAT4_BYTES;
            remainingBytes -= FLOAT4_BYTES;
            return tmp;
        }
};

void FLOAT4::printOn( ostream& outputStream ) const     {
        /*
        **      Although this "float" type (in BORLAND C++) can
        **      apparently accomodate more than the usual range
        **      of [ le-37 to le37 ], numbers outside of this
        **      range will be printed as "0" when using the C++
        **      Stream stuff. Using C's "%e" will show the actual
        **      value.
        */

        int     i, perLine = 0;
        int     items = length / FLOAT4_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << data[i];
            if ( perLine >= 4 ) {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  FLOAT8  -       DECLARATIONS
**
**              @(#)float8.h    13.4    11/12/91
*/

class FLOAT8 : public SECSELEMENT       {
protected:
        double  *data;
        double  *ptr;
public:
        FLOAT8( unsigned long Length, unsigned char *Float8 );
        FLOAT8( unsigned long Items, double DataArray[] );
        ~FLOAT8();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasDouble();
        double          returnDouble();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  FLOAT8  -       MEMBER FUNCTIONS
**
**              @(#)float8.cpp  13.7    11/12/91
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <float8.h>

FLOAT8::FLOAT8( unsigned long Length, unsigned char *Float8 ) :
    SECSELEMENT( "FLOAT8", float8Class )        {
        short   i, j, k, items = Length / FLOAT8_BYTES;
        unsigned char   *reverseFloat;

        if ( sizeof(double) != FLOAT8_BYTES )   {
            cerr << "FLOAT8: C++ type does not match length of FLOAT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Length;
        data = ptr = new double[items];
        itemCode = FLOAT8_ITEM;

        for ( i = 0; i < items; i++ )   {
            /*
            **  Point to byte after end of current FLOAT8 in
            **  input byte stream. We'll get bytes in reverse
            **  order from the byte stream.
            */
            Float8 += FLOAT8_BYTES;
            reverseFloat = (unsigned char *) &data[i];
            for ( j = 0; j < FLOAT8_BYTES; j++ )        {
                *reverseFloat++ = *--Float8;
            }
            /*
            **  Point to byte after end of current FLOAT8 in
            **  input byte stream.
            */
            Float8 += FLOAT8_BYTES;
        }
};

FLOAT8::FLOAT8( unsigned long Items, double DataArray[] ) :
    SECSELEMENT( "FLOAT8", float8Class )        {
        short   i;

        if ( sizeof(double) != FLOAT8_BYTES )    {
            cerr << "FLOAT8: C++ type does not match length of FLOAT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Items * FLOAT8_BYTES;
        data = ptr = new double[Items];
        itemCode = FLOAT8_ITEM;

        for ( i = 0; i < Items; i++ )    {
            data[i] = DataArray[i];
        }
};

FLOAT8::~FLOAT8()       {
        delete data;
};

short FLOAT8::bytesPerItem()    {
        return FLOAT8_BYTES;
};

unsigned char *FLOAT8::stringForMessage()       {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader, *reverseFloat;
        long    i, j, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        items = length / FLOAT8_BYTES;
        for ( i = 0; i < items; i++ )   {
            /*
            **  Point to byte after end of current FLOAT8 in
            **  output byte stream.
            */
            ptrOut += FLOAT8_BYTES;
            reverseFloat = (unsigned char *) &data[i];
            for ( j = 0; j < FLOAT8_BYTES; j++ )        {
                *--ptrOut = *reverseFloat++;
            }
            /*
            **  Point to byte after end of current FLOAT8 in
            **  output byte stream.
            */
            ptrOut += FLOAT8_BYTES;
        }

        return ptrSave;
};

int FLOAT8::hasDouble() {
        remainingBytes = length;
        ptr = data;
        return 1;
};

double FLOAT8::returnDouble()   {
        if ( remainingBytes <= 0 )      {
            cerr << "FLOAT8: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            double      tmp = *ptr;
            ptr += FLOAT8_BYTES;
            remainingBytes -= FLOAT8_BYTES;
            return tmp;
        }
};

void FLOAT8::printOn( ostream& outputStream ) const     {
        int     i, perLine = 0;
        int     items = length / FLOAT8_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << data[i];
            if ( perLine >= 4 ) {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  INTl    -       DECLARATIONS
**
**              @(#)intl.h      13.5    11/12/91
*/

class INTl : public SECSELEMENT {
protected:
        char    *data;
        char    *ptr;
public:
        INTl( unsigned long Length, unsigned char *Intl );
        INTl( unsigned long Items, long DataArray[] );
        ~INTl();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  INT1    -       MEMBER FUNCTIONS
**
**              @(#)intl.cpp    13.7    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <intl.h>

INT1::INT1( unsigned long Length, unsigned char *Intl ) :
    SECSELEMENT( "INT1", intlClass )    {
        int     i;

        length = remainingBytes = Length;
        data = ptr = new char[length];
        itemCode = INT1_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = *Intl++;
        }
};

INT1::INT1( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "INT1", intlClass )    {
        int     i;

        length = remainingBytes = Items * INT1_BYTES;
        data = ptr = new char[length];
        itemCode = INT1_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = (char ) DataArray[i];
        }
};

INT1::~INT1()   {
        delete data;
};

short INT1::bytesPerItem()      {
        return INT1_BYTES;
};

unsigned char *INT1::stringForMessage() {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
```

```
                delete( itemHeader );

                /*
                **      Point to first byte of actual data to be returned,
                **      then copy the data in.
                */
                ptrOut += headerLength;
                for ( i = 0; i < length; i++ )  {
                    *ptrOut++ = data[i];
                }

                return ptrSave;
        };

        int INT1::hasLong()     {
                remainingBytes = length;
                ptr = data;
                return 1;
        };

        long INT1::returnLong() {
                if ( remainingBytes <= 0 )      {
                    cerr << "INT1: trying to get data after end of contents" << endl;
                    exit( 1 );
                }
                else    {
                    long        tmp = (long) *ptr;
                    ptr += INT1_BYTES;
                    remainingBytes -= INT1_BYTES;
                    return tmp;
                }
        };

        void INT1::printOn( ostream& outputStream ) const       {
```

```
        int     i, perLine = 0;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < length; i++, perLine++ )       {
            outputStream.width( 5 );
            outputStream.fill( ' ' );
            outputStream.setf( ios::right, ios::adjustfield );
            outputStream << dec << (int) data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  INT2    -       DECLARATIONS
**
**              @(#)int2.h      13.5    11/12/91
*/

class INT2 : public SECSELEMENT {
protected:
        short   *data;
        short   *ptr;
public:
        INT2( unsigned long Length, unsigned char *Int2 );
        INT2( unsigned long Items, long DataArray[] );
        ~INT2();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  INT2    -       MEMBER FUNCTIONS
**
**              @(#)int2.cpp    13.8    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <int2.h>

INT2::INT2( unsigned long Length, unsigned char *Int2 ) :
    SECSELEMENT( "INT2", int2Class )    {
        int     i, j, items = Length / INT2_BYTES;

        length = remainingBytes = Length;
        data = ptr = new short[items];
        itemCode = INT2_ITEM;

        for ( i = 0; i < items; i++ )   {
            data[i] = 0;
            for ( j = 0; j < sizeof(short); j++, Int2++ )       {
                    data[i] = (data[i] << 8) + *Int2;
            }
        }
};

INT2::INT2( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "INT2", int2Class )    {
        int     i;

        length = remainingBytes = Items * INT2_BYTES;
        data = ptr = new short[Items];
        itemCode = INT2_ITEM;

        for ( i = 0; i < Items; i++ )   {
            data[i] = (short) DataArray[i];
        }
};

INT2::~INT2()   {
        delete data;
};

short INT2::bytesPerItem()      {
        return INT2_BYTES;
};

unsigned char *INT2::stringForMessage() {
        unsigned char  *ptrOut, *ptrSave;
        unsigned char  *itemHeader;
        long    i, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        items = length / INT2_BYTES;
        for ( i = 0; i < items; i++ )   {
            *ptrOut++ = data[i] >> 8 & 0xFF;
            *ptrOut++ = data[i] & 0xFF;
        }

        return ptrSave;
};

int INT2::hasLong()     {
        remainingBytes = length;
        ptr = data;
        return 1;
};

long INT2::returnLong() {
        if ( remainingBytes <= 0 )      {
            cerr << "INT2: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            long        tmp = (long) *ptr;
            ptr++;
            remainingBytes -= INT2_BYTES;
            return tmp;
        }
};

void INT2::printOn( ostream& outputStream ) const       {
        int     i, perLine = 0;
        int     items = length / INT2_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << dec << data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  INT4    -       DECLARATIONS
**
**              @(#)int4.h      13.4    11/12/91
*/

class INT4 : public SECSELEMENT {
protected:
        long    *data;
        long    *ptr;
public:
        INT4( unsigned long Length, unsigned char *Int4 );
        INT4( unsigned long Items, long DataArray[] );
        ~INT4();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  INT4    -       MEMBER FUNCTIONS
**
**              @(#)int4.cpp    13.7    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <int4.h>

INT4::INT4( unsigned long Length, unsigned char *Int4 ) :
    SECSELEMENT( "INT4", int4Class )    {
        int     i, j, items = Length / INT4_BYTES;

        length = remainingBytes = Length;
        data = ptr = new long[items];
        itemCode = INT4_ITEM;

        for ( i = 0; i < items; i++ )   {
            data[i] = 0;
            for ( j = 0; j < sizeof(long); j++, Int4++ )         {
                    data[i] = (data[i] << 8) + *Int4;
            }
        }
};

INT4::INT4( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "INT4", int4Class )    {
        int     i;

        length = remainingBytes = Items * INT4_BYTES;
        data = ptr = new long[Items];
        itemCode = INT4_ITEM;

        for ( i = 0; i < Items; i++ )   {
            data[i] = DataArray[i];
        }
};

INT4::~INT4()   {
        delete data;
};

short INT4::bytesPerItem()      {
        return INT4_BYTES;
};

unsigned char *INT4::stringForMessage() {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        items = length / INT4_BYTES;
        for ( i = 0; i < items; i++ )   {
            *ptrOut++ = data[i] >> 24 & 0xFF;
            *ptrOut++ = data[i] >> 16 & 0xFF;
            *ptrOut++ = data[i] >> 8 & 0xFF;
            *ptrOut++ = data[i] & 0xFF;
        }

        return ptrSave;
};

int INT4::hasLong()     {
        remainingBytes = length;
        ptr = data;
        return 1;
};

long INT4::returnLong() {
        if ( remainingBytes <= 0 )      {
            cerr << "INT4: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            long        tmp = *ptr;
            ptr++;
            remainingBytes -= INT4_BYTES;
            return tmp;
        }
};

void INT4::printOn( ostream& outputStream ) const         {
        int     i, perLine = 0;
        int     items = length / INT4_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )         {
            outputStream << " " << dec << data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  INT8    -       DECLARATIONS
**
**              @(#)int8.h      13.4    11/12/91
*/

class INT8 : public SECSELEMENT {
protected:
        double  *data;
        double  *ptr;
public:
        INT8( unsigned long Length, unsigned char *Int8 );
        INT8( unsigned long Items, double DataArray[] );
        ~INT8();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasDouble();
        double          returnDouble();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  INT8    -       MEMBER FUNCTIONS
**
**              @(#)int8.cpp    13.7    11/12/91
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <int8.h>

INT8::INT8( unsigned long Length, unsigned char *Int8 ) :
    SECSELEMENT( "INT8", int8Class )    {
        short   i, j, items = Length / INT8_BYTES;

        if ( sizeof(double) != INT8_BYTES )     {
            cerr << "INT8: C++ type does not match length of INT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Length;
        data = ptr = new double[items];
        itemCode = INT8_ITEM;

        for ( i = 0; i < items; i++ )   {
            long                mostSignif = 0;
            unsigned long       leastSignif = 0;

            for ( j = 0; j < sizeof(long); j++, Int8++ )        {
                leastSignif = (leastSignif << 8) + *Int8;
                mostSignif = (mostSignif << 8) + *(Int8 + sizeof(long));
            }

            data[i] = (mostSignif * 4294967295) + leastSignif;
        }
};

INT8::INT8( unsigned long Items, double DataArray[] ) :
    SECSELEMENT( "INT8", int8Class )    {
        short   i;

        if ( sizeof(double) != INT8_BYTES )     {
            cerr << "INT8: C++ type does not match length of INT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Items * INT8_BYTES;
        data = ptr = new double[Items];
        itemCode = INT8_ITEM;

        for ( i = 0; i < Items; i++ )   {
            data[i] = DataArray[i];
        }

};

INT8::~INT8()   {
        delete data;
};

short INT8::bytesPerItem()      {
        return INT8_BYTES;
};
```

```

unsigned char *INT8::stringForMessage() {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader, *reverseFloat;
        long    i, j, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;

        items = length / INT8_BYTES;
        for ( i = 0; i < items; i++ )   {
            long                mostSignif;
            unsigned long       leastSignif;

            mostSignif = data[i] / 4294967295;
            leastSignif = data[i] - (mostSignif * 4294967295);

            *ptrOut++ = leastSignif >> 24 & 0xFF;
            *ptrOut++ = leastSignif >> 16 & 0xFF;
            *ptrOut++ = leastSignif >> 8 & 0xFF;
            *ptrOut++ = leastSignif & 0xFF;

            *ptrOut++ = mostSignif >> 24 & 0xFF;
            *ptrOut++ = mostSignif >> 16 & 0xFF;
            *ptrOut++ = mostSignif >> 8 & 0xFF;
            *ptrOut++ = mostSignif & 0xFF;
        }

        return ptrSave;
};

int INT8::hasDouble()   {
        remainingBytes = length;
        ptr = data;
        return 1;
};

double INT8::returnDouble()     {
        if ( remainingBytes <= 0 )      {
            cerr << "INT8: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            double      tmp = *ptr;
            ptr += INT8_BYTES;
            remainingBytes -= INT8_BYTES;
            return tmp;
        }
```

```
};

void INT8::printOn( ostream& outputStream ) const       {
        int     i, perLine = 0;
        int     items = length / INT8_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << data[i];
            if ( perLine >= 4 ) {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  S1F1    -       DECLARATIONS
**
**              @(#)s1f1.h      13.1    11/24/91
*/

class S1F1 : public STREAMFUNCTION      {
public:
        S1F1();
        S1F1( SECSMESSAGE *Primary );
        ~S1F1();

        //      buildSendPrimary() will build a customized sendPrimary()
        //      Member Function for use in the S1F1C Class, by
        //      prompting the user through the SEMI-specified
        //      TREE. The user should answer based on how his
        //      specific equipment uses the SEMI-specified TREE.
        //
        //      buildSendSecondary() will build a customized
        //      sendSecondary() Member Function for use in the S1F1C
        //      Class.
        //
        //      buildSendPrimary()
        //      buildSendSecondary()

        void    buildSendPrimary();

        void    buildSendSecondary();
};
```

```
/*
**      class:  S1F1    -       MEMBER FUNCTIONS
**
**              @(#)s1f1.cpp    13.1    11/24/91
*/

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s1f1.h>

S1F1::S1F1() : STREAMFUNCTION() {
};

S1F1::S1F1( SECSMESSAGE *Primary ) : STREAMFUNCTION( Primary )  {
};

S1F1::~S1F1()   {
};

void S1F1::buildSendPrimary()   {
        TREE    *modelTree, *treePtr;
        char    dummyString[7];

        treePtr = modelTree = new TREE;
        sprintf( dummyString, "aDummy" );

        treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );

        this->buildSend( PRIMARY, 1, 1, modelTree );
};

void SlFl::buildSendSecondary() {
        TREE    *modelTree, *treePtr;
        char    dummyString[7];

        treePtr = modelTree = new TREE;
        sprintf( dummyString, "aDummy" );

        treePtr = treePtr->addChild( *new TREE ( *new SLIST( 2L ) ) );
        treePtr->setChildCnt( 2L );
        treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );
        treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );

        this->buildSend( SECONDARY, 1, 1, modelTree );
};
```

```
/*
**      class:  S1F1C (Custom)  -       DECLARATIONS
**
**              @(#)s1f1c.h     13.1    11/24/91
*/

class S1F1C : public S1F1       {
public:
        S1F1C();
        S1F1C( SECSMESSAGE *Primary );
        ~S1F1C();

        void    primaryUpdate();
        void    secondaryUpdate();

        void    sendPrimary();
        void    sendSecondary();
};
```

```
/*
**      class:  S1F1C (Custom)  -       MEMBER FUNCTIONS
**
**              @(#)s1f1c.cpp   13.1    11/24/91
*/

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s1f1.h>
#include <s1f1c.h>
#include <dasamac.h>

S1F1C::S1F1C() : S1F1() {
};

S1F1C::S1F1C( SECSMESSAGE *Primary ) : S1F1( Primary )  {
};

S1F1C::~S1F1C() {
};

void S1F1C::primaryUpdate()     {
        this->sendSecondary();
};

void S1F1C::secondaryUpdate()   {
        TREE    *TREE_PTR;
        char    model[7];
        char    revision[7];


        TREE_PTR = primary->returnTree();

        MOVE_DOWN;
          strncpy( model, SEPTR->returnString(), 6 );
          MOVE_RIGHT( 1 );
          strncpy( revision, SEPTR->returnString(), 6 );
        MOVE_UP;
};

void S1F1C::sendPrimary()       {
        TREE    *TREE_ROOT, *TREE_PTR;
        char    model[7];


        TREE_ROOT = TREE_PTR = new TREE;

        ADD_ASCII( 6L, model );

        primary = new SECSMESSAGE( 1, 1, REPLY, TREE_ROOT );

        primary->sendMessage();
};

void S1F1C::sendSecondary()     {
        TREE    *TREE_ROOT, *TREE_PTR;
        char    model[7];
        char    revision[7];



        TREE_ROOT = TREE_PTR = new TREE;

        ADD_LIST_AND_MOVEDOWN( 2L );
          ADD_ASCII( 6L, model );
          ADD_ASCII( 6L, revision );

        secondary = new SECSMESSAGE( 1, 2, NOREPLY, TREE_ROOT );

        secondary->sendMessage();
};
```

```
/*
**      class:  S2F23   -       DECLARATIONS
**
**              @(#)s2f23.h     13.2    11/12/91
*/

class S2F23 : public STREAMFUNCTION     {
public:
        S2F23();
        S2F23( SECSMESSAGE *Primary );
        ~S2F23();

        //      buildSendPrimary() will build a customized sendPrimary()
        //      Member Function for use in the S2F23C Class, by
        //      prompting the user through the SEMI-specified
        //      TREE. The user should answer based on how his
        //      specific equipment uses the SEMI-specified TREE.
        //
        //      buildSendSecondary() will build a customized
        //      sendSecondary() Member Function for use in the S2F23C
        //      Class.
        //
        //      buildSendPrimary()
        //      buildSendSecondary()

        //      S2F23 can only be sent from Host to Equipment, so
        //      these two Member Functions simply generate error
        //      messages.
        //
        void    sendSecondary();
        void    buildSendSecondary();

        //      Since S2F23 can be sent from Host to Equipment, this
        //      Member Function helps the user construct that message.
        //
        void    buildSendPrimary();
};
```

```
/*
**      class:  S2F23   -       MEMBER FUNCTIONS
**
**              @(#)s2f23.cpp   13.3    11/24/91
*/

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s2f23.h>

S2F23::S2F23() : STREAMFUNCTION()       {
};

S2F23::S2F23( SECSMESSAGE *Primary ) : STREAMFUNCTION( Primary )        {
};

S2F23::~S2F23() {
};

void S2F23::sendSecondary()     {
        cerr << "This message cannot be sent from Host to Equipment" << endl;
};

void S2F23::buildSendPrimary()  {
        TREE    *modelTree, *treePtr;
        long    dummy[1];
        char    samplePeriod[20];

        treePtr = modelTree = new TREE;
        dummy[0] = 0;
        sprintf( samplePeriod, "samplePeriod" );

        treePtr = treePtr->addChild( *new TREE ( *new SLIST( 5L ) ) );
        treePtr->setChildCnt( 5L );
        //      TRID == TRACE ID
        treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
        //      DSPER == SAMPLE PERIOD
        treePtr = treePtr->addChild( *new TREE ( *new ASCII( 20L, samplePeriod ) ) );
        //      TOTSMP == TOTAL SAMPLES
        treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
        //      REPGSZ == REPORTING GROUP SIZE
        treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
        //      SVIDs == STATUS VARIABLE IDs
        treePtr = treePtr->addChild( *new TREE ( *new SLIST( 0L ) ) );
        treePtr->setChildCnt( 1L );
        treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );

        this->buildSend( PRIMARY, 2, 23, modelTree );
};

void S2F23::buildSendSecondary()        {
        cerr << "This message cannot be sent from Host to Equipment" << endl;
};
```

```
/*
**      class:  S2F23C (Custom) -       DECLARATIONS
**
**              @(#)s2f23c.h    13.3    11/24/91
*/

class S2F23C : public S2F23     {
public:
        S2F23C();
        S2F23C( SECSMESSAGE *Primary );
        ~S2F23C();

        void    secondaryUpdate();

        void    sendPrimary();
};
```

```
/*
**      class:  S2F23C (Custom) -       MEMBER FUNCTIONS
**
**              @(#)s2f23c.cpp  13.4    11/24/91
*/

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s2f23.h>
#include <s2f23c.h>
#include <dasamac.h>

S2F23C::S2F23C() : S2F23()      {
};

S2F23C::S2F23C( SECSMESSAGE *Primary ) : S2F23( Primary )       {
};

S2F23C::~S2F23C()       {
};

void S2F23C::secondaryUpdate()  {
        TREE    *TREE_PTR;
        long    ack2;


        TREE_PTR = secondary->returnTree();
```

```

          ack2 = SEPTR->returnLong();
};

void S2F23C::sendPrimary()      {

          TREE    *TREE_ROOT, *TREE_PTR;
          long    traceID[1];
          char    samplePeriod[7];
          long    totalSamples[1];
          long    reportGrpSize[1];
          long    statusVarID[8];



          TREE_ROOT = TREE_PTR = new TREE;

          ADD_LIST_AND_MOVEDOWN( 5L );
            ADD_INT2( 1L, traceID );
            ADD_ASCII( 6L, samplePeriod );
            ADD_INT2( 1L, totalSamples );
            ADD_INT2( 1L, reportGrpSize );
            ADD_LIST_AND_MOVEDOWN( 8L );
            TREE  *TREE_MARK = TREE_PTR;
            for ( int i = 0; i < 8; i++ ) {
            TREE_PTR = TREE_MARK;
              ADD_INT2( 1L, &statusVarID[i] );
            }

          primary = new SECSMESSAGE( 2, 23, REPLY, TREE_ROOT );

          primary->sendMessage();
};
```

```
/*
**      class:  S6F9    -       DECLARATIONS
**
**                      @(#)s6f9.h      13.5    11/24/91
*/

class S6F9 : public STREAMFUNCTION      {
public:
        S6F9();
        S6F9( SECSMESSAGE *Primary );
        ~S6F9();

        //      buildSendPrimary() will build a customized sendPrimary()
        //      Member Function for use in the S6F9C Class, by
        //      prompting the user through the SEMI-specified
        //      TREE. The user should answer based on how his
        //      specific equipment uses the SEMI-specified TREE.
        //
        //      buildSendSecondary() will build a customized
        //      sendSecondary() Member Function for use in the S6F9C
        //      Class.
        //
        //      buildSendPrimary()
        //      buildSendSecondary()

        //      S6F9 can only be sent from Equipment to Host, so
        //      these two Member Functions simply generate error
        //      messages.
        //
        void    sendPrimary();
        void    buildSendPrimary();

        //      Since S6F9 can be sent from Equipment to Host, it
        //      follows that the Host can send the Secondary message,
        //      namely S6F10. This member function helps the user
        //      construct that message.
        //
        void    buildSendSecondary();
};
```

```
/*
**      class:  S6F9    -       MEMBER FUNCTIONS
**
**              @(#)s6f9.cpp    13.8    11/24/91
*/

#include <iostream.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s6f9.h>

S6F9::S6F9() : STREAMFUNCTION() {
};

S6F9::S6F9( SECSMESSAGE *Primary ) : STREAMFUNCTION( Primary )  {
};

S6F9::~S6F9()   {
};

void S6F9::sendPrimary()        {
        cerr << "This message cannot be sent from Host to Equipment" << endl;
};

void S6F9::buildSendPrimary()   {
        cerr << "This message cannot be sent from Host to Equipment" << endl;
};

void S6F9::buildSendSecondary() {
        TREE    *modelTree, *treePtr;
        long    dummy[1];

        treePtr = modelTree = new TREE;
        dummy[0] = 0;

        treePtr = treePtr->addChild( *new TREE ( *new BINARY( 1L, dummy ) ) );

        this->buildSend( SECONDARY, 6, 9, modelTree );
};
```

```
/*
**      class:  S6F9C (Custom)  -       DECLARATIONS
**
**              @(#)s6f9c.h     13.1    11/24/91
*/

class S6F9C : public S6F9       {
public:
        S6F9C();
        S6F9C( SECSMESSAGE *Primary );
        ~S6F9C();

        void    primaryUpdate();

        void    sendSecondary();
};
```

```
/*
**      class:  S6F9C (Custom)  -       MEMBER FUNCTIONS
**
**              @(#)s6f9c.cpp   13.1    11/24/91
*/

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>
#include <s6f9.h>
#include <s6f9c.h>
#include <dasamac.h>

S6F9C::S6F9C() : S6F9() {
};

S6F9C::S6F9C( SECSMESSAGE *Primary ) : S6F9( Primary )  {
};

S6F9C::~S6F9C() {
};

void S6F9C::primaryUpdate()     {
        this->sendSecondary();
};

void S6F9C::sendSecondary()     {
```

```
            TREE    *TREE_ROOT, *TREE_PTR;
            long    ack6[1];


            TREE_ROOT = TREE_PTR = new TREE;

            ADD_BINARY( 1L, &ack6[0] );

            secondary = new SECSMESSAGE( 6, 10, NOREPLY, TREE_ROOT );

            secondary->sendMessage();
    };
```

```
/*
**      class:  SECSELEMENT      -       DECLARATIONS
**
**      This is an "abstract class" - no instances of this class
**      may be declared, but classes may be derived from this class.
**
**      "Pure virtual functions" returnValue() & printOn() are
**      what make this an "abstract class". This means that these
**      functions must be defined in any classes derived from this
**      class.
**
**      The "= 0" after the function declarations for returnValue() &
**      printOn() is what identifies them as "pure virtual functions".
**
**              @(#)secselem.h  13.9    11/12/91
*/

#define WORKSPACE       8

class SECSELEMENT : public Object       {
protected:
        char            *className;
        classType       classTypeValue;
        unsigned char   itemCode;
        unsigned long   length; // Number of bytes in guts of item.
        unsigned long   remainingBytes;
public:
        SECSELEMENT( char *ClassName, classType ClassType );
        virtual ~SECSELEMENT();

        virtual int             isEqual( const Object& ) const;
        virtual classType       isA( ) const;
        virtual char            *nameOf( ) const;
        virtual hashValueType   hashValue( ) const;

        unsigned char           getItemCode();
        virtual unsigned char   *getItemHeader( short *HeaderLength,
                                    unsigned char *ItemHeader );
        virtual long            lengthInMessage();
        virtual unsigned char   *stringForMessage() = 0;

        virtual int             hasDouble();
        virtual int             hasLong();
        virtual int             hasString();
        virtual double          returnDouble();
        virtual long            returnLong();
        virtual char            *returnString();

        virtual long            itemsRemaining();
        virtual short           bytesPerItem() = 0;

        virtual void            printOn( ostream& ) const = 0;
};

void printBits( char *desc, int bytes, unsigned char *ptr );
```

```
/*
**      class:  SECSELEMENT     -       MEMBER FUNCTIONS
**
**              @(#)secselem.cpp        13.7    11/12/91
*/

#include <string.h>
#include <stdio.h>
#include <mem.h>
#include <stdlib.h>
#include <values.h>
#include <iostream.h>
#include <object.h>
#include <secselem.h>

SECSELEMENT::SECSELEMENT( char *ClassName, classType ClassType )        {
        className = ClassName;
        classTypeValue = ClassType;
};

SECSELEMENT::~SECSELEMENT()     {
};

SECSELEMENT::isEqual( const Object& testSECSELEMENT ) const     {
        //return ( data == (SECSELEMENT &)testSECSELEMENT );
        return ( 0 );
};

classType SECSELEMENT::isA() const      {
        return classTypeValue;
};

char *SECSELEMENT::nameOf() const       {
        return className;
```

```
};

hashValueType SECSELEMENT::hashValue() const    {
        hashValueType   value = hashValueType( 0 );
        return( value );
};

unsigned char SECSELEMENT::getItemCode()        {
        return itemCode;
};

unsigned char *SECSELEMENT::getItemHeader( short *HeaderLength,
    unsigned char *ItemHeader ) {
        unsigned char   formatByte;
        unsigned char   lengthBytes, *ptrSave;
        unsigned long   tmpLength;
        int     i;

        /*
        **      Create space to return the "Item Header"
        **      version of this item.
        */
        lengthBytes = this->lengthInMessage() - 1 - length;
        *HeaderLength = 1 + lengthBytes;
        ItemHeader = new unsigned char[*HeaderLength];
        ptrSave = ItemHeader;

        /*
        **      Determine the Format Byte and put it in the
        **      return string.
        */
        formatByte = this->getItemCode() << 2;
        formatByte |= lengthBytes;
        *ItemHeader = formatByte;

        /*
        **      Point to last length byte and populate the length
        **      bytes in reverse order.
        */
        ItemHeader += lengthBytes;
        tmpLength = length;
        for ( i = 0; i < lengthBytes; i++ )     {
            *ItemHeader = tmpLength & 0x000000FF;
            tmpLength = tmpLength >> 8;
            ItemHeader--;
        }

        return ptrSave;
};

long SECSELEMENT::lengthInMessage()     {
        unsigned char   lengthBytes;
        int     i;
        unsigned long   temp;

        if ( length == 0 )      {
            lengthBytes = 1;
        }
        else    {
            temp = length;
            for ( i = 0; i < sizeof(long); i++ )        {
                if ( temp == 0 )        {
                    i -= 1;
                    break;
                }
                temp = temp >> 8;
            }
            lengthBytes = i + 1;
        }

        return (long) 1 + lengthBytes + length;
};

int SECSELEMENT::hasDouble()    {
        return 0;
};

int SECSELEMENT::hasLong()      {
        return 0;
};

int SECSELEMENT::hasString()    {
        return 0;
};

double SECSELEMENT::returnDouble()      {
        return 0.0;
};

long SECSELEMENT::returnLong()  {
        return 0;
};

char *SECSELEMENT::returnString()       {
        return NULL;
};

long SECSELEMENT::itemsRemaining()      {
        return remainingBytes / this->bytesPerItem();
};

void
```

```
printBits( char *desc, int bytes, unsigned char *ptr )  {
        int     i, j;
        char    tmpChar;

        printf( "%s:\n\t", desc );

        for ( i = 0; i < bytes; i++ )   {

            tmpChar = *ptr++;
            for ( j = 0; j < BITSPERBYTE; j++ ) {
                if ( tmpChar & 0x80 )   {
                    printf( "1" );
                }
                else    {
                    printf( "0" );
                }
                if ( j % 2 )    {
                    printf( " " );
                }
                tmpChar = tmpChar << 1;
            }
            printf( "\n\t" );
        }
        printf( "\n" );
}
```

```
/*
**      class:  SECSMESSAGE     -       DECLARATIONS
**
**              @(#)secsmess.h  13.7    11/12/91
*/

#ifndef __CUSTOM_H
#include <custom.h>
#define __CUSTOM_H
#endif

#define MEMCHUNK        512


#define NOREPLY         0
#define REPLY           1


#define CHARCNTOVERHEAD         1
#define HEADEROVERHEAD          10
#define NONOVERHEADINBLOCK      243
#define CHECKSUMOVERHEAD        2

#define OVERHEADINBLOCK         ( CHARCNTOVERHEAD + HEADEROVERHEAD + CHECKSUMOVERHEAD )
#define BYTESINBLOCK            ( OVERHEADINBLOCK + NONOVERHEADINBLOCK )


class SECSMESSAGE       {
protected:
    unsigned char       *messageFlat;
    short               messageLength;
    TREE                *messageTree;

    /*
    **  This data is part of the incoming SECS-II message &
    **  will be used to instantiate the specific SECS
    **  object.
    */
    unsigned char       secsStream, secsFunction;
    unsigned char       upperDevID, lowerDevID;
    unsigned char       upperBlockNo, lowerBlockNo;
    unsigned char       systemByte[4];
    truthFlag           msgToHost, replyRequired;

    /*
    **  This data is part of the incoming SECS-II message,
    **  but will be used only help separate the wheat from
    **  the chaff of the message.
    */
    truthFlag   lastBlock;


    void        parseMessage( unsigned char *MessageIn, short Length );

    void        addItemToMessage( TREE *TreePtr, short *MessCharCnt );
    void        expandMessage();
    void        addSECSheader( short *MessCharCnt );

public:
    SECSMESSAGE( unsigned char *MessageIn, short Length );
    SECSMESSAGE( char *FileName );
    SECSMESSAGE( short Stream, short Function, int ReplyRequired, TREE *TreePtr );
    ~SECSMESSAGE();

    void        buildMessage();
    void        sendMessage();

    short       getSECSstream();
    short       getSECSfunction();

    int         isPrimary();

    TREE        *returnTree();

    void        printOn( ostream& ) const;
};

long    bytesValue( short numBytes, register unsigned char *fromString );
```

```
/*
**      class:  SECSMESSAGE     -       MEMBER FUNCTIONS
**
**              @(#)secsmess.cpp        13.10   11/24/91
*/

#include <string.h>
#include <stdio.h>
#include <alloc.h>
#include <mem.h>
#include <iostream.h>
#include <values.h>
#include <process.h>
#include <fstream.h>
#include <object.h>
#include <clstypes.h>

#ifndef __CUSTOM_H
#include <custom.h>
#define __CUSTOM_H
#endif

#include <secstype.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>

SECSMESSAGE::SECSMESSAGE( unsigned char *MessageIn, short Length ) {
        /*      CHANGE
        **
        **      messageTree->addParentType( "LIST" );
        */
        this->parseMessage( MessageIn, Length );
```

```
};

SECSMESSAGE::SECSMESSAGE( char *FileName )      {
        long    fileSize;
        unsigned char   charIn, *messageIn;
        char    fullFileName[SIZEFILE];
        short   i = 0;

        strcpy( fullFileName, SECSTESTDIR );
        strcat( fullFileName, FileName );

        ifstream sourceFile( fullFileName, ios::binary );
        if ( ! sourceFile )     {
            cerr << "SECSMESSAGE: cannot open file " <<
                fullFileName << " for input" << endl;
            exit( 1 );
        }

        /*
        **      Seek to end of the file, then request byte
        **      position in file to determine file size. Size
        **      messageIn[] based on this, after seeking back
        **      to the start of the file.
        */
        sourceFile.seekg( 0L, (seek_dir) 2 );
        fileSize = sourceFile.tellg();
        sourceFile.seekg( 0L, (seek_dir) 0 );
        messageIn = new unsigned char[fileSize];

        sourceFile.read( messageIn, fileSize );
        if ( sourceFile.gcount() != fileSize )  {
            cout << "SECSMESSAGE: did not get the expected number " <<
                "of characters from the file." << endl;
            exit( 1 );
        }

        sourceFile.close();

        /*      CHANGE
        **
        **      messageTree->addParentType( "LIST" );
        */
        this->parseMessage( messageIn, fileSize );
};

SECSMESSAGE::SECSMESSAGE( short Stream, short Function, int ReplyRequired,
TREE *TreePtr ) {
        /*      CHANGE
        **
        **      messageTree->addParentType( "LIST" );
        */
        messageTree = TreePtr;
        messageFlat = NULL;

        secsStream = Stream;
        secsFunction = Function;
        replyRequired = (truthFlag) ReplyRequired;

        /*      CHANGE
        **
        **      systemByte = ?????
        **      upperDevID = ????
        **      lowerDevID = ????
        */
        systemByte[0] = (unsigned char) 0;
        systemByte[1] = (unsigned char) 2;
        systemByte[2] = (unsigned char) 2;
        systemByte[3] = (unsigned char) 3;
        upperDevID = 54;
        lowerDevID = 59;
};

SECSMESSAGE::~SECSMESSAGE() {
        if ( messageFlat != NULL )       {
            delete messageFlat;
        }

        if ( messageTree != NULL )       {
            delete messageTree;
        }
};

void
SECSMESSAGE::parseMessage( unsigned char *MessageIn, short Length ) {
        unsigned char   *messInPtr = MessageIn;
        messageFlat = new unsigned char[Length];
        unsigned char   *messFlatPtr;

        short   i, blockCnt = 0;
        short   charCnt, blockCharCnt, messCharCnt = 0;
        short   formatByte, itemCode, numLengthBytes;

        messageLength = Length;


        lastBlock = FALSE;

        while ( ! lastBlock )   {
            blockCnt++;
            blockCharCnt = *messInPtr++ - 10; // subtract out Header bytes
            messCharCnt += blockCharCnt;

            msgToHost = ( *messInPtr & 0200 ) >> 7;
            upperDevID = *messInPtr++ & ~0200;
            lowerDevID = *messInPtr++;

            replyRequired = ( *messInPtr & 0200 ) >> 7;
            secsStream = *messInPtr++ & ~0200;
            secsFunction = *messInPtr++;

            lastBlock = ( *messInPtr & 0200 ) >> 7;
            upperBlockNo = *messInPtr++ & ~0200;
            lowerBlockNo = *messInPtr++;

            for ( i = 0; i < 4; i++ )   {
                systemByte[i] = *messInPtr++;
            }

            messFlatPtr = messageFlat + messCharCnt - blockCharCnt;

            while ( blockCharCnt -- )   {
                *messFlatPtr++ = *messInPtr++;
            }

            messInPtr += 2;     // Checksum
        }
        delete MessageIn;


        messageTree = new TREE;
        TREE    *treePtr = messageTree;

        messFlatPtr = messageFlat;
        charCnt = 0;
        while ( charCnt < messCharCnt ) {
            unsigned long       length;

        formatByte = ( short ) *messFlatPtr++;
        itemCode = formatByte >> 2;
        numLengthBytes = formatByte & 0003;

        length = bytesValue( numLengthBytes, messFlatPtr );
        messFlatPtr += numLengthBytes;

        switch ( itemCode ) {
            case LIST_ITEM:
                    treePtr = treePtr->addChild( *new TREE
                        ( *new SLIST ( length ) ) );
                    /*      INVESTIGATE:
                    **      which "part" of ROOT is this set for ?
                    */
                    treePtr->setChildCnt( length );
                    length = 0;
                    break;
            case ASCII_ITEM:
                    treePtr = treePtr->addChild( *new TREE
                        ( *new ASCII ( length, (char *) messFlatPtr ) ) );
                    break;
            case BINARY_ITEM:
                    treePtr = treePtr->addChild( *new TREE
                        ( *new BINARY ( length, messFlatPtr ) ) );
                    break;
            case BOOLEAN_ITEM:
                    treePtr = treePtr->addChild( *new TREE
                        ( *new BOOLEAN ( length, messFlatPtr ) ) );
                    break;
            case FLOAT4_ITEM:
                    treePtr = treePtr->addChild( *new TREE
                        ( *new FLOAT4 ( length, messFlatPtr ) ) );
                    break;
```

```
                case FLOAT8_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new FLOAT8 ( length, messFlatPtr ) ) );
                        break;
                case INT1_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new INT1 ( length, messFlatPtr ) ) );
                        break;
                case INT2_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new INT2 ( length, messFlatPtr ) ) );
                        break;
                case INT4_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new INT4 ( length, messFlatPtr ) ) );
                        break;
                case INT8_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new INT8 ( length, messFlatPtr ) ) );
                        break;
                case UINT8_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new UINT8 ( length, messFlatPtr ) ) );
                        break;
                case UINT1_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new UINT1 ( length, messFlatPtr ) ) );
                        break;
                case UINT2_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new UINT2 ( length, messFlatPtr ) ) );
                        break;
                case UINT4_ITEM:
                        treePtr = treePtr->addChild( *new TREE
                            ( *new UINT4 ( length, messFlatPtr ) ) );
                        break;
                default:
                        cout << "<<<< got an UNKNOWN >>>>\n" << endl;
                        break;
            }

            messFlatPtr += length;
            charCnt += ( 1 + numLengthBytes + length );

            while ( treePtr->gotAllChildren() ) {
                if ( ! treePtr->atRoot() )      {
                    treePtr = treePtr->moveUp();
                }
                else    {
                    if ( charCnt != messCharCnt )       {
                        cout << "AT ROOT AND ALL CHARS ARE NOT READ";
                    }
                    break;
                }
            }
        }

        /*
        free( messageFlat );
        */

        if ( messCharCnt != 0 && ! treePtr->atRoot() )
            cout << "NOT AT ROOT, BUT ALL CHARACTERS ARE READ";

};

void SECSMESSAGE::buildMessage()        {
        char    fullFileName[SIZEFILE];
        char    baseFileName[SIZEFILE];
        short   blockCharCnt;

        strcpy( fullFileName, SECSTESTDIR );
        sprintf( baseFileName, "X_S%d-F%d", this->getSECSstream(),
            this->getSECSfunction() );
        strcat( fullFileName, baseFileName );

        ofstream destFile( fullFileName, ios::binary );
        if( !destFile ) {
            cerr << "SECSMESSAGE: cannot open file " <<
                fullFileName << " for output" << endl;
        }

        /*
        **      Allocate memory for raw message to Equipment.
        */
        if ( messageFlat != NULL )      {
            delete messageFlat;
        }
        messageFlat = new unsigned char[MEMCHUNK];
        messageLength = MEMCHUNK;

        /*
        **      Run through the SECS TREE, adding the raw data
        **      from each item to the message that will be sent
        **      to the Equipment.
        */

        TREE    *treePtr = messageTree;
        short   messCharCnt = 0;

        if ( treePtr != NULL )  {
            /*
        **  If TREE has Children.
        */
        if ( treePtr->hasChildren() )       {
            this->addItemToMessage( treePtr, &messCharCnt );
            /*
            **      Move Down to Left-most Child.
            */
            treePtr = treePtr->moveDown();
            treePtr->resetChildCursor();
            /*
            **      While not at Root of Super-TREE ...
            */
            while( ! treePtr->atRoot() )    {
                /*
                **  If TREE has Children.
                */
                if ( treePtr->hasChildren() )       {
                    /*
                    **      If visited all Children of TREE.
                    */
                    if ( treePtr->gotAllChildren() )        {
                        /*
                        **  If TREE has Sibling to Right.
                        */
                        if ( treePtr->hasRightSibling() )   {
                            /*
                            **      Move to Sibling to Right.
                            */
                            treePtr = treePtr->moveRight( 1 );
                            treePtr->resetChildCursor();
                        }
                        else        {
                            /*
                            **      Move Up to Parent.
                    */
                    treePtr = treePtr->moveUp();
                }
            }
            else    {
                this->addItemToMessage( treePtr, &messCharCnt );
                /*
                **  Move Down to Child.
                */
                treePtr = treePtr->moveDown();
                treePtr->resetChildCursor();
            }
        }
        else        {
            this->addItemToMessage( treePtr, &messCharCnt );
            /*
            **      If TREE has Sibling to Right.
            */
            if ( treePtr->hasRightSibling() )       {
                /*
                **  Move to Sibling to Right.
                */
                treePtr = treePtr->moveRight( 1 );
                treePtr->resetChildCursor();
            }
            else    {
                /*
                **  Move Up to TREE's Parent.
                */
                treePtr = treePtr->moveUp();
            }
        }
    }
}
```

```
            else        {
                this->addItemToMessage( treePtr, &messCharCnt );
            }
        }


        this->addSECSheader( &messCharCnt );


        destFile.write( messageFlat, messageLength );

        destFile.close();
};

void SECSMESSAGE::sendMessage() {
        this->buildMessage();
};

void SECSMESSAGE::addItemToMessage( TREE *treePtr, short *MessCharCnt)  {
        SECSELEMENT     *sePtr;
        short           itemLength;

        sePtr = ((SECSELEMENT *) treePtr->nodeAddr());
        itemLength = (short) sePtr->lengthInMessage();

        if ( *MessCharCnt + itemLength > messageLength )        {
            this->expandMessage();
        }


        memcpy( messageFlat + *MessCharCnt,
            sePtr->stringForMessage(), itemLength );
        free( sePtr->stringForMessage() );
        *MessCharCnt += itemLength;
```

```
};

void SECSMESSAGE::expandMessage()       {
        unsigned char   *oldMessage, *newMessage;
        short           newMessageLength = messageLength + MEMCHUNK;


        oldMessage = messageFlat;       // save for delete

        newMessage = new unsigned char[newMessageLength];
        memcpy( newMessage, messageFlat, messageLength );

        messageFlat = newMessage;
        messageLength = newMessageLength;

        delete oldMessage;
};

void SECSMESSAGE::addSECSheader( short *MessCharCnt )   {
        int     i, bytesToCopy = NONOVERHEADINBLOCK;
        unsigned char   setBit = 1 << 7;
        unsigned char   *tmpMessage, *itemsPtr, *outPtr;

        int     fullBlocks = *MessCharCnt / NONOVERHEADINBLOCK;
        int     leftOvers = *MessCharCnt % NONOVERHEADINBLOCK;
        int     totalBlocks = fullBlocks + ( leftOvers ? 1 : 0 );

        int     sizeOfMessageOut = *MessCharCnt;

        /*
        **      Will happen when the message consists of only the
        **      header (e.g., S?-FO).
        */
        if ( *MessCharCnt == 0 )        {
            totalBlocks = 1;
            bytesToCopy = 0;
        }

        sizeOfMessageOut += totalBlocks * OVERHEADINBLOCK;

        outPtr = tmpMessage = new unsigned char[sizeOfMessageOut];

        itemsPtr = messageFlat; // Point to start of "items only".

        while ( totalBlocks-- ) {
            if ( ( totalBlocks == 0 ) && leftOvers )    {
                bytesToCopy = leftOvers;
            }
            *outPtr++ = (unsigned char) bytesToCopy + HEADEROVERHEAD;

            *outPtr++ = upperDevID;     // MESSAGE TO HOST - not set here
            *outPtr++ = lowerDevID;

            *outPtr++ = secsStream;     // REPLY REQUIRED
            *outPtr++ = secsFunction;

            *outPtr++ = upperBlockNo | ( totalBlocks == 0 ? setBit : 0 );
            *outPtr++ = lowerBlockNo;
            for ( i = 0; i < 4; i++ )    {
                *outPtr++ = systemByte[i];
            }
            memcpy( outPtr, itemsPtr, bytesToCopy );
            outPtr += bytesToCopy;
            itemsPtr += bytesToCopy;

            *outPtr++ = 0xFE;
            *outPtr++ = 0xFF;
        }

        itemsPtr = messageFlat; // save message address for delete

        messageFlat = tmpMessage;
        messageLength = sizeOfMessageOut;

        delete itemsPtr;        // delete old message
};

short SECSMESSAGE::getSECSstream()      {
        return (short) this->secsStream;
};

short SECSMESSAGE::getSECSfunction()    {
        return (short) this->secsFunction;
};

int SECSMESSAGE::isPrimary()    {
        return secsFunction & 1;
};

TREE *SECSMESSAGE::returnTree() {
        return this->messageTree;
};

void SECSMESSAGE::printOn( ostream& outputStream ) const        {
        outputStream << "S" << (short) this->secsStream;
        outputStream << "-F" << (short) this->secsFunction << endl;
        this->messageTree->printOn( outputStream );
};
```

```

/*
**      Get the requested number of bytes from a string of unsigned char's,
**      building the value as each byte is encountered. Return the total
**      value after all bytes have been visited.
**
**      Because we will be putting 1, 2, 3, ... bytes into a "long" type
**      we may need to extend the sign bit to fill a "long". We only need to
**      do this if the high-order bit is set in the first byte we encounter.
*/
long    bytesValue( short numBytes, register unsigned char *fromString )
{
        int     i;
        register        long            value = 0;
        long    mask, byteMask;
        short   hiBitValue = 1;
        truthFlag       isNegative = FALSE;



        /*
        **      Determine the value of a byte with only the highest bit set.
        **      If the first byte of the string we are evaluating has this
        **      bit set, we have a negative number. If we have a negative
        **      nummber, we will have to extend the SIGN BIT.
        */
        for ( i = 0; i < BITSPERBYTE - 1; i++ )
            hiBitValue *= 2;

        if ( *fromString & hiBitValue )
            isNegative = TRUE;



```

```
    /*
    **      Determine the value of the number of bytes in memory
    **      without regard to sign.
    */
    for ( i = 0; i < numBytes; i++ )
        value = (value << BITSPERBYTE) + *fromString++;



    if ( isNegative )       {
        /*
        **  Create a byte-sized mask of all 1's.
        */
        byteMask = ( 2 * hiBitValue ) - 1;

        /*
        **  Create a long-sized mask for sign extension of our final value.
        **  The highest-order bytes that are not used by our value are set
        **  to all 1's. Assuming 8-bit bytes and 4-byte "long"s, here are
        **  what the masks would look like:
        **
        **                  numBytes == 1           mask=0xffffff00
        **                  numBytes == 2           mask=0xffff0000
        **                  numBytes == 3           mask=0xff000000
        **                  numBytes == 4           mask=0x00000000
        */
        mask = 0;
        for ( i = 0; i < numBytes; i++ )    {
            mask = mask << BITSPERBYTE;
            mask |= byteMask;
        }
        mask = ~mask;


```

```

            /*
            **  Do the sign extension.
            */
            value = mask | value;
        }



        return( value );
}
```

```
/*
**      class:  SLIST   -       DECLARATIONS
**
**              @(#)slist.h     13.6    11/12/91
*/

class SLIST : public SECSELEMENT        {
protected:
public:
        SLIST( unsigned long Length );
        ~SLIST();

        short           bytesPerItem();

        unsigned char   *getItemHeader( short *HeaderLength,
                            unsigned char *ItemHeader );
        long            lengthInMessage();
        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        long            itemsRemaining();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  SLIST   -       MEMBER FUNCTIONS
**
**              @(#)slist.cpp   13.7    11/12/91
*/

#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <slist.h>

SLIST::SLIST( unsigned long Length ) :
    SECSELEMENT( "LIST", slistClass )   {
        length = Length;
        itemCode = LIST_ITEM;
};

SLIST::~SLIST() {
};

short SLIST::bytesPerItem()     {
        return LIST_BYTES;
};

/*
**      SLIST requires a different version of getItemHeader()
**      because the "length" is already subtracted out of the
**      "lengthInMessage()" for SLIST --- it must be subtracted
**      out for all SECSELEMENT subclasses other than SLIST.
*/
unsigned char *SLIST::getItemHeader( short *HeaderLength,
    unsigned char *ItemHeader ) {
```

```
    unsigned char   formatByte;
    unsigned char   lengthBytes, *ptrSave;
    unsigned long   tmpLength;
    int     i;

    /*
    **      Create space to return the "Item Header"
    **      version of this item.
    */
    lengthBytes = this->lengthInMessage() - 1;
    *HeaderLength = 1 + lengthBytes;
    ItemHeader = new unsigned char[*HeaderLength];
    ptrSave = ItemHeader;

    /*
    **      Determine the Format Byte and put it in the
    **      return string.
    */
    formatByte = this->getItemCode() << 2;
    formatByte |= lengthBytes;
    *ItemHeader = formatByte;

    /*
    **      Point to last length byte and populate the length
    **      bytes in reverse order.
    */
    ItemHeader += lengthBytes;
    tmpLength = length;
    for ( i = 0; i < lengthBytes; i++ )     {
        *ItemHeader = tmpLength & 0x000000FF;
        tmpLength = tmpLength >> 8;
        ItemHeader--;
    }

```

```
        return ptrSave;
};

long SLIST::lengthInMessage()   {
        return (long) SECSELEMENT::lengthInMessage() - length;
};

unsigned char *SLIST::stringForMessage()        {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        return ptrSave;
};

int SLIST::hasLong()    {
        return 1;
```

```
};

long SLIST::returnLong()        {
        return length;
};

long SLIST::itemsRemaining()    {
        /*
        **      There's always 1 item: the length of the LIST.
        */
        return 1L;
};

void SLIST::printOn( ostream& outputStream ) const      {
        outputStream << '<' << this->nameOf() << ' ' << dec << length << ">" << flush;
};
```

```
/*
**      class:  STREAMFUNCTION  -       DECLARATIONS
**
**              @(#)strfunc.h   13.5    11/24/91
*/

#define PRIMARY         0
#define SECONDARY       1

#define INDENTMAX       80

class STREAMFUNCTION {
protected:
        SECSMESSAGE     *primary;
        SECSMESSAGE     *secondary;

        char    Indent[INDENTMAX];
        int     IndentCnt;
        int     IndentInc;

        ofstream        *classFile;
        ofstream        *declFile;
        ofstream        *codeFile;

        void            classWrite( char *String );
        void            declWrite( char *String );
        void            codeWrite( char *String );
public:
        STREAMFUNCTION();
        STREAMFUNCTION( SECSMESSAGE *SECSmess );
        virtual ~STREAMFUNCTION();

        void            graftSecondary( SECSMESSAGE *Secondary );

```

```
//      Send SECS Message to Equipment.
//
virtual void    sendPrimary();
virtual void    sendSecondary();

virtual void    buildSendPrimary();
virtual void    buildSendSecondary();

//      Update the DASA based on SECS Message from the
//      Equipment:
//
//      primaryUpdate() - updates based on a Primary message
//                        received from the Equipment.
//
//      secondaryUpdate() - updates based on a Secondary message
//                        received from the Equipment.
//
virtual void    primaryUpdate();
virtual void    secondaryUpdate();

//      Prompt user through a TREE received from the
//      Equipment.
//
void            buildPrimaryUpdate();
void            buildSecondaryUpdate();
void            buildUpdate( int SMtype );
void            updateTreeWalk( TREE *TreePtr );
void            updatePromptUser( TREE *TreePtr );

//      Prompt user through a TREE to be sent to the
//      Equipment.
//
void            buildSend( int SMtype, short Stream,
                short Function, TREE *TreePtr );
```

```
        void            sendTreeWalk( TREE *TreePtr );
        void            sendPromptUser( TREE *TreePtr );

        void            printTreeLocale( TREE *TreePtr );
        void            getVariable( char *Type, int *Length, char *Name );
        int             variableList( TREE *TreePtr );
        int             parentVariableList( TREE *TreePtr );

        int             getStream();
        int             getPrimaryFunction();
        int             getSecondaryFunction();

        void            indentMore();
        void            indentLess();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  STREAMFUNCTION  -       MEMBER FUNCTIONS
**
**              @(#)strfunc.cpp 13.6    11/24/91
*/

#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <dbllist.h>
#include <dlstelem.h>
#include <tree.h>
#include <secsmess.h>
#include <secstype.h>
#include <strfunc.h>

STREAMFUNCTION::STREAMFUNCTION()        {
        primary = NULL;
        secondary = NULL;

        IndentCnt = 0;
        IndentInc = 2;
        memset( Indent, ' ', INDENTMAX );
        Indent[IndentCnt] = '\0';
};

STREAMFUNCTION::STREAMFUNCTION( SECSMESSAGE *SECSmess ) {
        if ( SECSmess->isPrimary() )    {
            primary = SECSmess;
            secondary = NULL;
        }
        /*
        **      This part of the constructor would only be exercised
        **      in "artificial" situtations, e.g., if we were sending
        **      Secondary messages from the Equipment (as files) that
        **      were not in response to Primary messages that the Host
        **      had sent in order to "buildSecondaryUpdate()".
        */
        else    {
            primary = NULL;
            secondary = SECSmess;
        }

        IndentCnt = 0;
        IndentInc = 2;
        memset( Indent, ' ', INDENTMAX );
        Indent[IndentCnt] = '\0';
};

STREAMFUNCTION::~STREAMFUNCTION()       {
        if ( primary != NULL )  {
            delete primary;
        }

        if ( secondary != NULL )        {
            delete secondary;
        }
};

void STREAMFUNCTION::graftSecondary( SECSMESSAGE *Secondary )   {
        secondary = Secondary;
};

void STREAMFUNCTION::sendPrimary()      {
};

void STREAMFUNCTION::sendSecondary()    {
        TREE    *treePtr = NULL;
        short   stream, function;

        /*
        **      Send S*-F0 to close the transaction on the Equipment.
        **
        **      This code will only be exercised if the user does not
        **      override with a more specific response.
        */
        stream = primary->getSECSstream();
        function = 0;

        secondary = new SECSMESSAGE( stream, function, NOREPLY, treePtr );

        secondary->sendMessage();
};

void STREAMFUNCTION::buildSendPrimary() {
};

void STREAMFUNCTION::buildSendSecondary()       {
};

void STREAMFUNCTION::primaryUpdate()    {
};

void STREAMFUNCTION::secondaryUpdate()  {
};

void STREAMFUNCTION::buildPrimaryUpdate()       {
        this->buildUpdate( PRIMARY );
};

void STREAMFUNCTION::buildSecondaryUpdate()     {
        this->buildUpdate( SECONDARY );
};

/*
**      Something in here causes "Null Pointer Assignment"
*/
void STREAMFUNCTION::buildUpdate( int SMtype )  {
        char    className[SIZEFILE];
        char    classFileName[SIZEFILE];
        char    declFileName[SIZEFILE];
        char    codeFileName[SIZEFILE];
        char    memFuncName[20];        // Name of Member Function to be
                                        //   generated.
        char    smTypeDecl[10];         // To initialize (TREE *) in
                                        //   code to be generated.
        char    fileSuffix[10];         // Suffix of file where generated
                                        //   code will be written to.
        char    genString[80];
        ifstream        *declIn, *codeIn;
        TREE            *treePtr;
        SECSMESSAGE     *smPtr;

        if ( SMtype == PRIMARY )        {
            smPtr = primary;
            sprintf( memFuncName, "primaryUpdate" );
            sprintf( smTypeDecl, "primary" );
            sprintf( fileSuffix, "pu" );
        }
        else    {
            smPtr = secondary;
            sprintf( memFuncName, "secondaryUpdate" );
            sprintf( smTypeDecl, "secondary" );
            sprintf( fileSuffix, "su" );
        }
```

```
	treePtr = smPtr->returnTree();

	sprintf( className, "S%dF%dC",
	    smPtr->getSECSstream(), smPtr->getSECSfunction() );

	sprintf( classFileName, "%s.%s", className, fileSuffix );

	sprintf( declFileName, "decl.tmp" );

	sprintf( codeFileName, "code.tmp" );

	/*
	**	Open the Class File and write the Member Function
	**	declaration and definition.
	*/
	classFile = new ofstream( classFileName, ios::in );
	if ( ! classFile )	{
	    cerr << "STREAMFUNCTION: cannot open file " <<
		classFileName << " for input" << endl;
	}

	sprintf( genString, "//\t%s() DECLARATION\n", memFuncName );
	this->classWrite( genString );
	sprintf( genString, "\tvoid\t%s();\n\n", memFuncName );
	this->classWrite( genString );
	sprintf( genString, "//\t%s() DEFINITION\n", memFuncName );
	this->classWrite( genString );
	sprintf( genString, "void %s::%s()\t{\n\n", className, memFuncName );
	this->classWrite( genString );



	/*
	**	Open temporary files to hold the declarations & code
```

```
**      that will be internal to the Member Function. Separate
**      files are necessary because the declaration & code will
**      not be generated in the order we want; we have to keep
**      them apart to get our desired order.
*/
declFile = new ofstream( declFileName, ios::out );
if ( ! declFile )        {
    cerr << "STREAMFUNCTION: cannot open file " <<
        declFileName << " for output" << endl;
}

codeFile = new ofstream( codeFileName, ios::out );
if ( ! codeFile )        {
    cerr << "STREAMFUNCTION: cannot open file " <<
        codeFileName << " for output" << endl;
}

cout << "\n\n\n========================================" <<
    "======================================\n" << endl;
cout << "BUILDING Member Function to update DASA based on" << endl;
cout << "\tStream " << smPtr->getSECSstream() << " - Function " <<
    smPtr->getSECSfunction() << " message received from Equipment.\n\n" << endl;
cout << "When prompted by '==> ' :" << endl;
cout << "\t- enter Variable Name to save data, OR" << endl;
cout << "\t- enter 'n' to NOT save data.\n\n" << endl;



/*
**      Generate declaration & initialization for (TREE *).
*/
this->declWrite( "\tTREE        *TREE_PTR;\n" );
sprintf( genString, "\tTREE_PTR = %s->returnTree();\n\n", smTypeDecl );
this->codeWrite( genString );
```

```

        /*
        **      Walk the TREE in prefix order, prompting the user
        **      for his desires, and generating declarations & code
        **      as we go.
        */
        this->updateTreeWalk( treePtr );

        /*
        **      We're finished putting into the temporary files:
        **      close them up.
        */
        // declFile->seekp( 0L, (seek_dir) 0 );
        // codeFile->seekp( 0L, (seek_dir) 0 );
        declFile->close();
        codeFile->close();
        delete declFile;
        delete codeFile;

        /*
        **      Open up the temporary files to get input FROM them,
        **      to be written into the Class File.
        */
        declIn = new ifstream( declFileName, ios::in );
        if ( ! declFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                declFileName << " for input" << endl;
        }

        codeIn = new ifstream( codeFileName, ios::in );
        if ( ! codeFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                codeFileName << " for input" << endl;
        }

        while ( declIn->getline( genString, 80 ) )      {
            this->classWrite( genString );
            this->classWrite( "\n" );
        }

        this->classWrite( "\n\n\n" );

        while ( codeIn->getline( genString, 80 ) )      {
            this->classWrite( genString );
            this->classWrite( "\n" );
        }

        /*
        **      We're finished putting into the temporary files:
        **      close them up.
        */
        declIn->close();
        codeIn->close();
        delete declIn;
        delete codeIn;

        /*
        **      Finish-up the Class File.
        */
        this->classWrite( "};\n" );
        classFile->close();
        delete classFile;
};

/*
**      Something in here causes "Null Pointer Assignment"
*/
void STREAMFUNCTION::buildSend( int SMtype, short Stream, short Function,
TREE *TreePtr ) {
        char    className[SIZEFILE];
        char    classFileName[SIZEFILE];
        char    declFileName[SIZEFILE];
        char    codeFileName[SIZEFILE];
        char    memFuncName[20];        // Name of Member Function to be
                                        //   generated.
        char    smTypeDecl[10];         // To initialize (TREE *) in
                                        //   code to be generated.
        char    fileSuffix[10];         // Suffix of file where generated
                                        //   code will be written to.
        char    genString[80];
        ifstream        *declIn, *codeIn;

        sprintf( className, "S%dF%dC", Stream, Function );

        if ( SMtype == PRIMARY )        {
            sprintf( memFuncName, "sendPrimary" );
            sprintf( smTypeDecl, "primary" );
            sprintf( fileSuffix, "sp" );
        }
        else    {
            Function++;
            sprintf( memFuncName, "sendSecondary" );
            sprintf( smTypeDecl, "secondary" );
            sprintf( fileSuffix, "ss" );
        }

        sprintf( classFileName, "%s.%s", className, fileSuffix );

        sprintf( declFileName, "decl.tmp" );

        sprintf( codeFileName, "code.tmp" );

        /*
        **      Open the Class File and write the Member Function
        **      declaration and definition.
        */
        classFile = new ofstream( classFileName, ios::in );
        if ( ! classFile )      {
            cerr << "STREAMFUNCTION: cannot open file " <<
                classFileName << " for input" << endl;
        }

        sprintf( genString, "//\t%s() DECLARATION\n", memFuncName );
        this->classWrite( genString );
        sprintf( genString, "\tvoid\t%s();\n\n", memFuncName );
        this->classWrite( genString );
        sprintf( genString, "//\t%s() DEFINITION\n", memFuncName );
        this->classWrite( genString );
        sprintf( genString, "void %s::%s()\t{\n\n", className, memFuncName );
        this->classWrite( genString );



        /*
        **      Open temporary files to hold the declarations & code
        **      that will be internal to the Member Function. Separate
        **      files are necessary because the declaration & code will
        **      not be generated in the order we want; we have to keep
        **      them apart to get our desired order.
        */
        declFile = new ofstream( declFileName, ios::out );
        if ( ! declFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                declFileName << " for output" << endl;
        }

        codeFile = new ofstream( codeFileName, ios::out );
        if ( ! codeFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                codeFileName << " for output" << endl;
        }

        cout << "\n\n\n=======================================" <<
            "=======================================\n" << endl;
        cout << "BUILDING Member Function to send\n" <<
            "\tStream " << Stream << " - Function " <<
            Function << " message to Equipment.\n\n" << endl;
        cout << "When prompted by '==> ' :" << endl;
        cout << "\t- for LIST, enter number of Items in LIST," << endl;
        cout << "\t- for Items, enter <type number variableName>," << endl;
        cout << "\t  e.g., INT2 1 waferID.\n\n" << endl;



        /*
        **      Generate declaration & initialization for TREE and (TREE *).
        */
        this->declWrite( "\tTREE\t*TREE_ROOT, *TREE_PTR;\n" );
        this->codeWrite( "\tTREE_ROOT = TREE_PTR = new TREE;\n\n" );

        /*
        **      Walk the TREE in prefix order, prompting the user
        **      for his desires, and generating declarations & code
        **      as we go.
        */
        this->sendTreeWalk( TreePtr );

        //      CHANGE: REPLY should also be a variable passed into
        //              this MF
        if ( SMtype == PRIMARY )        {
            sprintf( genString, "\n\t%s =
                new SECSMESSAGE( %d, %d, REPLY, TREE_ROOT );\n\n",
                smTypeDecl, Stream, Function );
        }
        else    {
            sprintf( genString, "\n\t%s =
                new SECSMESSAGE( %d, %d, NOREPLY, TREE_ROOT );\n\n",
                smTypeDecl, Stream, Function );
        }
        this->codeWrite( genString );
        sprintf( genString, "\t%s->sendMessage();\n", smTypeDecl );
        this->codeWrite( genString );

        /*
        **      We're finished putting into the temporary files:
        **      close them up.
        */
        // declFile->seekp( 0L, (seek_dir) 0 );
        // codeFile->seekp( 0L, (seek_dir) 0 );
        declFile->close();
        codeFile->close();
        delete declFile;
        delete codeFile;

        /*
        **      Open up the temporary files to get input FROM them,
        **      to be written into the Class File.
        */
        declIn = new ifstream( declFileName, ios::in );
        if ( ! declFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                declFileName << " for input" << endl;
        }

        codeIn = new ifstream( codeFileName, ios::in );
        if ( ! codeFile )       {
            cerr << "STREAMFUNCTION: cannot open file " <<
                codeFileName << " for input" << endl;
        }

        while ( declIn->getline( genString, 80 ) )      {
            this->classWrite( genString );
            this->classWrite( "\n" );
        }

        this->classWrite( "\n\n\n" );

        while ( codeIn->getline( genString, 80 ) )      {
            this->classWrite( genString );
            this->classWrite( "\n" );
        }

        /*
        **      We're finished putting into the temporary files:
        **      close them up.
        */
        declIn->close();
        codeIn->close();
        delete declIn;
        delete codeIn;

        /*
        **      Finish-up the Class File.
        */
        this->classWrite( "};\n" );
        classFile->close();
        delete classFile;
};
```

```

void STREAMFUNCTION::updateTreeWalk( TREE *TreePtr )    {
        char    genString[80];

        if ( TreePtr != NULL )  {
            cout << Indent;
            this->updatePromptUser( TreePtr );

            if ( TreePtr->hasChildren() )       {
                TreePtr = TreePtr->moveDown();
                TreePtr->resetChildCursor();
                sprintf( genString, "\t%sMOVE_DOWN;\n", Indent );
                this->codeWrite( genString );
                this->indentMore();

                this->updateTreeWalk( TreePtr );

                this->indentLess();
                TreePtr = TreePtr->moveUp();
                sprintf( genString, "\t%sMOVE_UP;\n", Indent );
                this->codeWrite( genString );
            }
            else if ( TreePtr->hasRightSibling() )      {
                TreePtr = TreePtr->moveRight( 1 );
                sprintf( genString, "\t%sMOVE_RIGHT( 1 );\n", Indent );
                this->codeWrite( genString );

                this->updateTreeWalk( TreePtr );
            }
        }
};

void STREAMFUNCTION::sendTreeWalk( TREE *TreePtr )      {
        char    genString[80];

        if ( TreePtr != NULL )  {
            cout << Indent;
            this->sendPromptUser( TreePtr );

            if ( TreePtr->hasChildren() )       {
                TreePtr = TreePtr->moveDown();
                TreePtr->resetChildCursor();
                this->indentMore();

                this->sendTreeWalk( TreePtr );

                this->indentLess();
                TreePtr = TreePtr->moveUp();
                if ( ((SECSELEMENT *)TreePtr->nodeAddr())->returnLong() == 0 )  {
                    sprintf( genString, "\t%s}\n", Indent );
                    this->codeWrite( genString );
                }
            }
            else if ( TreePtr->hasRightSibling() )      {
                TreePtr = TreePtr->moveRight( 1 );

                this->sendTreeWalk( TreePtr );
            }
        }
};

void STREAMFUNCTION::updatePromptUser( TREE *TreePtr )  {
        SECSELEMENT     *sePtr;
        long            itemsInObject;
        int             i;
        char            genString[80];  // General-purpose string.
        char            typeString[80]; // String for declaration.
        char            initString[80]; // String for instantiation.

        sePtr = ((SECSELEMENT *) TreePtr->nodeAddr());
        itemsInObject = sePtr->itemsRemaining();

        sePtr->printOn( cout );

        this->printTreeLocale( TreePtr );

        if ( sePtr->isA() == slistClass )       {
            cout << endl;
        }
        else if ( sePtr->hasDouble() )  {
            cout << " ==> " << flush;
            sprintf( typeString, "\tdouble\t" );
            sprintf( initString, "= SEPTR->returnDouble();" );
        }
        else if ( sePtr->hasLong() )    {
            cout << " ==> " << flush;
            sprintf( typeString, "\tlong\t" );
            sprintf( initString, "= SEPTR->returnLong();" );
        }
        else if ( sePtr->hasString() )  {
            cout << " ==> " << flush;
            itemsInObject += 1;         // make room for NULL at end
            sprintf( typeString, "\tchar\t" );
            sprintf( initString, "SEPTR->returnString()" );
        }
        else    {
            cout << " ERROR: Object has " << itemsInObject;
            cout << " item(s) of UNKNOWN type" << endl;
        }

        if ( sePtr->isA() != slistClass )       {
            /*
```

```
**  Prompt the user for a variable name.
*/
char        variableName[80];
cin >> variableName;

if ( ( strlen( variableName ) == 1 ) &&
    ( strncmp( "n", variableName, 1 ) == 0 ) )      {
        /*
        **  DO NOTHING
        */
}
else        {
    /*
    **      Write declaration for the variable.
    */
    strcat( typeString, variableName );
    if ( itemsInObject > 1 || sePtr->hasString() )  {
        sprintf( genString, "[%d];\n", itemsInObject );
    }
    else    {
        sprintf( genString, ";\n" );
    }
    strcat( typeString, genString );
    this->declWrite( typeString );

    /*
    **      Write code to populate the variable.
    */
    if ( sePtr->hasString() )       {
        sprintf( genString, "\t%sstrncpy( %s, %s, %d );\n",
            Indent, variableName, initString, itemsInObject - 1 );
        this->codeWrite( genString );
    }
    else if ( itemsInObject > 1 )   {
          sprintf( genString, "\t%sfor ( i = 0; i < %d; i++ )\t{\n",
            Indent, itemsInObject );
          this->codeWrite( genString );
          for ( i = 0; i < itemsInObject; i++ )       {
            sprintf( genString, "\t\t%s%s[%d] %s\n",
              Indent, variableName, i, initString );
            this->codeWrite( genString );
          }
          sprintf( genString, "\t%s}\n", Indent );
          this->codeWrite( genString );
        }
        else  {
          sprintf( genString, "\t%s%s %s\n",
            Indent, variableName, initString );
          this->codeWrite( genString );
        }
      }
    }
};

void STREAMFUNCTION::sendPromptUser( TREE *TreePtr )  {
    SECSELEMENT     *sePtr;
    long            itemsInObject;
    int             i;
    char            genString[80];  // General-purpose string.
    char            typeString[80]; // String for declaration.
    char            initString[80]; // String for instantiation.

    char            seType[10];
    int             length;
    char            variableName[30];

    sePtr = ((SECSELEMENT *) TreePtr->nodeAddr());
    itemsInObject = sePtr->itemsRemaining();
```

```

        if ( sePtr->isA() == slistClass )       {
            long        numberOfChildren;

            if ( this->variableList( TreePtr ) )        {
                cout << "<list>";
            }
            else        {
                sePtr->printOn( cout );
            }

            this->printTreeLocale( TreePtr );

            //  Prompt user for number of items they want in LIST:
            //  an answer of 0 items in LIST will effectively
            //  "prune" the TREE at this point.
            //
            cout << " items in LIST ==> " << flush;
            cin >> numberOfChildren;
            sprintf( genString, "\t%sADD_LIST_AND_MOVEDOWN( %ldL );\n",
                Indent, numberOfChildren );
            this->codeWrite( genString );

            if ( numberOfChildren == 0 )        {
                /*
                **      Delete children of the Model we're tracking.
                */
                while ( TreePtr->getItemsInContainer() != 0 )   {
                    TreePtr->destroyFromTail( TreePtr->peekAtTail() );
                }
                TreePtr->setChildCnt( 0 );

                /*
                **      This is an empty LIST --- move back up in
                **      the code we're generating.
                */
                sprintf( genString, "\t%sMOVEUP();\n", Indent );
                this->codeWrite( genString );
            }
            else if ( this->variableList( TreePtr ) && numberOfChildren > 1 )   {
                //      So we'll know size of array to declare for children.
                TreePtr->setChildCnt( numberOfChildren );
                sprintf( genString, "\t%sTREE\t*TREE_MARK = TREE_PTR;\n",
                    Indent );
                this->codeWrite( genString );
                sprintf( genString, "\t%sfor ( int i = 0; i < %d; i++ )\t{\n",
                    Indent, numberOfChildren );
                this->codeWrite( genString );
                sprintf( genString, "\t%sTREE_PTR = TREE_MARK;\n", Indent );
                this->codeWrite( genString );
            }
        }
        else if ( sePtr->hasDouble() )  {
            cout << "<double>";
            this->printTreeLocale( TreePtr );
            cout << " ==> " << flush;
            sprintf( typeString, "\tdouble\t" );
            this->getVariable( seType, &length, variableName );
            strcat( typeString, variableName );
            if ( this->parentVariableList( TreePtr ) )  {
                TREE    *tempTree = TreePtr->moveUp();
                sprintf( genString, "[%d];\n", tempTree->getChildCnt() );
                strcat( typeString, genString );
                sprintf( genString, "\t%sADD_%s( 1L, &%s[i] );\n",
                    Indent, seType, variableName );
            }
            else        {
                sprintf( genString, "[%d];\n", length );
                strcat( typeString, genString );
                sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
                    Indent, seType, length, variableName );
            }
            this->declWrite( typeString );
            this->codeWrite( genString );
        }
        else if ( sePtr->hasLong() )    {
            cout << "<long>";
            this->printTreeLocale( TreePtr );
            cout << " ==> " << flush;
            sprintf( typeString, "\tlong\t" );
            this->getVariable( seType, &length, variableName );
            strcat( typeString, variableName );
            if ( this->parentVariableList( TreePtr ) )  {
                TREE    *tempTree = TreePtr->moveUp();
                sprintf( genString, "[%d];\n", tempTree->getChildCnt() );
                tempTree->setChildCnt( 1 );
                strcat( typeString, genString );
                sprintf( genString, "\t%sADD_%s( 1L, &%s[i] );\n",
                    Indent, seType, variableName );
            }
            else        {
                sprintf( genString, "[%d];\n", length );
                strcat( typeString, genString );
                sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
                    Indent, seType, length, variableName );
            }
            this->declWrite( typeString );
            this->codeWrite( genString );
        }
        else if ( sePtr->hasString() )  {
            cout << "<string>";
            this->printTreeLocale( TreePtr );
            cout << " ==> " << flush;
            itemsInObject += 1;          // make room for NULL at end
            sprintf( typeString, "\tchar\t" );
            this->getVariable( seType, &length, variableName );
            strcat( typeString, variableName );
            sprintf( genString, "[%d];\n", length + 1 );
            strcat( typeString, genString );
            this->declWrite( typeString );
            sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
                Indent, seType, length, variableName );
            this->codeWrite( genString );
        }
        else    {
            cout << "<?????>";
            this->printTreeLocale( TreePtr );
            cout << " ERROR: Object has " << itemsInObject;
            cout << " item(s) of UNKNOWN type" << endl;
        }
};

void STREAMFUNCTION::classWrite( char *String ) {
        int     stringLength = strlen( String );

        classFile->write( String, stringLength );
};

void STREAMFUNCTION::declWrite( char *String )  {
        int     stringLength = strlen( String );

        declFile->write( String, stringLength );
};

void STREAMFUNCTION::codeWrite( char *String )  {
        int     stringLength = strlen( String );

        codeFile->write( String, stringLength );
};

void STREAMFUNCTION::printTreeLocale( TREE *TreePtr )    {
        cout << "\t" << TreePtr->getLevel() << ":" <<
            TreePtr->getParentsChildCursor();
};

void STREAMFUNCTION::getVariable( char *Type, int *Length, char *Name ) {
        cin >> Type;
        cin >> *Length;
        cin >> Name;
};

int STREAMFUNCTION::variableList( TREE *TreePtr )       {
        return ( ((SECSELEMENT *) TreePtr->nodeAddr())->returnLong() == 0 );
};

int STREAMFUNCTION::parentVariableList( TREE *TreePtr ) {
        if ( TreePtr->atRoot() )        {
            return 1;
        }
        else    {
            return( this->variableList( TreePtr->moveUp() ) );
        }
};

void STREAMFUNCTION::indentMore()       {
        Indent[IndentCnt] = ' ';
        IndentCnt += IndentInc;
        Indent[IndentCnt] = '\0';
};

```

```
void STREAMFUNCTION::indentLess()        {
        Indent[IndentCnt] = ' ';
        IndentCnt -= IndentInc;
        Indent[IndentCnt] = '\0';
};

int STREAMFUNCTION::getStream() {
        return primary->getSECSstream();
};

int STREAMFUNCTION::getPrimaryFunction()        {
        return primary->getSECSfunction();
};

int STREAMFUNCTION::getSecondaryFunction()      {
        return ( primary->getSECSfunction() + 1 );
};

void STREAMFUNCTION::printOn( ostream& outputStream ) const      {
        primary->printOn( outputStream );
};
```

```
/*
**      class:  TREE    -       DECLARATIONS
**
**              @(#)tree.h      13.6    11/12/91
*/

class TREE : public DoubleList  {
protected:
        Object  *node;
        TREE    *parent;
        int     level;
        int     childCursor;
        int     childCnt;
        char    *className;
        classType       classTypeValue;
public:
        TREE( );
        TREE( Object& Node );
        ~TREE();
        classType       isA() const;
        char    *nameOf() const;
        int     atRoot();
        int     mayBeParent();
        int     hasChildren();
        int     hasRightSibling();
        int     hasLeftSibling();
        TREE    *addChild( TREE& Child );
        void    setParent( TREE *Parent );
        TREE    *moveUp();
        TREE    *moveDown();
        TREE    *moveRight( int MoveNodes );
        TREE    *moveLeft( int MoveNodes );
        int     getLevel();
        void    setLevel( int Level );
```

```
                void    setChildCnt( int ChildCnt );
                void    resetChildCursor();
                int     getChildCnt();
                int     gotAllChildren();
                int     getChildCursor();
                int     getParentsChildCursor();
                Object  *nodeAddr();
                virtual void            printOn( ostream& ) const;
        };
```

```
/*
**      class:  TREE    -       MEMBER FUNCTIONS
**
**                  @(#)tree.cpp    13.8    11/12/91
*/

#include <stddef.h>
#include <string.h>
#include <iostream.h>
#include <alloc.h>
#include <mem.h>
#include <object.h>
#include <dbllist.h>
#include <clstypes.h>
#include <custom.h>
#include <tree.h>

#define INDENTSTART     5
#define INDENTPERLEVEL  2

/*
**      The first constructor is used to create an empty TREE -
**      this will be the "ROOT subtree". This empty TREE object
**      must be created so that the member functions to add nodes
**      (e.g., addChild()) can be employed - you have to have a
**      TREE before you can use its member functions !!!
**
**      A client of the TREE class would typically use the first
**      constructor to create the "ROOT subtree", then use the
**      second constructor for all other subtrees.
**
**      The ROOT of the "ROOT subtree" (i.e., the ROOT of the
**      entire tree) is grafted to the initial empty TREE:
**      a member function such as addChild() sees that the
**      "node" is empty & grafts the "node" of the passed TREE
**      to the empty "node".
*/

TREE::TREE( )   {
        node = NULL;
        parent = NULL;
        level = 0;
        childCnt = 0;
        childCursor = 0;
        className = "TREE";
        classTypeValue = treeClass;
};

TREE::TREE( Object& Node )      {
        node = &Node;
        parent = NULL;
        childCnt = 0;
        childCursor = 0;
        className = "TREE";
        classTypeValue = treeClass;
};

TREE::~TREE()   {
        if ( node != NULL )     {
            delete node;
        }
};

classType TREE::isA() const     {
        return classTypeValue;
};

char *TREE::nameOf() const      {
        return className;
};

int TREE::atRoot()      {
        return ( parent == NULL );
};

/*
**      HAVE TO FLESH THIS GUY OUT --- USE A (LIST *) AS
**      MEMBER DATA, ADD THE THING PASSED HERE TO THE LIST.
**
**      USE COLLECTION MEMBER FUNCTION hasMember() IN THE
**      mayBeParent() member function below.
**
void TREE::addParentType( char* ClassName )     {
};
*/

int TREE::mayBeParent() {
        return ( strcmp( node->nameOf(), "LIST" ) ? FALSE : TRUE );
};

int TREE::hasChildren() {
        return ( this->isEmpty() ? FALSE : TRUE );
};

int TREE::hasRightSibling()     {
        if ( this->atRoot() )   {
            return FALSE;
        }
        else if ( ( parent->childCnt - parent->childCursor ) < 1 )      {
            return FALSE;
        }
        else    {
                return TRUE;
            }
};

int TREE::hasLeftSibling()      {
        if ( this->atRoot() )   {
            return FALSE;
        }
        else if ( parent->childCursor < 2 )      {
            return FALSE;
        }
        else    {
            return TRUE;
        }
};

/*
**      Return (TREE *) so that client of TREE Class can move
**      around within super-TREE.
*/
TREE *TREE::addChild( TREE& Child )     {
        if ( node == NULL )     {
            /*
            **          ENHANCEMENT:
            **
            **  Could we point "this" to the Child, basically
            **  overlaying the root node with the child ???
            **
            **          this = &Child;   DID NOT WORK
            */
            /*
            **  There is no root node, so make the incoming TREE
            **  the root node.
            */
```

```
      node = Child.nodeAddr();
      //
      //  Is this setting the level of the orphaned child ?
      //
      Child.setLevel( 0 );
      return this;
    }
    else    {
      childCursor++;
      this->addAtTail( Child );
      Child.setParent( this );
      Child.setLevel( this->getLevel() + 1 );
      if ( Child.mayBeParent() )  {
        return &Child;
      }
      else        {
        return this;
      }
    }
};

void TREE::setParent( TREE *Parent )    {
    parent = Parent;
};

/*
**      Returns pointer to ROOT of super-TREE
**
TREE *TREE::toRoot()    {
    if ( ! atRoot() )       {
      return( parent->toRoot() );
    }
    else    {
      return this;
```

```
        }
};
*/

TREE *TREE::moveUp()    {
        if ( ! this->atRoot() ) {
            return parent;
        }
        return this;
};

TREE *TREE::moveDown()  {
        /*
        **      The cursor is always at the left-most child
        **      when we moveDown().
        */
        if ( this->hasChildren() )      {
            childCursor = 1;
            return (TREE *) &( this->peekAtHead() );
        }
        return this;
};

/*
**      moveRight() & moveLeft() are almost identical -- need to
**      combine them into one member function, perhaps called
**      by two "tailoring" functions, or perhaps called with
**      a LEFT | RIGHT flag.
*/
TREE *TREE::moveRight( int MoveNodes )  {
        int     tmpChildCursor;

        if ( this->atRoot() )   {
            cout << "moveRight(): at ROOT, cannot move" << endl;
        }
        else    {
            /*
            **  Make sure the requested move is valid; if it is not,
            **  return the address of the "current" node.
            */
            if ( parent->childCursor + MoveNodes > parent->childCnt )    {
                cout << "moveRight(): cannot move " << MoveNodes <<
                    " nodes" << endl;
                return this;
            }

            /*
            **  Initialize an iterator to the start of the list of
            **  the "current" node's parent's children. The fact
            **  that the "current" node may not be at the start of
            **  the list is a accounted for below.
            */
            DoubleListIterator childRowIter( (DoubleList &) *(this->parent), 1 );

            /*
            **  We want to move right from the start of the list of
            **  children to the "current" node PLUS the number of
            **  nodes we've been asked to move.
            */
            parent->childCursor += MoveNodes;
            tmpChildCursor = parent->childCursor;
            while ( tmpChildCursor-- > 1 )      {
                if ( childRowIter == 0 )        {
                    cout << "moveRight(): at end of list" << endl;
                    return this;
                }
                /*
                **      Move right one child.
                */
                childRowIter++;
            }

            /*
            **  Return a pointer to the new "current" node.
            */
            return (TREE *) &((Object&)(childRowIter));
        }
        return this;
};

TREE *TREE::moveLeft( int MoveNodes )    {
        int     tmpChildCursor;

        if ( this->atRoot() )    {
            cout << "moveLeft(): at ROOT, cannot move" << endl;
        }
        else    {
            if ( parent->childCursor - MoveNodes < 1 )  {
                cout << "moveLeft(): cannot move " << MoveNodes <<
                    " nodes" << endl;
                return this;
            }

            DoubleListIterator childRowIter( (DoubleList &) *(this->parent), 1 );

            parent->childCursor -= MoveNodes;
            tmpChildCursor = parent->childCursor;
            while ( tmpChildCursor-- > 1 )      {
                if ( childRowIter == 0 )        {
                    cout << "moveLeft(): at end of list" << endl;
                    return this;
                }
                childRowIter++;
            }

            return (TREE *) &((Object&)(childRowIter));
        }
        return this;
};

/*
void TREE::forAllChildren()     {
        return level;
};
*/

int TREE::getLevel()    {
        return level;
};

void TREE::setLevel( int Level )        {
        level = Level;
};

void TREE::setChildCnt( int ChildCnt )  {
        childCnt = ChildCnt;
};

void TREE::resetChildCursor()   {
        childCursor = 0;
};

int TREE::getChildCnt() {
        return childCnt;
};

```

```
int TREE::gotAllChildren()      {
        return ( childCnt == childCursor );
};

int TREE::getChildCursor()      {
        return childCursor;
};

int TREE::getParentsChildCursor()       {
        if ( ! this->atRoot() ) {
            return parent->childCursor;
        }
        else    {
            return 0;
        }
};

Object *TREE::nodeAddr()        {
        return node;
};

void TREE::printOn( ostream& outputStream ) const       {
        int     curIndent, nxtIndent;
        char    *indent;

        curIndent = (INDENTPERLEVEL * level) + INDENTSTART;
        nxtIndent = curIndent +  INDENTPERLEVEL;
        indent = (char *) malloc( nxtIndent + 1 );
        memset( indent, ' ', nxtIndent );
        indent[nxtIndent] = '\0';

        indent[curIndent] = '\0';
        cout << indent;

        if ( node == NULL )     {
            outputStream << "<empty>" << endl;
        }
        else    {
            /*
            **  Print the "current" node.
            */
            node->printOn( outputStream );
            outputStream << endl;

            indent[curIndent] = ' ';

            /* TREE needs to add a method for node classes to identify
            themselves as potential PARENT node & a method to check
            if a given node is one of those:
            if ( this->parentNode( node->isA() ) )      {
            */
                /*
                **      If there are any children, print them.
                */
                DoubleListIterator childRowIter( (DoubleList &) *this, 1 );
                while ( childRowIter != 0 )     {
                    if ( ((Object&)(childRowIter)).isA() != treeClass ) {
                        cout << indent;
                    }
                    ((Object&)(childRowIter)).printOn( outputStream );
                    childRowIter++;
                }
            /* } */

        }
        free( indent );
};
```

```
/*
**      class:  UINT1   -       DECLARATIONS
**
**              @(#)uint1.h     13.5    11/12/91
*/

class UINT1 : public SECSELEMENT        {
protected:
        unsigned char   *data;
        unsigned char   *ptr;
public:
        UINT1( unsigned long Length, unsigned char *Uint1 );
        UINT1( unsigned long Items, long DataArray[] );
        ~UINT1();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  UINT1   -       MEMBER FUNCTIONS
**
**              @(#)uint1.cpp   13.6    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <uint1.h>

UINT1::UINT1( unsigned long Length, unsigned char *Uint1 ) :
    SECSELEMENT( "UINT1", uint1Class )  {
        int     i;

        length = remainingBytes = Length;
        data = ptr = new unsigned char[length];
        itemCode = UINT1_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = *Uint1++;
        }
};

UINT1::UINT1( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "UINT1", uint1Class )  {
        int     i;

        length = remainingBytes = Items * UINT1_BYTES;
        data = ptr = new unsigned char[length];
        itemCode = UINT1_ITEM;

        for ( i = 0; i < length; i++ )  {
            data[i] = (unsigned char ) DataArray[i];
        }
};

UINT1::~UINT1() {
        delete data;
};

short UINT1::bytesPerItem()     {
        return UINT1_BYTES;
};

unsigned char *UINT1::stringForMessage()        {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
```

```
          delete( itemHeader );

          /*
          **      Point to first byte of actual data to be returned,
          **      then copy the data in.
          */
          ptrOut += headerLength;
          for ( i = 0; i < length; i++ )  {
              *ptrOut++ = data[i];
          }

          return ptrSave;
};

int UINT1::hasLong()    {
          remainingBytes = length;
          ptr = data;
          return 1;
};

long UINT1::returnLong()        {
          if ( remainingBytes <= 0 )      {
              cerr << "UINT1: trying to get data after end of contents" << endl;
              exit( 1 );
          }
          else    {
              long        tmp = (long) *ptr;
              ptr += UINT1_BYTES;
              remainingBytes -= UINT1_BYTES;
              return tmp;
          }
};

void UINT1::printOn( ostream& outputStream ) const      {
        int     i, perLine = 0;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < length; i++, perLine++ )        {
            outputStream.width( 4 );
            outputStream.fill( ' ' );
            outputStream.setf( ios::right, ios::adjustfield );
            outputStream << dec << (int) data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  UINT2   -       DECLARATIONS
**
**              @(#)uint2.h     13.4    11/12/91
*/

class UINT2 : public SECSELEMENT        {
protected:
        unsigned short  *data;
        unsigned short  *ptr;
public:
        UINT2( unsigned long Length, unsigned char *Uint2 );
        UINT2( unsigned long Items, long DataArray[] );
        ~UINT2();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  UINT2   -       MEMBER FUNCTIONS
**
**              @(#)uint2.cpp   13.7    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <uint2.h>

UINT2::UINT2( unsigned long Length, unsigned char *Uint2 ) :
    SECSELEMENT( "UINT2", uint2Class )  {
        int     i, j, items = Length / UINT2_BYTES;

        length = remainingBytes = Length;
        data = ptr = new unsigned short[items];
        itemCode = UINT2_ITEM;

        for ( i = 0; i < items; i++ )   {
            data[i] = 0;
            for ( j = 0; j < sizeof(unsigned short); j++, Uint2++ )     {
                    data[i] = (data[i] << 8) + *Uint2;
            }
        }
};

UINT2::UINT2( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "UINT2", uint2Class )  {
        int     i;

```

```
		length = remainingBytes = Items * UINT2_BYTES;
		data = ptr = new unsigned short[Items];
		itemCode = UINT2_ITEM;

		for ( i = 0; i < Items; i++ )	{
			data[i] = (unsigned short) DataArray[i];
		}
};

UINT2::~UINT2() {
		delete data;
};

short UINT2::bytesPerItem()	{
		return UINT2_BYTES;
};

unsigned char *UINT2::stringForMessage()	{
		unsigned char	*ptrOut, *ptrSave;
		unsigned char	*itemHeader;
		long	i, items, itemLength;
		short	headerLength;

		/*
		**	Create space to return the "SECS Message ready"
		**	version of this item.
		*/
		itemLength = this->lengthInMessage();
		ptrOut = new unsigned char[itemLength];
		ptrSave = ptrOut;

		/*
		**	Get the Item Header and put it in the
		**	return string.
```

```
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;
        items = length / UINT2_BYTES;
        for ( i = 0; i < items; i++ )   {
            *ptrOut++ = data[i] >> 8 & 0xFF;
            *ptrOut++ = data[i] & 0xFF;
        }

        return ptrSave;
};

int UINT2::hasLong()    {
        remainingBytes = length;
        ptr = data;
        return 1;
};

long UINT2::returnLong()        {
        if ( remainingBytes <= 0 )      {
            cerr << "UINT2: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            long        tmp = (long) *ptr;
            ptr++;
            remainingBytes -= UINT2_BYTES;
            return tmp;
        }
};

void UINT2::printOn( ostream& outputStream ) const      {
        int     i, perLine = 0;
        int     items = length / UINT2_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << dec << data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  UINT4   -       DECLARATIONS
**
**              @(#)uint4.h     13.4    11/12/91
*/

class UINT4 : public SECSELEMENT        {
protected:
        unsigned long   *data;
        unsigned long   *ptr;
public:
        UINT4( unsigned long Length, unsigned char *Uint4 );
        UINT4( unsigned long Items, long DataArray[] );
        ~UINT4();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasLong();
        long            returnLong();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  UINT4   -       MEMBER FUNCTIONS
**
**              @(#)uint4.cpp   13.7    11/12/91
*/

#include <stdlib.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <uint4.h>

UINT4::UINT4( unsigned long Length, unsigned char *Uint4 ) :
    SECSELEMENT( "UINT4", uint4Class )  {
        int     i, j, items = Length / UINT4_BYTES;

        length = remainingBytes = Length;
        data = ptr = new unsigned long[items];
        itemCode = UINT4_ITEM;

        for ( i = 0; i < items; i++ )   {
            data[i] = 0;
            for ( j = 0; j < sizeof(unsigned long); j++, Uint4++ )      {
                    data[i] = (data[i] << 8) + *Uint4;
            }
        }
};

UINT4::UINT4( unsigned long Items, long DataArray[] ) :
    SECSELEMENT( "UINT4", uint4Class )  {
        int     i;

        length = remainingBytes = Items * UINT4_BYTES;
        data = ptr = new unsigned long[Items];
        itemCode = UINT4_ITEM;

        for ( i = 0; i < Items; i++ )    {
            data[i] = (unsigned long) DataArray[i];
        }
};

UINT4::~UINT4() {
        delete data;
};

short UINT4::bytesPerItem()     {
        return UINT4_BYTES;
};

unsigned char *UINT4::stringForMessage()        {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader;
        long    i, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
```

```
         */
         itemHeader = this->getItemHeader( &headerLength, itemHeader );
         memcpy( ptrOut, itemHeader, headerLength );
         delete( itemHeader );

         /*
         **      Point to first byte of actual data to be returned,
         **      then copy the data in.
         */
         ptrOut += headerLength;
         items = length / UINT4_BYTES;
         for ( i = 0; i < items; i++ )   {
             *ptrOut++ = data[i] >> 24 & 0xFF;
             *ptrOut++ = data[i] >> 16 & 0xFF;
             *ptrOut++ = data[i] >> 8 & 0xFF;
             *ptrOut++ = data[i] & 0xFF;
         }

         return ptrSave;
};

int UINT4::hasLong()    {
         remainingBytes = length;
         ptr = data;
         return 1;
};

long UINT4::returnLong()        {
         if ( remainingBytes <= 0 )      {
             cerr << "UINT4: trying to get data after end of contents" << endl;
             exit( 1 );
         }
         else    {
             long        tmp = (long) *ptr;
```

```
            ptr++;
            remainingBytes -= UINT4_BYTES;
            return tmp;
        }
};

void UINT4::printOn( ostream& outputStream ) const      {
        int     i, perLine = 0;
        int     items = length / UINT4_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )        {
            outputStream << " " << dec << data[i];
            if ( perLine >= 10 )        {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

```
/*
**      class:  UINT8   -       DECLARATIONS
**
**              @(#)uint8.h     13.4    11/12/91
*/

class UINT8 : public SECSELEMENT        {
protected:
        double  *data;
        double  *ptr;
public:
        UINT8( unsigned long Length, unsigned char *Uint8 );
        UINT8( unsigned long Items, double DataArray[] );
        ~UINT8();

        short           bytesPerItem();

        unsigned char   *stringForMessage();

        int             hasDouble();
        double          returnDouble();

        virtual void    printOn( ostream& ) const;
};
```

```
/*
**      class:  UINT8   -       MEMBER FUNCTIONS
**
**              @(#)uint8.cpp   13.7    11/12/91
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <object.h>
#include <clstypes.h>
#include <custom.h>
#include <secselem.h>
#include <uint8.h>

UINT8::UINT8( unsigned long Length, unsigned char *Uint8 ) :
    SECSELEMENT( "UINT8", uint8Class )  {
        short   i, j, items = Length / UINT8_BYTES;

        if ( sizeof(double) != UINT8_BYTES )    {
            cerr << "UINT8: C++ type does not match length of UINT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Length;
        data = ptr = new double[items];
        itemCode = UINT8_ITEM;

        for ( i = 0; i < items; i++ )   {
            long                mostSignif = 0;
            unsigned long       leastSignif = 0;

            for ( j = 0; j < sizeof(long); j++, Uint8++ )       {
                leastSignif = (leastSignif << 8) + *Uint8;
                mostSignif = (mostSignif << 8) + *(Uint8 + sizeof(long));
            }

            data[i] = (mostSignif * 4294967295) + leastSignif;
        }
};

UINT8::UINT8( unsigned long Items, double DataArray[] ) :
    SECSELEMENT( "UINT8", uint8Class )  {
        short   i;

        if ( sizeof(double) != UINT8_BYTES )    {
            cerr << "UINT8: C++ type does not match length of UINT8" << endl;
            exit( 1 );
        }

        length = remainingBytes = Items * UINT8_BYTES;
        data = ptr = new double[Items];
        itemCode = UINT8_ITEM;

        for ( i = 0; i < Items; i++ )   {
            data[i] = DataArray[i];
        }

};

UINT8::~UINT8() {
        delete data;
};

short UINT8::bytesPerItem()     {
        return UINT8_BYTES;
};
```

```

unsigned char *UINT8::stringForMessage()        {
        unsigned char   *ptrOut, *ptrSave;
        unsigned char   *itemHeader, *reverseFloat;
        long    i, j, items, itemLength;
        short   headerLength;

        /*
        **      Create space to return the "SECS Message ready"
        **      version of this item.
        */
        itemLength = this->lengthInMessage();
        ptrOut = new unsigned char[itemLength];
        ptrSave = ptrOut;

        /*
        **      Get the Item Header and put it in the
        **      return string.
        */
        itemHeader = this->getItemHeader( &headerLength, itemHeader );
        memcpy( ptrOut, itemHeader, headerLength );
        delete( itemHeader );

        /*
        **      Point to first byte of actual data to be returned,
        **      then copy the data in.
        */
        ptrOut += headerLength;

        items = length / UINT8_BYTES;
        for ( i = 0; i < items; i++ )   {
            long                mostSignif;
            unsigned long       leastSignif;

            mostSignif = data[i] / 4294967295;
            leastSignif = data[i] - (mostSignif * 4294967295);

            *ptrOut++ = leastSignif >> 24 & 0xFF;
            *ptrOut++ = leastSignif >> 16 & 0xFF;
            *ptrOut++ = leastSignif >> 8 & 0xFF;
            *ptrOut++ = leastSignif & 0xFF;

            *ptrOut++ = mostSignif >> 24 & 0xFF;
            *ptrOut++ = mostSignif >> 16 & 0xFF;
            *ptrOut++ = mostSignif >> 8 & 0xFF;
            *ptrOut++ = mostSignif & 0xFF;
        }

        return ptrSave;
};

int UINT8::hasDouble()  {
        remainingBytes = length;
        ptr = data;
        return 1;
};

double UINT8::returnDouble()    {
        if ( remainingBytes <= 0 )      {
            cerr << "UINT8: trying to get data after end of contents" << endl;
            exit( 1 );
        }
        else    {
            double      tmp = *ptr;
            ptr += UINT8_BYTES;
            remainingBytes -= UINT8_BYTES;
            return tmp;
        }
```

```
};

void UINT8::printOn( ostream& outputStream ) const          {
        int     i, perLine = 0;
        int     items = length / UINT8_BYTES;

        outputStream << '<' << this->nameOf();

        for ( i = 0; i < items; i++, perLine++ )            {
            outputStream << " " << data[i];
            if ( perLine >= 4 ) {
                perLine = 0;
                outputStream << endl;
            }
        }

        outputStream << ">" << flush;
};
```

APPENDIX B: Vita

Don Breisch was born on April 5, 1958 in Allentown, Pennsylvania to Charles F. and Natalie V. (Wagner) Breisch. He received a B.S. in Industrial Engineering from Lehigh University in 1980. On the Dean's List thrice, Mr. Breisch received the Bethlehem Fabricator's Award for most improved engineering student from his freshman to senior years. Since 1980, he has worked as a software engineer for Arthur Andersen, Hewlett-Packard, Western Electric, AT&T Network Systems, and AT&T Microelectronics. Mr. Breisch was part of the Tactical Cell Controller project, a DASA done for SEMATECH in 1990. He is currently responsible for worldwide computer network administration and design for AT&T Microelectronics.