

Theses and Dissertations

1992

Object-oriented interface to SECS-II

Don Breisch
Lehigh University

Follow this and additional works at: <https://preserve.lehigh.edu/etd>



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Breisch, Don, "Object-oriented interface to SECS-II" (1992). *Theses and Dissertations*. 5535.
<https://preserve.lehigh.edu/etd/5535>

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

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 1981
Date

M Spezialetti
Professor M. Spezialetti
Thesis Advisor

Kenneth J. G.
Chairman of Department

CONTENTS

ABSTRACT	1
1. OVERVIEW	3
1.1 The Need for SECS-I and SECS-II	4
1.2 DASA Development	6
1.3 SECS-II Internals	7
1.4 SECS-II Message Example	14
1.5 The Importance of Programming Ease in a SECS-II Interface.	16
2. DESIGN GOALS	18
2.1 GOAL 1: Reduce the Number of Interface Data Types	18
2.2 GOAL 2: Streamline the Process of Agreeing with Equipment on Message Form	19
2.3 GOAL 3: Make It Easier to Build SECS Messages to Send to the Equipment	20
2.4 GOAL 4: Make It Easier to Parse SECS Messages to Received from the Equipment	22
2.5 GOAL 5: Isolate Non-portable Code within SECS-II Data Types	23
2.6 Summary	24
3. OBJECT-ORIENTED PROGRAMMING PARADIGM	25
3.1 Example of a Class	25
3.2 Class Inheritance	26
3.3 Object-oriented Programming and the SECS-II Interface	27
4. CLASS ORGANIZATION	28
4.1 TREE Class	30
4.2 SECSELEMENT Classes	30
4.3 SECSMESSAGE Class	30
4.4 STREAM-FUNCTION Classes	31
5. INTERACTIVE MEMBER FUNCTION BUILDING WITH THE OOSI	33
5.1 Customizing Member Functions to Parse Messages from Equipment	33
5.2 Customizing Member Functions to Send Messages to Equipment	37
6. TREE CLASS	42
6.1 Member Data	42
6.2 Constructors	43
6.3 Node-Adding Member Functions	43
6.4 Movement Member Functions	43
6.5 Boolean Member Functions	44
7. SECSELEMENT CLASSES	45
7.1 SECSELEMENT Member Data	46
7.2 SECSELEMENT Constructor	46
7.3 Identification Member Functions	47
7.4 Member Functions to Build SECS-II Message	47
7.5 Boolean Member Functions	47
7.6 Data Access Member Functions	48
7.7 SECSELEMENT Subclasses	48

8. SECSMESSAGE CLASS	50
8.1 Member Data	50
8.2 Constructors	50
8.3 Member Functions for Internal Use by SECSMESSAGE	51
8.4 Member Functions for SECSMESSAGE Client Use	51
9. STREAM-FUNCTION CLASSES	53
9.1 STREAMFUNCTION Member Data	55
9.2 STREAMFUNCTION Constructors	55
9.3 Member Functions for Internal Use by STREAMFUNCTION	56
9.4 Specific Stream-Function Classes	56
9.5 Customized Stream-Function Classes	57
10. CONCLUSION	58
BIBLIOGRAPHY	59
APPENDIX A: Source Code for the Object-oriented SECS-II Interface	60
APPENDIX B: Vita	242

LIST OF FIGURES

Figure 1. Semiconductor Cleanroom Computer System	4
Figure 2. Software Layers on Automation Cell Computer	5
Figure 3. Byte Detail in One Block of a SECS-II Message	8
Figure 4. SECS-II Streams	10
Figure 5. SECS-II Message as a Tree Data Structure	12
Figure 6. SECS-II Data Types	13
Figure 7. Stream 6 Function 9 SEMI Specification	14
Figure 8. Stream 6 Function 9 SEMI Specification as a Tree	15
Figure 9. Mapping SECS-II Data Types to C Data Types	19
Figure 10. C Code to Build SECS-II Message	21
Figure 11. Tree Representation of Stream 6 Function 3 Message	22
Figure 12. C Code to Access Target Node in Figure 11	23
Figure 13. Object-Oriented SECS-II Interface Class Organization	28
Figure 14. Borland C++ Class Library	29
Figure 15. Sample Session to Build S6-F9 Member Function	35
Figure 16. Stream 6 Function 9 Tree with Variable Names	35
Figure 17. Code Generated by S6-F9 Session	37
Figure 18. Sample Session to Build S2-F23 Member Function	39
Figure 19. Stream 2 Function 23 Tree with Variable Names	40
Figure 20. Code Generated by S2-F23 Session	41
Figure 21. Mapping SECS-II Data Types to OOSI Types	46
Figure 22. Member Functions that Build Customized S1F1C Class	53

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-II¹ 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.

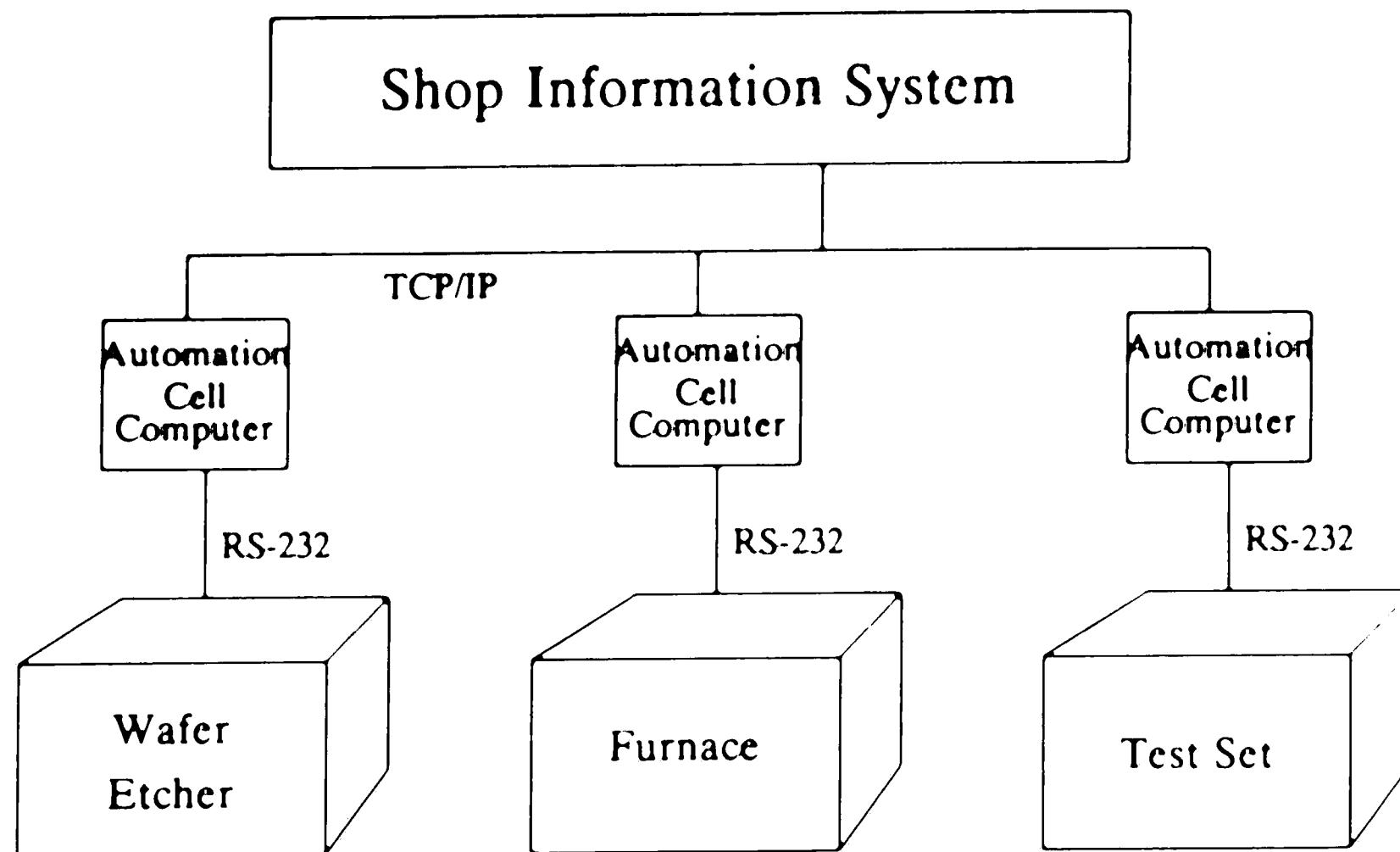


Figure 1. Semiconductor Cleanroom Computer System

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.

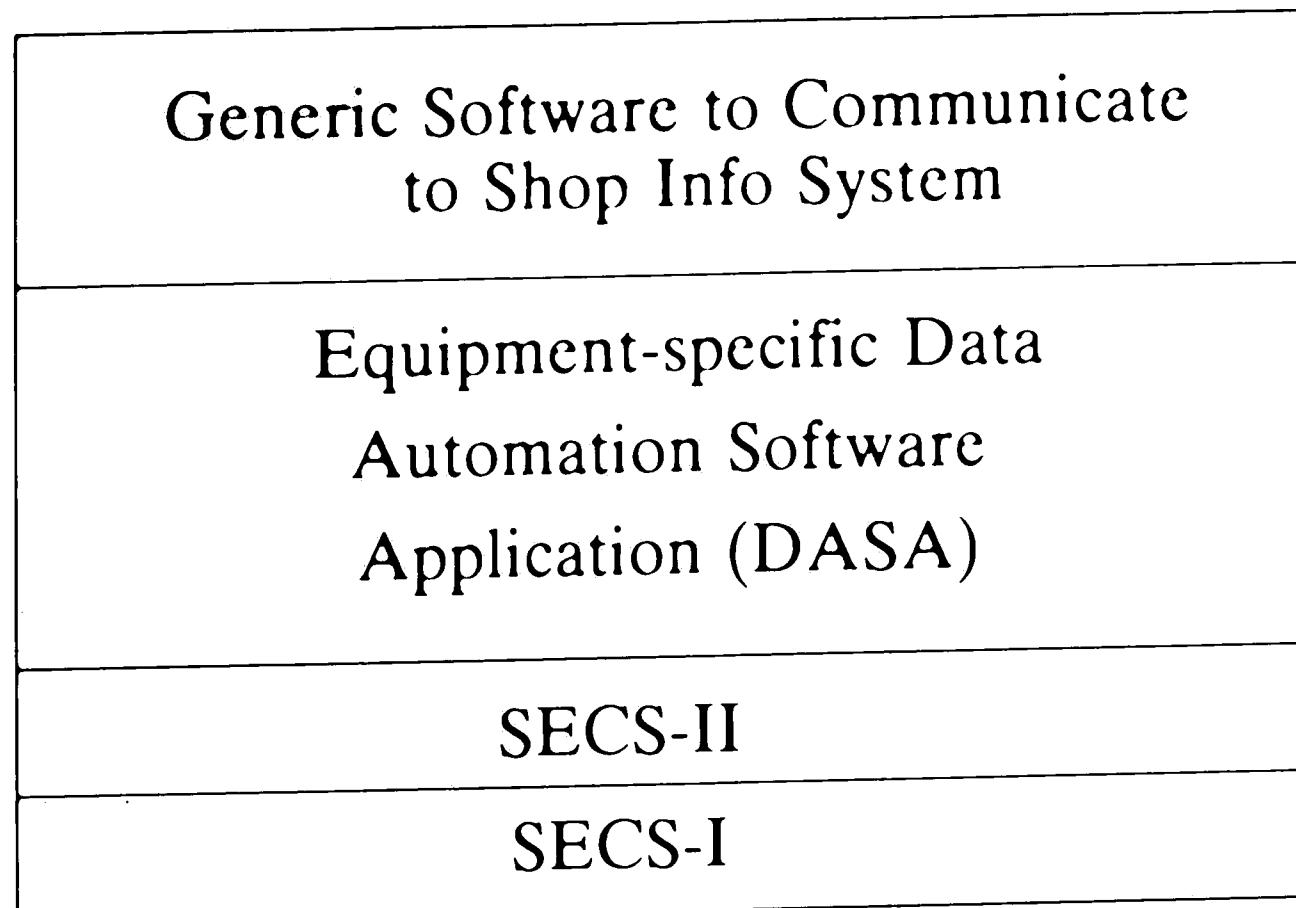


Figure 2. Software Layers on Automation Cell Computer

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.

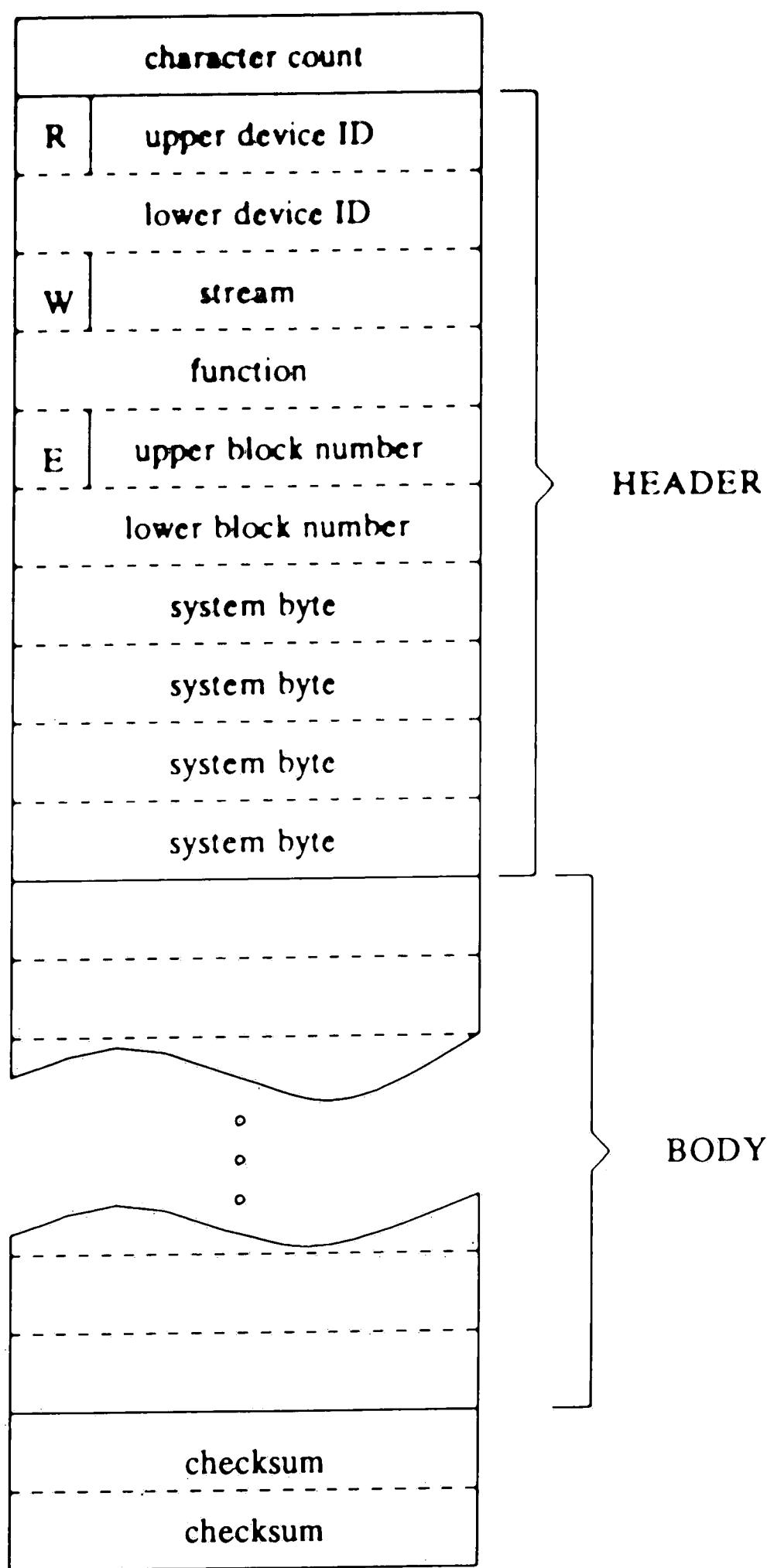


Figure 3. Byte Detail in One Block of a SECS-II Message

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.

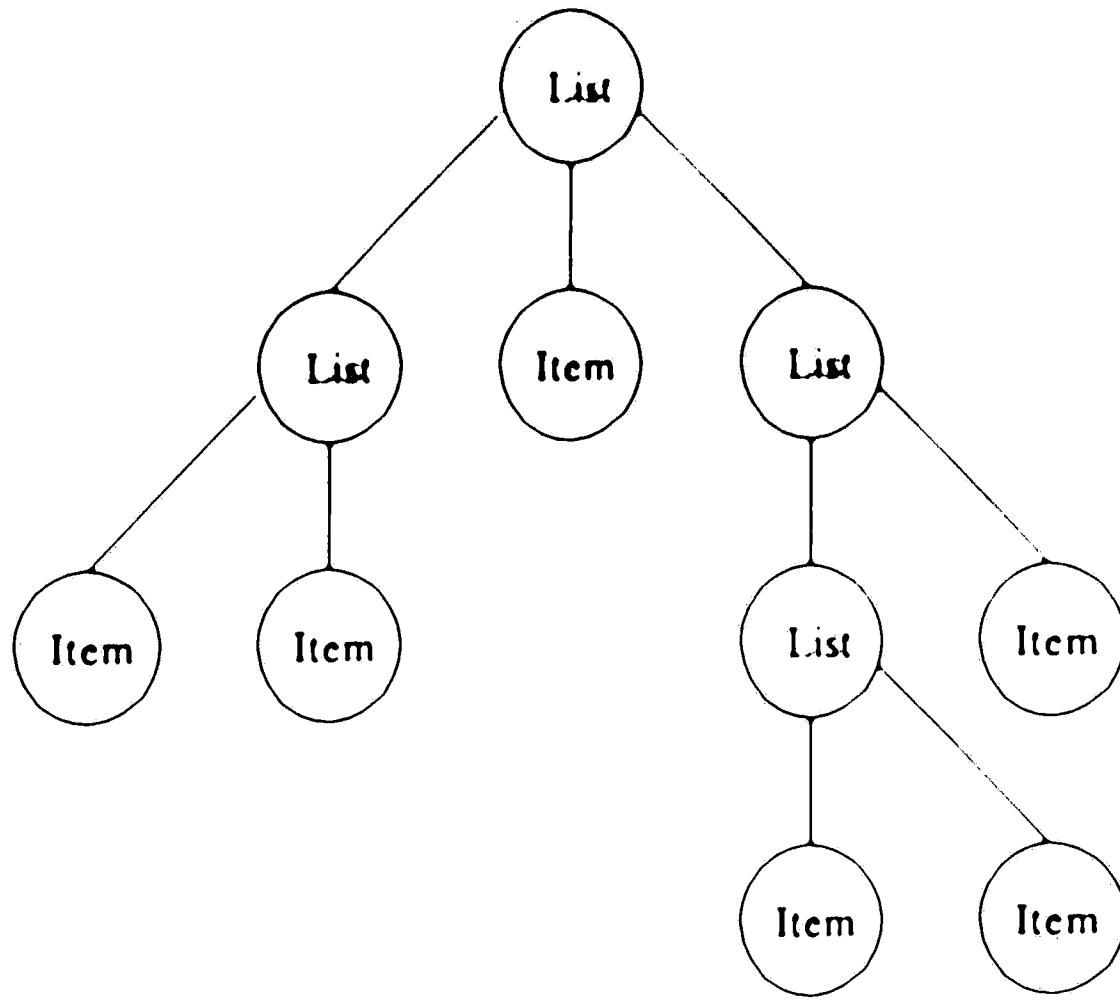


Figure 5. SECS-II Message as a Tree Data Structure

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 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.

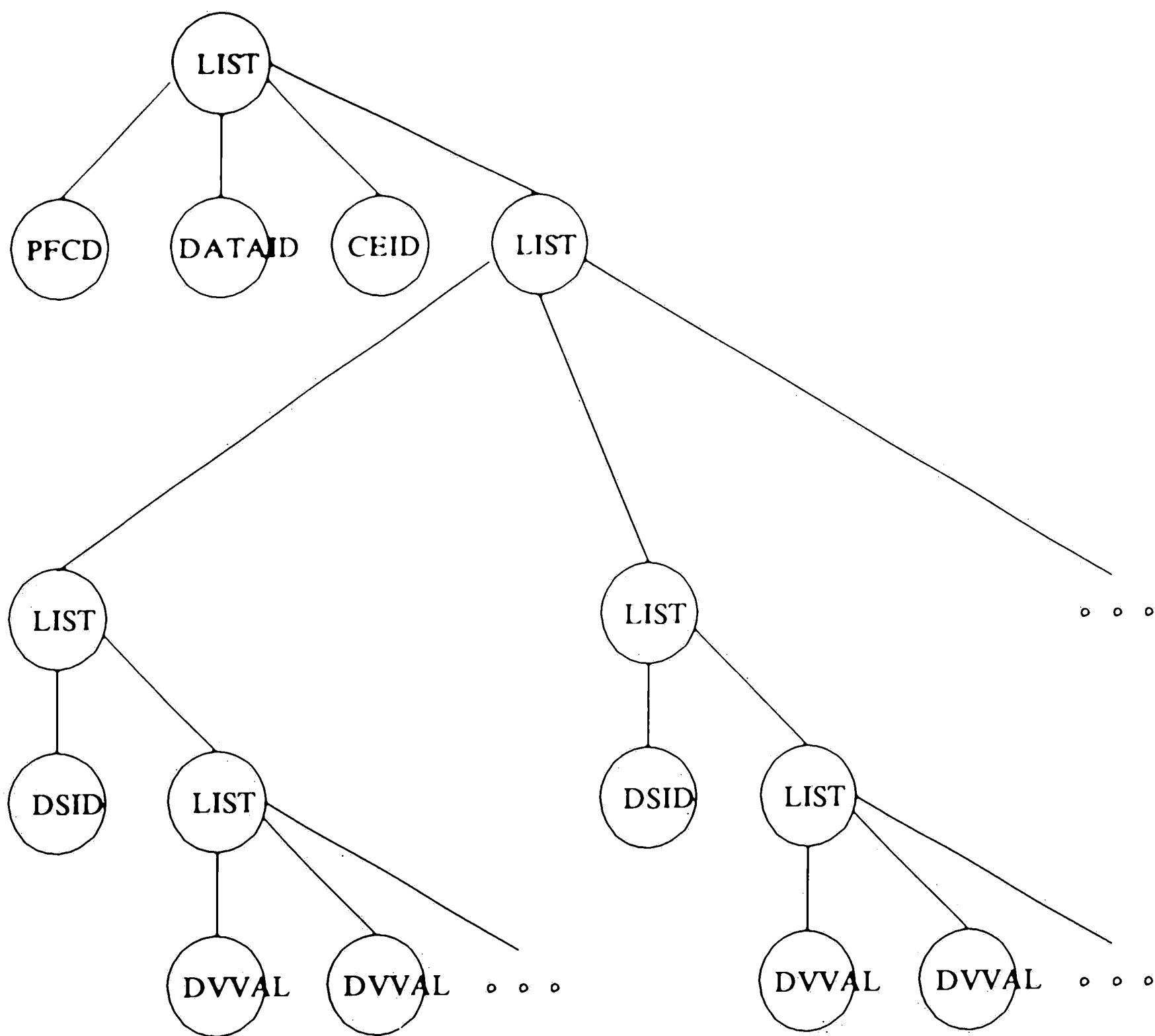


Figure 8. Stream 6 Function 9 SEMI Specification as a Tree

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 programmer 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[NODE_OF_CONSTANTS];
long          ecVal[NODE_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`.

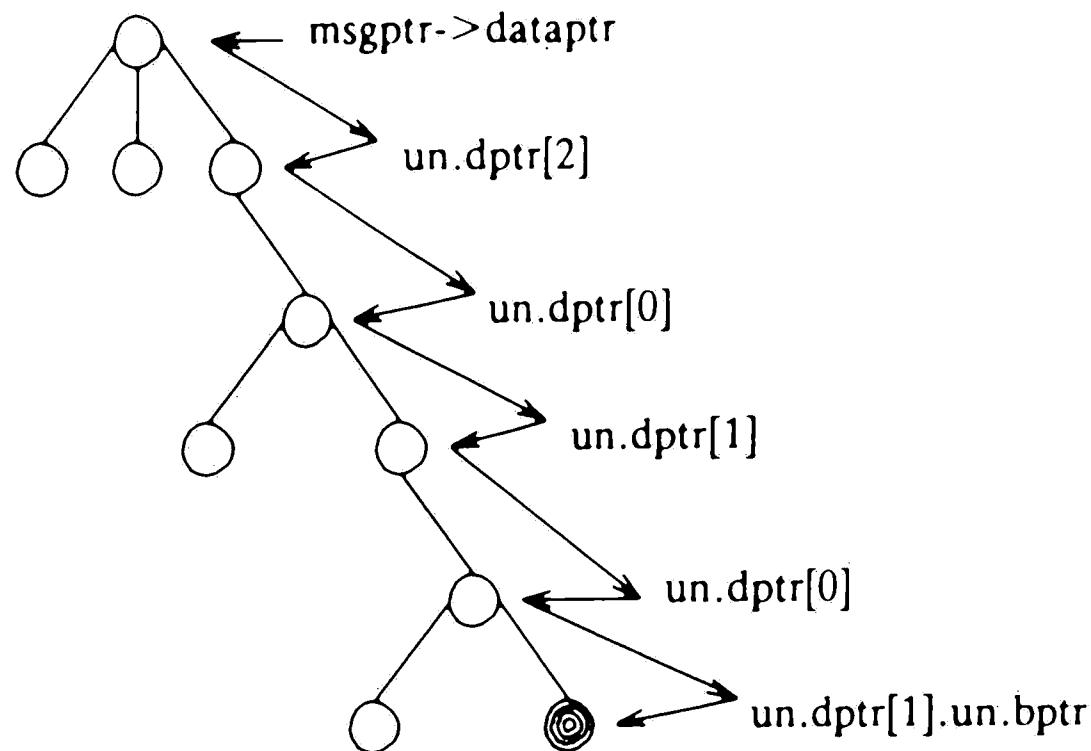


Figure 11. Tree Representation of Stream 6 Function 3 Message

```
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*.

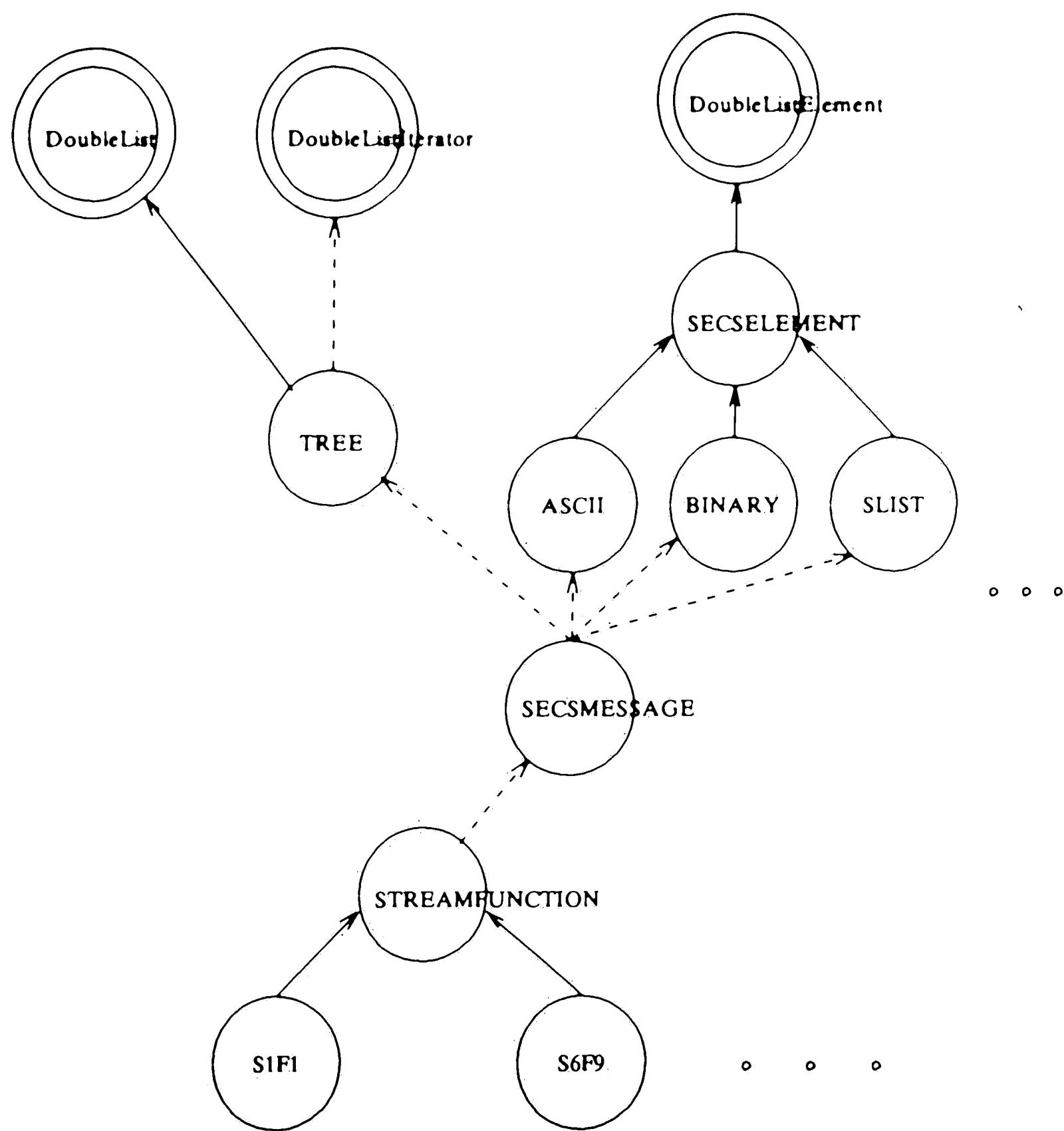


Figure 13. Object-Oriented SECS-II Interface Class Organization

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.

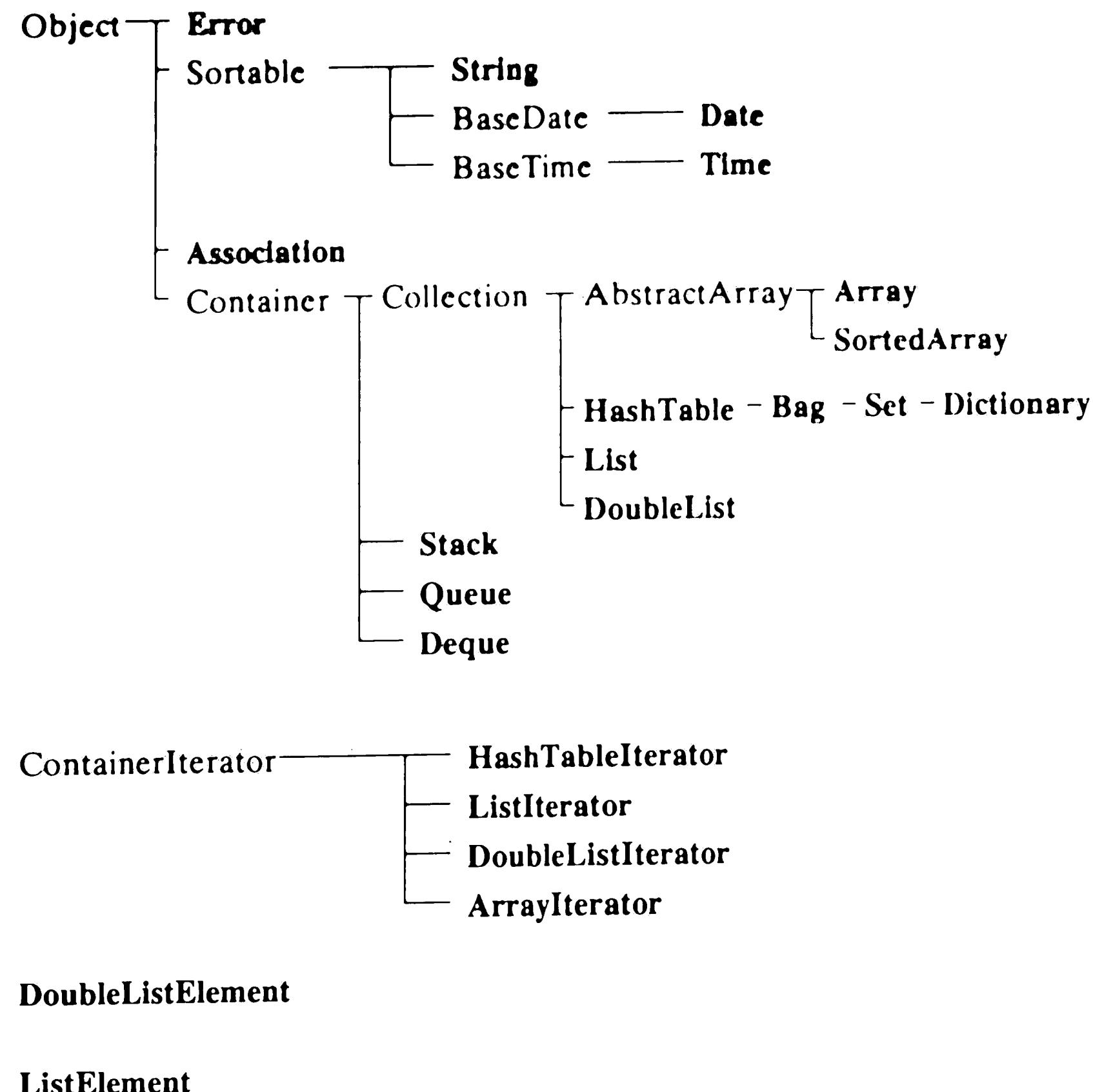


Figure 14. Borland C++ Class Library

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 SECMESSAGE Class

The SECMESSAGE 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 ++> ■
<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**.

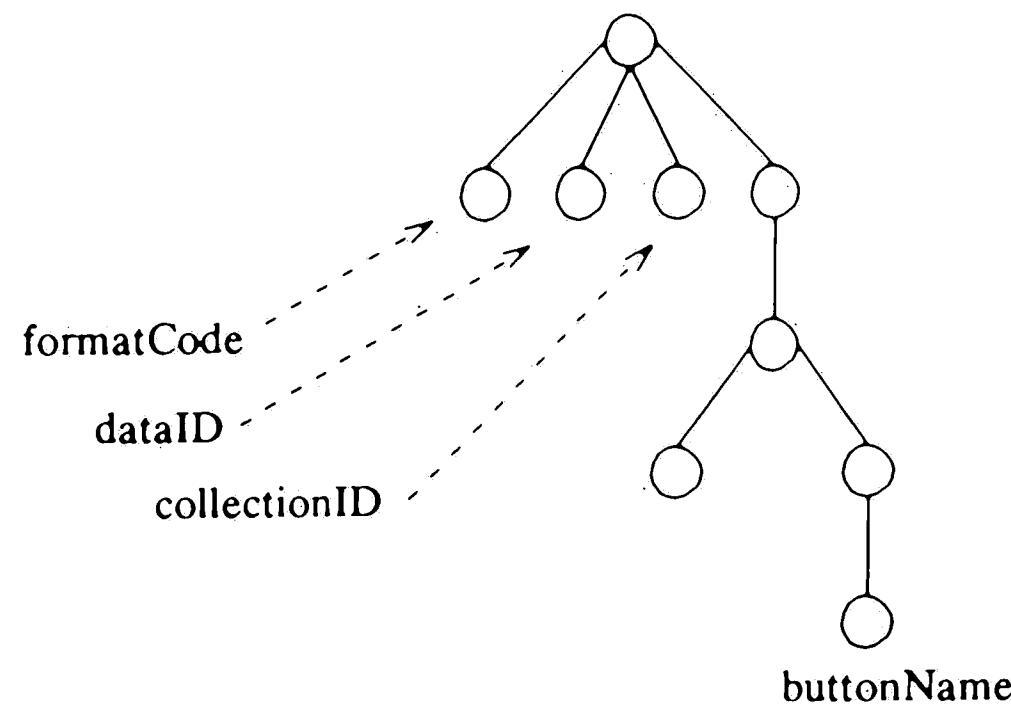


Figure 16. Stream 6 Function 9 Tree with Variable Names

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**.

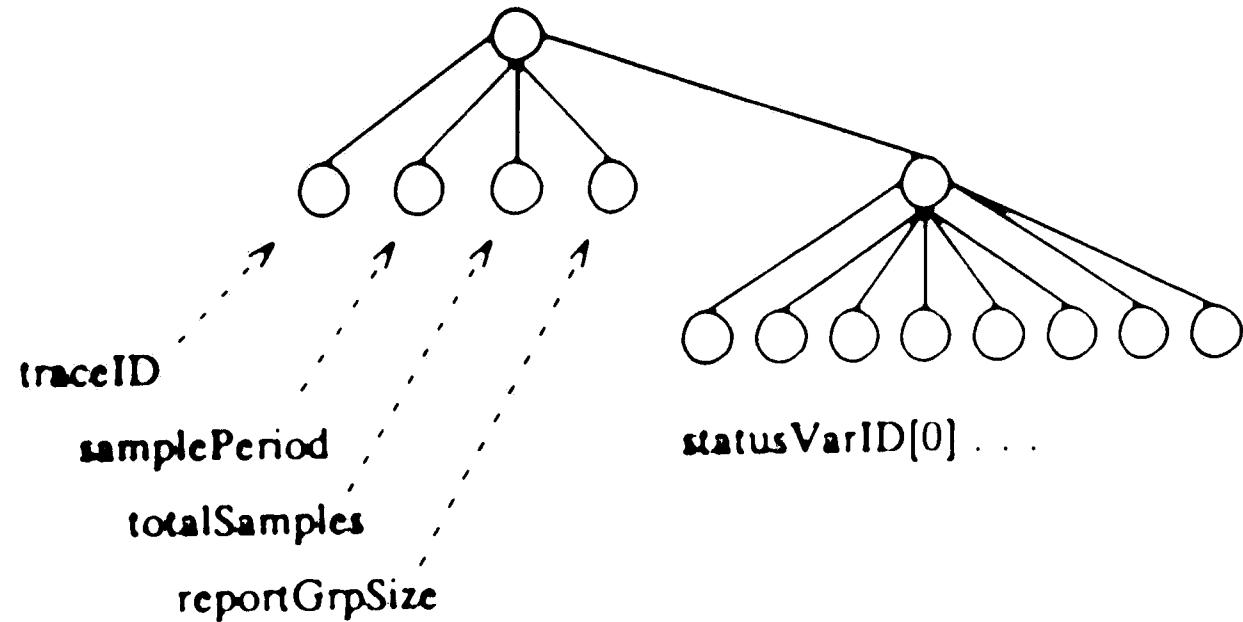


Figure 19. Stream 2 Function 23 Tree with Variable Names

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(SL)` 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

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

isRoot() 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.

CLASS NAME	MEMBER FUNCTION
STREAMFUNCTION	buildPrimaryUpdate() buildSecondaryUpdate() virtual sendPrimary() virtual sendSecondary() virtual primaryUpdate() virtual secondaryUpdate()
S1F1	buildSendPrimary() buildSendSecondary()
S1F1C	sendPrimary() sendSecondary() primaryUpdate() secondaryUpdate()

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. *IndentCntr* 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(SECMESSAGE *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 within 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.

Nov 24 10:48 1991 main.cpp Page 1

16

```
1  /*
2  ** This version of main is a mini-DASA.
3  **
4  ** Two type of Primary messages are expected from the
5  ** Equipment: S1-F1 and S6-F9. Specific Secondary
6  ** messages are sent in response to these. Generic S?-F0
7  ** Secondaries are sent in response to all other Primary
8  ** messages.
9  **
10 **
11 ** This DASA sends a S2-F23 to the Equipment when it
12 ** gets a S1-F1. This STREAMFUNCTION is put on a LIST,
13 ** awaiting the Secondary reply from the Equipment. If
14 ** the reply comes, the S2F23C object is removed from
15 ** the LIST. When the user quits out of this DASA, all
16 ** "orphaned" Primary messages sent to the Equipment are
17 ** reported.
18 **
19 */
20
21 #include <alloc.h>
22 #include <stdlib.h>
23 #include <stdio.h>
24 #include <string.h>
25 #include <iostream.h>
26 #include <process.h>
27 #include <fstream.h>
28 #include <dbllist.h>
29 #include <dlstelem.h>
30 #include <tree.h>
31 #include <secsmess.h>
32 #include <secselem.h>
33 #include <messages.h>
34 #include <list.h>
```

Nov 24 10:48 1991 main.cpp Page 1

1 /*
2 ** This version of main is a mini-DASA.
3 **
4 ** Two type of Primary messages are expected from the
5 ** Equipment: S1-F1 and S6-F9. Specific Secondary
6 ** messages are sent in response to these. Generic S?-F0
7 ** Secondaries are sent in response to all other Primary
8 ** messages.
9 **
10 ** This DASA sends a S2-F23 to the Equipment when it
11 ** gets a S1-F1. This STREAMFUNCTION is put on a LIST,
12 ** awaiting the Secondary reply from the Equipment. If
13 ** the reply comes, the S2F23C object is removed from
14 ** the LIST. When the user quits out of this DASA, all
15 ** "orphaned" Primary messages sent to the Equipment are
16 ** reported.
17 **
18 ** @(#)main.cpp 13.15 11/24/91
19 */
20
21 #include <alloc.h>
22 #include <stdlib.h>
23 #include <stdio.h>
24 #include <string.h>
25 #include <iostream.h>
26 #include <process.h>
27 #include <fstream.h>
28 #include <dbllist.h>
29 #include <dlistelem.h>
30 #include <tree.h>
31 #include <secsmess.h>
32 #include <secselem.h>
33 #include <messages.h>
34 #include <list.h>

Nov 24 10:48 1991 main.cpp Page 2

```
35 #include <listelem.h>
36
37
38 void
39 main()
40 {
41     SECSMESSAGE    *Incoming, *Outgoing;
42     STREAMFUNCTION *streamFunction, *sfSaved;
43
44     /*
45      ** LIST to store Primary messages sent to Equipment.
46      ** Secondary messages from the Equipment will be compared
47      ** to this LIST: if their Primary is not on the LIST they
48      ** will be thrown away, if their Primary is on the LIST
49      ** they will update the DASA & the STREAMFUNCTION will be
50      ** removed from the LIST.
51     */
52     List          sfList;
53     ListIterator  sfIterator = (ListIterator &) sfList.initIterator();
54     short         stream, function;
55     char          fileName[80];
56
57
58     while( 1 )      {
59         /*
60          ** Prompt user for incoming SECS-II message.
61          */
62         cout << "\nEnter filename for incoming SECS-II message" <<
63             " ('q' to quit) ==> " << flush;
64         cin >> fileName;
65
66         if ( ( strlen( fileName ) == 1 &&
67               ( strncmp( "q", fileName, 1 ) == 0 ) ) )
68             break;
```

Nov 24 10:48 1991 main.cpp Page 3

```
69      }
70  else      {
71      /*
72      ** If the user did not type 'q' for QUIT,
73      ** deal with incoming SECS-II message in a
74      ** manner depending on what Stream-Function it is.
75      */
76      Incoming = new SECSMESSAGE( fileName );
77
78      stream = Incoming->getSECSstream();
79      function = Incoming->getSECSfunction();
80
81      if ( Incoming->isPrimary() )      {
82          /*
83          ** Primary messages received from Equipment.
84          */
85      if ( stream == 1 && function == 1 ) {
86          /*
87          ** Create S1F1C object, send the
88          ** Secondary message, delete the
89          ** STREAMFUNCTION.
90          */
91          streamFunction = new S1F1C( Incoming );
92          streamFunction->sendSecondary();
93          delete streamFunction;
94          /*
95          ** Create S2F23C object, send it to the
96          ** Equipment, and add it to the LIST of
97          ** outstanding Primary messages.
98          */
99          Outgoing = new SECSMESSAGE( "_s2f23" );
100         streamFunction = new S2F23C( Outgoing );
101         streamFunction->sendPrimary();
102         sfList.add( (Object &) *streamFunction );
```

Nov 24 10:48 1991 main.cpp Page 4

```
103 }
104     else if ( stream == 6 && function == 9 ) {
105         /*
106          ** Create S6F9C object, send the
107          ** Secondary message, delete the
108          ** STREAMFUNCTION.
109         */
110         streamFunction = new S6F9C( Incoming );
111         streamFunction->sendSecondary();
112         delete streamFunction;
113     }
114     else {
115         /*
116          ** Create generic STREAMFUNCTION object,
117          ** send the Secondary message, delete the
118          ** STREAMFUNCTION.
119         */
120         streamFunction = new STREAMFUNCTION( Incoming );
121         streamFunction->sendSecondary();
122         delete streamFunction;
123     }
124 }
125     else {
126         /*
127          ** Secondary messages received from Equipment.
128          **
129          ** Look for the matching Primary message on the
130          ** LIST.
131         */
132         sfIterator = (ListIterator &)sfList.initIterator();
133         while( sfIterator ) {
134             sfSaved = (STREAMFUNCTION *)&((Object &)(sfIterator));
135             if ( sfSaved->getStream() == stream &&
136                 sfSaved->getSecondaryFunction() == function ) {
```

Nov 24 10:48 1991 main.cpp Page 5

```
137          /*  
138           ** The matching Primary is found;  
139           ** update the DASA & remove the  
140           ** STREAMFUNCTION from the LIST.  
141           */  
142           sfSaved->secondaryUpdate();  
143           sfList.destroy( (Object &) *sfSaved );  
144           break;  
145       }  
146     else {  
147       /*  
148       ** This STREAMFUNCTION did not match;  
149       ** go to the next item on the LIST.  
150       */  
151       sfIterator++;  
152     }  
153   }  
154 }  
155 }  
156 }  
157 /*  
158 ** Print out "orphaned" Primary messages still on the  
159 ** List of SECS-II messages:  
160 */  
161 sfIterator = (ListIterator &)sfList.initIterator();  
162 while( sfIterator ) {  
163   sfSaved = (STREAMFUNCTION *)&((Object &)(sfIterator));  
164   sfSaved->printOn( cout );  
165   sfIterator++;  
166 }  
167 }  
168 exit( 0 );  
170 }
```

/
Nov 24 10:48 1991 main1.cpp Page 1

```
1  /*
2   ** This version of main simply accepts incoming SECS-II
3   ** messages and prints out their structure and contents.
4   **
5   ** @(#)main1.cpp 13.1 11/24/91
6   */
7
8 #include <alloc.h>
9 #include <stdlib.h>
10 #include <stdio.h>
11 #include <string.h>
12 #include <iostream.h>
13 #include <process.h>
14 #include <fstream.h>
15 #include <dbllist.h>
16 #include <dlstelem.h>
17 #include <tree.h>
18 #include <secsmess.h>
19 #include <secselem.h>
20
21
22 void
23 main()
24 {
25     char          fileName[80];
26     SECMESSAGE    *Incoming;
27
28
29     while( 1 )      {
30         /*
31         ** Prompt user for incoming SECS-II message.
32         */
33         cout << "\nEnter filename for incoming SECS-II message" <<
34             " ('q' to quit) ==> " << flush;
```

Nov 24 10:48 1991 main1.cpp Page 2

```
35         cin >> fileName;
36
37         if ( ( strlen( fileName ) == 1 &&
38             ( strncmp( "q", fileName, 1 ) == 0 ) ) )
39             break;
40         }
41     else
42     {
43         /*
44         ** If the user did not type 'q' for QUIT,
45         ** create a new SECSMESSAGE object, print it
46         ** to COUT, then delete the object.
47         */
48         Incoming = new SECSMESSAGE( fileName );
49         Incoming->printOn( cout );
50         delete Incoming;
51     }
52
53     exit( 0 );
54
55 }
```

Nov 24 10:48 1991 main2.cpp Page 1

```
1  /*
2  **          This version of main accepts incoming SECS-II messages
3  **          and prompts the user to build the appropriate update
4  **          Member Function.
5  **
6  **          @(#)main2.cpp    13.1    11/24/91
7  */
8
9  #include <alloc.h>
10 #include <stdlib.h>
11 #include <stdio.h>
12 #include <string.h>
13 #include <iostream.h>
14 #include <process.h>
15 #include <fstream.h>
16 #include <dbllist.h>
17 #include <dlstelem.h>
18 #include <tree.h>
19 #include <secsmess.h>
20 #include <secselem.h>
21 #include <strfunc.h>
22
23
24 void
25 main()
26 {
27     char          fileName[80];
28     SECSMESSAGE *Incoming;
29     STREAMFUNCTION *streamFunction;
30
31
32     while( 1 )      {
33         /*
34          ** Prompt user for incoming SECS-II message.
```

Nov 24 10:48 1991 main2.cpp Page 2

```
35      */
36      cout << "\nEnter filename for incoming SECS-II message" <<
37          " ('q' to quit) ==> " << flush;
38      cin >> fileName;
39
40      if ( ( strlen( fileName ) == 1 &&
41             ( strncmp( "q", fileName, 1 ) == 0 ) ) )
42          break;
43      }
44      else      {
45          /*
46          ** If the user did not type 'q' for QUIT,
47          ** create a new SECMESSAGE object, create a
48          ** STREAMFUNCTION object from it, then call
49          ** the appropriate "build*Update() Member
50          ** Function to prompt the user to build a
51          ** Member Function.
52          */
53      Incoming = new SECMESSAGE( fileName );
54      streamFunction = new STREAMFUNCTION( Incoming );
55
56      if ( Incoming->isPrimary() )      {
57          /*
58          ** Send S?-F0 so to end conversation with
59          ** the Equipment.
60          */
61          streamFunction->sendSecondary();
62          streamFunction->buildPrimaryUpdate();
63      }
64      else      {
65          streamFunction->buildSecondaryUpdate();
66      }
67      delete streamFunction;
68  }
```

Nov 24 10:48 1991 main2.cpp Page 3

```
69      }
70
71     exit( 0 );
72
73 }
```

70

Nov 24 10:48 1991 main3.cpp Page 1

1 /*
2 ** This version of main prompts the user to:
3 ** - buildSendPrimary() for S1F1 & S2F23, and
4 ** - buildSendSecondary() for S1F1 & S6F9.
5 **
6 ** For a full-fledged production system, the user
7 ** would have to be prompted for all Stream-Functions
8 ** that the Host could possibly send to the Equipment.
9 **
10 ** @(#)main3.cpp 13.1 11/24/91
11 */
12
13 #include <alloc.h>
14 #include <stdlib.h>
15 #include <stdio.h>
16 #include <string.h>
17 #include <iostream.h>
18 #include <process.h>
19 #include <fstream.h>
20 #include <dbllist.h>
21 #include <dlstelem.h>
22 #include <tree.h>
23 #include <secsmess.h>
24 #include <secselem.h>
25 #include <messages.h>
26
27
28 void
29 main()
30 {
31 STREAMFUNCTION *streamFunction;
32
33 streamFunction = new S2F23();

Nov 24 10:48 1991 main3.cpp Page 2

```
35     streamFunction->buildSendPrimary();
36     delete streamFunction;
37
38     streamFunction = new S1F1();
39     streamFunction->buildSendPrimary();
40     streamFunction->buildSendSecondary();
41     delete streamFunction;
42
43     streamFunction = new S6F9();
44     streamFunction->buildSendSecondary();
45     delete streamFunction;
46
47     exit( 0 );
48
49 }
```

Sep 19 08:49 1991 custom.h Page 1

```
1  /*
2  **          @(#)custom.h    13.5    9/19/91
3  */
4
5  #include <clatypes.h>
6
7  // #define      secsElemClass _firstUserClass
8  #define      treeClass      255
9  #define      secsElemClass (treeClass+1)
10 #define     asciiClass      (secsElemClass+1)
11 #define     binaryClass     (asciiClass+1)
12 #define     booleanClass    (binaryClass+1)
13 #define     float4Class     (booleanClass+1)
14 #define     float8Class     (float4Class+1)
15 #define     int1Class       (float8Class+1)
16 #define     int2Class       (int1Class+1)
17 #define     int4Class       (int2Class+1)
18 #define     int8Class       (int4Class+1)
19 #define     slistClass      (int8Class+1)
20 #define     uint1Class      (slistClass+1)
21 #define     uint2Class      (uint1Class+1)
22 #define     uint4Class      (uint2Class+1)
23 #define     uint8Class      (uint4Class+1)
24
25 #define SIZEFILE      80
26 #define SECSTESTDIR   "e:\ame\secstest\
27
28 enum   truthFlag   { FALSE, TRUE };
29
30 #define ASCII_ITEM   020
31 #define BINARY_ITEM  010
32 #define BOOLEAN_ITEM 011
33 #define INT1_ITEM    031
34 #define INT2_ITEM    032
```

Sep 19 08:49 1991 custom.h Page 2

35 #define INT4_ITEM 034
36 #define INT8_ITEM 030
37 #define FLOAT4_ITEM 044
38 #define FLOAT8_ITEM 040
39 #define LIST_ITEM 000
40 #define UINT1_ITEM 051
41 #define UINT2_ITEM 052
42 #define UINT4_ITEM 054
43 #define UINT8_ITEM 050
44
45 #define ASCII_BYTES 1
46 #define BINARY_BYTES 1
47 #define BOOLEAN_BYTES 1
48 #define INT1_BYTES 1
49 #define INT2_BYTES 2
50 #define INT4_BYTES 4
51 #define INT8_BYTES 8
52 #define FLOAT4_BYTES 4
53 #define FLOAT8_BYTES 8
54 #define LIST_BYTES 0
55 #define UINT1_BYTES 1
56 #define UINT2_BYTES 2
57 #define UINT4_BYTES 4
58 #define UINT8_BYTES 8

Nov 12 09:28 1991 dasamac.h Page 1

```
1  /*
2   **      Macros needed by customized S-P Classes.
3   **
4   **      @(#)dasamac.h    13.2    11/12/91
5   */
6
7
8 #define TREE_PTR          treePtr
9 #define TREE_ROOT         treeRoot
10 #define TREE_MARK        treeMark
11
12
13
14 #define SEPTR    ((SECSELEMENT *) TREE_PTR->nodeAddr())
15
16
17
18 #define MOVE_DOWN        TREE_PTR = TREE_PTR->moveDown()
19 #define MOVE_UP          TREE_PTR = TREE_PTR->moveUp()
20 #define MOVE_RIGHT(x)    TREE_PTR = TREE_PTR->moveRight(x)
21 #define MOVE_LEFT(x)    TREE_PTR = TREE_PTR->moveLeft(x)
22
23
24
25 #define ADD_CHILD(x)     TREE_PTR = TREE_PTR->addChild(x)
26 #define ADD_NODE(x)      *new TREE(x)
27
28 #define ADD_ASCII(x,y)   *new ASCII(x,y)
29 #define ADD_ASCII(x,y)   ADD_CHILD( ADD_NODE( ADD_ASCII(x,y)))
30
31 #define ADD_BINARY(x,y)  *new BINARY(x,y)
32 #define ADD_BINARY(x,y) ADD_CHILD( ADD_NODE( ADD_BINARY(x,y)))
33
34 #define ADD_BOOLEAN(x,y) *new BOOLEAN(x,y)
```

Nov 12 09:28 1991 dasamac.h Page 2

```
35 #define ADD_BOOLEAN(x,y)           ADD_CHILD( ADD_NODE( ADD_BOOLEANX(x,y)))
36
37 #define ADD_FLOAT4X(x,y)          *new FLOAT4(x,y)
38 #define ADD_FLOAT4(x,y) ADD_CHILD( ADD_NODE( ADD_FLOAT4X(x,y)))
39
40 #define ADD_FLOAT8X(x,y)          *new FLOAT8(x,y)
41 #define ADD_FLOAT8(x,y) ADD_CHILD( ADD_NODE( ADD_FLOAT8X(x,y)))
42
43 #define ADD_INT1X(x,y)           *new INT1(x,y)
44 #define ADD_INT1(x,y) ADD_CHILD( ADD_NODE( ADD_INT1X(x,y)))
45
46 #define ADD_INT2X(x,y)           *new INT2(x,y)
47 #define ADD_INT2(x,y) ADD_CHILD( ADD_NODE( ADD_INT2X(x,y)))
48
49 #define ADD_INT4X(x,y)           *new INT4(x,y)
50 #define ADD_INT4(x,y) ADD_CHILD( ADD_NODE( ADD_INT4X(x,y)))
51
52 #define ADD_INT8X(x,y)           *new INT8(x,y)
53 #define ADD_INT8(x,y) ADD_CHILD( ADD_NODE( ADD_INT8X(x,y)))
54
55 #define ADD_UINT1X(x,y)          *new UINT1(x,y)
56 #define ADD_UINT1(x,y) ADD_CHILD( ADD_NODE( ADD_UINT1X(x,y)))
57
58 #define ADD_UINT2X(x,y)          *new UINT2(x,y)
59 #define ADD_UINT2(x,y) ADD_CHILD( ADD_NODE( ADD_UINT2X(x,y)))
60
61 #define ADD_UINT4X(x,y)          *new UINT4(x,y)
62 #define ADD_UINT4(x,y) ADD_CHILD( ADD_NODE( ADD_UINT4X(x,y)))
63
64 #define ADD_UINT8X(x,y)          *new UINT8(x,y)
65 #define ADD_UINT8(x,y) ADD_CHILD( ADD_NODE( ADD_UINT8X(x,y)))
66
67 #define ADD_LIST(x)              *new SLIST(x)
68 #define SET_CNT(x)               TREE_PTR->setChildCnt(x)
```

Nov 12 09:28 1991 dasamac.h Page 3

69 #define ADD_LIST_AND_MOVEDOWN(x) ADD_CHILD(ADD_NODE(ADD_LIST(x))),SET_CNT(x)

Nov 24 10:43 1991 messages.h Page 1

```
1  /*
2  **      Contains all includes necessary for SECMESSAGE client Classes.
3  **
4  **      @(#)messages.h 13.6    11/24/91
5  */
6 #include <strfunc.h>
7 #include <s1f1.h>
8 #include <s1f1c.h>
9 #include <s2f23.h>
10 #include <s2f23c.h>
11 #include <s6f9.h>
12 #include <s6f9c.h>
```

Aug 26 17:09 1991 secstype.h Page 1

```
1  /*
2  **      Contains all includes necessary for SECSELEMENT Class and its
3  **      subclasses.
4  **
5  **      @(#)secstype.h 13.1    8/26/91
6  */
7  #include <secselem.h>
8  #include <ascii.h>
9  #include <binary.h>
10 #include <boolean.h>
11 #include <float4.h>
12 #include <float8.h>
13 #include <int1.h>
14 #include <int2.h>
15 #include <int4.h>
16 #include <int8.h>
17 #include <slist.h>
18 #include <uint1.h>
19 #include <uint2.h>
20 #include <uint4.h>
21 #include <uint8.h>
```

Nov 12 09:27 1991 ascii.h Page 1

```
1  /*
2   **      class: ASCII -      DECLARATIONS
3   **
4   **          @(#)ascii.h    13.4    11/12/91
5   */
6
7  class ASCII : public SECSELEMENT      {
8 protected:
9     char   *data;
10 public:
11     ASCII( unsigned long Length, char *Ascii );
12     ~ASCII();
13
14     short      bytesPerItem();
15
16     unsigned char *stringForMessage();
17
18     int       hasString();
19     char   *returnString();
20
21     virtual void printOn( ostream& ) const;
22 };
```

Nov 12 09:32 1991 ascii.cpp Page 1

```
1  /*
2  **      class: ASCII -      MEMBER FUNCTIONS
3  **
4  **          @(#)ascii.cpp    13.6    11/12/91
5  */
6
7 #include <iostream.h>
8 #include <string.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <ascii.h>
14
15 ASCII::ASCII( unsigned long Length, char *Ascii ) :
16     SECSELEMENT( "ASCII", asciiClass ) {
17     length = remainingBytes = Length;
18     data = new char[length+1];
19     strncpy( data, Ascii, length );
20     data[length] = '\0';
21     itemCode = ASCII_ITEM;
22 }
23
24 ASCII::~ASCII() {
25     delete data;
26 }
27
28 short ASCII::bytesPerItem() {
29     return ASCII_BYTES;
30 }
31
32 unsigned char *ASCII::stringForMessage() {
33     unsigned char *ptrOut, *ptrSave;
34     unsigned char *itemHeader;
```

Nov 12 09:32 1991 ascii.cpp Page 2

```
35     long    i, itemLength;
36     short   headerLength;
37
38     /*
39      ** Create space to return the "SECS Message ready"
40      ** version of this item.
41     */
42     itemLength = this->lengthInMessage();
43     ptrOut = new unsigned char[itemLength];
44     ptrSave = ptrOut;
45
46     /*
47      ** Get the Item Header and put it in the
48      ** return string.
49     */
50     itemHeader = this->getItemHeader( &headerLength, itemHeader );
51     memcpy( ptrOut, itemHeader, headerLength );
52     delete( itemHeader );
53
54     /*
55      ** Point to first byte of actual data to be returned,
56      ** then copy the data in.
57     */
58     ptrOut += headerLength;
59     strncpy( (char *) ptrOut, data, length );
60
61     return ptrSave;
62 };
63
64 int ASCII::hasString()  {
65     return 1;
66 };
67
68 char *ASCII::returnString()      {
```

Nov 12 09:32 1991 ascii.cpp Page 3

```
69         return data;
70     };
71
72 void ASCII::printOn( ostream& outputStream ) const    {
73     outputStream << '<' << this->nameOf() << " '" << data << "'>" << flush;
74 }
```

Nov 12 09:27 1991 binary.h Page 1

```
1  /*
2   **      class: BINARY -      DECLARATIONS
3   **
4   **          @(#)binary.h    13.6    11/12/91
5   */
6
7  class BINARY : public SECSELEMENT      {
8 protected:
9     unsigned char   *data;
10    unsigned char   *ptr;
11 public:
12    BINARY( unsigned long Length, unsigned char *Binary );
13    BINARY( unsigned long Items, long DataArray[] );
14    ~BINARY();
15
16    short          bytesPerItem();
17
18    unsigned char   *stringForMessage();
19
20    int            hasLong();
21    long           returnLong();
22
23    virtual void   printOn( ostream& ) const;
24 }
```

Nov 12 09:32 1991 binary.cpp Page 1

```
1  /*
2   **      class: BINARY -      MEMBER FUNCTIONS
3   **
4   **          @(#)binary.cpp 13.8    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <string.h>
10 #include <object.h>
11 #include <clstypes.h>
12 #include <custom.h>
13 #include <secselem.h>
14 #include <binary.h>
15
16 BINARY::BINARY( unsigned long Length, unsigned char *Binary ) :
17     SECSELEMENT( "BINARY", binaryClass )      {
18     length = remainingBytes = Length;
19     data = ptr = new unsigned char[length];
20     itemCode = BINARY_ITEM;
21
22     memcpy( data, Binary, length );
23 }
24
25 BINARY::BINARY( unsigned long Items, long DataArray[] ) :
26     SECSELEMENT( "BINARY", binaryClass )      {
27     int      i;
28
29     length = remainingBytes = Items * BINARY_BYTES;
30     data = ptr = new unsigned char[length];
31     itemCode = BINARY_ITEM;
32
33     for ( i = 0; i < length; i++ )  {
34         data[i] = (unsigned char) DataArray[i];
```

Nov 12 09:32 1991 binary.cpp Page 2

```
35      }
36  };
37
38  BINARY::~BINARY()
39      delete data;
40  };
41
42  short BINARY::bytesPerItem()
43      return BINARY_BYTES;
44  };
45
46  unsigned char *BINARY::stringForMessage()
47      unsigned char    *ptrOut, *ptrSave;
48      unsigned char    *itemHeader;
49      long    i, itemLength;
50      short   headerLength;
51
52  /*
53  **      Create space to return the "SECS Message ready"
54  **      version of this item.
55  */
56  itemLength = this->lengthInMessage();
57  ptrOut = new unsigned char[itemLength];
58  ptrSave = ptrOut;
59
60  /*
61  **      Get the Item Header and put it in the
62  **      return string.
63  */
64  itemHeader = this->getItemHeader( &headerLength, itemHeader );
65  memcpy( ptrOut, itemHeader, headerLength );
66  delete( itemHeader );
67
68  /*
```

Nov 12 09:32 1991 binary.cpp Page 3

```
69          **      Point to first byte of actual data to be returned,
70          **      then copy the data in.
71          */
72          ptrOut += headerLength;
73          for ( i = 0; i < length; i++ ) {
74              *ptrOut++ = data[i];
75          }
76
77          return ptrSave;
78      };
79
80      int BINARY::hasLong()    {
81          remainingBytes = length;
82          ptr = data;
83          return 1;
84      };
85
86      long BINARY::returnLong() {
87          if ( remainingBytes <= 0 ) {
88              cerr << "BINARY: trying to get data after end of contents" << endl;
89              exit( 1 );
90          }
91          else {
92              long      tmp = (long) *ptr;
93              ptr += BINARY_BYTES;
94              remainingBytes -= BINARY_BYTES;
95              return tmp;
96          }
97      };
98
99      void BINARY::printOn( ostream& outputStream ) const {
100          int      i, perLine = 0;
101
102          outputStream << '<' << this->nameOf();
```

Nov 12 09:32 1991 binary.cpp Page 4

```
103     for ( i = 0; i < length; i++, perLine++ )      {
104         outputStream.width( 3 );
105         outputStream.fill( ' ' );
106         outputStream.setf( ios::right, ios::adjustfield );
107         outputStream << hex << (int) data[i];
108         if ( perLine >= 16 )    {
109             perLine = 0;
110             outputStream << endl;
111         }
112     }
113     outputStream << ">" << flush;
114 }
```

88

Nov 12 09:27 1991 boolean.h Page 1

```
1  /*
2   **      class: BOOLEAN -      DECLARATIONS
3   **
4   **          @(#)boolean.h    13.4    11/12/91
5   */
6
7  class BOOLEAN : public SECSELEMENT      {
8  protected:
9      unsigned char  *data;
10     unsigned char  *ptr;
11 public:
12     BOOLEAN( unsigned long Length, unsigned char *Boolean );
13     BOOLEAN( unsigned long Items, long DataArray[] );
14     ~BOOLEAN();
15
16     short          bytesPerItem();
17
18     unsigned char  *stringForMessage();
19
20     int            hasLong();
21     long           returnLong();
22
23     virtual void   printOn( ostream& ) const;
24 }
```

Nov 12 09:32 1991 boolean.cpp Page 1

```
1  /*
2   **      class: BOOLEAN -      MEMBER FUNCTIONS
3   **
4   **          @(#)boolean.cpp 13.6    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <boolean.h>
14
15 BOOLEAN::BOOLEAN( unsigned long Length, unsigned char *Boolean ) :
16     SECSELEMENT( "BOOLEAN", booleanClass ) {
17     int      i;
18
19     length = remainingBytes = Length;
20     data = ptr = new unsigned char[length];
21     itemCode = BOOLEAN_ITEM;
22
23     for ( i = 0; i < length; i++ ) {
24         data[i] = *Boolean++;
25     }
26 }
27
28 BOOLEAN::BOOLEAN( unsigned long Items, long DataArray[] ) :
29     SECSELEMENT( "BOOLEAN", booleanClass ) {
30     int      i;
31
32     length = remainingBytes = Items * BOOLEAN_BYTES;
33     data = ptr = new unsigned char[length];
34     itemCode = BOOLEAN_ITEM;
```

Nov 12 09:32 1991 boolean.cpp Page 2

```
35
36     for ( i = 0; i < length; i++ ) {
37         data[i] = (unsigned char) DataArray[i];
38     }
39 }
40
41 BOOLEAN::~BOOLEAN() {
42     delete data;
43 }
44
45 short BOOLEAN::bytesPerItem() {
46     return BOOLEAN_BYTES;
47 }
48
49 unsigned char *BOOLEAN::stringForMessage() {
50     unsigned char    *ptrOut, *ptrSave;
51     unsigned char    *itemHeader;
52     long      i, itemLength;
53     short     headerLength;
54
55     /*
56     **      Create space to return the "SECS Message ready"
57     **      version of this item.
58     */
59     itemLength = this->lengthInMessage();
60     ptrOut = new unsigned char[itemLength];
61     ptrSave = ptrOut;
62
63     /*
64     **      Get the Item Header and put it in the
65     **      return string.
66     */
67     itemHeader = this->getItemHeader( &headerLength, itemHeader );
68     memcpy( ptrOut, itemHeader, headerLength );
```

Nov 12 09:32 1991 boolean.cpp Page 3

```
69         delete( itemHeader );
70
71         /*
72          **      Point to first byte of actual data to be returned,
73          **      then copy the data in.
74         */
75         ptrOut += headerLength;
76         for ( i = 0; i < length; i++ )  {
77             *ptrOut++ = data[i];
78         }
79
80         return ptrSave;
81     };
82
83     int BOOLEAN::hasLong()  {
84         remainingBytes = length;
85         ptr = data;
86         return 1;
87     };
88
89     long BOOLEAN::returnLong()      {
90         if ( remainingBytes <= 0 )      {
91             cerr << "BOOLEAN: trying to get data after end of contents" << endl;
92             exit( 1 );
93         }
94         else      {
95             long      tmp = (long) *ptr;
96             ptr += BOOLEAN_BYTES;
97             remainingBytes -= BOOLEAN_BYTES;
98             return tmp;
99         }
100    };
101
102    void BOOLEAN::printOn( ostream& outputStream ) const      {
```

Nov 12 09:32 1991 boolean.cpp Page 4

```
103     int      i, perLine = 0;
104     char     booleanValue;
105
106     outputStream << '<' << this->nameOf();
107
108     for ( i = 0; i < length; i++, perLine++ ) {
109         booleanValue = ( (int) data[i] == 0 ) ? 'F' : 'T';
110         outputStream << " " << booleanValue;
111         if ( perLine >= 30 ) {
112             perLine = 0;
113             outputStream << endl;
114         }
115     }
116
117     outputStream << ">" << flush;
118 }
```

Nov 12 09:28 1991 float4.h Page 1

```
1  /*
2  **      class: FLOAT4 -      DECLARATIONS
3  **
4  **          @(#)float4.h    13.5    11/12/91
5  */
6
7  class FLOAT4 : public SECSELEMENT      {
8  protected:
9      float   *data;
10     float   *ptr;
11 public:
12     FLOAT4( unsigned long Length, unsigned char *Float4 );
13     FLOAT4( unsigned long Items, double DataArray[] );
14     ~FLOAT4();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int       hasDouble();
21     double    returnDouble();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 float4.cpp Page 1

1 /*
2 ** class: FLOAT4 - MEMBER FUNCTIONS
3 **
4 ** @(#)float4.cpp 13.8 11/12/91
5 */
6
7 #include <stdlib.h>
8 #include <math.h>
9 #include <stdio.h>
10 #include <iostream.h>
11 #include <object.h>
12 #include <clatypes.h>
13 #include <custom.h>
14 #include <secselem.h>
15 #include <float4.h>
16
17 FLOAT4::FLOAT4(unsigned long Length, unsigned char *Float4) :
18 SECSELEMENT("FLOAT4", float4Class) {
19 short i, j, k, items = Length / FLOAT4_BYTES;
20 unsigned char *reverseFloat;
21
22 length = remainingBytes = Length;
23 data = ptr = new float[items];
24 itemCode = FLOAT4_ITEM;
25
26 for (i = 0; i < items; i++) {
27 /*
28 ** Point to byte after end of current FLOAT4 in
29 ** input byte stream. We'll get bytes in reverse
30 ** order from the byte stream.
31 */
32 Float4 += sizeof(float);
33 reverseFloat = (unsigned char *) &data[i];
34 for (j = 0; j < sizeof(float); j++) {

Nov 12 09:33 1991 float4.cpp Page 2

```
35          *reverseFloat++ = *---Float4;
36      }
37      /*
38      ** Point to byte after end of current FLOAT4 in
39      ** input byte stream.
40      */
41      Float4 += sizeof(float);
42  }
43 }
44
45 FLOAT4::FLOAT4( unsigned long Items, double DataArray[] ) :
46     SECSELEMENT( "FLOAT4", float4Class ) {
47     short i;
48
49     length = remainingBytes = Items * FLOAT4_BYTES;
50     data = ptr = new float[Items];
51     itemCode = FLOAT4_ITEM;
52
53     for ( i = 0; i < Items; i++ ) {
54         data[i] = (float) DataArray[i];
55     }
56 }
57
58 FLOAT4::~FLOAT4() {
59     delete data;
60 }
61
62 short FLOAT4::bytesPerItem() {
63     return FLOAT4_BYTES;
64 }
65
66 unsigned char *FLOAT4::stringForMessage() {
67     unsigned char *ptrOut, *ptrSave;
68     unsigned char *itemHeader, *reverseFloat;
```

Nov 12 09:33 1991 float4.cpp Page 3

```
69     long    i, j, items, itemLength;
70     short   headerLength;
71
72     /*
73      ** Create space to return the "SECS Message ready"
74      ** version of this item.
75     */
76     itemLength = this->lengthInMessage();
77     ptrOut = new unsigned char[itemLength];
78     ptrSave = ptrOut;
79
80     /*
81      ** Get the Item Header and put it in the
82      ** return string.
83     */
84     itemHeader = this->getItemHeader( &headerLength, itemHeader );
85     memcpy( ptrOut, itemHeader, headerLength );
86     delete( itemHeader );
87
88     /*
89      ** Point to first byte of actual data to be returned,
90      ** then copy the data in.
91     */
92     ptrOut += headerLength;
93     items = length / FLOAT4_BYTES;
94     for ( i = 0; i < items; i++ ) {
95         /*
96          ** Point to byte after end of current FLOAT4 in
97          ** output byte stream.
98        */
99        ptrOut += sizeof(float);
100       reverseFloat = (unsigned char *) &data[i];
101       for ( j = 0; j < sizeof(float); j++ ) {
102           *--ptrOut = *reverseFloat++;

L6
L7
```

Nov 12 09:33 1991 float4.cpp Page 4

```
103      }
104      /*
105      ** Point to byte after end of current FLOAT4 in
106      ** output byte stream.
107      */
108      ptrOut += sizeof(float);
109  }
110
111      return ptrSave;
112  };
113
114  int FLOAT4::hasDouble() {
115      remainingBytes = length;
116      ptr = data;
117      return 1;
86
118  };
119
120  double FLOAT4::returnDouble() {
121      if ( remainingBytes <= 0 ) {
122          cerr << "FLOAT4: trying to get data after end of contents" << endl;
123          exit( 1 );
124      }
125      else {
126          double      tmp = (double) *ptr;
127          ptr += FLOAT4_BYTES;
128          remainingBytes -= FLOAT4_BYTES;
129          return tmp;
130      }
131  };
132
133  void FLOAT4::printOn( ostream& outputStream ) const {
134      /*
135      ** Although this "float" type (in BORLAND C++) can
136      ** apparently accomodate more than the usual range
```

Nov 12 09:33 1991 float4.cpp Page 5

```
137      **      of [ 1e-37 to 1e37 ], numbers outside of this
138      **      range will be printed as "0" when using the C++
139      **      Stream stuff. Using C's "%e" will show the actual
140      **      value.
141      */
142
143      int      i, perLine = 0;
144      int      items = length / FLOAT4_BYTES;
145
146      outputStream << '<' << this->nameOf();
147
148      for ( i = 0; i < items; i++, perLine++ )          {
149          outputStream << " " << data[i];
150          if ( perLine >= 4 ) {
151              perLine = 0;
152              outputStream << endl;
153          }
154      }
155
156      outputStream << ">" << flush;
157  };
```

Nov 12 09:28 1991 float8.h Page 1

```
1  /*
2   **      class: FLOAT8 -      DECLARATIONS
3   **
4   **          @(#)float8.h    13.4    11/12/91
5   */
6
7  class FLOAT8 : public SECSELEMENT      {
8  protected:
9      double *data;
10     double *ptr;
11 public:
12     FLOAT8( unsigned long Length, unsigned char *Float8 );
13     FLOAT8( unsigned long Items, double DataArray[] );
14     ~FLOAT8();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int      hasDouble();
21     double   returnDouble();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 float8.cpp Page 1

```
1  /*
2   **      class: FLOAT8 -          MEMBER FUNCTIONS
3   **
4   **          @(#)float8.cpp 13.7    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <math.h>
9 #include <stdio.h>
10 #include <iostream.h>
11 #include <object.h>
12 #include <clatypes.h>
13 #include <custom.h>
14 #include <secselem.h>
15 #include <float8.h>
16
17 FLOAT8::FLOAT8( unsigned long Length, unsigned char *Float8 ) :
18     SECSELEMENT( "FLOAT8", float8Class ) {
19     short i, j, k, items = Length / FLOAT8_BYTES;
20     unsigned char *reverseFloat;
21
22     if ( sizeof(double) != FLOAT8_BYTES ) {
23         cerr << "FLOAT8: C++ type does not match length of FLOAT8" << endl;
24         exit( 1 );
25     }
26
27     length = remainingBytes = Length;
28     data = ptr = new double[items];
29     itemCode = FLOAT8_ITEM;
30
31     for ( i = 0; i < items; i++ ) {
32         /*
33         ** Point to byte after end of current FLOAT8 in
34         ** input byte stream. We'll get bytes in reverse
```

Nov 12 09:33 1991 float8.cpp Page 2

```
35      ** order from the byte stream.
36      */
37      Float8 += FLOAT8_BYTES;
38      reverseFloat = (unsigned char *) &data[i];
39      for ( j = 0; j < FLOAT8_BYTES; j++ )           {
40          *reverseFloat++ = *--Float8;
41      }
42      /*
43      ** Point to byte after end of current FLOAT8 in
44      ** input byte stream.
45      */
46      Float8 += FLOAT8_BYTES;
47  }
48  };
49
50 FLOAT8::FLOAT8( unsigned long Items, double DataArray[] ) :
51     SECSELEMENT( "FLOAT8", float8Class )           {
52     short    i;
53
54     if ( sizeof(double) != FLOAT8_BYTES )   {
55         cerr << "FLOAT8: C++ type does not match length of FLOAT8" << endl;
56         exit( 1 );
57     }
58
59     length = remainingBytes = Items * FLOAT8_BYTES;
60     data = ptr = new double[Items];
61     itemCode = FLOAT8_ITEM;
62
63     for ( i = 0; i < Items; i++ )   {
64         data[i] = DataArray[i];
65     }
66  };
67
68 FLOAT8::~FLOAT8()           {
```

Nov 12 09:33 1991 float8.cpp Page 3

```
69         delete data;
70     };
71
72     short FLOAT8::bytesPerItem()      {
73         return FLOAT8_BYTES;
74     };
75
76     unsigned char *FLOAT8::stringForMessage()          {
77         unsigned char    *ptrOut, *ptrSave;
78         unsigned char    *itemHeader, *reverseFloat;
79         long    i, j, items, itemLength;
80         short   headerLength;
81
82         /*
83         **      Create space to return the "SECS Message ready"
84         **      version of this item.
85         */
86         itemLength = this->lengthInMessage();
87         ptrOut = new unsigned char[itemLength];
88         ptrSave = ptrOut;
89
90         /*
91         **      Get the Item Header and put it in the
92         **      return string.
93         */
94         itemHeader = this->getItemHeader( &headerLength, itemHeader );
95         memcpy( ptrOut, itemHeader, headerLength );
96         delete( itemHeader );
97
98         /*
99         **      Point to first byte of actual data to be returned,
100        **      then copy the data in.
101        */
102        ptrOut += headerLength;
```

Nov 12 09:33 1991 float8.cpp Page 4

```
103     items = length / FLOAT8_BYTES;
104     for ( i = 0; i < items; i++ )    {
105         /*
106          ** Point to byte after end of current FLOAT8 in
107          ** output byte stream.
108         */
109         ptrOut += FLOAT8_BYTES;
110         reverseFloat = (unsigned char *) &data[i];
111         for ( j = 0; j < FLOAT8_BYTES; j++ )      {
112             *--ptrOut = *reverseFloat++;
113         }
114         /*
115          ** Point to byte after end of current FLOAT8 in
116          ** output byte stream.
117         */
118         ptrOut += FLOAT8_BYTES;
119     }
120
121     return ptrSave;
122 };
123
124 int FLOAT8::hasDouble() {
125     remainingBytes = length;
126     ptr = data;
127     return 1;
128 };
129
130 double FLOAT8::returnDouble() {
131     if ( remainingBytes <= 0 )      {
132         cerr << "FLOAT8: trying to get data after end of contents" << endl;
133         exit( 1 );
134     }
135     else    {
136         double      tmp = *ptr;
```

Nov 12 09:33 1991 float8.cpp Page 5

```
137         ptr += FLOAT8_BYTES;
138         remainingBytes -= FLOAT8_BYTES;
139         return tmp;
140     }
141 }
142
143 void FLOAT8::printOn( ostream& outputStream ) const {
144     int i, perLine = 0;
145     int items = length / FLOAT8_BYTES;
146
147     outputStream << '<' << this->nameOf();
148
149     for ( i = 0; i < items; i++, perLine++ )
150         outputStream << " " << data[i];
151     if ( perLine >= 4 ) {
152         perLine = 0;
153         outputStream << endl;
154     }
155 }
156
157     outputStream << ">" << flush;
158 }
```

Nov 12 09:28 1991 int1.h Page 1

```
1  /*
2   **      class: INT1      -      DECLARATIONS
3   **
4   **          @(#)int1.h      13.5      11/12/91
5   */
6
7  class INT1 : public SECSELEMENT {
8  protected:
9      char    *data;
10     char    *ptr;
11 public:
12     INT1( unsigned long Length, unsigned char *Int1 );
13     INT1( unsigned long Items, long DataArray[] );
14     ~INT1();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int       hasLong();
21     long      returnLong();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 int1.cpp Page 1

```
1  /*
2   *      class: INT1      -      MEMBER FUNCTIONS
3   *
4   *      @(#)int1.cpp    13.7    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <int1.h>
14
15 INT1::INT1( unsigned long Length, unsigned char *Int1 ) :
16     SECSELEMENT( "INT1", int1Class ) {
17     int i;
18
19     length = remainingBytes = Length;
20     data = ptr = new char[length];
21     itemCode = INT1_ITEM;
22
23     for ( i = 0; i < length; i++ ) {
24         data[i] = *Int1++;
25     }
26 }
27
28 INT1::INT1( unsigned long Items, long DataArray[] ) :
29     SECSELEMENT( "INT1", int1Class ) {
30     int i;
31
32     length = remainingBytes = Items * INT1_BYTES;
33     data = ptr = new char[length];
34     itemCode = INT1_ITEM;
```

Nov 12 09:33 1991 int1.cpp Page 2

```
35
36     for ( i = 0; i < length; i++ ) {
37         data[i] = (char ) DataArray[i];
38     }
39 }
40
41 INT1::~INT1() {
42     delete data;
43 }
44
45 short INT1::bytesPerItem() {
46     return INT1_BYTES;
47 }
48
49 unsigned char *INT1::stringForMessage() {
50     unsigned char *ptrOut, *ptrSave;
51     unsigned char *itemHeader;
52     long i, itemLength;
53     short headerLength;
54
55     /*
56      ** Create space to return the "SECS Message ready"
57      ** version of this item.
58     */
59     itemLength = this->lengthInMessage();
60     ptrOut = new unsigned char[itemLength];
61     ptrSave = ptrOut;
62
63     /*
64      ** Get the Item Header and put it in the
65      ** return string.
66     */
67     itemHeader = this->getItemHeader( &headerLength, itemHeader );
68     memcpy( ptrOut, itemHeader, headerLength );
```

Nov 12 09:33 1991 int1.cpp Page 3

```
69         delete( itemHeader );
70
71         /*
72          **      Point to first byte of actual data to be returned,
73          **      then copy the data in.
74         */
75         ptrOut += headerLength;
76         for ( i = 0; i < length; i++ )  {
77             *ptrOut++ = data[i];
78         }
79
80         return ptrSave;
81     };
82
83     int INT1::hasLong()      {
84         remainingBytes = length;
85         ptr = data;
86         return 1;
87     };
88
89     long INT1::returnLong() {
90         if ( remainingBytes <= 0 )      {
91             cerr << "INT1: trying to get data after end of contents" << endl;
92             exit( 1 );
93         }
94         else   {
95             long      tmp = (long) *ptr;
96             ptr += INT1_BYTES;
97             remainingBytes -= INT1_BYTES;
98             return tmp;
99         }
100    };
101
102    void INT1::printOn( ostream& outputStream ) const {
```

Nov 12 09:33 1991 intl.cpp Page 4

```
103     int      i, perLine = 0;
104
105     outputStream << '<' << this->nameOf();
106
107     for ( i = 0; i < length; i++, perLine++ )      {
108         outputStream.width( 5 );
109         outputStream.fill( ' ' );
110         outputStream.setf( ios::right, ios::adjustfield );
111         outputStream << dec << (int) data[i];
112         if ( perLine >= 10 )           {
113             perLine = 0;
114             outputStream << endl;
115         }
116     }
117
118     outputStream << ">" << flush;
119 }
```

Nov 12 09:28 1991 int2.h Page 1

```
1  /*
2   **      class: INT2      -      DECLARATIONS
3   **
4   **          @(#)int2.h      13.5      11/12/91
5   */
6
7  class INT2 : public SECSELEMENT {
8  protected:
9      short    *data;
10     short    *ptr;
11 public:
12     INT2( unsigned long Length, unsigned char *Int2 );
13     INT2( unsigned long Items, long DataArray[] );
14     ~INT2();
15
16     short        bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int         hasLong();
21     long        returnLong();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 int2.cpp Page 1

```
1  /*
2   **      class: INT2      -      MEMBER FUNCTIONS
3   **
4   **      .          @(#)int2.cpp    13.8     11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <int2.h>
14
15 INT2::INT2( unsigned long Length, unsigned char *Int2 ) :
16     SECSELEMENT( "INT2", int2Class ) {
17     int i, j, items = Length / INT2_BYTES;
18
19     length = remainingBytes = Length;
20     data = ptr = new short[items];
21     itemCode = INT2_ITEM;
22
23     for ( i = 0; i < items; i++ ) {
24         data[i] = 0;
25         for ( j = 0; j < sizeof(short); j++, Int2++ )
26             data[i] = (data[i] << 8) + *Int2;
27     }
28 }
29
30
31 INT2::INT2( unsigned long Items, long DataArray[] ) :
32     SECSELEMENT( "INT2", int2Class ) {
33     int i;
```

Nov 12 09:33 1991 int2.cpp Page 2

```
35     length = remainingBytes = Items * INT2_BYTES;
36     data = ptr = new short[Items];
37     itemCode = INT2_ITEM;
38
39     for ( i = 0; i < Items; i++ )  {
40         data[i] = (short) DataArray[i];
41     }
42 }
43
44 INT2::~INT2()  {
45     delete data;
46 }
47
48 short INT2::bytesPerItem()      {
49     return INT2_BYTES;
50 }
51
52 unsigned char *INT2::stringForMessage() {
53     unsigned char *ptrOut, *ptrSave;
54     unsigned char *itemHeader;
55     long i, items, itemLength;
56     short headerLength;
57
58     /*
59     **      Create space to return the "SECS Message ready"
60     **      version of this item.
61     */
62     itemLength = this->lengthInMessage();
63     ptrOut = new unsigned char[itemLength];
64     ptrSave = ptrOut;
65
66     /*
67     **      Get the Item Header and put it in the
68     **      return string.
```

Nov 12 09:33 1991 int2.cpp Page 3

```
69      */
70      itemHeader = this->getItemHeader( &headerLength, itemHeader );
71      memcpy( ptrOut, itemHeader, headerLength );
72      delete( itemHeader );
73
74      /*
75      **      Point to first byte of actual data to be returned,
76      **      then copy the data in.
77      */
78      ptrOut += headerLength;
79      items = length / INT2_BYTES;
80      for ( i = 0; i < items; i++ ) {
81          *ptrOut++ = data[i] >> 8 & 0xFF;
82          *ptrOut++ = data[i] & 0xFF;
83      }
84
85      return ptrSave;
86  };
87
88  int INT2::hasLong() {
89      remainingBytes = length;
90      ptr = data;
91      return 1;
92  };
93
94  long INT2::returnLong() {
95      if ( remainingBytes <= 0 ) {
96          cerr << "INT2: trying to get data after end of contents" << endl;
97          exit( 1 );
98      }
99      else {
100          long tmp = (long) *ptr;
101          ptr++;
102          remainingBytes -= INT2_BYTES;
```

Nov 12 09:33 1991 int2.cpp Page 4

```
103             return tmp;
104         }
105     };
106
107 void INT2::printOn( ostream& outputStream ) const {
108     int      i, perLine = 0;
109     int      items = length / INT2_BYTES;
110
111     outputStream << '<' << this->nameOf();
112
113     for ( i = 0; i < items; i++, perLine++ )
114         outputStream << " " << dec << data[i];
115     if ( perLine >= 10 ) {
116         perLine = 0;
117         outputStream << endl;
118     }
119 }
120
121     outputStream << ">" << flush;
122 }
```

115

Nov 12 09:28 1991 int4.h Page 1

```
1  /*
2   **      class: INT4      -      DECLARATIONS
3   **
4   **          @(#)int4.h      13.4      11/12/91
5   */
6
7  class INT4 : public SECSELEMENT {
8  protected:
9      long    *data;
10     long    *ptr;
11 public:
12     INT4( unsigned long Length, unsigned char *Int4 );
13     INT4( unsigned long Items, long DataArray[] );
14     ~INT4();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int       hasLong();
21     long      returnLong();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 int4.cpp Page 1

```
1  /*
2   **      class: INT4      -      MEMBER FUNCTIONS
3   **
4   **          @(#)int4.cpp    13.7    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <int4.h>
14
15 INT4::INT4( unsigned long Length, unsigned char *Int4 ) :
16     SECSELEMENT( "INT4", int4Class ) {
17     int i, j, items = Length / INT4_BYTES;
18
19     length = remainingBytes = Length;
20     data = ptr = new long[items];
21     itemCode = INT4_ITEM;
22
23     for ( i = 0; i < items; i++ ) {
24         data[i] = 0;
25         for ( j = 0; j < sizeof(long); j++, Int4++ ) {
26             data[i] = (data[i] << 8) + *Int4;
27         }
28     }
29 }
30
31 INT4::INT4( unsigned long Items, long DataArray[] ) :
32     SECSELEMENT( "INT4", int4Class ) {
33     int i;
```

Nov 12 09:33 1991 int4.cpp Page 2

```
35     length = remainingBytes = Items * INT4_BYTES;
36     data = ptr = new long[Items];
37     itemCode = INT4_ITEM;
38
39     for ( i = 0; i < Items; i++ )    {
40         data[i] = DataArray[i];
41     }
42 };
43
44 INT4::~INT4()    {
45     delete data;
46 };
47
48 short INT4::bytesPerItem()      {
49     return INT4_BYTES;
50 };
51
52 unsigned char *INT4::stringForMessage() {
53     unsigned char *ptrOut, *ptrSave;
54     unsigned char *itemHeader;
55     long i, items, itemLength;
56     short headerLength;
57
58     /*
59      ** Create space to return the "SECS Message ready"
60      ** version of this item.
61     */
62     itemLength = this->lengthInMessage();
63     ptrOut = new unsigned char[itemLength];
64     ptrSave = ptrOut;
65
66     /*
67      ** Get the Item Header and put it in the
68      ** return string.
```

Nov 12 09:33 1991 int4.cpp Page 3

```
69      */
70      itemHeader = this->getItemHeader( &headerLength, itemHeader );
71      memcpy( ptrOut, itemHeader, headerLength );
72      delete( itemHeader );
73
74      /*
75      **      Point to first byte of actual data to be returned,
76      **      then copy the data in.
77      */
78      ptrOut += headerLength;
79      items = length / INT4_BYTES;
80      for ( i = 0; i < items; i++ ) {
81          *ptrOut++ = data[i] >> 24 & 0xFF;
82          *ptrOut++ = data[i] >> 16 & 0xFF;
83          *ptrOut++ = data[i] >> 8 & 0xFF;
84          *ptrOut++ = data[i] & 0xFF;
85      }
86
87      return ptrSave;
88  };
89
90  int INT4::hasLong()      {
91      remainingBytes = length;
92      ptr = data;
93      return 1;
94  };
95
96  long INT4::returnLong() {
97      if ( remainingBytes <= 0 )      {
98          cerr << "INT4: trying to get data after end of contents" << endl;
99          exit( 1 );
100     }
101     else      {
102         long      tmp = *ptr;
```

Nov 12 09:33 1991 int4.cpp Page 4

```
103     ptr++;
104     remainingBytes -= INT4_BYTES;
105     return tmp;
106 }
107 );
108
109 void INT4::printOn( ostream& outputStream ) const {
110     int i, perLine = 0;
111     int items = length / INT4_BYTES;
112
113     outputStream << '<' << this->nameOf();
114
115     for ( i = 0; i < items; i++, perLine++ ) {
116         outputStream << " " << dec << data[i];
117         if ( perLine >= 10 ) {
118             perLine = 0;
119             outputStream << endl;
120         }
121     }
122
123     outputStream << ">" << flush;
124 }
```

Nov 12 09:28 1991 int8.h Page 1

```
1  /*
2   **      class: INT8      -      DECLARATIONS
3   **
4   **          @(#)int8.h      13.4      11/12/91
5   */
6
7  class INT8 : public SECSELEMENT {
8  protected:
9      double *data;
10     double *ptr;
11 public:
12     INT8( unsigned long Length, unsigned char *Int8 );
13     INT8( unsigned long Items, double DataArray[] );
14     ~INT8();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int       hasDouble();
21     double    returnDouble();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:33 1991 int8.cpp Page 1

122

```
1  /*
2  **      class: INT8      -      MEMBER FUNCTIONS
3  **
4  **          @(#)int8.cpp    13.7    11/12/91
5  */
6
7 #include <stdlib.h>
8 #include <math.h>
9 #include <stdio.h>
10 #include <iostream.h>
11 #include <object.h>
12 #include <clatypes.h>
13 #include <custom.h>
14 #include <secselem.h>
15 #include <int8.h>
16
17 INT8::INT8( unsigned long Length, unsigned char *Int8 ) :
18     SECSELEMENT( "INT8", int8Class )   {
19     short i, j, items = Length / INT8_BYTES;
20
21     if ( sizeof(double) != INT8_BYTES )      {
22         cerr << "INT8: C++ type does not match length of INT8" << endl;
23         exit( 1 );
24     }
25
26     length = remainingBytes = Length;
27     data = ptr = new double[items];
28     itemCode = INT8_ITEM;
29
30     for ( i = 0; i < items; i++ )   {
31         long             mostSignif = 0;
32         unsigned long     leastSignif = 0;
33
34         for ( j = 0; j < sizeof(long); j++, Int8++ )      {
```

Nov 12 09:33 1991 int8.cpp Page 2

```
35         leastSignif = (leastSignif << 8) + *Int8;
36         mostSignif = (mostSignif << 8) + *(Int8 + sizeof(long));
37     }
38
39     data[i] = (mostSignif * 4294967295) + leastSignif;
40 }
41 };
42
43 INT8::INT8( unsigned long Items, double dataArray[] ) :
44     SECSELEMENT( "INT8", int8Class ) {
45     short i;
46
47     if ( sizeof(double) != INT8_BYTES ) {
48         cerr << "INT8: C++ type does not match length of INT8" << endl;
49         exit( 1 );
50     }
51
52     length = remainingBytes = Items * INT8_BYTES;
53     data = ptr = new double[Items];
54     itemCode = INT8_ITEM;
55
56     for ( i = 0; i < Items; i++ ) {
57         data[i] = dataArray[i];
58     }
59
60 };
61
62 INT8::~INT8() {
63     delete data;
64 }
65
66 short INT8::bytesPerItem() {
67     return INT8_BYTES;
68 }
```

Nov 12 09:33 1991 int8.cpp Page 3

```
69
70     unsigned char *INT8::stringForMessage() {
71         unsigned char   *ptrOut, *ptrSave;
72         unsigned char   *itemHeader, *reverseFloat;
73         long      i, j, items, itemLength;
74         short     headerLength;
75
76         /*
77          ** Create space to return the "SECS Message ready"
78          ** version of this item.
79         */
80         itemLength = this->lengthInMessage();
81         ptrOut = new unsigned char[itemLength];
82         ptrSave = ptrOut;
83
84         /*
85          ** Get the Item Header and put it in the
86          ** return string.
87         */
88         itemHeader = this->getItemHeader( &headerLength, itemHeader );
89         memcpy( ptrOut, itemHeader, headerLength );
90         delete( itemHeader );
91
92         /*
93          ** Point to first byte of actual data to be returned,
94          ** then copy the data in.
95         */
96         ptrOut += headerLength;
97
98         items = length / INT8_BYTES;
99         for ( i = 0; i < items; i++ ) {
100             long            mostSignif;
101             unsigned long    leastSignif;
```

124

Nov 12 09:33 1991 int8.cpp Page 4

```
103         mostSignif = data[i] / 4294967295;
104         leastSignif = data[i] - (mostSignif * 4294967295);
105
106         *ptrOut++ = leastSignif >> 24 & 0xFF;
107         *ptrOut++ = leastSignif >> 16 & 0xFF;
108         *ptrOut++ = leastSignif >> 8 & 0xFF;
109         *ptrOut++ = leastSignif & 0xFF;
110
111         *ptrOut++ = mostSignif >> 24 & 0xFF;
112         *ptrOut++ = mostSignif >> 16 & 0xFF;
113         *ptrOut++ = mostSignif >> 8 & 0xFF;
114         *ptrOut++ = mostSignif & 0xFF;
115     }
116
117     return ptrSave;
118 }
119
120 int INT8::hasDouble() {
121     remainingBytes = length;
122     ptr = data;
123     return 1;
124 }
125
126 double INT8::returnDouble() {
127     if ( remainingBytes <= 0 ) {
128         cerr << "INT8: trying to get data after end of contents" << endl;
129         exit( 1 );
130     }
131     else {
132         double tmp = *ptr;
133         ptr += INT8_BYTES;
134         remainingBytes -= INT8_BYTES;
135         return tmp;
136     }
}
```

Nov 12 09:33 1991 int8.cpp Page 5

```
137    };
138
139 void INT8::printOn( ostream& outputStream ) const {
140     int      i, perLine = 0;
141     int      items = length / INT8_BYTES;
142
143     outputStream << '<' << this->nameOf();
144
145     for ( i = 0; i < items; i++, perLine++ )
146         outputStream << " " << data[i];
147         if ( perLine >= 4 ) {
148             perLine = 0;
149             outputStream << endl;
150         }
151     }
152
153     outputStream << ">" << flush;
154 }
```

Nov 24 10:44 1991 slf1.h Page 1

127

```
1  /*
2  **      class: S1F1      -      DECLARATIONS
3  **
4  **          @(#)slf1.h      13.1      11/24/91
5  */
6
7  class S1F1 : public STREAMFUNCTION      {
8  public:
9      S1F1();
10     S1F1( SECMESSAGE *Primary );
11     ~S1F1();
12
13     //      buildSendPrimary() will build a customized sendPrimary()
14     //      Member Function for use in the S1F1C Class, by
15     //      prompting the user through the SEMI-specified
16     //      TREE. The user should answer based on how his
17     //      specific equipment uses the SEMI-specified TREE.
18     //
19     //      buildSendSecondary() will build a customized
20     //      sendSecondary() Member Function for use in the S1F1C
21     //      Class.
22     //
23     //      buildSendPrimary()
24     //      buildSendSecondary()
25
26     void    buildSendPrimary();
27
28     void    buildSendSecondary();
29 }
```

Nov 24 10:48 1991 s1f1.cpp Page 1

```
1  /*
2  **      class: S1F1      -      MEMBER FUNCTIONS
3  **
4  **          @(#)s1f1.cpp    13.1    11/24/91
5  */
6
7 #include <stdio.h>
8 #include <iostream.h>
9 #include <fstream.h>
10 #include <dbllist.h>
11 #include <dlistelem.h>
12 #include <tree.h>
13 #include <secsmess.h>
14 #include <secstype.h>
15 #include <strfunc.h>
16 #include <s1f1.h>
17
18 S1F1::S1F1() : STREAMFUNCTION() {
19 }
20
21 S1F1::S1F1( SECMESSAGE *Primary ) : STREAMFUNCTION( Primary ) {
22 }
23
24 S1F1::~S1F1() {
25 }
26
27 void S1F1::buildSendPrimary() {
28     TREE    *modelTree, *treePtr;
29     char    dummyString[7];
30
31     treePtr = modelTree = new TREE;
32     sprintf( dummyString, "aDummy" );
33
34     treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );
```

Nov 24 10:48 1991 s1f1.cpp Page 2

```
35      this->buildSend( PRIMARY, 1, 1, modelTree );
36
37  };
38
39 void S1F1::buildSendSecondary() {
40     TREE    *modelTree, *treePtr;
41     char    dummyString[7];
42
43     treePtr = modelTree = new TREE;
44     sprintf( dummyString, "aDummy" );
45
46     treePtr = treePtr->addChild( *new TREE ( *new SLIST( 2L ) ) );
47     treePtr->setChildCnt( 2L );
48     treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );
49     treePtr = treePtr->addChild( *new TREE ( *new ASCII( 6L, dummyString ) ) );
50
51     this->buildSend( SECONDARY, 1, 1, modelTree );
52 }
```

129

Nov 24 10:44 1991 s1f1c.h Page 1

```
1  /*
2   **      class: S1F1C (Custom) -      DECLARATIONS
3   **
4   **          @(#)s1f1c.h    13.1    11/24/91
5   */
6
7  class S1F1C : public S1F1      {
8  public:
9      S1F1C();
10     S1F1C( SECSMESSAGE *Primary );
11     ~S1F1C();
12
13     void    primaryUpdate();
14     void    secondaryUpdate();
15
16     void    sendPrimary();
17     void    sendSecondary();
18 }
```

130

Nov 24 10:48 1991 s1f1c.cpp Page 1

131

```
1  /*
2  **      class: S1F1C (Custom) -      MEMBER FUNCTIONS
3  **
4  **          @(#)s1f1c.cpp    13.1    11/24/91
5  */
6
7 #include <stdio.h>
8 #include <iostream.h>
9 #include <fstream.h>
10 #include <string.h>
11 #include <dbllist.h>
12 #include <dlistelem.h>
13 #include <tree.h>
14 #include <secsmess.h>
15 #include <secstype.h>
16 #include <strfunc.h>
17 #include <s1f1.h>
18 #include <s1f1c.h>
19 #include <dasamac.h>
20
21 S1F1C::S1F1C() : S1F1() {
22 }
23
24 S1F1C::S1F1C( SECSMESSAGE *Primary ) : S1F1( Primary ) {
25 }
26
27 S1F1C::~S1F1C() {
28 }
29
30 void S1F1C::primaryUpdate() {
31     this->sendSecondary();
32 }
33
34 void S1F1C::secondaryUpdate() {
```

Nov 24 10:48 1991 s1f1c.cpp Page 2

```
35      TREE    *TREE_PTR;
36      char    model[7];
37      char    revision[7];
38
39
40      TREE_PTR = primary->returnTree();
41
42      MOVE_DOWN;
43      strncpy( model, SEPTR->returnString(), 6 );
44      MOVE_RIGHT( 1 );
45      strncpy( revision, SEPTR->returnString(), 6 );
46      MOVE_UP;
47  };
48
49 void S1F1C::sendPrimary() {
50     TREE    *TREE_ROOT, *TREE_PTR;
51     char    model[7];
52
53
54     TREE_ROOT = TREE_PTR = new TREE;
55
56     ADD_ASCII( 6L, model );
57
58     primary = new SECMESSAGE( 1, 1, REPLY, TREE_ROOT );
59
60     primary->sendMessage();
61  };
62
63 void S1F1C::sendSecondary() {
64     TREE    *TREE_ROOT, *TREE_PTR;
65     char    model[7];
66     char    revision[7];
67
68
```

Nov 24 10:48 1991 sflc.cpp Page 3

```
69
70     TREE_ROOT = TREE_PTR = new TREE;
71
72     ADD_LIST_AND_MOVEDOWN( 2L );
73     ADD_ASCII( 6L, model );
74     ADD_ASCII( 6L, revision );
75
76     secondary = new SECSMESSAGE( 1, 2, NOREPLY, TREE_ROOT );
77
78     secondary->sendMessage();
79 }
```

Nov 12 09:29 1991 s2f23.h Page 1

```
1  /*
2  **      class: S2F23 -      DECLARATIONS
3  **
4  **          @(#)s2f23.h    13.2    11/12/91
5  */
6
7  class S2F23 : public STREAMFUNCTION      {
8  public:
9      S2F23();
10     S2F23( SECMESSAGE *Primary );
11     ~S2F23();
12
13     //      buildSendPrimary() will build a customized sendPrimary()
14     //      Member Function for use in the S2F23C Class, by
15     //      prompting the user through the SEMI-specified
16     //      TREE. The user should answer based on how his
17     //      specific equipment uses the SEMI-specified TREE.
18     //
19     //      buildSendSecondary() will build a customized
20     //      sendSecondary() Member Function for use in the S2F23C
21     //      Class.
22     //
23     //      buildSendPrimary()
24     //      buildSendSecondary()
25
26     //      S2F23 can only be sent from Host to Equipment, so
27     //      these two Member Functions simply generate error
28     //      messages.
29     //
30     void    sendSecondary();
31     void    buildSendSecondary();
32
33     //      Since S2F23 can be sent from Host to Equipment, this
34     //      Member Function helps the user construct that message.
```

Nov 12 09:29 1991 82f23.h Page 2

```
35      //  
36      void      buildSendPrimary();  
37  };
```

Nov 24 10:49 1991 s2f23.cpp Page 1

```
1  /*
2   **      class: S2F23 - MEMBER FUNCTIONS
3   **
4   **          @(#)s2f23.cpp    13.3    11/24/91
5   */
6
7 #include <stdio.h>
8 #include <iostream.h>
9 #include <fstream.h>
10 #include <dbllist.h>
11 #include <dlistelem.h>
12 #include <tree.h>
13 #include <secsmess.h>
14 #include <secstype.h>
15 #include <strfunc.h>
16 #include <s2f23.h>
17
18 S2F23::S2F23() : STREAMFUNCTION()
19 {
20
21 S2F23::S2F23( SECMESSAGE *Primary ) : STREAMFUNCTION( Primary )
22 {
23
24 S2F23::~S2F23()
25 {
26
27 void S2F23::sendSecondary()
28 {
29     cerr << "This message cannot be sent from Host to Equipment" << endl;
30 }
31 void S2F23::buildSendPrimary()
32 {
33     TREE    *modelTree, *treePtr;
34     long    dummy[1];
35     char    samplePeriod[20];
```

Nov 24 10:49 1991 s2f23.cpp Page 2

```
35
36     treePtr = modelTree = new TREE;
37     dummy[0] = 0;
38     sprintf( samplePeriod, "samplePeriod" );
39
40     treePtr = treePtr->addChild( *new TREE ( *new SLIST( 5L ) ) );
41     treePtr->setChildCnt( 5L );
42     //      TRID == TRACE ID
43     treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
44     //      DSPER == SAMPLE PERIOD
45     treePtr = treePtr->addChild( *new TREE ( *new ASCII( 20L, samplePeriod ) ) );
46     //      TOTSMP == TOTAL SAMPLES
47     treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
48     //      REPGSZ == REPORTING GROUP SIZE
49     treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
50     //      SVIDs == STATUS VARIABLE IDs
51     treePtr = treePtr->addChild( *new TREE ( *new SLIST( 0L ) ) );
52     treePtr->setChildCnt( 1L );
53     treePtr = treePtr->addChild( *new TREE ( *new INT4( 1L, dummy ) ) );
54
55     this->buildSend( PRIMARY, 2, 23, modelTree );
56 }
57
58 void S2F23::buildSendSecondary()          {
59     cerr << "This message cannot be sent from Host to Equipment" << endl;
60 }
```

Nov 24 10:44 1991 s2f23c.h Page 1

```
1  /*
2  **      class: S2F23C (Custom) -      DECLARATIONS
3  **
4  **          @(#)s2f23c.h    13.3    11/24/91
5  */
6
7  class S2F23C : public S2F23      {
8  public:
9      S2F23C();
10     S2F23C( SECMESSAGE *Primary );
11     ~S2F23C();
12
13     void    secondaryUpdate();
14
15     void    sendPrimary();
16 }
```

Nov 24 10:49 1991 s2f23c.cpp Page 1

130

```
1  /*
2  **      class: S2F23C (Custom) -      MEMBER FUNCTIONS
3  **
4  **          @(#)s2f23c.cpp 13.4    11/24/91
5  */
6
7 #include <stdio.h>
8 #include <iostream.h>
9 #include <fstream.h>
10 #include <dbllist.h>
11 #include <dlstelem.h>
12 #include <tree.h>
13 #include <secsmess.h>
14 #include <secstype.h>
15 #include <strfunc.h>
16 #include <s2f23.h>
17 #include <s2f23c.h>
18 #include <dasamac.h>
19
20 S2F23C::S2F23C() : S2F23()
21 {
22
23 S2F23C::S2F23C( SECMESSAGE *Primary ) : S2F23( Primary )
24 {
25
26 S2F23C::~S2F23C()
27 {
28
29 void S2F23C::secondaryUpdate()
30 {
31     TREE    *TREE_PTR;
32     long    ack2;
33
34     TREE_PTR = secondary->returnTree();
```

Nov 24 10:49 1991 s2f23c.cpp Page 2

```
35
36     ack2 = SEPTR->returnLong();
37 }
38
39 void S2F23C::sendPrimary() {
40
41     TREE *TREE_ROOT, *TREE_PTR;
42     long traceID[1];
43     char samplePeriod[7];
44     long totalSamples[1];
45     long reportGrpSize[1];
46     long statusVarID[8];
47
48
49
50     TREE_ROOT = TREE_PTR = new TREE;
51
52     ADD_LIST_AND_MOVEDOWN( 5L );
53         ADD_INT2( 1L, traceID );
54         ADD_ASCII( 6L, samplePeriod );
55         ADD_INT2( 1L, totalSamples );
56         ADD_INT2( 1L, reportGrpSize );
57         ADD_LIST_AND_MOVEDOWN( 8L );
58         TREE *TREE_MARK = TREE_PTR;
59         for ( int i = 0; i < 8; i++ ) {
60             TREE_PTR = TREE_MARK;
61             ADD_INT2( 1L, &statusVarID[i] );
62         }
63
64     primary = new SECSMESSAGE( 2, 23, REPLY, TREE_ROOT );
65
66     primary->sendMessage();
67 }
```

Nov 24 10:44 1991 s6f9.h Page 1

```
1  /*
2  **      class: S6F9      -      DECLARATIONS
3  **
4  **          @(#)s6f9.h      13.5      11/24/91
5  */
6
7  class S6F9 : public STREAMFUNCTION      {
8  public:
9      S6F9();
10     S6F9( SECMESSAGE *Primary );
11     ~S6F9();
12
13     //      buildSendPrimary() will build a customized sendPrimary()
14     //      Member Function for use in the S6F9C Class, by
15     //      prompting the user through the SEMI-specified
16     //      TREE. The user should answer based on how his
17     //      specific equipment uses the SEMI-specified TREE.
18     //
19     //      buildSendSecondary() will build a customized
20     //      sendSecondary() Member Function for use in the S6F9C
21     //      Class.
22     //
23     //      buildSendPrimary()
24     //      buildSendSecondary()
25
26     //      S6F9 can only be sent from Equipment to Host, so
27     //      these two Member Functions simply generate error
28     //      messages.
29     //
30     void    sendPrimary();
31     void    buildSendPrimary();
32
33     //      Since S6F9 can be sent from Equipment to Host, it
34     //      follows that the Host can send the Secondary message,
```

Nov 24 10:44 1991 86f9.h Page 2

```
35      //      namely S6F10. This member function helps the user
36      //      construct that message.
37      //
38      void    buildSendSecondary();
39  };
```

Nov 24 10:49 1991 s6f9.cpp Page 1

143

```
1  /*
2  **      class: S6F9      -      MEMBER FUNCTIONS
3  **
4  **          @(#)s6f9.cpp    13.8    11/24/91
5  */
6
7  #include <iostream.h>
8  #include <fstream.h>
9  #include <dbllist.h>
10 #include <dlistelem.h>
11 #include <tree.h>
12 #include <secsmess.h>
13 #include <secstype.h>
14 #include <strfunc.h>
15 #include <s6f9.h>
16
17 S6F9::S6F9() : STREAMFUNCTION() {
18 }
19
20 S6F9::S6F9( SECMESSAGE *Primary ) : STREAMFUNCTION( Primary ) {
21 }
22
23 S6F9::~S6F9() {
24 }
25
26 void S6F9::sendPrimary() {
27     cerr << "This message cannot be sent from Host to Equipment" << endl;
28 }
29
30 void S6F9::buildSendPrimary() {
31     cerr << "This message cannot be sent from Host to Equipment" << endl;
32 }
33
34 void S6F9::buildSendSecondary() {
```

Nov 24 10:49 1991 86f9.cpp Page 2

```
35     TREE    *modelTree, *treePtr;
36     long    dummy[1];
37
38     treePtr = modelTree = new TREE;
39     dummy[0] = 0;
40
41     treePtr = treePtr->addChild( *new TREE ( *new BINARY( 1L, dummy ) ) );
42
43     this->buildSend( SECONDARY, 6, 9, modelTree );
44 }
```

Nov 24 10:45 1991 s6f9c.h Page 1

```
1  /*
2   **      class: S6F9C (Custom) -      DECLARATIONS
3   **
4   **          @(#)s6f9c.h    13.1    11/24/91
5   */
6
7  class S6F9C : public S6F9
8  public:
9      S6F9C();
10     S6F9C( SECMESSAGE *Primary );
11     ~S6F9C();
12
13     void    primaryUpdate();
14
15     void    sendSecondary();
16 }
```

14

Nov 24 10:49 1991 s6f9c.cpp Page 1

1 /*
2 * class: S6F9C (Custom) - MEMBER FUNCTIONS
3 *
4 * @(#)s6f9c.cpp 13.1 11/24/91
5 */
6
7 #include <stdio.h>
8 #include <iostream.h>
9 #include <fstream.h>
10 #include <string.h>
11 #include <dbllist.h>
12 #include <dlistelem.h>
13 #include <tree.h>
14 #include <secsmess.h>
15 #include <secstype.h>
16 #include <strfunc.h>
17 #include <s6f9.h>
18 #include <s6f9c.h>
19 #include <dasamac.h>
20
21 S6F9C::S6F9C() : S6F9()
22 {
23
24 S6F9C::S6F9C(SECMESSAGE *Primary) : S6F9(Primary)
25 {
26
27 S6F9C::~S6F9C()
28 {
29
30 void S6F9C::primaryUpdate()
31 {
32 this->sendSecondary();
33 }
34 void S6F9C::sendSecondary()

Nov 24 10:49 1991 s6f9c.cpp Page 2

```
35      TREE      *TREE_ROOT, *TREE_PTR;
36      long      ack6[1];
37
38
39      TREE_ROOT = TREE_PTR = new TREE;
40
41      ADD_BINARY( 1L, &ack6[0] );
42
43      secondary = new SECSMESSAGE( 6, 10, NOREPLY, TREE_ROOT );
44
45      secondary->sendMessage();
46  };
```

Nov 12 09:29 1991 secselem.h Page 1

```
1  /*
2  **      class: SECSELEMENT      -      DECLARATIONS
3  **
4  **      This is an "abstract class" - no instances of this class
5  **      may be declared, but classes may be derived from this class.
6  **
7  **      "Pure virtual functions" returnValue() & printOn() are
8  **      what make this an "abstract class". This means that these
9  **      functions must be defined in any classes derived from this
10 **      class.
11 **
12 **      The "= 0" after the function declarations for returnValue() &
13 **      printOn() is what identifies them as "pure virtual functions".
14 **
15 **          @(#)secselem.h 13.9    11/12/91
16 */
17
18 #define WORKSPACE      8
19
20 class SECSELEMENT : public Object      {
21 protected:
22     char          *className;
23     classType     classTypeValue;
24     unsigned char itemCode;
25     unsigned long length; // Number of bytes in guts of item.
26     unsigned long remainingBytes;
27 public:
28     SECSELEMENT( char *ClassName, classType ClassType );
29     virtual ~SECSELEMENT();
30
31     virtual int           isEqual( const Object& ) const;
32     virtual classType     isA( ) const;
33     virtual char          *nameOf( ) const;
34     virtual hashValueType hashCode( ) const;
```

Nov 12 09:29 1991 secselem.h Page 2

```
35
36     unsigned char           getItemCode();
37     virtual unsigned char   *getItemHeader( short *HeaderLength,
38                                         unsigned char *ItemHeader );
39     virtual long            lengthInMessage();
40     virtual unsigned char   *stringForMessage() = 0;
41
42     virtual int             hasDouble();
43     virtual int             hasLong();
44     virtual int             hasString();
45     virtual double          returnDouble();
46     virtual long            returnLong();
47     virtual char             *returnString();
48
49     virtual long            itemsRemaining();
50     virtual short           bytesPerItem() = 0;
51
52     virtual void            printOn( ostream& ) const = 0;
53 };
54
55 void printBits( char *desc, int bytes, unsigned char *ptr );
```

Nov 12 09:36 1991 secselem.cpp Page 1

```
1  /*
2   **      class: SECSELEMENT      -      MEMBER FUNCTIONS
3   **
4   **          @(#)secselem.cpp      13.7      11/12/91
5   */
6
7  #include <string.h>
8  #include <stdio.h>
9  #include <mem.h>
10 #include <stdlib.h>
11 #include <values.h>
12 #include <iostream.h>
13 #include <object.h>
14 #include <secselem.h>
15
16 SECSELEMENT::SECSELEMENT( char *ClassName, classType ClassType )      {
17     className = ClassName;
18     classTypeValue = ClassType;
19 }
20
21 SECSELEMENT::~SECSELEMENT()      {
22 }
23
24 SECSELEMENT::isEqual( const Object& testSECSELEMENT ) const      {
25     //return ( data == (SECSELEMENT &)testSECSELEMENT );
26     return ( 0 );
27 }
28
29 classType SECSELEMENT::isA() const      {
30     return classTypeValue;
31 }
32
33 char *SECSELEMENT::nameOf() const      {
34     return className;
```

Nov 12 09:36 1991 secselem.cpp Page 2

```
35  };
36
37 hashValueType SECSELEMENT::hashValue() const {
38     hashValueType value = hashValueType( 0 );
39     return( value );
40 }
41
42 unsigned char SECSELEMENT::getItemCode() {
43     return itemCode;
44 }
45
46 unsigned char *SECSELEMENT::getItemHeader( short *HeaderLength,
47     unsigned char *ItemHeader ) {
48     unsigned char formatByte;
49     unsigned char lengthBytes, *ptrSave;
50     unsigned long tmpLength;
51     int i;
52
53     /*
54      ** Create space to return the "Item Header"
55      ** version of this item.
56     */
57     lengthBytes = this->lengthInMessage() - 1 - length;
58     *HeaderLength = 1 + lengthBytes;
59     ItemHeader = new unsigned char[*HeaderLength];
60     ptrSave = ItemHeader;
61
62     /*
63      ** Determine the Format Byte and put it in the
64      ** return string.
65     */
66     formatByte = this->getItemCode() << 2;
67     formatByte |= lengthBytes;
68     *ItemHeader = formatByte;
```

151

Nov 12 09:36 1991 secselem.cpp Page 3

```
69
70     /*
71      **      Point to last length byte and populate the length
72      **      bytes in reverse order.
73      */
74     ItemHeader += lengthBytes;
75     tmpLength = length;
76     for ( i = 0; i < lengthBytes; i++ )      {
77         *ItemHeader = tmpLength & 0x000000FF;
78         tmpLength = tmpLength >> 8;
79         ItemHeader--;
80     }
81
82     return ptrSave;
83 }
84
85 long SECSELEMENT::lengthInMessage()      {
86     unsigned char    lengthBytes;
87     int      i;
88     unsigned long   temp;
89
90     if ( length == 0 )      {
91         lengthBytes = 1;
92     }
93     else      {
94         temp = length;
95         for ( i = 0; i < sizeof(long); i++ )      {
96             if ( temp == 0 )      {
97                 i -= 1;
98                 break;
99             }
100            temp = temp >> 8;
101        }
102        lengthBytes = i + 1;
```

Nov 12 09:36 1991 secselem.cpp Page 4

```
103         }
104
105         return (long) 1 + lengthBytes + length;
106     );
107
108     int SECSELEMENT::hasDouble()      {
109         return 0;
110     );
111
112     int SECSELEMENT::hasLong()       {
113         return 0;
114     );
115
116     int SECSELEMENT::hasString()    {
117         return 0;
118     );
119
120     double SECSELEMENT::returnDouble()   {
121         return 0.0;
122     );
123
124     long SECSELEMENT::returnLong()    {
125         return 0;
126     );
127
128     char *SECSELEMENT::returnString()  {
129         return NULL;
130     );
131
132     long SECSELEMENT::itemsRemaining() {
133         return remainingBytes / this->bytesPerItem();
134     );
135
136     void
```

Nov 12 09:36 1991 secselem.cpp Page 5

```
137     printBits( char *desc, int bytes, unsigned char *ptr ) {
138         int      i, j;
139         char    tmpChar;
140
141         printf( "\s:\n\t", desc );
142
143         for ( i = 0; i < bytes; i++ ) {
144
145             tmpChar = *ptr++;
146             for ( j = 0; j < BITSPERBYTE; j++ ) {
147                 if ( tmpChar & 0x80 ) {
148                     printf( "1" );
149                 }
150                 else {
151                     printf( "0" );
152                 }
153                 if ( j & 2 ) {
154                     printf( " " );
155                 }
156                 tmpChar = tmpChar << 1;
157             }
158             printf( "\n\t" );
159         }
160         printf( "\n" );
161     }
```

Nov 25 14:04 1991 secsmess.h Page 1

```
1  /*
2   *      class: SECMESSAGE      -      DECLARATIONS
3   *
4   *      @(#)secsmess.h 13.7    11/12/91
5   */
6
7 #ifndef __CUSTOM_H
8 #include <custom.h>
9 #define __CUSTOM_H
10#endif
11
12#define MEMCHUNK      512
13
14
15#define NOREPLY      0
16#define REPLY        1
17
18
19#define CHARCNOTOVERHEAD      1
20#define HEADEROVERHEAD      10
21#define NONOVERHEADINBLOCK  243
22#define CHECKSUMOVERHEAD    2
23
24#define OVERHEADINBLOCK    ( CHARCNOTOVERHEAD + HEADEROVERHEAD + CHECKSUMOVERHEAD )
25#define BYTESINBLOCK      ( OVERHEADINBLOCK + NONOVERHEADINBLOCK )
26
27
28 class SECMESSAGE      {
29 protected:
30     unsigned char      *messageFlat;
31     short             messageLength;
32     TREE              *messageTree;
33
34     /*
```

Nov 25 14:04 1991 secsmess.h Page 2

```
35      ** This data is part of the incoming SECS-II message &
36      ** will be used to instantiate the specific SECS
37      ** object.
38      */
39      unsigned char      secsStream, secsFunction;
40      unsigned char      upperDevID, lowerDevID;
41      unsigned char      upperBlockNo, lowerBlockNo;
42      unsigned char      systemByte[4];
43      truthFlag          msgToHost, replyRequired;
44
45      /*
46      ** This data is part of the incoming SECS-II message,
47      ** but will be used only help separate the wheat from
48      ** the chaff of the message.
49      */
50      truthFlag          lastBlock;
51
52
53      void      parseMessage( unsigned char *MessageIn, short Length );
54
55      void      addItemToMessage( TREE *TreePtr, short *MessCharCnt );
56      void      expandMessage();
57      void      addSECSheader( short *MessCharCnt );
58
59  public:
60      SECSMESSAGE( unsigned char *MessageIn, short Length );
61      SECSMESSAGE( char *FileName );
62      SECSMESSAGE( short Stream, short Function, int ReplyRequired, TREE *TreePtr );
63      ~SECSMESSAGE();
64
65      void      buildMessage();
66      void      sendMessage();
67
68      short     getSECSstream();
```

Nov 25 14:04 1991 secsmess.h Page 3

```
69     short      getSECSfunction();
70
71     int       isPrimary();
72
73     TREE      *returnTree();
74
75     void      printOn( ostream& ) const;
76 }
77
78 long    bytesValue( short numBytes, register unsigned char *fromString );
```

Nov 25 14:11 1991 secsmess.cpp Page 1

158

```
1  /*
2   **      class: SECMESSAGE      -      MEMBER FUNCTIONS
3   **
4   **          @(#)secsmess.cpp      13.10  11/24/91
5   */
6
7  #include <string.h>
8  #include <stdio.h>
9  #include <alloc.h>
10 #include <mem.h>
11 #include <iostream.h>
12 #include <values.h>
13 #include <process.h>
14 #include <fstream.h>
15 #include <object.h>
16 #include <clstypes.h>
17
18 #ifndef __CUSTOM_H
19 #include <custom.h>
20 #define __CUSTOM_H
21 #endif
22
23 #include <secstype.h>
24 #include <dbllist.h>
25 #include <dlstelem.h>
26 #include <tree.h>
27 #include <secsmess.h>
28
29 SECMESSAGE::SECMESSAGE( unsigned char *MessageIn, short Length ) {
30     /*      CHANGE
31     **
32     **          messageTree->addParentType( "LIST" );
33     */
34     this->parseMessage( MessageIn, Length );
```

Nov 25 14:11 1991 secsmess.cpp Page 2

```
35    };
```

```
36
```

```
37    SECMESSAGE::SECMESSAGE( char *fileName )      {
```

```
38        long   fileSize;
```

```
39        unsigned char  charIn, *messageIn;
```

```
40        char   fullFileName[SIZEFILE];
```

```
41        short  i = 0;
```

```
42
```

```
43        strcpy( fullFileName, SECSTESTDIR );
```

```
44        strcat( fullFileName, fileName );
```

```
45
```

```
46        ifstream sourceFile( fullFileName, ios::binary );
```

```
47        if ( ! sourceFile )      {
```

```
48            cerr << "SECMESSAGE: cannot open file " <<
```

```
49            fullFileName << " for input" << endl;
```

```
50            exit( 1 );
```

```
51        }
```

```
52
```

```
53        /*
```

```
54         ** Seek to end of the file, then request byte
```

```
55         ** position in file to determine file size. Size
```

```
56         ** messageIn[] based on this, after seeking back
```

```
57         ** to the start of the file.
```

```
58     */
```

```
59     sourceFile.seekg( 0L, (seek_dir) 2 );
```

```
60     fileSize = sourceFile.tellg();
```

```
61     sourceFile.seekg( 0L, (seek_dir) 0 );
```

```
62     messageIn = new unsigned char[fileSize];
```

```
63
```

```
64     sourceFile.read( messageIn, fileSize );
```

```
65     if ( sourceFile.gcount() != fileSize ) {
```

```
66         cout << "SECMESSAGE: did not get the expected number " <<
```

```
67         "of characters from the file." << endl;
```

```
68         exit( 1 );
```

Nov 25 14:11 1991 secsmess.cpp Page 3

```
69      }
70
71      sourceFile.close();
72
73      /*      CHANGE
74      **
75      **      messageTree->addParentType( "LIST" );
76      */
77      this->parseMessage( messageIn, fileSize );
78  };
79
80  SECSMESSAGE::SECSMESSAGE( short Stream, short Function, int ReplyRequired,
81  TREE *TreePtr ) {
82      /*      CHANGE
83      **
84      **      messageTree->addParentType( "LIST" );
85      */
86      messageTree = TreePtr;
87      messageFlat = NULL;
88
89      secsStream = Stream;
90      secsFunction = Function;
91      replyRequired = (truthFlag) ReplyRequired;
92
93      /*      CHANGE
94      **
95      **      systemByte = ??????
96      **      upperDevID = ??????
97      **      lowerDevID = ??????
98      */
99      systemByte[0] = (unsigned char) 0;
100     systemByte[1] = (unsigned char) 2;
101     systemByte[2] = (unsigned char) 2;
102     systemByte[3] = (unsigned char) 3;
```

Nov 25 14:11 1991 secsmess.cpp Page 4

```
103         upperDevID = 54;
104         lowerDevID = 59;
105     };
106
107     SECSMESSAGE::~SECSMESSAGE() {
108         if ( messageFlat != NULL ) {
109             delete messageFlat;
110         }
111
112         if ( messageTree != NULL ) {
113             delete messageTree;
114         }
115     };
116
117     void
118     SECSMESSAGE::parseMessage( unsigned char *MessageIn, short Length ) {
119         unsigned char    *messInPtr = MessageIn;
120         messageFlat = new unsigned char[Length];
121         unsigned char    *messFlatPtr;
122
123         short    i, blockCnt = 0;
124         short    charCnt, blockCharCnt, messCharCnt = 0;
125         short    formatByte, itemCode, numLengthBytes;
126
127         messageLength = Length;
128
129
130         lastBlock = FALSE;
131
132         while ( ! lastBlock ) {
133             blockCnt++;
134             blockCharCnt = *messInPtr++ - 10; // subtract out Header bytes
135             messCharCnt += blockCharCnt;
136     }
```

Nov 25 14:11 1991 secsmess.cpp Page 5

```
137     msgToHost = ( *messInPtr & 0200 ) >> 7;
138     upperDevID = *messInPtr++ & ~0200;
139     lowerDevID = *messInPtr++;
140
141     replyRequired = ( *messInPtr & 0200 ) >> 7;
142     secsStream = *messInPtr++ & ~0200;
143     secsFunction = *messInPtr++;
144
145     lastBlock = ( *messInPtr & 0200 ) >> 7;
146     upperBlockNo = *messInPtr++ & ~0200;
147     lowerBlockNo = *messInPtr++;
148
149     for ( i = 0; i < 4; i++ ) {
150         systemByte[i] = *messInPtr++;
151     }
152
153     messFlatPtr = messageFlat + messCharCnt - blockCharCnt;
154
155     while ( blockCharCnt -- ) {
156         *messFlatPtr++ = *messInPtr++;
157     }
158
159     messInPtr += 2;      // Checksum
160 }
161 delete MessageIn;
162
163
164     messageTree = new TREE;
165     TREE    *treePtr = messageTree;
166
167     messFlatPtr = messageFlat;
168     charCnt = 0;
169     while ( charCnt < messCharCnt ) {
170         unsigned long      length;
```

Nov 25 14:11 1991 secmess.cpp Page 6

```
171
172     formatByte = ( short ) *messFlatPtr++;
173     itemCode = formatByte >> 2;
174     numLengthBytes = formatByte & 0003;
175
176     length = bytesValue( numLengthBytes, messFlatPtr );
177     messFlatPtr += numLengthBytes;
178
179     switch ( itemCode ) {
180         case LIST_ITEM:
181             treePtr = treePtr->addChild( *new TREE
182                 ( *new SLIST ( length ) ) );
183             /*           INVESTIGATE:
184             **           which "part" of ROOT is this set for ?
185             */
186             treePtr->setChildCnt( length );
187             length = 0;
188             break;
189         case ASCII_ITEM:
190             treePtr = treePtr->addChild( *new TREE
191                 ( *new ASCII ( length, (char *) messFlatPtr ) ) );
192             break;
193         case BINARY_ITEM:
194             treePtr = treePtr->addChild( *new TREE
195                 ( *new BINARY ( length, messFlatPtr ) ) );
196             break;
197         case BOOLEAN_ITEM:
198             treePtr = treePtr->addChild( *new TREE
199                 ( *new BOOLEAN ( length, messFlatPtr ) ) );
200             break;
201         case FLOAT4_ITEM:
202             treePtr = treePtr->addChild( *new TREE
203                 ( *new FLOAT4 ( length, messFlatPtr ) ) );
204             break;
```

Nov 25 14:11 1991 secmess.cpp Page 7

```
205     case FLOAT8_ITEM:
206         treePtr = treePtr->addChild( *new TREE
207             ( *new FLOAT8 ( length, messFlatPtr ) ) );
208         break;
209     case INT1_ITEM:
210         treePtr = treePtr->addChild( *new TREE
211             ( *new INT1 ( length, messFlatPtr ) ) );
212         break;
213     case INT2_ITEM:
214         treePtr = treePtr->addChild( *new TREE
215             ( *new INT2 ( length, messFlatPtr ) ) );
216         break;
217     case INT4_ITEM:
218         treePtr = treePtr->addChild( *new TREE
219             ( *new INT4 ( length, messFlatPtr ) ) );
220         break;
221     case INT8_ITEM:
222         treePtr = treePtr->addChild( *new TREE
223             ( *new INT8 ( length, messFlatPtr ) ) );
224         break;
225     case UINT8_ITEM:
226         treePtr = treePtr->addChild( *new TREE
227             ( *new UINT8 ( length, messFlatPtr ) ) );
228         break;
229     case UINT1_ITEM:
230         treePtr = treePtr->addChild( *new TREE
231             ( *new UINT1 ( length, messFlatPtr ) ) );
232         break;
233     case UINT2_ITEM:
234         treePtr = treePtr->addChild( *new TREE
235             ( *new UINT2 ( length, messFlatPtr ) ) );
236         break;
237     case UINT4_ITEM:
238         treePtr = treePtr->addChild( *new TREE
```

Nov 25 14:11 1991 secsmess.cpp Page 8

165

```
239             ( *new UINT4 ( length, messFlatPtr ) ) );
240             break;
241         default:
242             cout << "||||< got an UNKNOWN >>>\n" << endl;
243             break;
244         }
245
246         messFlatPtr += length;
247         charCnt += ( 1 + numLengthBytes + length );
248
249         while ( treePtr->gotAllChildren() ) {
250             if ( ! treePtr->atRoot() ) {
251                 treePtr = treePtr->moveUp();
252             }
253             else {
254                 if ( charCnt != messCharCnt ) {
255                     cout << "AT ROOT AND ALL CHARS ARE NOT READ";
256                 }
257                 break;
258             }
259         }
260     }
261
262     /*
263     free( messageFlat );
264     */
265
266     if ( messCharCnt != 0 && ! treePtr->atRoot() )
267         cout << "NOT AT ROOT, BUT ALL CHARACTERS ARE READ";
268
269     };
270
271 void SECSMESSAGE::buildMessage() {
272     char    fullFileName[SIZEFILE];
```

Nov 25 14:11 1991 secsmess.cpp Page 9

```
273     char    baseFileName[SIZEFILE];
274     short   blockCharCnt;
275
276     strcpy( fullFileName, SECSTESTDIR );
277     sprintf( baseFileName, "X_S%d-F%d", this->getSECSstream(),
278             this->getSECSfunction() );
279     strcat( fullFileName, baseFileName );
280
281     ofstream destFile( fullFileName, ios::binary );
282     if( !destFile ) {
283         cerr << "SECSMESSAGE: cannot open file " <<
284             fullFileName << " for output" << endl;
285     }
286
287     /*
288      **      Allocate memory for raw message to Equipment.
289      */
290     if ( messageFlat != NULL )      {
291         delete messageFlat;
292     }
293     messageFlat = new unsigned char[MEMCHUNK];
294     messageLength = MEMCHUNK;
295
296     /*
297      **      Run through the SECS TREE, adding the raw data
298      **      from each item to the message that will be sent
299      **      to the Equipment.
300      */
301
302     TREE    *treePtr = messageTree;
303     short   messCharCnt = 0;
304
305     if ( treePtr != NULL )  {
306         /*
```

Nov 25 14:11 1991 secsmess.cpp Page 10

```
307      ** If TREE has Children.  
308      */  
309      if ( treePtr->hasChildren() )      {  
310          this->addItemToMessage( treePtr, &messCharCnt );  
311          /*  
312          ** Move Down to Left-most Child.  
313          */  
314          treePtr = treePtr->moveDown();  
315          treePtr->resetChildCursor();  
316          /*  
317          ** While not at Root of Super-TREE ...  
318          */  
319          while( ! treePtr->atRoot() )      {  
320              /*  
321              ** If TREE has Children.  
322              */  
323              if ( treePtr->hasChildren() )      {  
324                  /*  
325                  ** If visited all Children of TREE.  
326                  */  
327                  if ( treePtr->gotAllChildren() )      {  
328                      /*  
329                      ** If TREE has Sibling to Right.  
330                      */  
331                      if ( treePtr->hasRightSibling() )  {  
332                          /*  
333                          ** Move to Sibling to Right.  
334                          */  
335                          treePtr = treePtr->moveRight( 1 );  
336                          treePtr->resetChildCursor();  
337                      }  
338                      else          {  
339                          /*  
340                          ** Move Up to Parent.
```

Nov 25 14:11 1991 secsmess.cpp Page 11

```
341          */
342          treePtr = treePtr->moveUp();
343      }
344  }
345  else {
346      this->addItemToMessage( treePtr, &messCharCnt );
347      /*
348      ** Move Down to Child.
349      */
350      treePtr = treePtr->moveDown();
351      treePtr->resetChildCursor();
352  }
353 }
354 else {
355     this->addItemToMessage( treePtr, &messCharCnt );
356     /*
357     ** If TREE has Sibling to Right.
358     */
359     if ( treePtr->hasRightSibling() ) {
360         /*
361         ** Move to Sibling to Right.
362         */
363         treePtr = treePtr->moveRight( 1 );
364         treePtr->resetChildCursor();
365     }
366     else {
367         /*
368         ** Move Up to TREE's Parent.
369         */
370         treePtr = treePtr->moveUp();
371     }
372 }
373 }
374 }
```

Nov 25 14:11 1991 secsmess.cpp Page 12

```
375         else          {
376             this->addItemToMessage( treePtr, &messCharCnt );
377         }
378     }
379
380     this->addSECSheader( &messCharCnt );
381
382
383
384     destFile.write( messageFlat, messageLength );
385
386     destFile.close();
387 }
388
389 void SECSMESSAGE::sendMessage() {
390     this->buildMessage();
391 }
392
393 void SECSMESSAGE::addItemToMessage( TREE *treePtr, short *MessCharCnt ) {
394     SECSELEMENT      *sePtr;
395     short            itemLength;
396
397     sePtr = ((SECSELEMENT *) treePtr->nodeAddr());
398     itemLength = (short) sePtr->lengthInMessage();
399
400     if ( *MessCharCnt + itemLength > messageLength )           {
401         this->expandMessage();
402     }
403
404
405     memcpy( messageFlat + *MessCharCnt,
406            sePtr->stringForMessage(), itemLength );
407     free( sePtr->stringForMessage() );
408     *MessCharCnt += itemLength;
```

Nov 25 14:11 1991 secsmess.cpp Page 13

```
409     };
410
411 void SECSMESSAGE::expandMessage()          {
412     unsigned char   *oldMessage, *newMessage;
413     short           newMessageLength = messageLength + MEMCHUNK;
414
415     oldMessage = messageFlat;           // save for delete
416
417     newMessage = new unsigned char[newMessageLength];
418     memcpy( newMessage, messageFlat, messageLength );
419
420     messageFlat = newMessage;
421     messageLength = newMessageLength;
422
423     delete oldMessage;
424 }
425
426
427 void SECSMESSAGE::addSECSheader( short *MessCharCnt )  {
428     int      i, bytesToCopy = NONOVERHEADINBLOCK;
429     unsigned char  setBit = 1 << 7;
430     unsigned char  *tmpMessage, *itemsPtr, *outPtr;
431
432     int      fullBlocks = *MessCharCnt / NONOVERHEADINBLOCK;
433     int      leftOvers = *MessCharCnt % NONOVERHEADINBLOCK;
434     int      totalBlocks = fullBlocks + ( leftOvers ? 1 : 0 );
435
436     int      sizeOfMessageOut = *MessCharCnt;
437
438     /*
439     ** Will happen when the message consists of only the
440     ** header (e.g., S?-F0).
441     */
442     if ( *MessCharCnt == 0 )      {
```

Nov 25 14:11 1991 secsmess.cpp Page 14

```
443         totalBlocks = 1;
444         bytesToCopy = 0;
445     }
446
447     sizeOfMessageOut += totalBlocks * OVERHEADINBLOCK;
448
449     outPtr = tmpMessage = new unsigned char[sizeOfMessageOut];
450
451     itemsPtr = messageFlat; // Point to start of "items only".
452
453     while ( totalBlocks-- ) {
454         if ( ( totalBlocks == 0 ) && leftOvers )      {
455             bytesToCopy = leftOvers;
456         }
457         *outPtr++ = (unsigned char) bytesToCopy + HEADEROVERHEAD;
458
459         *outPtr++ = upperDevID;           // MESSAGE TO HOST - not set here
460         *outPtr++ = lowerDevID;
461
462         *outPtr++ = secsStream;          // REPLY REQUIRED
463         *outPtr++ = secsFunction;
464
465         *outPtr++ = upperBlockNo | ( totalBlocks == 0 ? setBit : 0 );
466         *outPtr++ = lowerBlockNo;
467         for ( i = 0; i < 4; i++ )    {
468             *outPtr++ = systemByte[i];
469         }
470         memcpy( outPtr, itemsPtr, bytesToCopy );
471         outPtr += bytesToCopy;
472         itemsPtr += bytesToCopy;
473
474         *outPtr++ = 0xFE;
475         *outPtr++ = 0xFF;
476     }
```

Nov 25 14:11 1991 secsmess.cpp Page 15

```
477
478     itemsPtr = messageFlat; // save message address for delete
479
480     messageFlat = tmpMessage;
481     messageLength = sizeOfMessageOut;
482
483     delete itemsPtr;           // delete old message
484 }
485
486 short SECSMESSAGE::getSECSstream() {
487     return (short) this->secsStream;
488 }
489
490 short SECSMESSAGE::getSECSfunction() {
491     return (short) this->secsFunction;
492 }
493
494 int SECSMESSAGE::isPrimary() {
495     return secsFunction & 1;
496 }
497
498 TREE *SECSMESSAGE::returnTree() {
499     return this->messageTree;
500 }
501
502 void SECSMESSAGE::printOn( ostream& outputStream ) const {
503     outputStream << "S" << (short) this->secsStream;
504     outputStream << "-F" << (short) this->secsFunction << endl;
505     this->messageTree->printOn( outputStream );
506 }
507
508
509
510
```

```
511
512  /*
513  **      Get the requested number of bytes from a string of unsigned char's,
514  **      building the value as each byte is encountered. Return the total
515  **      value after all bytes have been visited.
516  **
517  **      Because we will be putting 1, 2, 3, ... bytes into a "long" type
518  **      we may need to extend the sign bit to fill a "long". We only need to
519  **      do this if the high-order bit is set in the first byte we encounter.
520  */
521 long bytesValue( short numBytes, register unsigned char *fromString )
522 {
523     int     i;
524     register long          value = 0;
525     long    mask, byteMask;
526     short   hiBitValue = 1;
527     truthFlag      isNegative = FALSE;
528
529
530
531  /*
532  **      Determine the value of a byte with only the highest bit set.
533  **      If the first byte of the string we are evaluating has this
534  **      bit set, we have a negative number. If we have a negative
535  **      number, we will have to extend the SIGN BIT.
536  */
537 for ( i = 0; i < BITSPERBYTE - 1; i++ )
538     hiBitValue *= 2;
539
540 if ( *fromString & hiBitValue )
541     isNegative = TRUE;
542
543
544
```

174

```
545      /*
546      ** Determine the value of the number of bytes in memory
547      ** without regard to sign.
548      */
549      for ( i = 0; i < numBytes; i++ )
550          value = (value << BITSPERBYTE) + *fromString++;
551
552
553
554      if ( isNegative )      {
555          /*
556          ** Create a byte-sized mask of all 1's.
557          */
558          byteMask = ( 2 * hiBitValue ) - 1;
559
560          /*
561          ** Create a long-sized mask for sign extension of our final value.
562          ** The highest-order bytes that are not used by our value are set
563          ** to all 1's. Assuming 8-bit bytes and 4-byte "long"s, here are
564          ** what the masks would look like:
565          **
566          **           numBytes == 1           mask=0xfffffff00
567          **           numBytes == 2           mask=0xfffff0000
568          **           numBytes == 3           mask=0xff0000000
569          **           numBytes == 4           mask=0x000000000
570          */
571          mask = 0;
572          for ( i = 0; i < numBytes; i++ )      {
573              mask = mask << BITSPERBYTE;
574              mask |= byteMask;
575          }
576          mask = ~mask;
577
578
```

Nov 25 14:11 1991 secmess.cpp Page 18

```
579
580         /*
581          ** Do the sign extension.
582          */
583         value = mask | value;
584     }
585
586
587
588     return( value );
589 }
```

Nov 12 09:30 1991 slist.h Page 1

```
1  /*
2  **      class: SLIST -      DECLARATIONS
3  **
4  **          @(#)slist.h    13.6    11/12/91
5  */
6
7  class SLIST : public SECSELEMENT
8 protected:
9 public:
10    SLIST( unsigned long Length );
11    ~SLIST();
12
13    short      bytesPerItem();
14
15    unsigned char *getItemHeader( short *HeaderLength,
16                                unsigned char *ItemHeader );
17    long       lengthInMessage();
18    unsigned char *stringForMessage();
19
20    int        hasLong();
21    long       returnLong();
22
23    long       itemsRemaining();
24
25    virtual void printOn( ostream& ) const;
26 }
```

Nov 12 09:37 1991 slist.cpp Page 1

```
1  /*
2   **      class: SLIST -      MEMBER FUNCTIONS
3   **
4   **          @(#)slist.cpp    13.7    11/12/91
5   */
6
7  #include <iostream.h>
8  #include <object.h>
9  #include <clstypes.h>
10 #include <custom.h>
11 #include <secselem.h>
12 #include <slist.h>
13
14 SLIST::SLIST( unsigned long Length ) :
15     SECSELEMENT( "LIST", slistClass ) {
16     length = Length;
17     itemCode = LIST_ITEM;
18 }
19
20 SLIST::~SLIST() {
21 }
22
23 short SLIST::bytesPerItem() {
24     return LIST_BYTES;
25 }
26
27 /*
28 **      SLIST requires a different version of getItemHeader()
29 **      because the "length" is already subtracted out of the
30 **      "lengthInMessage()" for SLIST --- it must be subtracted
31 **      out for all SECSELEMENT subclasses other than SLIST.
32 */
33 unsigned char *SLIST::getItemHeader( short *HeaderLength,
34                                     unsigned char *ItemHeader ) {
```

LL1

Nov 12 09:37 1991 slist.cpp Page 2

```
35     unsigned char    formatByte;
36     unsigned char    lengthBytes, *ptrSave;
37     unsigned long    tmpLength;
38     int      i;
39
40     /*
41     **      Create space to return the "Item Header"
42     **      version of this item.
43     */
44     lengthBytes = this->lengthInMessage() - 1;
45     *HeaderLength = 1 + lengthBytes;
46     ItemHeader = new unsigned char[*HeaderLength];
47     ptrSave = ItemHeader;
48
49     /*
50     **      Determine the Format Byte and put it in the
51     **      return string.
52     */
53     formatByte = this->getItemCode() << 2;
54     formatByte |= lengthBytes;
55     *ItemHeader = formatByte;
56
57     /*
58     **      Point to last length byte and populate the length
59     **      bytes in reverse order.
60     */
61     ItemHeader += lengthBytes;
62     tmpLength = length;
63     for ( i = 0; i < lengthBytes; i++ )      {
64         *ItemHeader = tmpLength & 0x000000FF;
65         tmpLength = tmpLength >> 8;
66         ItemHeader--;
67     }
68
```

Nov 12 09:37 1991 slist.cpp Page 3

```
69         return ptrSave;
70     };
71
72     long SLIST::lengthInMessage()    {
73         return (long) SECSELEMENT::lengthInMessage() - length;
74     };
75
76     unsigned char *SLIST::stringForMessage()           {
77         unsigned char   *ptrOut, *ptrSave;
78         unsigned char   *itemHeader;
79         long      i, itemLength;
80         short     headerLength;
81
82         /*
83         **      Create space to return the "SECS Message ready"
84         **      version of this item.
85         */
86         itemLength = this->lengthInMessage();
87         ptrOut = new unsigned char[itemLength];
88         ptrSave = ptrOut;
89
90         /*
91         **      Get the Item Header and put it in the
92         **      return string.
93         */
94         itemHeader = this->getItemHeader( &headerLength, itemHeader );
95         memcpy( ptrOut, itemHeader, headerLength );
96         delete( itemHeader );
97
98         return ptrSave;
99     };
100
101    int SLIST::hasLong()      {
102        return 1;
```

Nov 12 09:37 1991 slist.cpp Page 4

```
103  };
104
105 long SLIST::returnLong()          {
106     return length;
107 }
108
109 long SLIST::itemsRemaining()     {
110     /*
111      **      There's always 1 item: the length of the LIST.
112      */
113     return 1L;
114 }
115
116 void SLIST::printOn( ostream& outputStream ) const    {
117     outputStream << '<' << this->nameOf() << ' ' << dec << length << ">" << flush;
118 }
```

Nov 24 10:45 1991 strfunc.h Page 1

```
1  /*
2   **      class: STREAMFUNCTION -      DECLARATIONS
3   **
4   **          @(#)strfunc.h    13.5    11/24/91
5   */
6
7 #define PRIMARY          0
8 #define SECONDARY         1
9
10#define INDENTMAX        80
11
12 class STREAMFUNCTION {
13 protected:
14     SECSMESSAGE *primary;
15     SECSMESSAGE *secondary;
16
17     char Indent[INDENTMAX];
18     int IndentCnt;
19     int IndentInc;
20
21     ofstream *classFile;
22     ofstream *declFile;
23     ofstream *codeFile;
24
25     void classWrite( char *String );
26     void declWrite( char *String );
27     void codeWrite( char *String );
28 public:
29     STREAMFUNCTION();
30     STREAMFUNCTION( SECSMESSAGE *SECSmess );
31     virtual ~STREAMFUNCTION();
32
33     void graftSecondary( SECSMESSAGE *Secondary );
34
```

Nov 24 10:45 1991 strfunc.h Page 2

18

```
35      //      Send SECS Message to Equipment.  
36      //  
37      virtual void    sendPrimary();  
38      virtual void    sendSecondary();  
39  
40      virtual void    buildSendPrimary();  
41      virtual void    buildSendSecondary();  
42  
43      //      Update the DASA based on SECS Message from the  
44      //      Equipment:  
45      //  
46      //      primaryUpdate() - updates based on a Primary message  
47      //                      received from the Equipment.  
48      //  
49      //      secondaryUpdate() - updates based on a Secondary message  
50      //                      received from the Equipment.  
51      //  
52      virtual void    primaryUpdate();  
53      virtual void    secondaryUpdate();  
54  
55      //      Prompt user through a TREE received from the  
56      //      Equipment.  
57      //  
58      void            buildPrimaryUpdate();  
59      void            buildSecondaryUpdate();  
60      void            buildUpdate( int SMtype );  
61      void            updateTreeWalk( TREE *TreePtr );  
62      void            updatePromptUser( TREE *TreePtr );  
63  
64      //      Prompt user through a TREE to be sent to the  
65      //      Equipment.  
66      //  
67      void            buildSend( int SMtype, short Stream,  
68                      short Function, TREE *TreePtr );
```

Nov 24 10:45 1991 strfunc.h Page 3

```
69      void      sendTreeWalk( TREE *TreePtr );
70      void      sendPromptUser( TREE *TreePtr );
71
72      void      printTreeLocale( TREE *TreePtr );
73      void      getVariable( char *Type, int *Length, char *Name );
74      int       variableList( TREE *TreePtr );
75      int       parentVariableList( TREE *TreePtr );
76
77      int       getStream();
78      int       getPrimaryFunction();
79      int       getSecondaryFunction();
80
81      void      indentMore();
82      void      indentLess();
83
84      virtual void      printOn( ostream& ) const;
85  };
```

Nov 25 14:18 1991 strfunc.cpp Page 1

```
1  /*
2  **      class: STREAMFUNCTION -      MEMBER FUNCTIONS
3  **
4  **          @(#)strfunc.cpp 13.6    11/24/91
5  */
6
7  #include <stdio.h>
8  #include <string.h>
9  #include <iostream.h>
10 #include <fstream.h>
11 #include <dbllist.h>
12 #include <dlstelem.h>
13 #include <tree.h>
14 #include <secsmess.h>
15 #include <secstype.h>
16 #include <strfunc.h>
17
18 STREAMFUNCTION::STREAMFUNCTION() {
19     primary = NULL;
20     secondary = NULL;
21
22     IndentCnt = 0;
23     IndentInc = 2;
24     memset( Indent, ' ', INDENTMAX );
25     Indent[IndentCnt] = '\0';
26 }
27
28 STREAMFUNCTION::STREAMFUNCTION( SECSMESSAGE *SECSmess ) {
29     if ( SECSmess->isPrimary() ) {
30         primary = SECSmess;
31         secondary = NULL;
32     }
33     /*
34     **      This part of the constructor would only be exercised
```

Nov 25 14:18 1991 strfunc.cpp Page 2

```
35      ::      in "artificial" situations, e.g., if we were sending
36      ::      Secondary messages from the Equipment (as files) that
37      ::      were not in response to Primary messages that the Host
38      ::      had sent in order to "buildSecondaryUpdate()".
39      */
40      else {
41          primary = NULL;
42          secondary = SECSMESS;
43      }
44
45      IndentCnt = 0;
46      IndentInc = 2;
47      memset( Indent, ' ', INDENTMAX );
48      Indent[IndentCnt] = '\0';
49  };
50
51 STREAMFUNCTION::~STREAMFUNCTION() {
52     if ( primary != NULL ) {
53         delete primary;
54     }
55
56     if ( secondary != NULL ) {
57         delete secondary;
58     }
59 }
60
61 void STREAMFUNCTION::graftSecondary( SECSMESSAGE *Secondary ) {
62     secondary = Secondary;
63 }
64
65 void STREAMFUNCTION::sendPrimary() {
66 }
67
68 void STREAMFUNCTION::sendSecondary() {
```

Nov 25 14:18 1991 strfunc.cpp Page 3

```
69      TREE    *treePtr = NULL;
70      short   stream, function;
71
72      /*
73      **      Send S**FO to close the transaction on the Equipment.
74      **
75      **      This code will only be exercised if the user does not
76      **      override with a more specific response.
77      */
78      stream = primary->getSECSstream();
79      function = 0;
80
81      secondary = new SECSMESSAGE( stream, function, NOREPLY, treePtr );
82
83      secondary->sendMessage();
18
84  };
85
86 void STREAMFUNCTION::buildSendPrimary() {
87 };
88
89 void STREAMFUNCTION::buildSendSecondary() {
90 };
91
92 void STREAMFUNCTION::primaryUpdate() {
93 };
94
95 void STREAMFUNCTION::secondaryUpdate() {
96 };
97
98 void STREAMFUNCTION::buildPrimaryUpdate() {
99     this->buildUpdate( PRIMARY );
100 };
101
102 void STREAMFUNCTION::buildSecondaryUpdate() {
```

Nov 25 14:18 1991 strfunc.cpp Page 4

```
103         this->buildUpdate( SECONDARY );
104     };
105
106     /*
107      **      Something in here causes "Null Pointer Assignment"
108      */
109 void STREAMFUNCTION::buildUpdate( int SMtype ) {
110     char    className[SIZEFILE];
111     char    classFileName[SIZEFILE];
112     char    declFileName[SIZEFILE];
113     char    codeFileName[SIZEFILE];
114     char    memFuncName[20];           // Name of Member Function to be
115                                     // generated.
116     char    smTypeDecl[10];          // To initialize (TREE *) in
117                                     // code to be generated.
118     char    fileSuffix[10];          // Suffix of file where generated
119                                     // code will be written to.
120     char    genString[80];
121     ifstream      *declIn, *codeIn;
122     TREE          *treePtr;
123     SECMESSAGE    *smPtr;
124
125     if ( SMtype == PRIMARY )        {
126         smPtr = primary;
127         sprintf( memFuncName, "primaryUpdate" );
128         sprintf( smTypeDecl, "primary" );
129         sprintf( fileSuffix, "pu" );
130     }
131     else   {
132         smPtr = secondary;
133         sprintf( memFuncName, "secondaryUpdate" );
134         sprintf( smTypeDecl, "secondary" );
135         sprintf( fileSuffix, "su" );
136     }
```

Nov 25 14:18 1991 strfunc.cpp Page 5

```
137     treePtr = smPtr->returnTree();
138
139     sprintf( className, "StdFndC",
140             smPtr->getSECSstream(), smPtr->getSECSfunction() );
141
142     sprintf( classFileName, "%s.%s", className, fileSuffix );
143
144     sprintf( declFileName, "decl.tmp" );
145
146     sprintf( codeFileName, "code.tmp" );
147
148     /*
149      **      Open the Class File and write the Member Function
150      **      declaration and definition.
151      */
152     classFile = new ofstream( classFileName, ios::in );
153     if ( ! classFile )      {
154         cerr << "STREAMFUNCTION: cannot open file " <<
155             classFileName << " for input" << endl;
156     }
157
158     sprintf( genString, "/*\t%s() DECLARATION\n", memFuncName );
159     this->classWrite( genString );
160     sprintf( genString, "\tvoid\t%s();\n\n", memFuncName );
161     this->classWrite( genString );
162     sprintf( genString, "/*\t%s() DEFINITION\n", memFuncName );
163     this->classWrite( genString );
164     sprintf( genString, "void %s::%s()\t{\n\n", className, memFuncName );
165     this->classWrite( genString );
166
167
168
169     /*
170      **      Open temporary files to hold the declarations & code
```

Nov 25 14:18 1991 strfunc.cpp Page 6

```
171      **      that will be internal to the Member Function. Separate
172      **      files are necessary because the declaration & code will
173      **      not be generated in the order we want; we have to keep
174      **      them apart to get our desired order.
175      */
176      declFile = new ofstream( declFileName, ios::out );
177      if ( ! declFile )      {
178          cerr << "STREAMFUNCTION: cannot open file " <<
179              declFileName << " for output" << endl;
180      }
181
182      codeFile = new ofstream( codeFileName, ios::out );
183      if ( ! codeFile )      {
184          cerr << "STREAMFUNCTION: cannot open file " <<
185              codeFileName << " for output" << endl;
186      }
187
188      cout << "\n\n\n====="
189          "=====\n" << endl;
190      cout << "BUILDING Member Function to update DASA based on" << endl;
191      cout << "\tStream " << smPtr->getSECSstream() << " - Function "
192          smPtr->getSECSfunction() << " message received from Equipment.\n\n" << endl;
193      cout << "When prompted by '==> ' :" << endl;
194      cout << "\t- enter Variable Name to save data, OR" << endl;
195      cout << "\t- enter 'n' to NOT save data.\n\n" << endl;
196
197
198
199      /*
200      **      Generate declaration & initialization for (TREE *).
201      */
202      this->declWrite( "\tTREE      *TREE_PTR;\n" );
203      sprintf( genString, "\tTREE_PTR = %s->returnTree();\n\n", smTypeDecl );
204      this->codeWrite( genString );
```

18

Nov 25 14:18 1991 strfunc.cpp Page 7

```
205
206     /*
207      **      Walk the TREE in prefix order, prompting the user
208      **      for his desires, and generating declarations & code
209      **      as we go.
210     */
211     this->updateTreeWalk( treePtr );
212
213     /*
214      **      We're finished putting into the temporary files:
215      **      close them up.
216     */
217     // declFile->seekp( 0L, (seek_dir) 0 );
218     // codeFile->seekp( 0L, (seek_dir) 0 );
219     declFile->close();
220     codeFile->close();
221     delete declFile;
222     delete codeFile;
223
224     /*
225      **      Open up the temporary files to get input FROM them,
226      **      to be written into the Class File.
227     */
228     declIn = new ifstream( declFileName, ios::in );
229     if ( ! declFile )      {
230         cerr << "STREAMFUNCTION: cannot open file " <<
231             declFileName << " for input" << endl;
232     }
233
234     codeIn = new ifstream( codeFileName, ios::in );
235     if ( ! codeFile )      {
236         cerr << "STREAMFUNCTION: cannot open file " <<
237             codeFileName << " for input" << endl;
238 }
```

Nov 25 14:18 1991 strfunc.cpp Page 8

```
239
240     while ( declIn->getline( genString, 80 ) )      {
241         this->classWrite( genString );
242         this->classWrite( "\n" );
243     }
244
245     this->classWrite( "\n\n\n" );
246
247     while ( codeIn->getline( genString, 80 ) )      {
248         this->classWrite( genString );
249         this->classWrite( "\n" );
250     }
251
252     /*
253      **      We're finished putting into the temporary files:
254      **      close them up.
255     */
256     declIn->close();
257     codeIn->close();
258     delete declIn;
259     delete codeIn;
260
261     /*
262      **      Finish-up the Class File.
263     */
264     this->classWrite( "};\n" );
265     classFile->close();
266     delete classFile;
267 };
268
269 /*
270  **      Something in here causes "Null Pointer Assignment"
271 */
272 void STREAMFUNCTION::buildSend( int SMtype, short Stream, short Function,
```

Nov 25 14:18 1991 strfunc.cpp Page 9

192

```
273 TREE *TreePtr ) {
274     char    className[SIZEFILE];
275     char    classFileName[SIZEFILE];
276     char    declFileName[SIZEFILE];
277     char    codeFileName[SIZEFILE];
278     char    memFuncName[20];           // Name of Member Function to be
279                                         // generated.
280     char    smTypeDecl[10];          // To initialize (TREE *) in
281                                         // code to be generated.
282     char    fileSuffix[10];          // Suffix of file where generated
283                                         // code will be written to.
284     char    genString[80];
285     ifstream      *declIn, *codeIn;
286
287     sprintf( className, "S%dF%dC", Stream, Function );
288
289     if ( SMtype == PRIMARY )        {
290         sprintf( memFuncName, "sendPrimary" );
291         sprintf( smTypeDecl, "primary" );
292         sprintf( fileSuffix, "sp" );
293     }
294     else   {
295         Function++;
296         sprintf( memFuncName, "sendSecondary" );
297         sprintf( smTypeDecl, "secondary" );
298         sprintf( fileSuffix, "ss" );
299     }
300
301     sprintf( classFileName, "%s.%s", className, fileSuffix );
302
303     sprintf( declFileName, "decl.tmp" );
304
305     sprintf( codeFileName, "code.tmp" );
306
```

Nov 25 14:18 1991 strfunc.cpp Page 10

```
307      /*
308      **      Open the Class File and write the Member Function
309      **      declaration and definition.
310      */
311      classFile = new ofstream( className, ios::in );
312      if ( ! classFile )      {
313          cerr << "STREAMFUNCTION: cannot open file " <<
314              className << " for input" << endl;
315      }
316
317      sprintf( genString, "/*\t%s() DECLARATION\n", memFuncName );
318      this->classWrite( genString );
319      sprintf( genString, "\tvoid\t%s();\n\n", memFuncName );
320      this->classWrite( genString );
321      sprintf( genString, "/*\t%s() DEFINITION\n", memFuncName );
322      this->classWrite( genString );
323      sprintf( genString, "void %s::%s()\t{\n\n", className, memFuncName );
324      this->classWrite( genString );
325
326
327
328      /*
329      **      Open temporary files to hold the declarations & code
330      **      that will be internal to the Member Function. Separate
331      **      files are necessary because the declaration & code will
332      **      not be generated in the order we want; we have to keep
333      **      them apart to get our desired order.
334      */
335      declFile = new ofstream( declFileName, ios::out );
336      if ( ! declFile )      {
337          cerr << "STREAMFUNCTION: cannot open file " <<
338              declFileName << " for output" << endl;
339      }
340
```

Nov 25 14:18 1991 strfunc.cpp Page 11

```
194
341     codeFile = new ofstream( codeFileName, ios::out );
342     if ( ! codeFile )      {
343         cerr << "STREAMFUNCTION: cannot open file " <<
344             codeFileName << " for output" << endl;
345     }
346
347     cout << "\n\n\n=====\n=====\n" <<
348         "=====\n=====\n" << endl;
349     cout << "BUILDING Member Function to send\n" <<
350         "\tStream " << Stream << " - Function " <<
351         Function << " message to Equipment.\n\n" << endl;
352     cout << "When prompted by '==> ' :" << endl;
353     cout << "\t- for LIST, enter number of Items in LIST," << endl;
354     cout << "\t- for Items, enter <type number variableName>," << endl;
355     cout << "\t e.g., INT2 1 waferID.\n\n" << endl;
356
357
358
359 /*
360 **      Generate declaration & initialization for TREE and (TREE *).
361 */
362 this->declWrite( "\tTREE\t*TREE_ROOT, *TREE_PTR;\n" );
363 this->codeWrite( "\tTREE_ROOT = TREE_PTR = new TREE;\n\n" );
364
365 /*
366 **      Walk the TREE in prefix order, prompting the user
367 **      for his desires, and generating declarations & code
368 **      as we go.
369 */
370 this->sendTreeWalk( TreePtr );
371
372 //      CHANGE: REPLY should also be a variable passed into
373 //              this MF
374 if ( SMtype == PRIMARY )      {
```

Nov 25 14:18 1991 strfunc.cpp Page 12

```
375     sprintf( genString, "\n\t@s =\n"
376             new SECSMESSAGE( &d, &d, REPLY, TREE_ROOT );\n\n",
377             smTypeDecl, Stream, Function );
378 }
379 else {
380     sprintf( genString, "\n\t@s =\n"
381             new SECSMESSAGE( &d, &d, NOREPLY, TREE_ROOT );\n\n",
382             smTypeDecl, Stream, Function );
383 }
384 this->codeWrite( genString );
385 sprintf( genString, "\t@s->sendMessage();\n", smTypeDecl );
386 this->codeWrite( genString );
387
388 /*
389 **      We're finished putting into the temporary files:
390 **      close them up.
391 */
392 // declFile->seekp( 0L, (seek_dir) 0 );
393 // codeFile->seekp( 0L, (seek_dir) 0 );
394 declFile->close();
395 codeFile->close();
396 delete declFile;
397 delete codeFile;
398
399 /*
400 **      Open up the temporary files to get input FROM them,
401 **      to be written into the Class File.
402 */
403 declIn = new ifstream( declFileName, ios::in );
404 if ( ! declFile ) {
405     cerr << "STREAMFUNCTION: cannot open file " <<
406         declFileName << " for input" << endl;
407 }
408
```

Nov 25 14:18 1991 strfunc.cpp Page 13

106

```
409     codeIn = new ifstream( codeFileName, ios::in );
410     if ( ! codeFile )          {
411         cerr << "STREAMFUNCTION: cannot open file " <<
412             codeFileName << " for input" << endl;
413     }
414
415     while ( declIn->getline( genString, 80 ) )      {
416         this->classWrite( genString );
417         this->classWrite( "\n" );
418     }
419
420     this->classWrite( "\n\n\n" );
421
422     while ( codeIn->getline( genString, 80 ) )      {
423         this->classWrite( genString );
424         this->classWrite( "\n" );
425     }
426
427     /*
428     **      We're finished putting into the temporary files:
429     **      close them up.
430     */
431     declIn->close();
432     codeIn->close();
433     delete declIn;
434     delete codeIn;
435
436     /*
437     **      Finish-up the Class File.
438     */
439     this->classWrite( "};\n" );
440     classFile->close();
441     delete classFile;
442 }
```

Nov 25 14:18 1991 strfunc.cpp Page 14

167
443
444 void STREAMFUNCTION::updateTreeWalk(TREE *TreePtr) {
445 char genString[80];
446
447 if (TreePtr != NULL) {
448 cout << Indent;
449 this->updatePromptUser(TreePtr);
450
451 if (TreePtr->hasChildren()) {
452 TreePtr = TreePtr->moveDown();
453 TreePtr->resetChildCursor();
454 sprintf(genString, "\t%sMOVE_DOWN;\n", Indent);
455 this->codeWrite(genString);
456 this->indentMore();
457
458 this->updateTreeWalk(TreePtr);
459
460 this->indentLess();
461 TreePtr = TreePtr->moveUp();
462 sprintf(genString, "\t%sMOVE_UP;\n", Indent);
463 this->codeWrite(genString);
464 }
465 else if (TreePtr->hasRightSibling()) {
466 TreePtr = TreePtr->moveRight(1);
467 sprintf(genString, "\t%sMOVE_RIGHT(1);\n", Indent);
468 this->codeWrite(genString);
469
470 this->updateTreeWalk(TreePtr);
471 }
472 }
473 };
474
475 void STREAMFUNCTION::sendTreeWalk(TREE *TreePtr) {
476 char genString[80];

Nov 25 14:18 1991 strfunc.cpp Page 15

108
477
478 if (TreePtr != NULL) {
479 cout << Indent;
480 this->sendPromptUser(TreePtr);
481
482 if (TreePtr->hasChildren()) {
483 TreePtr = TreePtr->moveDown();
484 TreePtr->resetChildCursor();
485 this->indentMore();
486
487 this->sendTreeWalk(TreePtr);
488
489 this->indentLess();
490 TreePtr = TreePtr->moveUp();
491 if (((SECSELEMENT *)TreePtr->nodeAddr())->returnLong() == 0) {
492 sprintf(genString, "\t%s\n", Indent);
493 this->codeWrite(genString);
494 }
495 }
496 else if (TreePtr->hasRightSibling()) {
497 TreePtr = TreePtr->moveRight(1);
498
499 this->sendTreeWalk(TreePtr);
500 }
501 }
502 };
503
504 void STREAMFUNCTION::updatePromptUser(TREE *TreePtr) {
505 SECSELEMENT *sePtr;
506 long itemsInObject;
507 int i;
508 char genString[80]; // General-purpose string.
509 char typeString[80]; // String for declaration.
510 char initString[80]; // String for instantiation.

Nov 25 14:18 1991 strfunc.cpp Page 16

```
511
512     sePtr = ((SECSELEMENT *) TreePtr->nodeAddr());
513     itemsInObject = sePtr->itemsRemaining();
514
515     sePtr->printOn( cout );
516
517     this->printTreeLocale( TreePtr );
518
519     if ( sePtr->isA() == slistClass )          {
520         cout << endl;
521     }
522     else if ( sePtr->hasDouble() )   {
523         cout << " ==> " << flush;
524         sprintf( typeString, "\tdouble\t" );
525         sprintf( initString, "= SEPTR->returnDouble();" );
526     }
527     else if ( sePtr->hasLong() )    {
528         cout << " ==> " << flush;
529         sprintf( typeString, "\tlong\t" );
530         sprintf( initString, "= SEPTR->returnLong();" );
531     }
532     else if ( sePtr->hasString() )  {
533         cout << " ==> " << flush;
534         itemsInObject += 1;           // make room for NULL at end
535         sprintf( typeString, "\tchar\t" );
536         sprintf( initString, "SEPTR->returnString()" );
537     }
538     else   {
539         cout << " ERROR: Object has " << itemsInObject;
540         cout << " item(s) of UNKNOWN type" << endl;
541     }
542
543     if ( sePtr->isA() != slistClass )      {
544         /*
```

Nov 25 14:18 1991 strfunc.cpp Page 17

```
545      /* Prompt the user for a variable name.
546      */
547      char      variableName[80];
548      cin >> variableName;
549
550      if ( ( strlen( variableName ) == 1 ) &&
551          ( strncmp( "n", variableName, 1 ) == 0 ) )      {
552          /*
553          ** DO NOTHING
554          */
555      }
556      else      {
557          /*
558          ** Write declaration for the variable.
559          */
560          strcat( typeString, variableName );
561          if ( itemsInObject > 1 || sePtr->hasString() ) {
562              sprintf( genString, "[%d];\n", itemsInObject );
563          }
564          else      {
565              sprintf( genString, ";"\n );
566          }
567          strcat( typeString, genString );
568          this->declWrite( typeString );
569
570          /*
571          ** Write code to populate the variable.
572          */
573          if ( sePtr->hasString() )      {
574              sprintf( genString, "\t%sstrncpy( %s, %s, %d );\n",
575                  Indent, variableName, initString, itemsInObject - 1 );
576              this->codeWrite( genString );
577          }
578          else if ( itemsInObject > 1 )      {
```

Nov 25 14:18 1991 strfunc.cpp Page 18

```
579         sprintf( genString, "\t@sfor ( i = 0; i < \d; i++ )\t{\n",
580                 Indent, itemsInObject );
581         this->codeWrite( genString );
582         for ( i = 0; i < itemsInObject; i++ ) {
583             sprintf( genString, "\t\t@s(\d) @s\n",
584                     Indent, variableName, i, initString );
585             this->codeWrite( genString );
586         }
587         sprintf( genString, "\t@s)\n", Indent );
588         this->codeWrite( genString );
589     }
590     else {
591         sprintf( genString, "\t@s@ s\n",
592                 Indent, variableName, initString );
593         this->codeWrite( genString );
594     }
595 }
596 }
597 };
598
599 void STREAMFUNCTION::sendPromptUser( TREE *TreePtr ) {
600     SECSELEMENT      *sePtr;
601     long              itemsInObject;
602     int               i;
603     char              genString[80]; // General-purpose string.
604     char              typeString[80]; // String for declaration.
605     char              initString[80]; // String for instantiation.
606
607     char              seType[10];
608     int               length;
609     char              variableName[30];
610
611     sePtr = ((SECSELEMENT *) TreePtr->nodeAddr());
612     itemsInObject = sePtr->itemsRemaining();
```

Nov 25 14:18 1991 strfunc.cpp Page 19

```
613
614     if ( sePtr->isA() == slistClass )          {
615         long      numberOfChildren;
616
617         if ( this->variableList( TreePtr ) )        {
618             cout << "<list>";
619         }
620         else      {
621             sePtr->printOn( cout );
622         }
623
624         this->printTreeLocale( TreePtr );
625
626         // Prompt user for number of items they want in LIST:
627         // an answer of 0 items in LIST will effectively
628         // "prune" the TREE at this point.
629         //
630         cout << " items in LIST ==> " << flush;
631         cin >> numberOfChildren;
632         sprintf( genString, "\t%sADD_LIST_AND_MOVEDOWN( %ldL );\n",
633                 Indent, numberOfChildren );
634         this->codeWrite( genString );
635
636         if ( numberOfChildren == 0 )          {
637             /*
638              **      Delete children of the Model we're tracking.
639              */
640             while ( TreePtr->getItemsInContainer() != 0 )  {
641                 TreePtr->destroyFromTail( TreePtr->peekAtTail() );
642             }
643             TreePtr->setChildCnt( 0 );
644
645             /*
646              **      This is an empty LIST --- move back up in
```

Nov 25 14:18 1991 strfunc.cpp Page 20

```
647          ..      the code we're generating.  
648          /*  
649             sprintf( genString, "\t\tMOVEUP();\n", Indent );  
650             this->codeWrite( genString );  
651         }  
652     else if ( this->variableList( TreePtr ) && numberOfChildren > 1 ) {  
653         //      So we'll know size of array to declare for children.  
654         TreePtr->setChildCnt( numberOfChildren );  
655         sprintf( genString, "\t\tTREE\t*TREE_MARK = TREE_PTR;\n",  
656                     Indent );  
657         this->codeWrite( genString );  
658         sprintf( genString, "\t\tfor ( int i = 0; i < %d; i++ )\t{\n",  
659                     Indent, numberOfChildren );  
660         this->codeWrite( genString );  
661         sprintf( genString, "\t\tTREE_PTR = TREE_MARK;\n", Indent );  
662         this->codeWrite( genString );  
663     }  
664 }  
665 else if ( sePtr->hasDouble() ) {  
666     cout << "<double>";  
667     this->printTreeLocale( TreePtr );  
668     cout << " ==> " << flush;  
669     sprintf( typeString, "\tdouble\t" );  
670     this->getVariable( seType, &length, variableName );  
671     strcat( typeString, variableName );  
672     if ( this->parentVariableList( TreePtr ) ) {  
673         TREE *tempTree = TreePtr->moveUp();  
674         sprintf( genString, "[%d];\n", tempTree->getChildCnt() );  
675         strcat( typeString, genString );  
676         sprintf( genString, "\t\tADD_%s( 1L, &%s[i] );\n",  
677                     Indent, seType, variableName );  
678     }  
679     else {  
680         sprintf( genString, "[%d];\n", length );
```

Nov 25 14:18 1991 strfunc.cpp Page 21

```
681         strcat( typeString, genString );
682         sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
683                  Indent, seType, length, variableName );
684     }
685     this->declWrite( typeString );
686     this->codeWrite( genString );
687 }
688 else if ( sePtr->hasLong() ) {
689     cout << "<long>";
690     this->printTreeLocale( TreePtr );
691     cout << " ==> " << flush;
692     sprintf( typeString, "\tlong\t" );
693     this->getVariable( seType, &length, variableName );
694     strcat( typeString, variableName );
695     if ( this->parentVariableList( TreePtr ) ) {
696         TREE *tempTree = TreePtr->moveUp();
697         sprintf( genString, "[%d];\n", tempTree->getChildCnt() );
698         tempTree->setChildCnt( 1 );
699         strcat( typeString, genString );
700         sprintf( genString, "\t%sADD_%s( 1L, &%s[i] );\n",
701                  Indent, seType, variableName );
702     }
703     else {
704         sprintf( genString, "[%d];\n", length );
705         strcat( typeString, genString );
706         sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
707                  Indent, seType, length, variableName );
708     }
709     this->declWrite( typeString );
710     this->codeWrite( genString );
711 }
712 else if ( sePtr->hasString() ) {
713     cout << "<string>";
714     this->printTreeLocale( TreePtr );
```

Nov 25 14:18 1991 strfunc.cpp Page 22

205

```
715         cout << " ==> " << flush;
716         itemsInObject += 1;           // make room for NULL at end
717         sprintf( typeString, "\tchar\t" );
718         this->getVariable( seType, &length, variableName );
719         strcat( typeString, variableName );
720         sprintf( genString, "[\d];\n", length + 1 );
721         strcat( typeString, genString );
722         this->declWrite( typeString );
723         sprintf( genString, "\t%sADD_%s( %dL, %s );\n",
724                  Indent, seType, length, variableName );
725         this->codeWrite( genString );
726     }
727     else {
728         cout << "<?????>";
729         this->printTreeLocale( TreePtr );
730         cout << " ERROR: Object has " << itemsInObject;
731         cout << " item(s) of UNKNOWN type" << endl;
732     }
733 }
734
735 void STREAMFUNCTION::classWrite( char *String ) {
736     int      stringLength = strlen( String );
737
738     classFile->write( String, stringLength );
739 }
740
741 void STREAMFUNCTION::declWrite( char *String ) {
742     int      stringLength = strlen( String );
743
744     declFile->write( String, stringLength );
745 }
746
747 void STREAMFUNCTION::codeWrite( char *String ) {
748     int      stringLength = strlen( String );
```

Nov 25 14:18 1991 strfunc.cpp Page 23

```
749
750         codeFile->write( String, stringLength );
751     };
752
753     void STREAMFUNCTION::printTreeLocale( TREE *TreePtr ) {
754         cout << "\t" << TreePtr->getLevel() << ":" <<
755             TreePtr->getParentsChildCursor();
756     };
757
758     void STREAMFUNCTION::getVariable( char *Type, int *Length, char *Name ) {
759         cin >> Type;
760         cin >> *Length;
761         cin >> Name;
762     };
763
206    764     int STREAMFUNCTION::variableList( TREE *TreePtr ) {
765         return ( ((SECSELEMENT *) TreePtr->nodeAddr())->returnLong() == 0 );
766     };
767
768     int STREAMFUNCTION::parentVariableList( TREE *TreePtr ) {
769         if ( TreePtr->atRoot() ) {
770             return 1;
771         }
772         else {
773             return( this->variableList( TreePtr->moveUp() ) );
774         }
775     };
776
777     void STREAMFUNCTION::indentMore() {
778         Indent[IndentCnt] = ' ';
779         IndentCnt += IndentInc;
780         Indent[IndentCnt] = '\0';
781     };
782
```

Nov 25 14:18 1991 strfunc.cpp Page 24

```
783 void STREAMFUNCTION::indentLess()           {
784     Indent[IndentCnt] = ' ';
785     IndentCnt -= IndentInc;
786     Indent[IndentCnt] = '\0';
787 }
788
789 int STREAMFUNCTION::getStream() {
790     return primary->getSECSstream();
791 }
792
793 int STREAMFUNCTION::getPrimaryFunction()      {
794     return primary->getSECSfunction();
795 }
796
797 int STREAMFUNCTION::getSecondaryFunction()    {
798     return ( primary->getSECSfunction() + 1 );
799 }
800
801 void STREAMFUNCTION::printOn( ostream& outputStream ) const {
802     primary->printOn( outputStream );
803 }
```

Nov 12 09:31 1991 tree.h Page 1

```
1  /*
2   **      class: TREE      -      DECLARATIONS
3   **
4   **          @(#)tree.h      13.6      11/12/91
5   */
6
7  class TREE : public DoubleList {
8  protected:
9      Object *node;
10     TREE *parent;
11     int level;
12     int childCursor;
13     int childCnt;
14     char *className;
15     classType    classTypeValue;
16 public:
17     TREE( );
18     TREE( Object& Node );
19     ~TREE();
20     classType    isA() const;
21     char *nameOf() const;
22     int atRoot();
23     int mayBeParent();
24     int hasChildren();
25     int hasRightSibling();
26     int hasLeftSibling();
27     TREE *addChild( TREE& Child );
28     void setParent( TREE *Parent );
29     TREE *moveUp();
30     TREE *moveDown();
31     TREE *moveRight( int MoveNodes );
32     TREE *moveLeft( int MoveNodes );
33     int getLevel();
34     void setLevel( int Level );
```

Nov 12 09:31 1991 tree.h Page 2

```
35      void      setChildCnt( int ChildCnt );
36      void      resetChildCursor();
37      int       getChildCnt();
38      int       gotAllChildren();
39      int       getChildCursor();
40      int       getParentsChildCursor();
41      Object   *nodeAddr();
42      virtual void      printOn( ostream& ) const;
43  };
```

Nov 12 09:44 1991 tree.cpp Page 1

210

```
1  /*
2   *      class: TREE - MEMBER FUNCTIONS
3   *
4   *          @(#)tree.cpp    13.8    11/12/91
5   */
6
7 #include <stddef.h>
8 #include <string.h>
9 #include <iostream.h>
10 #include <alloc.h>
11 #include <mem.h>
12 #include <object.h>
13 #include <dbllist.h>
14 #include <clstypes.h>
15 #include <custom.h>
16 #include <tree.h>
17
18 #define INDENTSTART      5
19 #define INDENTPERLEVEL   2
20
21 /*
22  *      The first constructor is used to create an empty TREE -
23  *      this will be the "ROOT subtree". This empty TREE object
24  *      must be created so that the member functions to add nodes
25  *      (e.g., addChild()) can be employed - you have to have a
26  *      TREE before you can use its member functions !!!
27  *
28  *      A client of the TREE class would typically use the first
29  *      constructor to create the "ROOT subtree", then use the
30  *      second constructor for all other subtrees.
31  *
32  *      The ROOT of the "ROOT subtree" (i.e., the ROOT of the
33  *      entire tree) is grafted to the initial empty TREE:
34  *      a member function such as addChild() sees that the
```

Nov 12 09:44 1991 tree.cpp Page 2

```
35    **      "node" is empty & grafts the "node" of the passed TREE
36    **      to the empty "node".
37    */
38
39    TREE::TREE( )      {
40        node = NULL;
41        parent = NULL;
42        level = 0;
43        childCnt = 0;
44        childCursor = 0;
45        className = "TREE";
46        classTypeValue = treeClass;
47    };
48
49    TREE::TREE( Object& Node )      {
50        node = &Node;
51        parent = NULL;
52        childCnt = 0;
53        childCursor = 0;
54        className = "TREE";
55        classTypeValue = treeClass;
56    };
57
58    TREE::~TREE()      {
59        if ( node != NULL )      {
60            delete node;
61        }
62    };
63
64    classType TREE::isA() const      {
65        return classTypeValue;
66    };
67
68    char *TREE::nameOf() const      {
```

Nov 12 09:44 1991 tree.cpp Page 3

```
69         return className;
70     };
71
72     int TREE::atRoot()      {
73         return ( parent == NULL );
74     };
75
76     /**
77     ** HAVE TO FLESH THIS GUY OUT --- USE A (LIST *) AS
78     ** MEMBER DATA, ADD THE THING PASSED HERE TO THE LIST.
79     **
80     ** USE COLLECTION MEMBER FUNCTION hasMember() IN THE
81     ** mayBeParent() member function below.
82     **
83     void TREE::addParentType( char* className )      {
84     };
85     */
86
87     int TREE::mayBeParent() {
88         return ( strcmp( node->nameOf(), "LIST" ) ? FALSE : TRUE );
89     };
90
91     int TREE::hasChildren() {
92         return ( this->isEmpty() ? FALSE : TRUE );
93     };
94
95     int TREE::hasRightSibling()      {
96         if ( this->atRoot() )      {
97             return FALSE;
98         }
99         else if ( ( parent->childCnt - parent->childCursor ) < 1 )      {
100            return FALSE;
101        }
102        else      {
```

Nov 12 09:44 1991 tree.cpp Page 4

```
103             return TRUE;
104         }
105     };
106
107     int TREE::hasLeftSibling()      {
108         if ( this->atRoot() )    {
109             return FALSE;
110         }
111         else if ( parent->childCursor < 2 )    {
112             return FALSE;
113         }
114         else    {
115             return TRUE;
116         }
117     };
118
119 /**
120 **      Return (TREE *) so that client of TREE Class can move
121 **      around within super-TREE.
122 */
123     TREE *TREE::addChild( TREE& Child )      {
124         if ( node == NULL )    {
125             /**
126             **          ENHANCEMENT:
127             **
128             **  Could we point "this" to the Child, basically
129             **  overlaying the root node with the child ???
130             **
131             **          this = &Child;    DID NOT WORK
132             */
133             /**
134             **  There is no root node, so make the incoming TREE
135             **  the root node.
136             */

```

Nov 12 09:44 1991 tree.cpp Page 5

```
137         node = Child.nodeAddr();
138         //
139         // Is this setting the level of the orphaned child ?
140         //
141         Child.setLevel( 0 );
142         return this;
143     }
144     else {
145         childCursor++;
146         this->addAtTail( Child );
147         Child.setParent( this );
148         Child.setLevel( this->getLevel() + 1 );
149         if ( Child.mayBeParent() ) {
150             return &Child;
151         }
152         else {
153             return this;
154         }
155     }
156 }
157
158 void TREE::setParent( TREE *Parent ) {
159     parent = Parent;
160 }
161
162 /**
163 ** Returns pointer to ROOT of super-TREE
164 */
165 TREE *TREE::toRoot() {
166     if ( ! atRoot() ) {
167         return( parent->toRoot() );
168     }
169     else {
170         return this;
```

Nov 12 09:44 1991 tree.cpp Page 6

```
171         }
172     };
173     */
174
175     TREE *TREE::moveUp()    {
176         if ( ! this->atRoot() ) {
177             return parent;
178         }
179         return this;
180     };
181
182     TREE *TREE::moveDown()  {
183         /*
184         **      The cursor is always at the left-most child
185         **      when we moveDown().
186         */
187         if ( this->hasChildren() ) {
188             childCursor = 1;
189             return (TREE *) &( this->peekAtHead() );
190         }
191         return this;
192     };
193
194     /*
195     **      moveRight() & moveLeft() are almost identical -- need to
196     **      combine them into one member function, perhaps called
197     **      by two "tailoring" functions, or perhaps called with
198     **      a LEFT | RIGHT flag.
199     */
200     TREE *TREE::moveRight( int MoveNodes )  {
201         int      tmpChildCursor;
202
203         if ( this->atRoot() )  {
204             cout << "moveRight(): at ROOT, cannot move" << endl;
```

Nov 12 09:44 1991 tree.cpp Page 7

```
205      }
206      else {
207          /*
208          ** Make sure the requested move is valid; if it is not,
209          ** return the address of the "current" node.
210          */
211          if ( parent->childCursor + MoveNodes > parent->childCnt )    {
212              cout << "moveRight(): cannot move " << MoveNodes <<
213                  " nodes" << endl;
214              return this;
215          }
216
217          /*
218          ** Initialize an iterator to the start of the list of
219          ** the "current" node's parent's children. The fact
220          ** that the "current" node may not be at the start of
221          ** the list is accounted for below.
222          */
223          DoubleListIterator childRowIter( (DoubleList &) *(this->parent), 1 );
224
225          /*
226          ** We want to move right from the start of the list of
227          ** children to the "current" node PLUS the number of
228          ** nodes we've been asked to move.
229          */
230          parent->childCursor += MoveNodes;
231          tmpChildCursor = parent->childCursor;
232          while ( tmpChildCursor-- > 1 )    {
233              if ( childRowIter == 0 )        {
234                  cout << "moveRight(): at end of list" << endl;
235                  return this;
236              }
237              /*
238                  Move right one child.
```

Nov 12 09:44 1991 tree.cpp Page 8

```
239          */
240          childRowIter++;
241      }
242
243      /*
244      ** Return a pointer to the new "current" node.
245      */
246      return (TREE *) &((Object&)(childRowIter));
247  }
248  return this;
249 }
250
251 TREE *TREE::moveLeft( int MoveNodes )  {
252     int tmpChildCursor;
253
254     if ( this->atRoot() )  {
255         cout << "moveLeft(): at ROOT, cannot move" << endl;
256     }
257     else  {
258         if ( parent->childCursor - MoveNodes < 1 )  {
259             cout << "moveLeft(): cannot move " << MoveNodes <<
260                 " nodes" << endl;
261             return this;
262         }
263
264         DoubleListIterator childRowIter( (DoubleList &) *(this->parent), 1 );
265
266         parent->childCursor -= MoveNodes;
267         tmpChildCursor = parent->childCursor;
268         while ( tmpChildCursor-- > 1 )  {
269             if ( childRowIter == 0 )  {
270                 cout << "moveLeft(): at end of list" << endl;
271                 return this;
272             }
```

Nov 12 09:44 1991 tree.cpp Page 9

```
273             childRowIter++;
274         }
275
276         return (TREE *) &((Object&)(childRowIter));
277     }
278     return this;
279 }
280
281 /*
282 void TREE::forAllChildren() {
283     return level;
284 }
285 */
286
287 int TREE::getLevel() {
288     return level;
289 }
290
291 void TREE::setLevel( int Level ) {
292     level = Level;
293 }
294
295 void TREE::setChildCnt( int ChildCnt ) {
296     childCnt = ChildCnt;
297 }
298
299 void TREE::resetChildCursor() {
300     childCursor = 0;
301 }
302
303 int TREE::getChildCnt() {
304     return childCnt;
305 }
306
```

Nov 12 09:44 1991 tree.cpp Page 10

```
307     int TREE::gotAllChildren()      {
308         return ( childCnt == childCursor );
309     };
310
311     int TREE::getChildCursor()      {
312         return childCursor;
313     };
314
315     int TREE::getParentsChildCursor()      {
316         if ( ! this->atRoot() ) {
317             return parent->childCursor;
318         }
319         else {
320             return 0;
321         }
322     };
323
324     Object *TREE::nodeAddr()      {
325         return node;
326     };
327
328     void TREE::printOn( ostream& outputStream ) const      {
329         int      curIndent, nxtIndent;
330         char    *indent;
331
332         curIndent = (INDENTPERLEVEL * level) + INDENTSTART;
333         nxtIndent = curIndent + INDENTPERLEVEL;
334         indent = (char *) malloc( nxtIndent + 1 );
335         memset( indent, ' ', nxtIndent );
336         indent[nxtIndent] = '\0';
337
338         indent[curIndent] = '\0';
339         cout << indent;
340     }
```

Nov 12 09:44 1991 tree.cpp Page 11

```
341     if ( node == NULL )      {
342         outputStream << "<empty>" << endl;
343     }
344     else    {
345         /*
346          ** Print the "current" node.
347          */
348         node->printOn( outputStream );
349         outputStream << endl;
350
351         indent[curIndent] = ' ';
352
353         /* TREE needs to add a method for node classes to identify
354            themselves as potential PARENT node & a method to check
355            if a given node is one of those:
356            if ( this->parentNode( node->isA() ) )      {
357            */
358            /*
359             ** If there are any children, print them.
360             */
361            DoubleListIterator childRowIter( (DoubleList &) *this, 1 );
362            while ( childRowIter != 0 )      {
363                if ( ((Object&)(childRowIter)).isA() != treeClass ) {
364                    cout << indent;
365                }
366                ((Object&)(childRowIter)).printOn( outputStream );
367                childRowIter++;
368            }
369            /* } */
370
371        }
372        free( indent );
373    };
```

Nov 12 09:31 1991 uintl.h Page 1

```
1  /*
2   **      class: (UINT1 - DECLARATIONS
3   **
4   **          @(#)uintl.h    13.5    11/12/91
5   */
6
7  class UINT1 : public SECSELEMENT
8  {
9  protected:
10     unsigned char *data;
11     unsigned char *ptr;
12 public:
13     UINT1( unsigned long Length, unsigned char *Uint1 );
14     UINT1( unsigned long Items, long DataArray[] );
15     ~UINT1();
16     short bytesPerItem();
17     unsigned char *stringForMessage();
18     int hasLong();
19     long returnLong();
20     virtual void printOn( ostream& ) const;
21 };
22
23
24 }
```

Nov 12 09:44 1991 uintl.cpp Page 1

```
1  /*
2   **      class: (UINT1) - MEMBER FUNCTIONS
3   **
4   **          @(#)uintl.cpp    13.6    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <uintl.h>
14
15 UINT1::UINT1( unsigned long Length, unsigned char *Uint1 ) :
222
16     SECSELEMENT( "UINT1", uint1Class ) {
17     int i;
18
19     length = remainingBytes = Length;
20     data = ptr = new unsigned char[length];
21     itemCode = UINT1_ITEM;
22
23     for ( i = 0; i < length; i++ ) {
24         data[i] = *Uint1++;
25     }
26 }
27
28 UNT1::UINT1( unsigned long Items, long DataArray[] ) :
29     SECSELEMENT( "UINT1", uint1Class ) {
30     int i;
31
32     length = remainingBytes = Items * UNT1_BYTES;
33     data = ptr = new unsigned char[length];
34     itemCode = UNT1_ITEM;
```

Nov 12 09:44 1991 uint1.cpp Page 2

```
35
36     for ( i = 0; i < length; i++ ) {
37         data[i] = (unsigned char ) DataArray[i];
38     }
39 }
40
41 UINT1::~UINT1() {
42     delete data;
43 }
44
45 short UINT1::bytesPerItem() {
46     return UINT1_BYTES;
47 }
48
49 unsigned char *UINT1::stringForMessage() {
50     unsigned char    *ptrOut, *ptrSave;
51     unsigned char    *itemHeader;
52     long      i, itemLength;
53     short     headerLength;
54
55     /*
56     **      Create space to return the "SECS Message ready"
57     **      version of this item.
58     */
59     itemLength = this->lengthInMessage();
60     ptrOut = new unsigned char[itemLength];
61     ptrSave = ptrOut;
62
63     /*
64     **      Get the Item Header and put it in the
65     **      return string.
66     */
67     itemHeader = this->getItemHeader( &headerLength, itemHeader );
68     memcpy( ptrOut, itemHeader, headerLength );
```

223

Nov 12 09:44 1991 uint1.cpp Page 3

```
69         delete( itemHeader );
70
71         /*
72          **      Point to first byte of actual data to be returned,
73          **      then copy the data in.
74         */
75         ptrOut += headerLength;
76         for ( i = 0; i < length; i++ )  {
77             *ptrOut++ = data[i];
78         }
79
80         return ptrSave;
81     };
82
83     int UINT1::hasLong()      {
84         remainingBytes = length;
85         ptr = data;
86         return 1;
87     };
88
89     long UINT1::returnLong()      {
90         if ( remainingBytes <= 0 )      {
91             cerr << "UINT1: trying to get data after end of contents" << endl;
92             exit( 1 );
93         }
94         else      {
95             long      tmp = (long) *ptr;
96             ptr += UINT1_BYTES;
97             remainingBytes -= UINT1_BYTES;
98             return tmp;
99         }
100    };
101
102    void UINT1::printOn( ostream& outputStream ) const      {
```

Nov 12 09:44 1991 uintl.cpp Page 4

```
103     int      i, perLine = 0;
104
105     outputStream << '<' << this->nameOf();
106
107     for ( i = 0; i < length; i++, perLine++ )          {
108         outputStream.width( 4 );
109         outputStream.fill( ' ' );
110         outputStream.setf( ios::right, ios::adjustfield );
111         outputStream << dec << (int) data[i];
112         if ( perLine >= 10 )           {
113             perLine = 0;
114             outputStream << endl;
115         }
116     }
117
118     outputStream << ">" << flush;
119 }
```

Nov 12 09:31 1991 uint2.h Page 1

```
1  /*
2  **      class: (UINT2) -      DECLARATIONS
3  **
4  **          @(#)uint2.h    13.4    11/12/91
5  */
6
7  class UINT2 : public SECSELEMENT {
8  protected:
9      unsigned short *data;
10     unsigned short *ptr;
11 public:
12     UINT2( unsigned long Length, unsigned char *Uint2 );
13     UINT2( unsigned long Items, long DataArray[] );
14     ~UINT2();
15
16     short        bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int          hasLong();
21     long         returnLong();
22
23     virtual void printOn( ostream& ) const;
24 }
```

Nov 12 09:44 1991 uint2.cpp Page 1

227

```
1  /*
2  ..      class: (UINT2  -      MEMBER FUNCTIONS
3  ..
4  ..          @(#)uint2.cpp    13.7    11/12/91
5  */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clatypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <uint2.h>
14
15 UINT2::UINT2( unsigned long Length, unsigned char *Uint2 ) :
16     SECSELEMENT( "UINT2", uint2Class ) {
17     int      i, j, items = Length / UINT2_BYTES;
18
19     length = remainingBytes = Length;
20     data = ptr = new unsigned short[items];
21     itemCode = UINT2_ITEM;
22
23     for ( i = 0; i < items; i++ ) {
24         data[i] = 0;
25         for ( j = 0; j < sizeof(unsigned short); j++, Uint2++ ) {
26             data[i] = (data[i] << 8) + *Uint2;
27         }
28     }
29 };
30
31 UINT2::UINT2( unsigned long Items, long DataArray[] ) :
32     SECSELEMENT( "UINT2", uint2Class ) {
33     int      i;
```

Nov 12 09:44 1991 uint2.cpp Page 2

228

```
35     length = remainingBytes = Items * UINT2_BYTES;
36     data = ptr = new unsigned short[Items];
37     itemCode = UINT2_ITEM;
38
39     for ( i = 0; i < Items; i++ )    {
40         data[i] = (unsigned short) DataArray[i];
41     }
42 }
43
44 UINT2::~UINT2() {
45     delete data;
46 }
47
48 short UINT2::bytesPerItem()      {
49     return UINT2_BYTES;
50 }
51
52 unsigned char *UINT2::stringForMessage()      {
53     unsigned char *ptrOut, *ptrSave;
54     unsigned char *itemHeader;
55     long i, items, itemLength;
56     short headerLength;
57
58     /*
59      ** Create space to return the "SECS Message ready"
60      ** version of this item.
61     */
62     itemLength = this->lengthInMessage();
63     ptrOut = new unsigned char[itemLength];
64     ptrSave = ptrOut;
65
66     /*
67      ** Get the Item Header and put it in the
68      ** return string.
```

Nov 12 09:44 1991 uint2.cpp Page 3

```
69      */
70      itemHeader = this->getItemHeader( &headerLength, itemHeader );
71      memcpy( ptrOut, itemHeader, headerLength );
72      delete( itemHeader );
73
74      /*
75      **      Point to first byte of actual data to be returned,
76      **      then copy the data in.
77      */
78      ptrOut += headerLength;
79      items = length / UINT2_BYTES;
80      for ( i = 0; i < items; i++ )    {
81          *ptrOut++ = data[i] >> 8 & 0xFF;
82          *ptrOut++ = data[i] & 0xFF;
83      }
84
85      return ptrSave;
86  };
87
88  int UINT2::hasLong()      {
89      remainingBytes = length;
90      ptr = data;
91      return 1;
92  };
93
94  long UINT2::returnLong()      {
95      if ( remainingBytes <= 0 )      {
96          cerr << "UINT2: trying to get data after end of contents" << endl;
97          exit( 1 );
98      }
99      else      {
100         long      tmp = (long) *ptr;
101         ptr++;
102         remainingBytes -= UINT2_BYTES;
```

Nov 12 09:44 1991 uint2.cpp Page 4

```
103         return tmp;
104     }
105 }
106
107 void UINT2::printOn( ostream& outputStream ) const {
108     int      i, perLine = 0;
109     int      items = length / UINT2_BYTES;
110
111     outputStream << '<' << this->nameOf();
112
113     for ( i = 0; i < items; i++, perLine++ )
114         outputStream << " " << dec << data[i];
115     if ( perLine >= 10 ) {
116         perLine = 0;
117         outputStream << endl;
118     }
119 }
120
121     outputStream << ">" << flush;
122 }
```

Nov 12 09:31 1991 uint4.h Page 1

```
1  /*
2  **      class:  UINT4    -      DECLARATIONS
3  **
4  **          @(#)uint4.h     13.4    11/12/91
5  */
6
7  class UINT4 : public SECSELEMENT
8 protected:
9     unsigned long   *data;
10    unsigned long   *ptr;
11 public:
12    UINT4( unsigned long Length, unsigned char *Uint4 );
13    UINT4( unsigned long Items, long DataArray[] );
14    ~UINT4();
15
16    short           bytesPerItem();
17
18    unsigned char   *stringForMessage();
19
20    int             hasLong();
21    long            returnLong();
22
23    virtual void   printOn( ostream& ) const;
24 }
```

Nov 12 09:44 1991 uint4.cpp Page 1

```
1  /*
2   *      class:  UINT4    -      MEMBER FUNCTIONS
3   *
4   *          @(#)uint4.cpp    13.7    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <iostream.h>
9 #include <object.h>
10 #include <clstypes.h>
11 #include <custom.h>
12 #include <secselem.h>
13 #include <uint4.h>
14
15 UINT4::UINT4( unsigned long Length, unsigned char *Uint4 ) :
16     SECSELEMENT( "UINT4", uint4Class ) {
17     int      i, j, items = Length / UINT4_BYTES;
18
19     length = remainingBytes = Length;
20     data = ptr = new unsigned long[items];
21     itemCode = UINT4_ITEM;
22
23     for ( i = 0; i < items; i++ ) {
24         data[i] = 0;
25         for ( j = 0; j < sizeof(unsigned long); j++, Uint4++ )
26             data[i] = (data[i] << 8) + *Uint4;
27     }
28 }
29 }
30
31 UINT4::UINT4( unsigned long Items, long DataArray[] ) :
32     SECSELEMENT( "UINT4", uint4Class ) {
33     int      i;
```

Nov 12 09:44 1991 uint4.cpp Page 2

```
35     length = remainingBytes = Items * UINT4_BYTES;
36     data = ptr = new unsigned long[Items];
37     itemCode = UINT4_ITEM;
38
39     for ( i = 0; i < Items; i++ )    {
40         data[i] = (unsigned long) DataArray(i);
41     }
42 };
43
44 UINT4::~UINT4() {
45     delete data;
46 };
47
48 short UINT4::bytesPerItem()      {
49     return UINT4_BYTES;
50 };
51
52 unsigned char *UINT4::stringForMessage()      {
53     unsigned char *ptrOut, *ptrSave;
54     unsigned char *itemHeader;
55     long i, items, itemLength;
56     short headerLength;
57
58     /*
59     **      Create space to return the "SECS Message ready"
60     **      version of this item.
61     */
62     itemLength = this->lengthInMessage();
63     ptrOut = new unsigned char[itemLength];
64     ptrSave = ptrOut;
65
66     /*
67     **      Get the Item Header and put it in the
68     **      return string.
```

Nov 12 09:44 1991 uint4.cpp Page 3

```
69          */
70      itemHeader = this->getItemHeader( &headerLength, itemHeader );
71      memcpy( ptrOut, itemHeader, headerLength );
72      delete( itemHeader );
73
74      /*
75      **      Point to first byte of actual data to be returned,
76      **      then copy the data in.
77      */
78      ptrOut += headerLength;
79      items = length / UINT4_BYTES;
80      for ( i = 0; i < items; i++ ) {
81          *ptrOut++ = data[i] >> 24 & 0xFF;
82          *ptrOut++ = data[i] >> 16 & 0xFF;
83          *ptrOut++ = data[i] >> 8 & 0xFF;
84          *ptrOut++ = data[i] & 0xFF;
85      }
86
87      return ptrSave;
88  };
89
90  int UINT4::hasLong()    {
91      remainingBytes = length;
92      ptr = data;
93      return 1;
94  };
95
96  long UINT4::returnLong() {
97      if ( remainingBytes <= 0 ) {
98          cerr << "UINT4: trying to get data after end of contents" << endl;
99          exit( 1 );
100     }
101     else {
102         long      tmp = (long) *ptr;
```

Nov 12 09:44 1991 uint4.cpp Page 4

```
103     ptr++;
104     remainingBytes -= UINT4_BYTES;
105     return tmp;
106 }
107 };
108
109 void UINT4::printOn( ostream& outputStream ) const {
110     int i, perLine = 0;
111     int items = length / UINT4_BYTES;
112
113     outputStream << '<' << this->nameOf();
114
115     for ( i = 0; i < items; i++, perLine++ )
116         outputStream << " " << dec << data[i];
117     if ( perLine >= 10 )
118         perLine = 0;
119     outputStream << endl;
120
121 }
122
123     outputStream << ">" << flush;
124 }
```

Nov 12 09:31 1991 uint8.h Page 1

```
1  /*
2  **      class:  UINT8   -      DECLARATIONS
3  **
4  **          @(#)uint8.h    13.4    11/12/91
5  */
6
7  class UINT8 : public SECSELEMENT      {
8  protected:
9      double *data;
10     double *ptr;
11 public:
12     UINT8( unsigned long Length, unsigned char *Uint8 );
13     UINT8( unsigned long Items, double DataArray[] );
14     ~UINT8();
15
16     short      bytesPerItem();
17
18     unsigned char *stringForMessage();
19
20     int       hasDouble();
21     double    returnDouble();
22
23     virtual void printOn( ostream& ) const;
24 }
```

236

Nov 12 09:44 1991 uint8.cpp Page 1

237

```
1  /*
2   *      class:  UINT8    -      MEMBER FUNCTIONS
3   *
4   *      E(\#)uint8.cpp  13.7    11/12/91
5   */
6
7 #include <stdlib.h>
8 #include <math.h>
9 #include <stdio.h>
10 #include <iostream.h>
11 #include <object.h>
12 #include <clatypes.h>
13 #include <custom.h>
14 #include <secselem.h>
15 #include <uint8.h>
16
17 UINT8::UINT8( unsigned long Length, unsigned char *Uint8 ) :
18     SECSELEMENT( "UINT8", uint8Class ) {
19     short i, j, items = Length / UINT8_BYTES;
20
21     if ( sizeof(double) != UINT8_BYTES ) {
22         cerr << "UINT8: C++ type does not match length of UINT8" << endl;
23         exit( 1 );
24     }
25
26     length = remainingBytes = Length;
27     data = ptr = new double[items];
28     itemCode = UINT8_ITEM;
29
30     for ( i = 0; i < items; i++ ) {
31         long          mostSignif = 0;
32         unsigned long leastSignif = 0;
33
34         for ( j = 0; j < sizeof(long); j++, Uint8++ ) {
```

Nov 12 09:44 1991 uint8.cpp Page 2

```
35         leastSignif = (leastSignif << 8) + *Uint8;
36         mostSignif = (mostSignif << 8) + *(Uint8 + sizeof(long));
37     }
38
39     data[i] = (mostSignif * 4294967295) + leastSignif;
40 }
41 };
42
43 UINT8::UINT8( unsigned long Items, double DataArray[] ) :
44     SECSELEMENT( "UINT8", uint8Class ) {
45     short i;
46
47     if ( sizeof(double) != UINT8_BYTES ) {
48         cerr << "UINT8: C++ type does not match length of UINT8" << endl;
49         exit( 1 );
50     }
51
52     length = remainingBytes = Items * UINT8_BYTES;
53     data = ptr = new double[Items];
54     itemCode = UINT8_ITEM;
55
56     for ( i = 0; i < Items; i++ ) {
57         data[i] = DataArray[i];
58     }
59
60 };
61
62 UINT8::~UINT8() {
63     delete data;
64 };
65
66 short UINT8::bytesPerItem() {
67     return UINT8_BYTES;
68 };
```

Nov 12 09:44 1991 uint8.cpp Page 3

```
69
70     unsigned char *UINT8::stringForMessage()          {
71         unsigned char   *ptrOut, *ptrSave;
72         unsigned char   *itemHeader, *reverseFloat;
73         long      i, j, items, itemLength;
74         short     headerLength;
75
76         /*
77          ** Create space to return the "SECS Message ready"
78          ** version of this item.
79         */
80         itemLength = this->lengthInMessage();
81         ptrOut = new unsigned char[itemLength];
82         ptrSave = ptrOut;
83
84         /*
85          ** Get the Item Header and put it in the
86          ** return string.
87         */
88         itemHeader = this->getItemHeader( &headerLength, itemHeader );
89         memcpy( ptrOut, itemHeader, headerLength );
90         delete( itemHeader );
91
92         /*
93          ** Point to first byte of actual data to be returned,
94          ** then copy the data in.
95         */
96         ptrOut += headerLength;
97
98         items = length / UINT8_BYTES;
99         for ( i = 0; i < items; i++ ) {
100             long          mostSignif;
101             unsigned long leastSignif;
```

Nov 12 09:44 1991 uint8.cpp Page 4

240

```
103         mostSignif = data[i] / 4294967295;
104         leastSignif = data[i] - (mostSignif * 4294967295);
105
106         *ptrOut++ = leastSignif >> 24 & 0xFF;
107         *ptrOut++ = leastSignif >> 16 & 0xFF;
108         *ptrOut++ = leastSignif >> 8 & 0xFF;
109         *ptrOut++ = leastSignif & 0xFF;
110
111         *ptrOut++ = mostSignif >> 24 & 0xFF;
112         *ptrOut++ = mostSignif >> 16 & 0xFF;
113         *ptrOut++ = mostSignif >> 8 & 0xFF;
114         *ptrOut++ = mostSignif & 0xFF;
115     }
116
117     return ptrSave;
118 }
119
120 int UINT8::hasDouble() {
121     remainingBytes = length;
122     ptr = data;
123     return 1;
124 }
125
126 double UINT8::returnDouble() {
127     if (remainingBytes <= 0) {
128         cerr << "UINT8: trying to get data after end of contents" << endl;
129         exit(1);
130     }
131     else {
132         double tmp = *ptr;
133         ptr += UINT8_BYTES;
134         remainingBytes -= UINT8_BYTES;
135         return tmp;
136     }
}
```

Nov 12 09:44 1991 uint8.cpp Page 5

```
137  };
138
139 void UINT8::printOn( ostream& outputStream ) const {
140     int      i, perLine = 0;
141     int      items = length / UINT8_BYTES;
142
143     outputStream << '<' << this->nameOf();
144
145     for ( i = 0; i < items; i++, perLine++ ) {
146         outputStream << " " << data[i];
147         if ( perLine >= 4 ) {
148             perLine = 0;
149             outputStream << endl;
150         }
151     }
152
153     outputStream << ">" << flush;
154 }
```

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.