Final Report for the Virtual Channel Encryptor
Laboratory Directed Research and Development Project

Dean J. Gibson, Ricardo A. Sarfaty

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia
Corporation, a Lockheed Martin Company, for the United States
Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; distribution is unlimited.
Final Report for the
Virtual Channel Encryptor
Laboratory Directed Research and Development Project

Dean J. Gibson and Ricardo A. Sarfaty
Data Transport and Network Design Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM  87185

Abstract
A workstation with a single physical connection to a data communications network may have a requirement for simultaneous “virtual” communication channels to more than one destination. This report describes the development of techniques based on the Data Encryption Standard (DES) which encrypt these virtual channels to secure the data being transmitted against unauthorized access. A software module has been developed for the UNIX operating system using these techniques for encryption, and some development has also been done on a hardware device to be included between the workstation and network which can also provide these functions. The material presented in this report will be useful to those with a need to protect information in data communications systems from unauthorized access.
10. APPENDICES................................................................................................................... 39
10.1 VCE LDRD PROPOSAL........................................................................................................ 39
10.2 VCE LDRD UPDATE STATUS REPORT........................................................................... 44
10.3 PROCESS FLOWCHARTS..................................................................................................... 47
10.2. LISTING OF CODE FOR VIRTUAL ENCRYPTION PROCESS ........................................ 49
Figures

FIGURE 1: DATA NETWORK USING ENCRYPTION ................................................................. 1
FIGURE 2: SINGLE CHANNEL ENCRYPTION WITH MULTIPLEXER .................................. 2
FIGURE 3: NETWORK USING VIRTUAL CHANNEL ENCRYPTION PROCESSORS .................. 3
FIGURE 4: BLOCK DIAGRAM OF DES ALGORITHM .......................................................... 9
FIGURE 5: CONTENTS OF “ROUND PROCESS” BOX ......................................................... 10
FIGURE 6: INTERMEDIATE KEY GENERATION ............................................................ 12
FIGURE 7: INITIAL PERMUTATION ............................................................................... 13
FIGURE 8: DES INITIAL PERMUTATION .................................................................... 14
FIGURE 9: DES ROUND ............................................................................................... 15
FIGURE 10: DES ROUND FLOWCHART ...................................................................... 16
FIGURE 11: FINAL PERMUTATION ............................................................................... 17
FIGURE 12: ELECTRONIC CODEBOOK MODE (ECB) .................................................. 18
FIGURE 13: CIPHER BLOCK CHAINING MODE (CBC) .............................................. 18
FIGURE 14: CIPHER FEEDBACK MODE (CFB) ............................................................. 19
FIGURE 15: OUTPUT FEEDBACK MODE (OFB) ........................................................... 20
FIGURE 16: KERBEROS FLOW DIAGRAM .................................................................. 27
FIGURE 17: BLOCK DIAGRAM OF FIRST IMPLEMENTATION ................................. 30
FIGURE 18: BLOCK DIAGRAM OF SECOND IMPLEMENTATION .............................. 31
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
1. Introduction

With the increasing use of data communications systems in our society, there has also been a growing need for techniques which can provide protection from unauthorized access for the information being transmitted through these networks. Traditionally, this protection has been provided by link encryption devices located at the end of each communications link interconnecting a pair of nodes in the network. This requires a large number of encryption devices, especially in a large network, increasing the cost of building and operating the network.

This report discusses a different approach to network encryption in which encryption processors are co-located with the workstations and servers connected to the network to provide an end-to-end encryption capability that eliminates the need for link encryption devices. These processors can be provided as either software packages for installation within the workstations and servers, or as hardware devices connected between the workstations, servers, and the network. These processors have the added feature that they can process multiple virtual channels simultaneously so that the workstation connected to the network can communicate with many different servers simultaneously or a server can simultaneously communicate with many different workstations.

1.1. Current Encryption Technology

Most applications of encryption to data communications networks take one of two forms. The first is link encryption. In this case, the entire data packet, including the header, is encrypted and encryption devices are located at each end of a link connecting two nodes (e.g., network routers, network hubs, servers, workstations, etc.) in the network to decrypt the message to allow routing through the network. This means that at every routing point in the network, the message must be decrypted. The second form is end-to-end encryption. In this case, only the data within the data packet, not the header, is encrypted, thus allow the network to route the message to the destination without decrypting the message. In this case, encryption devices are located only at the source and destination of the message. A typical example of the latter, which is the focus of this research, would be a hospital network, in which the workstations can communicate with an e-mail server, a financial records database, and a medical records server. Encryption devices would be placed between the workstations and the network and the servers and the network as illustrated in Fig. 1.

If the workstations run a multitasking operating system, such as UNIX, they can support multiple client programs running simultaneously, each communicating with one or more of the available servers on the network.

![Figure 1: Data Network Using Encryption](image-url)
Since there is typically only one physical connection between each workstation and the network, such as in an Ethernet system, the workstation-network connection has to support multiple virtual circuits simultaneously, with a separate virtual circuit for each connection between a client and the server with which it communicates. This is done by using communications protocols between the workstation and the network that divide the traffic on each of the virtual channels into packets. Each packet carries, among other things, addressing information, so the network can route each packet to its proper destination.

Most encryption equipment currently available cannot manage multiple virtual channels on the same physical link between the user's workstation and the network, so, to encrypt these channels, it is necessary to demultiplex (demux) the virtual circuits from the workstation into corresponding physical channels, encrypt these separately, and then remultiplex (mux) them for transmission over the network, as illustrated in Fig 2.

In many cases, the amount of multiplexing and demultiplexing, illustrated here for a user's workstation, may be much greater at the server, since a server typically has to handle many more simultaneous virtual channels then the user's workstation.

![Figure 2: Single Channel Encryption with Multiplexers.](image)

### 1.2. Virtual Channel Encryption

This report covers the development of a virtual channel encryption processor for workstations and/or servers. The processor will handle multiple virtual channels without the separate multiplexers, demultiplexers, and single channel encryption devices shown in Fig. 2. It is implemented as either a software module (within the server or workstation) or an hardware processor (external to the server or workstation) that is inserted in the communication link between a workstation, or server attached to a network, and the network itself, as illustrated in Fig. 3.

The encryption algorithm in the virtual channel encryption processor is based on the Data Encryption Standard (DES). The Outerbridge implementation of the DES algorithm was modified to remove the state variables from the code, placing the state variables in an array that contains a separate set of state variables for each channel supported by the virtual channel encryption processor. With this modification, the virtual channel encryption code can support multiple "threads" of execution, one for each virtual channel.
The basic virtual channel encryption algorithm is platform independent, but the interfaces between it and the operating system in the computer on which it runs is platform dependent. An example of this is SOLARIS’s use of the STREAMS protocol as a communication process for passing information to the network, where SUNOS uses socket calls. A software module has been developed for installation on a SUN SPARC workstation running the SOLARIS (UNIX based) Operating System (OS). Some design work has also been done on a software module that runs on a VME-based processor running the proprietary VxWorks operating system supplied by Windriver Systems. The VME processor can be installed in the link between a workstation or a server and the network (preferably close to the workstation or server) to provide the virtual encryption function when it is not desirable to install it directly in the workstation or server.

1.3. Supporting Activities

Key distribution is a problem in any application using data encryption. In our case, key distribution is implemented using concepts incorporated from the Kerberos network security architecture developed at the Massachusetts Institute of Technology (MIT). Keys are generated in a central “key distribution center” and distributed to each user as required when that user applies to the central authority (i.e., key distribution center) key distribution center for access to a particular resource on the network. Kerberos not only authenticates users needing access to servers through a password scheme, but also provides key distribution protocols that encrypt keys being transmitted across the network, so that they cannot be intercepted or used by unauthorized individuals.

Some conceptual work has also been done to implement the encryption algorithm in a separate VME-based processor running the VxWorks OS so as to remove the burden of OS system, and many of the details of the interface between the VME-based processor and the host workstation have been completed.

Future work to be performed includes testing of the VME-based encryption algorithm, extension to other applications on the SUN workstation, extension to encryption algorithms other than DES, applications in networks using personal computers, and use in networks based on transmission media other than Ethernet (e.g., FDDI, ATM, etc.). Also, the user interface for the virtual encryption algorithm should be developed to the point where applications such as telnet and FTP can be run, from the user’s point of view, in the same manner as when virtual channel encryption is not involved.
2. **Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode — a new digital communications technology based on cell switching</td>
</tr>
<tr>
<td>demux</td>
<td>Demultiplexer or Demultiplex</td>
</tr>
<tr>
<td>DES</td>
<td>Digital Encryption Standard — a standard endorsed by NIST.</td>
</tr>
<tr>
<td>FDDI</td>
<td>Fiber Distributed Data Interface</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Internetwork Protocol — a protocol that specifies peer-to-peer communications across logical subnetworks</td>
</tr>
<tr>
<td>LDRD</td>
<td>Laboratory Directed Research and Development — a Department of Energy program that allows its national laboratories to invest funds into new, “leading edge” research programs</td>
</tr>
<tr>
<td>mux</td>
<td>Multiplexer or Multiplex</td>
</tr>
<tr>
<td>NBS</td>
<td>National Bureau of Standards</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute for Standards and Technology</td>
</tr>
<tr>
<td>NSA</td>
<td>National Security Agency — the developers of the Millbury prototype ATM encryption hardware</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>SPARC</td>
<td>Scalable Processor ARCitecture — the processor/hardware architecture used by the latest Sun Microsystems workstations</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol — a protocol that provides assured, in-order delivery of PDUs over IP.</td>
</tr>
<tr>
<td>VME</td>
<td>versa-module, Europe</td>
</tr>
</tbody>
</table>
3. The Encryption Algorithm

To develop the virtual channel encryption algorithm, a version of DES was chosen in which the state variables could be readily separated from the code and a state variable array was set up to store the state variables separately for each virtual encryption channel. Once this is done, each virtual encryption channel can be implemented by a separate thread of execution, so that each of the virtual encryption channels operates independently.

3.1. Criteria for the Algorithm

In general, an encryption algorithm is a set of rules, which, when combined with a unique key, scrambles a length of "cleartext" to produce a length of "cryptotext" in such a manner that it is difficult to recover the cleartext without knowledge of the key, even if the algorithm is known. In our case, we would like to choose an algorithm that can also be readily modified to be suitable for the virtual encryption processor. This requires that the state variables be extracted from the algorithm, leaving pure code, and that the state variables can then be assembled into arrays that will preserve the state variable information independently for each of the virtual encryption channels.

Other criteria that the encryption algorithm should meet is that it be well-known and accepted in the user community, so that a virtual channel encryption device based on the algorithm will also be readily accepted. There should be no known methods to easily "break" the algorithm; that is, given the cleartext and corresponding cryptotext, there are no known methods to easily determine the encryption key. This will provide the virtual channel encryption device the level of trust it needs to succeed in the user community. The algorithm should also be non-proprietary, so that it can be adapted for the virtual encryption system without concerns over copyright or patent issues. This will allow the virtual encryption algorithm free and unimpeded distribution throughout the user community. Finally, the encryption algorithm should have many different hardware and software implementations available in the literature, to maximize the number of starting points for modification for virtual encryption.

3.2. Choice of the Algorithm

Of the many different encryption algorithms available, such as those described in Schneier's text [13], the Data Encryption Standard (DES) algorithm seems to meet these criteria best. There are different versions of the algorithm available in both hardware and software, and many of these are of a form in which the state variables can be easily extracted, leaving pure code which is the primary criterion for use in the virtual encryption processor. The algorithm is very well known, and is used in many different applications, such as in the banking industry, so it is trusted by those with large sums of money at risk. While the algorithm was developed by International Business Machines (IBM) and others, it is now non-proprietary, and it appears in hardware and software available from different sources.

3.3. History of the Data Encryption Standard

The history of the Data Encryption Standard dates from 1972, when the National Bureau of Standards (NBS), initiated a program to protect both computer and communications data, ideally with a single algorithm that could perform both functions. After requesting contributions from private industry and other sources, NBS received a proposal from IBM based on LUCIFER, an internal IBM development that could be implemented easily in both hardware and software. NBS evaluated this algorithm with the assistance of the National Security Administration (NSA), concluded that it was suitable, and published it as a federal standard in 1976.

There have been some concerns raised by the involvement of NSA in the development of DES, specifically in the length of the key used and in the design of the "S-boxes" that provide most of the security in the algorithm, which is dependent on the length of the key (the longer the key, the more secure the algorithm). The concern expressed was that if the key was short enough (e.g., 56 bits), an organization with extensive resources (such as NSA) could eventually break the algorithm and recover the key. There were also reservations expressed about the "S-boxes", where NSA had considerable design input. It was known that there were certain classified considerations involved in the design of the S-boxes, and this generated the fear that "trapdoors" had been built in these S-boxes which an organization (such as NSA) could easily exploit to break the algorithm. At this time, it appears that the concerns regarding the key length may be well-founded, so that the DES may eventually have to be replaced when computing power has advanced to where DES can be broken by brute force (i.e., try all the possible key values). No weaknesses have been found in the S-boxes, thus there does not appear to be any methods by which the DES algorithm can be broken other than brute force or poor implementation (e.g., provide variables to the user by storing in user accessible memory).
3.4. Principles of Operation

The Data Encryption Standard is a block cipher - it encrypts data in 64-bit blocks. It consists of three parts: the initial permutation, 16 identical "rounds", each using a combination of permutation and substitution operations, followed by a final permutation to finish the encryption. The initial permutation shuffles the 64 incoming bits following a table in the DES definition to produce another 64-bit word with the bits in a different order. This 64-bit word is then split into two 32-bit words in preparation for the first round. The first step of the round is to expand the 32 bits in the left word to a 48-bit word following a permutation table that duplicates some of the bits. This 48-bit word is then combined with a 48-bit intermediate key, formed for this round from the 64-bit master key by a key generation process. The 48-bit result of this operation is then applied to a set of eight "S-boxes" which splits this 48-bit word into eight 6-bit words, then performs a substitution operation to generate a set of eight 4-bit words which are combined to form a 32-bit word. The 32 bits from the S-boxes are then passed through a "P-box" permutation, then combined with the right 32-bit incoming word for this round to form the 32-bit right-side output for this round (note that the left and right words are switched at the output of each round). The left 32-bit output word for this round is simply the 32-bit right input word for this round, taken without any modification. The operations that make up this round are repeated 15 more times to complete the 16 rounds required by the DES algorithm, with each round using a different key generated from the master 64-bit key. After the 16th round, the two 32-bit words are recombined into a 64-bit word for output through a final permutation function, which is the exact inverse of the input algorithm at the beginning of the encryption operation. This process is illustrated in Figs. 4 and 5.
clear text 64 bits

initial permutation

Round 1 Input
Left 32 bits
Round 1 Input
Right 32 bits

Intermediate
Key 1

Round Process

Round 2 Input
Left 32 bits
Round 2 Input
Right 32 bits

Intermediate
Key 2

Round Process

Round 3 Input
Left 32 bits
Round 3 Input
Right 32 bits

Intermediate
Key 16

Round Process

Round 16 Input
Left 32 bits
Round 16 Input
Right 32 bits

Round 16 output
Left 32 bits
Round 16 output
Right 32 bits

final permutation

ciphertext 64 bits
3.5. Comparison of DES Algorithms

Several different software implementations of the DES algorithm were considered for this project, including those developed by Outerbridge, Tanenbaum, and Young. The primary criterion for choice between these algorithms was a tradeoff between memory storage space requirements and speed. Those with larger memory requirements tended to be the fastest, because most of the processing was done in lookup tables as opposed to the manipulation of Boolean equations. The tables, however, tended to take up more memory. The Outerbridge DES algorithm provided a balance between memory requirements and speed, so it was selected for this project.
3.6. Generation of the Intermediate Keys

Generation of the intermediate keys for each round can be done separately from the main encryption algorithm, and has to be done only when a new master key is supplied to the DES algorithm. The master key is usually given as 64 bits, although only 56 of these are actually used. Generation of the intermediate keys then consists of three parts: an initial key permutation that reduces the length of the key from 64 bits to 56 bits while shuffling the bits, a set of 16 shifting operations that generate individual 56 bit keys for each round of the DES algorithm, and a final permutation that reduces each intermediate key from 56 to 48 bits, in a form suitable for the DES algorithm. This process is illustrated in Fig. 6.

In the Outerbridge DES implementation, the initial key permutation takes the 64 bit master key, stored as an array of eight 8-bit bytes, and converts it to an array of 56 entries, each containing one bit. In the process, the 8 bits not used in the master key are eliminated. These 56 bits are divided into two 28-bit halves, each of which is shifted one or two bits to the left for each succeeding round of the encryption operation. The two 28-bit halves obtained for each round are then recombed to form a 56 bit word which is processed by another permutation operation to yield the 48-bit intermediate key for each round.

Decryption is done by exactly the same algorithm that performed the encryption, except that the 16 intermediate keys are supplied to the 16 rounds in reverse order.

3.7. Initial Permutation

As mentioned above, the DES encryption algorithm is implemented in three steps: an initial permutation, the 16 rounds of the encryption algorithm itself, and a final permutation. The initial permutation is not implemented using a look-up table as implied in the DES standard. Instead, the 64 bits to be encrypted are divided into a high-order and a low-order word, each containing 32 bits, which are referred to in the following discussion as “left” and “right”. A third variable “work” is also introduced for use as a temporary storage area. The initial permutation is then implemented by a series of shifting and masking operations as defined by the section of code in Fig. 7.
Figure 6: Intermediate Key Generation
work = ((left >> 4) ^ right) & 0x0f0f0f0fL;
right = right ^ work;
left = left ^ (work << 4);
work = ((left >> 16) ^ right) & 0x0000ffffL;
right = right ^ work;
left = left ^ (work << 16);
work = ((right >> 2) ^ left) & 0x33333333L;
left = left ^ work;
right = right ^ (work << 2);
work = ((right >> 8) ^ left) & 0x0f0f0f0fL;
left = left ^ work;
right = right ^ (work << 8);
right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
work = (left ^ right) & 0xaaa aaa aLL;
left = left ^ work;
right = right ^ work;
left = ((left << 1) | ((left >> 31) & 1L)) & 0xffffffffL;

Figure 7: Initial Permutation

For those who are not familiar with the operators used above, they are the standard "C" language bitwise operators defined as follows:

&       bitwise AND
|       bitwise inclusive OR
^       bitwise exclusive OR
<<      left shift (shift the value on the left of the operator by the number of bits given by the value on the right)
>>      right shift

A block diagram of the initial permutation is given in Fig. 8.
Figure 8: DES Initial Permutation
3.8. Round Implementation

The 16 rounds of the DES algorithm are combined into 8 steps, with each step performing two successive DES rounds. Each step consists of a combination of shifting and table-lookup operations, which also make use of the intermediate keys produced by the key generation function. The section of code in Fig. 9 defines the process used in each of the 8 steps:

```
work = ( right << 28 ) | ( right >> 4 );
work = work ^ keys[0];
fval = SP7[ work & 0x3fL ];
fval = fval | SP5[ ( work >> 8 ) & 0x3fL ];
fval = fval | SP3[ ( work >> 16 ) & 0x3fL ];
fval = fval | SP1[ ( work >> 24 ) & 0x3fL ];
work = right ^ keys[1];
fval = fval | SP8[ work & 0x3fL ];
fval = fval | SP6[ ( work >> 8 ) & 0x3fL ];
fval = fval | SP4[ ( work >> 16 ) & 0x3fL ];
fval = fval | SP2[ ( work >> 24 ) & 0x3fL ];
left = left ^ fval;
work = ( left << 28 ) | ( left >> 4 );
work = work ^ keys[2];
fval = SP7[ work & 0x3fL ];
fval = fval | SP5[ ( work >> 8 ) & 0x3fL ];
fval = fval | SP3[ ( work >> 16 ) & 0x3fL ];
fval = fval | SP1[ ( work >> 24 ) & 0x3fL ];
work = left ^ keys[3];
fval = fval | SP8[ work & 0x3fL ];
fval = fval | SP6[ ( work >> 8 ) & 0x3fL ];
fval = fval | SP4[ ( work >> 16 ) & 0x3fL ];
fval = fval | SP2[ ( work >> 24 ) & 0x3fL ];
right = right ^ fval;
```

Figure 9: DES Round

The variable “fval” is a fourth temporary variable used to store intermediate results, and the arrays SP1 through SP8 are used to implement the “S-boxes” of the DES algorithm. Once these eight steps are completed, the two 32-bit words “left” and “right” contain the encrypted information which must still be processed by the final permutation to produce the final encrypted result. The flowchart in Fig. 10 illustrates this process.
Figure 10: DES Round Flowchart
3.9. Final Permutation

As mentioned before, the final permutation is the exact inverse of the initial permutation. It is implemented by the code in Fig. 11:

```
right  =  (right << 31)!(right >> 1);
work   =  (left ^ right) & 0xffffffffL;
left   =  left ^ work;
right  =  right ^ work;
left   =  (left << 31)!(left >> 1);
work   =  ((left >> 8) ^ right) & 0x00ff00ffL;
right  =  right ^ work;
left   =  left ^ (work << 8);
work   =  ((left >> 2) ^ right) & 0x33333333L;
right  =  right ^ work;
left   =  left ^ (work << 2);
work   =  ((right >> 16) ^ left) & 0x0000ffffL;
left   =  left ^ work;
right  =  right ^ (work << 16);
work   =  ((right >> 4) ^ left) & 0xff00ff00L;
left   =  left ^ work;
right  =  right ^ (work << 4);
```

Figure 11: Final Permutation.

At the completion of the final permutation, “left” and “right” contain the left 32 bits and the right 32 bits, respectively, of the encrypted input. These may be combined into a single 64 bit work for further processing.

3.10. Modes of DES

There are four modes in which DES can be operated. These are known as the electronic codebook mode, the cipher block chaining mode, the cipher feedback mode, and the output feedback mode. These are illustrated in Figs. 12 through 15.

Any of these four modes could be used in the VCE encryption processor; however, the only one used at present is the electronic code book mode. The other three modes would require an additional state variable to handle the feedback. An initial value for this feedback variable has to be supplied as well as the key.
DES Encryption

- cleartext block
  - DES encrypt
  - ciphertext block

DES Decryption

- ciphertext block
  - DES decrypt
  - cleartext block

Figure 12: Electronic Codebook Mode (ECB)

Time = 1
- Initialization Vector
  - clear-text block
  - XOR
  - DES encrypt
  - clear-text block

Time = 2
- Initialization Vector
  - clear-text block
  - XOR
  - DES encrypt
  - clear-text block

Time
- Initialization Vector
  - clear-text block
  - XOR
  - DES encrypt
  - clear-text block

Figure 13: Cipher Block Chaining Mode (CBC)
3.11. Modification of the DES algorithm for the VCE project

To modify the DES algorithm for the virtual channel encryption project, one must modify the code so that it can support multiple encryption processes simultaneously without conflicts between the state variables for each encryption channel. There are two ways that this can be done. The first approach is to duplicate the entire algorithm, both code and state variables, so that each encryption channel has its own code and state variables. The second approach is to extract the state variables from the code, leaving only pure code to be shared by all encryption channels, then setting up a separate structure for only the state variables for each encryption channel.

![Diagram of Cipher Feedback Mode (CFB)](image)

Figure 14: Cipher Feedback Mode (CFB)
3.12. Threads and Processes

There are two methods used in UNIX to support the simultaneous execution of multiple programs. The first method, the process, assigns each executing program its own stack to store processor variables, its own piece of virtual memory, its own file descriptors, and its own code for execution, which is not shared with any other process. Different processes running on the same computer have little in common. Switching between processes must be done in the kernel (the primary element of the UNIX OS), since a process switch involves a change between virtual memory spaces.

In a multi-threaded environment, each program can be assigned its own thread for execution, but different threads share more in common when compared with different processes. While each thread has its own processor stack, it shares its virtual memory space, file structures, and code with the other threads running for the same user on the same machine. The main advantage of threads over processes is that it is much easier, and takes less time, for the processor to switch between threads than between processes. The switch can be done in user space because much less information is involved in the context switch. On the other hand, there is not as much isolation between threads as there is between processes, and the threads can interfere with each other’s variables if care is not used, as they share the same virtual memory space. However, interference between variables in different processes cannot occur, because processes do not share virtual memory space. A computer using an operating system that supports threads still makes use of the process concept. Threads are contained within processes, and one process can contain many threads. When a switch is made between threads running under different processes, such as when users are changed, it is necessary to make a full process context change. In the interest of speed, the threads concept is used to separate the encryption channels in the virtual encryption processor, but care must be taken to separate the state variables for each encryption channel.
3.13. Separation of State Variables

Since we are using the threads concept to structure the virtual channel encryption software, we need only one block of code which can be shared by all the threads. A separate set of state variables must be established for each channel. This is done by putting them into a two-dimensional array, with one index being the channel number and the other, the state variable within that channel. In the current implementation of the virtual channel encryption software, there are eight virtual channels supported, and each virtual channel has the following state variables: "left", "right", "work", and "fval".

3.14. Programming with Threads

A thread of execution is created using the call

```c
int thr_create (void *stack_base, size_t stacksize, void *(*startroutine) (void *), void *arg, long flags,
    thread_t *new_thread)
```

where

"stack_base" contains the address for the stack used by the new thread. This is usually NULL, which causes the function to create a stack of at least "stacksize" bytes.

"stack_size" is the size in bytes of the stack for the new thread. In most cases, this is set to zero, which causes the function to use a default.

"start_routine" contains the function with which the new thread begins execution.

"flags" specifies the attributes for the created thread. In most cases, a default value of zero can be used. Otherwise, this variable is constructed from the bitwise inclusive OR of the following:

"THR-DETACHED" This detaches the new thread so that its resources can be used as soon as the thread terminates. This is set when one does not want to stop execution for the thread to terminate.

"THR_SUSPENDED" This suspends the new thread and does not execute the "start_routine" until requested.

"THR_BOUND" This causes the new thread to be bound to a processor.

"THR_NEW_LWP" Increases the concurrency level for unbound threads by one.

"THR_DAEMON" This marks the new thread as a daemon.

"new_thread" points to a location where the ID of the new thread is stored. Normally set to zero and not used.

Return values. "Thr_create" returns a value of zero when it completes successfully. Another value indicates an error. Possible values are as follows:

"EAGAIN" A system limit has been exceeded.

"ENOMEM" Not enough memory available

"EINVAL" "Stack_base" is not zero, and "stacksize" is less than the value returned by "Thr_minstack".
3.15. Termination of a Thread

The call "void thr_exit (void *status)" is used to terminate a thread; "status" is generally ignored unless this thread is the last non-daemon thread, in which case it is set to zero.

The calls to create and terminate threads do not appear in the virtual encryption code itself. They appear in the code that calls the virtual encryption process, and set up one or more threads to enter the virtual encryption code, with a maximum of eight threads.
4. Application to the SUN Workstation Environment

4.1. Overview of Sockets

The interface between the VCE encryption software and user applications running on a SUN workstation using the Solaris UNIX-based operating system is through the "sockets" concept supported by UNIX.

Sockets are a basic component of interprocess communication. Users who have a need to communicate between different programs on the same host or different programs on different computers can use the "socket" concept to transfer data between the programs in a manner very similar to writing to and from files on a hard disk. To use the socket concept for communications, the user must establish a socket in each program requiring communication. Once this is done, a link must be set up between the two sockets over the network to provide communications. The TCP/IP internet protocols are used to support socket communications over a communications link.

4.2. Creation of Sockets

The user creates a socket using the "socket" function, which has the format

\[ s = \text{socket (domain, type, protocol)} \]

In our applications, the domain is usually set to "AF_INET" to indicate an internet application. This can also be set to "AF_UNIX" to indicate a UNIX application, which is used only between different programs running on the same host. The second entry type refers to the type of socket being created. If type is set to "SOCK_STREAM", the socket will provide a bi-directional, reliable, sequenced, and unduplicated flow of data with no record boundaries. If type is set to "SOCK_DGRAM", a datagram socket is created, which also supports the bi-directional flow of messages. Unlike the stream socket, no guarantee is given that the messages will be received in the same order as they are sent, and messages can be duplicated or even lost due to network problems. It is the user's responsibility to ensure that all messages sent are received in the proper order and that none are lost. For this reason, most applications use the "stream" socket. The third argument protocol is used to specify the communications protocol to be used between a socket pair. This is generally set to 0 to give the default protocol for the type of socket in use, which, for the stream socket operating over almost any kind of network, will be TCP/IP.

4.3. Binding of Sockets

Once the socket has been created, a name must be bound to it so that remote programs can identify and communicate with the socket. In this case, a communications channel is defined by five pieces of information: the protocol, the local address, the local port, the remote address, and the remote port. The protocol argument is the protocol to be used by the communications link, and is defined by the protocol requested when the sockets to be used at each end of the link are established. Needless to say, the same protocol must be used at each end. The "local address" is the host name or IP address of the host, and the "port" is the TCP port number used to refer to the process in the local host that contains this socket. On the other end of the communications link, the "remote address" is the host name or IP address of the remote host, and the port number is the TCP port number used by the program in the remote host using this socket. There can be no more than one communications link in a network described by the same set of five unique identifiers.

The binding operation is performed by the "bind" function, which has the form

\[ \text{bind (s, (struct sockaddr *) &sin, sizeof sin)} \]

where "s" is the socket handle that was generated when the "socket" function created the socket, and "sin" is a structure containing the elements "sin_addr.sin_addr" and "sin_port". The first entry in the structure must contain the IP address of the host supporting this socket, and the second entry is the port number of the program containing this socket. If the IP address of the host is not immediately known, there are functions supplied by UNIX that can provide the IP address, given the name of the host.
4.4. Accepting Connections

Once the sockets have been identified through the binding process, it is possible to set up a communications channel between them. This is usually done in a client-server environment, with the client initiating the connection to the server. Before this can be done, the server must perform two additional steps that allow it to receive connections from clients. The first is to indicate the number of connections that can be queued at the server socket. This is done by executing the “listen” function, which takes the form

\[ \text{listen} \left( s, n \right) \]

The integer “s” is the socket handle for this socket which was returned when the socket was created by the “socket” function, and “n” is the number of connection requests that can be queued up for connection to the socket before additional requests are refused. The second step that has to be executed to set up the server socket is to execute the “accept” function. This function takes the form

\[ \text{accept} \left( s, (\text{struct sockaddr}) \& \text{from}, \&\text{from \ len} \right) \]

where “s” is the socket handle for the program containing the socket in the server. Once this function is called, it will block any further execution of the server program, until a client socket makes a connection to it. Once this happens, the “from” structure will contain the IP address and TCP port number of the client, and the function will return a new socket handle which must be used for any further writing or reading from this communications channel. This feature allows the server program to return to the “accept” function to process further connections to this socket, while it is simultaneously handling traffic through the socket which has just had a connection made to it by a client.

When a client wants to connect its socket to a server, it executes the “connect” function, which has the form

\[ \text{connect} \left( s, \text{struct sockaddr \ *} \&\text{server}, \&\text{sizeof server} \right) \]

where “s” is the socket handle of the socket in the client program that wants to make the connection, and “server” is a structure containing the IP address and TCP port number of the server program in the server host. This function will block until the connection is made, and ready for use.

4.5. Communication through the Sockets

Data is actually transferred over a connection between two sockets using the normal “read” and “write” system calls, which are of the form

\[ \text{read} \left( s, \text{buf, sizeof buf} \right) \]

and

\[ \text{write} \left( s, \text{buf, sizeof buf} \right) \]

The integer “s” is the socket handle for the socket, and “buf” is a buffer (usually an array) containing the information being transferred.

4.6. Closing a Socket Connection

Once communication is completed, the connection between the sockets should be closed by the “shutdown” function, which takes the form

\[ \text{shutdown} \left( s, \text{how} \right) \]

The integer “s” is the socket handle, and “how” is an integer where 0 disallows further receives, 1 disallows further transmits, and 2 disallows both.
5. Key Handling

5.1. Approaches to Key Handling

Traditionally, key handling in a client/server network has been handled through direct negotiation between the servers and clients requesting access to them. Each time a client requests connection to a server, it has to provide a username/password combination to the server to identify and authenticate itself. This has two major disadvantages. First, the username/password combination has to be transmitted over the network every time access to a server is required, which poses potential security problems. These can be partially resolved by encrypting the passwords, but this raises a key distribution problem. The second problem is that each server has to maintain its own username/password table which will increase the work required to administer the network.

Kerberos is an application developed by Project Athena at the MIT to eliminate some of these problems. All password information is stored in a central location known as a Key Distribution Center (KDC), and passwords are never transmitted over the network, even in encrypted form. It also provides a method for key distribution so that each session between a client and a server can be encrypted, hence a session can not be monitored even by someone who has access to the network communication media.

5.2. Overview of Kerberos

As mentioned above, Kerberos is basically a method to control access to servers from clients in a communications network. To get access to a Kerberos network, a user must first log on to a terminal or workstation associated with the network. While Kerberos can control access to any terminal or workstation connected to the network, the usual procedure is to use the local operating system in the terminal or workstation to control access to itself, typically by requesting a password from each user when they log onto the workstation.

After a customer has logged onto his workstation or terminal through his local operating system, they are ready to log on to the Kerberos network. They do this by issuing a "kinit" command, in which they identify themselves in the command line, and provide a network password when prompted. The user's workstation makes contact with the key distribution center, and identifies the user in its request for network access. The key distribution center returns an initial "ticket" back to the customer, which contains a session key encrypted using the user's network password. The workstation uses the password supplied by the user to decrypt the ticket, which contains, among other things, a session key to be used for any further communications with the key distribution center. If the user cannot supply the correct password, the session key is destroyed, and the user cannot make any further attempts to access any resource on the Kerberos network. After successfully receiving their initial session key from the key distribution center, the user can use this key to encrypt further communications with the key distribution center to request access to the servers on the network.

When users requests a remote service, they do so by sending request to the key distribution center for access to the server. If the user has authorized access to that particular server, they will be supplied with a session key. A copy of the session key is also supplied to the server requested by the user, so that an encrypted session can be set up between the user's workstation and the server.

5.3. Application of Kerberos to this Project

A simplified version of Kerberos is used to provide key generation and key distribution. Keys are generated in a key distribution center for use by the virtual channel encryption processors distributed throughout the network. Once generated, the keys are distributed to the encryption processors by a simplified version of the key distribution protocols used by Kerberos.

When an encrypted channel is to be set up through the virtual channel encryption processor, a request is made to the key distribution center for the DES keys necessary for the session. Upon verification that the user is authorized access to the server, copies of the session key are sent to both the user's workstation and the server. Once the key is received at both ends, the key is loaded into the key matrix in the virtual encryption process at both ends, and communication can then begin over the new link. When there is no longer a need for communication between the user's workstation and the server, the communication link is dismantled and the keys erased.
5.4. Key Generation

Key generation in the key distribution center is done by applying the DES algorithm repeatably to a “seed” supplied to the key distribution center, when it is established and started. This method generates unique 64-bit keys for the DES algorithm for each session running through the virtual channel encryption processor.

5.5. Key Distribution

The user and the server are provided with fixed key distribution keys which are used only for the purposes of key distribution. The key distribution key for the user is known only by the user and the key distribution center. The key distribution key for the server is known only by the server and the key distribution center. The user and server do not know each other’s key distribution keys. Only the key distribution center knows all of the keys, and for that reason, they must be kept secure. The session key for the user is encrypted at the key distribution center by the user’s key distribution key, and the user decrypts the session key using the same key distribution key. The session key for the server is encrypted at the key distribution center using the server’s key distribution key, and is decrypted at the server by the server’s key distribution key. Once the keys are decrypted, they are loaded into the virtual encryption algorithm, and the link between the user and server is ready for secure operation.

The encrypted keys are distributed through network sockets separate from those used by the virtual channel encryptor to transmit encrypted data from one host to another. The key communication socket links are kept permanently in existence, whereas those used by the virtual channel encryptor for encrypted communications are taken down when the data link is no longer needed. This frees VCE channels for other use.

Key generation and distribution are illustrated in Fig. 16.
user requests secure connection to server

message requesting keys is sent to the Key Distribution Center

Key Distribution Center verifies that request is permitted

Key Distribution Center generates session key

- session key encrypted by user's key distribution key
- session key encrypted by server's key distribution key

encrypted key sent to user's workstation

encrypted key sent to server

- session key decrypted by user's workstation
- session key decrypted by server

session key loaded into virtual channel encryption process in user's workstation

session key loaded into virtual channel encryption process in server.

session established between user's workstation and server

Figure 16: Kerberos flow diagram
6. Extension to an External VME-based Processor

6.1. Justification

The primary justification for moving the encryption process out of the workstation to a separate VME-based chassis is to remove the encryption processing load from the workstation processor. The VME-based encryption process will use a Motorola MC68030 microprocessor to run the encryption algorithm which is virtually identical to that used in the workstation. In general, an external encryption processor will be more useful at a server than at a user's workstation due to the greater number of channels handled by a server.

6.2. Architecture

The VME-based encryption processor will be implemented on a single board VME based processor using the Motorola MC68030 microprocessor. The board also contains RAM for variable storage, on-board PROM for program storage and an Ethernet interface for communications between the VME board, the host workstation, and the rest of the network.

The VME-based encryption processor will be first implemented on a board with a single Ethernet interface. This interface will handle the traffic between the encryption processor and the user, as well as between the encryption processor and the network, which will simplify debugging. This is an obvious security problem, since both clear text and encrypted text appear on the network. Therefore, once the encryption processor with a single interface operates properly, a second Ethernet interface can be installed on the board to separate the clear and encrypted text. The clear text then passes over only a very small intra-network consisting of the encryption processor and its associated workstation, and no clear text is transmitted over the local or wide area network.

The VME-based encryption processor will be running the encryption process under the VxWorks OS. When the code for the virtual channel encryptor is working properly, it can be loaded into the read-only memory (ROM) on the processor board, eliminating any need for a disk drive, which reduces costs. VxWorks offers a number of utilities which simplify the programming required to implement the VME-based encryptor, with the most important being those which support the “socket” concept. The VxWorks sockets are compatible with the Solaris sockets used in the Sun workstation, so the socket concept can be used to transfer data between the VME encryption processor and its associated workstation, and to maintain connections over the network to a distant workstation.

6.3. Implementation

The VME-based encryption processor can be implemented in two stages. In the first, the encrypted data is sent back to the workstation for routing to a distant workstation over the network. In the second, the VME-based encryption processor takes care of the routing as well, and maintains the encrypted connections to the distant workstation by itself. These two implementations are shown in Figures 17 and 18.

In the first implementation, 16 socket pairs are maintained between the VME-based encryption processor and the associated workstation to handle the 8 virtual channels that can be processed by the encryption processor. Each full-duplex link will then have two socket pairs, one for transmission of cleartext data from the user’s application to the encryption processor. A second socket pair will be used to transmit the encrypted data back to the host workstation. (Each socket pair is a full duplex link that can carry data in both directions simultaneously.) The workstation will then route the encrypted data to another socket for transmission to the distant workstation. The sixteen socket pairs that terminate in the VME encryption processor will be permanent. They will be set up when the encryption processor is started, and will run continuously afterwards.

Once the first implementation is working properly, it can be converted to the second phase, where the VME encryption processor makes connections directly to the remote workstation. There will be eight sockets set up to handle the encrypted data, but, initially, they are not connected to any destination. Once users on the workstation associated with the encryption processor take control, they can direct the VME encryption processor to make the actual connection to the remote workstation. This requires a separate control channel between the workstation and the VME processor to direct the sockets to make the appropriate connections. This option reduces the load on the workstation processor, since it no longer has to handle encrypted data.
Figure 17: Block Diagram of First Implementation
Figure 18: Block Diagram of Second Implementation.
7. Areas for Future Work

There are a number of areas in which additional work can be done to make the virtual channel encryption concept more useful. Some of them are listed below.

7.1. Hardware Implementation

Work needs to be completed on the VME implementation of the virtual channel encryption algorithm. The virtual channel encryption code must be interfaced with the VxWorks operating system and tested to verify that it operates properly. Work must also be completed on the control protocols that operate between the VME processor and the associated workstation or server.

7.2. UNIX Applications

The necessary user interfaces should be developed so that UNIX commands such as telnet and FTP work with the virtual encryption processor in the same manner as they do without virtual channel encryption, from the user’s point of view. It would also be a good demonstration of the virtual channel encryption concept if it could be demonstrated in a client-server link between a workstation and a server running an SQL database application.

7.3. Extension to Personal Computers

The virtual encryption processor should be installed in a personal computer running one of Microsoft’s operating systems such as Windows NT. The personal computer could then be used as a client to get access to a database server over a network with the communications channel protected by encryption.

It could also be demonstrated with LINUX, a version of UNIX that will run on personal computers. Using the virtual encryption process with LINUX may be difficult, however, since current versions of LINUX do not support threads. Of course, a potentially useful project would be to extend the threads concept to LINUX.

7.4. Extension to Networks Other than Ethernet

Two obvious choices are FDDI and ATM. Both networks can operate at speeds much greater than those found in Ethernet, and may replace Ethernet as workstations and servers demand higher network bandwidths. Most of the effort would be in the modification to the device drivers that control the FDDI and ATM interfaces.
8. Conclusion

This research helped in the understanding of making multi-threaded encryption algorithms in support of multi-context switching encryption devices. The research allowed us to develop a software based virtual channel encryptor for the SUN SPARC workstation to show proof-of-concept. The software implementation demonstrated that we could maintain multiple encrypted channel contexts through a single DES algorithm. However, this mechanism was not made transparent to the application. It required a knowledgeable user to setup the channels or a modification to the application to have it call the encrypted socket or stream head. Although we were able to demonstrate the system with several channels, we did not have the time to obtain performance measurements that could be included in this report.

Although the work successfully completed the software implementation on the Sun workstation, we were not successful in completing the development of a working hardware implementation of the VCE. As described in the paper, the hardware implementation would be based on the VME back-plane using the Motorola 68030 microprocessor. The hardware was procured and software modification had begun when the LDRD ended. This research has led to another research project called the Robustness Agile Encryptor that will deliver hardware this year. Although we were unable to complete this work the research has made an impact.

The final delivery was to try and integrate the VCE concept into a database. This work was never started. It was determined that the approach would not have worked within a medium to large data base. This was the first application dropped to assure that concept work would be completed.

Although additional work remains to be performed in this area, the concept remains valid and there continues to be a quantifiable demand for this type of technology capability, both in the defense and non-defense arenas.
9. References


## 10. Appendices

### 10.1. LDRD Proposal

**LDRD PROPOSED WORK**

**PROPOSAL COVER PAGE**

<table>
<thead>
<tr>
<th>Project Title:</th>
<th>Virtual Channel Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible Project Manager (PM):</td>
<td>Michael Sjulin, 9417</td>
</tr>
<tr>
<td>Principal Investigator(s):</td>
<td>Dean J. Gibson, 9417, James A. Davis, 1423</td>
</tr>
</tbody>
</table>

**Note to Preparer:** These three topics (two for new proposals) may be any length. However, they must fit in the space provided (this box).

**Abstract (Nature of Work):**
Modern networked computers typically support multiple simultaneous “virtual” communication channels to different destinations over a single physical connection to a data communication network. Inherently, it has been difficult to provide adequate security to virtual connections, although, the current solutions have employed encryption of everything passing over a link between two nodes of the network (link encryption) or encryption of data upon entry and exit from the network (end-to-end encryption). Both link and end-to-end encryption currently only support a single channel, either virtual or direct connect. Another approach has been to encrypt the data files stored in a computer before transmission over the network (file encryption).

This research proposes to develop a proof-of-principle system that will support encrypting multiple virtual channels between the user’s computer and the network without physically separating them, eliminating the need for multiplexing and demultiplexing operations and for individual encryption devices to support each data channel. This approach would reduce the bulk and cost of the equipment required to support a multi-channel communication system.

The issues to be addressed include the maintenance of multiple encryption algorithms inside a single encryption device, synchronization of multiple encrypted virtual channels, transmission of routing information through the encryption units, and key handling. Extensions of these concepts to protect SQL database will also be investigated. Two approaches to implementing this system for a database will be considered: storage of the database objects in encrypted form, which would require encryption and decryption for the data entering and exiting the database, and encryption on access to the database, so that clear text information in the database is transmitted in encrypted form over the network.

**Work Proposed for Next Year:**
A proof-of-principle system will be established in Department 9417’s Information Technology Lab (ITL) in Building 835, using the Sun workstations as multi-user terminals and servers. A software-based implementation of the control system, utilizing DES, will be installed onto several workstations to support multiple virtual channels. Key-handling issues, transmission of routing information through the encryption algorithm, process control algorithms or protocol development, and channel synchronization are topics that will be specifically addressed. An SQL database will also be installed on one of the machines for integration into the communications network, so that research can be extended to support database access control and data protection.
LDRD Data Input Form

Please ensure that an entry has been provided in every data field.

Proposal Number: [ ]
Duration: [ ] 1 Year [ ] 2 Year [ ] 3 Year
(Leave blank, can only be assigned by the LDRD Office)

If Renewal Proposal: [ ] 2nd Year - (Go to Proposal Title) [ ] 3rd Year - (Go to Proposal Title)

Involves Living Subjects?: [ ] Yes [ ] No

Do you wish to give an Oral Presentation for this Proposal? [ ] No [ ] Yes
Will it be classified? [ ] Yes [ ] No

Is this an NFRD proposal? [ ] Yes - (Select a Technology Area and Program Area)
(Leave blank, can only be assigned by the LDRD Office)

Technology Area (Check only one)

- [ ] Computational, Computer, & Mathematical Science
- [ ] Manufacturing Systems
- [ ] Structural Materials Development
- [ ] Solid State Sciences & Technology
- [ ] Engineering Mechanics
- [ ] Information Technologies
- [ ] Engineered Systems and Devices
- [ ] Non Structural Materials Development
- [ ] Large-Scale Systems Analysis, Design, and Integration
- [ ] Environmental Sciences

Program Area (Check only one)

- [ ] Engineered Processes and Materials
- [ ] Microelectronics and Photonics
- [ ] Integrated Capabilities
- [ ] Electronics
- [ ] National Security Technology
- [ ] Counter-proliferation
- [ ] Biomedical Engineering
- [ ] Computational and Information Sciences
- [ ] Engineering Sciences
- [ ] Information Science and Technology
- [ ] Advanced Manufacturing Technologies
- [ ] Energy & Environmental Sciences
- [ ] Transportation

Proposal Title: [ ] Virtual Channel Encryption

Lead Principal Investigator: [ ] Dean J. Gibson
Project (Subcase) Manager: [ ] Michael R. Sjulin

Funding Requested: (From LDRD Cost Estimate Worksheet)

<table>
<thead>
<tr>
<th>FY95</th>
<th>Total ($)</th>
<th>Manpower (FTE)</th>
<th>Total Labor ($)</th>
<th>Total DC ($)</th>
<th>Total SC ($)</th>
<th>Total Assessments ($)</th>
<th>Inflation ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>1.2</td>
<td>192K</td>
<td>20K</td>
<td>5K</td>
<td>12K</td>
<td>11K</td>
<td></td>
</tr>
<tr>
<td>236K</td>
<td>1.2</td>
<td>192K</td>
<td>10K</td>
<td>12K</td>
<td>22K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Breakdown by Center for FY95:

<table>
<thead>
<tr>
<th>Center #</th>
<th>Total ($)</th>
<th>Manpower (FTE)</th>
<th>Total Labor ($)</th>
<th>Total DC ($)</th>
<th>Total SC ($)</th>
<th>Total Assessments ($)</th>
<th>Inflation ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td>36K</td>
<td>0.2</td>
<td>32K</td>
<td>2K</td>
<td>2K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9400</td>
<td>204K</td>
<td>1.0</td>
<td>160K</td>
<td>5K</td>
<td>10K</td>
<td>9K</td>
<td></td>
</tr>
</tbody>
</table>
Scientific and Technical Soundness

Problem: A client or server workstation connected to a data communication network often needs to establish simultaneous communication channels to more than one destination. Even with only one physical connection between the workstation and the network, multiple “virtual” channels can be supported over this link because typical data communication technologies such as ATM and FDDI use data packets that contain addressing information which allows the network to route the data to the proper destination.

Inherently, it has been difficult to provide adequate security to virtual connections, although, the current solutions have employed encryption of everything passing over a link between two nodes of the network (link encryption) or encryption of data upon entry and exit from the network (end-to-end encryption). Both link and end-to-end encryption currently only support a single channel, either virtual or direct connect. Another approach has been to encrypt the data files stored in a computer before transmission over the network (file encryption). Protection of information passing over the connection between the workstation and the network is still a research topic, since present encryption techniques cannot handle multiple virtual channels going to different destinations over a single physical link. As an example, link encryption, a current approach, encrypts everything passing over a link between two nodes of a network, but also decrypts everything before the data can be processed by a network node for transmission to the next destination. Another approach, end-to-end encryption, encrypts data upon entry and exit from the network, but this approach has limited the number of virtual channels on a physical link to one. A third approach, file encryption, encrypts a file in the workstation before it is broken into packets for transmission over the network.

This research proposes to develop a proof-of-principle system that will support encrypting multiple virtual channels between the user’s computer and the network without physically separating them, eliminating the need for multiplexing and demultiplexing operations and for individual encryption devices to support each data channel. The encryption algorithm will be placed in the communications link between a workstation and a network to process multiple virtual connections using an end-to-end encryption technology for each virtual channel. The encryption device will need to maintain a separate encryption process for each virtual communications channel, passing the routing address for each virtual channel around the encryption process.

Issues: Several technical issues need to be addressed as part of this project. The first is that of developing a control algorithm to control several simultaneous virtual channel encryption processes using a single encryption engine or kernel. Each of these processes need to have the encryption keys and state variables maintained, the network address information maintained, and each encryption process maintained in synchronization. There are issues surrounding the generation and distribution of key data and the method by which this data is updated to support a secure open network (Kerberos will be the starting methodology used to implement this task). The technique by which the routing address information is maintained and passed around the encryption process will be derived from current end-to-end encryption devices, but the actual implementation will more than likely require a change to support the multiple virtual channels running concurrently. Not only must methods be developed to maintain synchronization for N encryption processes at each end of a virtual link, but a need to establish the practical value of N for a given system implementation is required. Finally, the integration of this approach into query processes of an SQL database needs to be addressed. One major issue when encrypting data for storage in a database, is how will data be tagged to support plan text queries.

Approach: This proposal will support a two fold approach to implementing the multi-virtual channel encryption system. To support the approaches, a prototype data communications network will be established in Building 835 using the Department 9417’s Information Technology Lab (ITL).

In the first approach, a software-based encryption algorithm based on DES will be the encryption demel used on the Sun workstations to perform the encryption process. A kernel control algorithm will be developed to control the encryption processes required to maintain the encryption state variables, key variables, and virtual link address information. As a daemon to this process, an algorithm to maintain synchronization between the processes at each end of a virtual link will be developed. A key-handling scheme, initially based on Kerberos, will be used to distribute and update the necessary encryption keys throughout the network. The Kerberos security process will be modified as required to better support the multi-virtual channel encryption process.

In the second approach, a hardware version of the control system described above will be prototyped for insertion in the link between a workstation and the network. The goal is to develop a device that performs these functions without any need for software modifications in the workstation and will support a greater number of virtual channels. This will also eliminate the additional load that would be placed on the workstation processor in performing the encryption and
decryption functions. Once the two approaches are implemented, the research to evaluate system performance and the number of virtual channels supportable by the two approaches will be determined.

The final experiment will be to install an SQL database on one of the workstations in the ITL and augment the algorithms to support the query functions for access into the database. This implementation will use encryption in storing objects to and retrieving objects from the database. A means by which the system can tag data objects that are encrypted to support the query functions of the database will need to be developed. The encryption processor will also be installed between the database and the network to demonstrate that a user on the network will have full access to the database with adequate network security.

Expected Results: At the end of this research, we expect to have developed in software the necessary algorithms to handle multiple virtual links between workstations on a network, and to have developed a hardware prototype which will duplicate these functions. We will also have demonstrated that a user can access a database over a network using a similar encryption process and will also have developed an approach to providing access separation in an SQL database. We will have an understanding of the maximum number of virtual circuits supportable by a software and hardware implementation of the system.

Creativity and Innovation

This project is an extension of the current state-of-the-art in encryption technology into several new areas. The concept of a multi-channel encrypter, along with the key distribution and encryption channel synchronization technologies necessary to support it is new in supporting network virtual channels. As mentioned before, encryption technologies are available for link and end-to-end encryption, but this work will extend to handle multiple virtual circuits over a single physical link using the same encryption engine running concurrent encryption processes. Extension of these concepts onto the access process for an SQL database provides some unique approaches to data security. The use of multi-channel encryption technologies will lead to some new concepts for the separation and protection of database objects, and to new methods for controlling the necessary access of multiple, simultaneous users to the database.

Project Plan

Milestones:

- Implement DES in software on workstation 10/94
- Modify encryption object to pass routing information 3/95
- Create Encryption Process Control Algorithm 6/95
- Set up key distribution scheme 9/95
- Demonstrate hardware encryption prototype 3/96
- Install SQL database on workstation 3/96
- Modify SQL database to use encryption during storage/retrieval 4/96
- Install encryption process between database and network 7/96
- Write SAND report 9/96
Staffing:
Dean Gibson, Lead Principal Investigator, 9417
Contribution: Project management, Key distribution and virtual channel control process, and network design.
Biography: Education: PhD in Electrical Engineering Work Related: digital and analog circuit design, signal processing, cryptographic equipment support, communication networks, and software design.

James A. Davis, Principal Investigator, 1423
Contribution: cryptographic support (key handling and algorithm implementation)
Biography: Education: PhD in Probability and Statistics Work Related: weapon system analysis, applied mathematics, and cryptographic application (for Command & Control projects).

Budget:
<table>
<thead>
<tr>
<th></th>
<th>FY95 FTEs</th>
<th>FY96 FTEs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gibson (0.8), Davis (0.2), 9400 staff (0.2) - Total of 1.2 FTEs</td>
<td>Gibson (0.8), Davis (0.2), 9400 staff (0.2) - Total of 1.2 FTEs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FY95 DCs</th>
<th>FY96 DCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Implementation Components - 20K</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FY95 SCs</th>
<th>FY96 SCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drafting Definition and Network Definition - 5K</td>
<td>Shop Support - 7K</td>
<td></td>
</tr>
<tr>
<td>SAND Reports Processing - 3K</td>
<td>Total - 10K</td>
<td></td>
</tr>
</tbody>
</table>

Impact
Virtual channel encryption technologies could have a substantial impact on the protection of data on communication networks and database storage systems. They would eliminate the need for the physical separation of communication networks handling different types of sensitive information. Extended to database applications, these technologies could permit the storage of different levels of sensitive information on the same computer, especially if the data is stored in encrypted form. If one can use the appropriately approved encryption algorithms from NSA, it might be possible to extend these concepts to the communication and storage of classified data.

These same concepts will be very useful in the evolving National Information Infrastructure. Here, they could be used to provide protection for financial, medical, and other types of sensitive information, in both data communication and data storage applications, eliminating much of the need for physical separation of the information to provide adequate security. They would also provide a more seamless means to providing encryption on a Wide Area Network (WAN) or Local Area Network (LAN) where many user nodes are involved.

Signatures

James A. Davis, 1423, Principle Investigator

Dean J. Gibson, 9417, Lead Principal Investigator

Michael Sjulin, 9417, Project Manager

Michael Eaton, 9400, Center Director
10.2. FY 1995 Progress Report

<table>
<thead>
<tr>
<th>Case Number</th>
<th>3517.270</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>Virtual Channel Encryption</td>
</tr>
<tr>
<td>Project Manager</td>
<td>M. R. Sjulin, 9417</td>
</tr>
<tr>
<td>Principal Investigators</td>
<td>Dean J. Gibson, 9417</td>
</tr>
</tbody>
</table>

Abstract - A descriptive project abstract (100-200 words)

Modern networked computers typically support multiple simultaneous "virtual" communication channels to different destinations over a single physical connection to a data communication network. Protection of such connections has been difficult due to the lack of encryption equipment capable of simultaneously processing multiple virtual channels over a single physical connection.

This project is the development of a proof-of-concept system that will support the encryption of multiple virtual channels over a single physical link between a user's computer and a network, eliminating the need for multiplexing and demultiplexing operations and individual encryption devices for each channel. Both a software module for installation in the user's computer and an external hardware module to perform these functions are being developed. Issues being addressed include: maintenance of multiple encryption algorithms inside a single encryption device, synchronization of multiple encrypted virtual channels, transmission of routing information through the encryption units, and key handling.
FY95 Accomplishments - (150-400 words)

**Background** - This project addresses the need for security in the transmission of data over a communications network from a workstation which can support multiple virtual channels over a single network connection. Past approaches to this problem have required the demultiplexing of the virtual channels into distinct physical channels, each to be processed by its own encryption device. This project is the development of software and hardware which can encrypt multiple virtual channels simultaneously, eliminating the need for demultiplexing and separate encryption devices for each channel.

**Main conclusions/findings** - A DES (Data Encryption Standard) algorithm has been modified to handle multiple virtual data channels simultaneously, and has been installed on a workstation. A copy of the Kerberos network security software has also been installed on this workstation to provide key handling and link control functions. From these activities, we conclude that one can perform the virtual channel encryption functions in a software module that resides in the user’s workstation.

**Recommendations, if any** - It is recommended that this project be continued for another year to permit the development of a hardware virtual channel encryptor to be placed in the communication link between the workstation and the network. This will unload the encryption computations from the workstation processor, substantially reducing the workstation’s workload.

**A review of data** - To handle multiple data channels simultaneously, the DES encryption algorithm chosen for this project was modified by stripping out all of its internal variables, leaving only pure code, so that it can handle multiple data streams simultaneously, with a separate data structure handling the state variables for each data stream. The modified DES algorithm has been inserted into a "STREAMS" Ethernet driver to provide the encryption function needed for the virtual channel encryptor. The STREAMS device driver concept features a modular approach to the design of device driver components which simplifies the insertion of the encryption software.

Kerberos is a network security concept which controls access to the resources attached to a communications network. A "key distribution center" is used to generate a set of "tickets" which are given to each user to gain access to resources attached to the network. Each time a user requests access to a resource, one of these tickets is used to establish a secure link between the user and that resource.

**Refereed publications resulting from the work**: Required info is as follows: Author(s), title of article, publisher, where published, volume number, page numbers, date of publication (mo/yr) (see attached bibliographic reference sheet for more detail)

| None |

**All other publications resulting from the work**: Required info is as follows: Author(s), title of article, name of conference where paper was presented, date of presentation (mo/day/yr), location (city, state, country), volume number, page numbers (see attached bibliographic reference sheet for more detail)

| None |
| **Number of patent disclosures:** (where the invention was at least in part attributable to LDRD support) | None |
| **Number of patent applications:** (where the invention was at least in part attributable to LDRD support) | None |
| **Number of patents:** (where the invention was at least in part attributable to LDRD support) | None |
| **Number of copyrights on computer software:** (where the code was at least in part attributable to LDRD support) | None |
| **Number of students:** (if any) supported by the project | None |
| **Number of post docs:** (if any) supported by the project | None |
| **Number of permanent technical or scientific staff hired:** (if any) supported by the project | None |
| **Number of awards (and their names):** by organizations outside the laboratory to an individual or team attributed at least in part to LDRD support | None |
| **Number of new non-LDRD funded projects:** their amounts and source of funding | None |

**Your qualitative assessment about the completion of your milestones for the year in percent:** 100%

**Your qualitative assessment about the direction of the project as a result of research or other findings:** (Please place an X over the number of the statement that best describes your results)

| **Goals met, hypothesis proved** |
| **Goals partially met, hypothesis modified** |
| **Goals substantially modified, hypothesis redefined** |
| **Goals not met, hypothesis disproved** |
| **Project terminated because:** |
10.3. Process Flowcharts

**NOTE:** The diagrams illustrate the flow of data and operations in a distributed computing environment. The process begins with a login request from a client, which is processed by the server. The server then initiates a session and distributes keys, which are encrypted and transmitted to ensure security. The flowchart outlines the steps from the initialization of a session to the completion of a transaction, ensuring the integrity and confidentiality of the data.
10.4. Listing of Code for the Virtual Encryption Process
DISTRIBUTION:

1 MS 0431  S. G. Varnado
1 MS 0451  J. H. Moore
5 MS 0451  M. R. Sjulin
2 MS 0100  Document Processing for DOE/OSTI, 7613-2
1 MS 0619  Print Media, 12615
5 MS 0899  Technical Library, 4414
1 MS 9018  Central Technical Files, 8523-2
3 MS 0449  R. A. Sarfaty
:bi1-version 1.0
:project project5
{
  :files {vce5.bil}
  :root-window vce5.mainwindow
  :session-management {
    :session-terminating-method :none
  }
  :tooltalk {
    :desktop-level :none
  }
  :internationalization {
    :enabled :false
  }
  :res-file-types {}
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <Xm/Xm.h>
#include <Xm/Mwmutil.h>
#include <Xm/Protocols.h>
#include <Dt/Help.h>
#include <Dt/HelpQuickD.h>
#include <Dt/HelpDialog.h>
#include "vce5-ui.h"
#include "project5.h"
#include "dtb-utils.h"

XtAppcontext app-usr = (XtAppContext)NULL;

static XrmoptionDescRec optionDescList[I] = {
    {"-session", "+session", XrmoptionSepArg, (XP0inter)NULL}
};

static XtResource resources[I] = {
    {"session", "Session", XrRString, sizeof (String),
    XtOffsetOf(DtbAppResourceRec, session_file), XtRImmediate, (XtP0inter)NULL}
};

DtbAppResourceRec dtb_app_resource_rec;

int main(int argc, char **argv)
{ 
    Widget toplevel = (Widget)NULL;
    Display *display = (Display*)NULL;
    XtAppContext app = (XtAppContext)NULL;
    String *fallback_resources = (String*)NULL;
    ArgList init_args = (ArgList)NULL;
    Cardial num_init_args = (Cardial)0;
    ArgList get_resources_args = (ArgList)NULL;
    Cardial num_get_resources_args = (Cardial)0;
    Atom save_yourself_atom = (Atom)NULL;

    /* main for application project5 */
    int
    main(argc, argv);
/* Get display and verify initialization was successful. */
if (toplevel != NULL) {
    display = XtDisplayOfObject(toplevel);
} else {  
    fprintf(stderr, "Could not open display.\n");
    exit(1);
}
/* Save the toplevel widget so it can be fetched later as needed. */
dtb_save_toplevel_widget(toplevel);
/* Save the command used to invoke the application. */
dtb_save_command(argv[0]);

XGetApplicationResources(toplevel, (XtPointer)tstub-app-resource-rec, 
resources, XtNumber(resources), 
get_resources_args, num_get_resources_args);

/* Initialize all global variables. */
dtb_vce5_mainwindow_info_clear(&dtb_vce5_mainwindow);
/* Set up the application's root window. */
dtb_vce5_mainwindow = toplevel;
XtVaSetValues(dtb_vce5_mainwindow, 
XmNallowShellResize, True, 
XmNtitle, "VCE Receiving Window", 
XmNinitialState, NormalState, 
XmNbackground, dtb_cv_string_to_pixel(dtb_vce5_mainwindow, "white"), 
NULL);
dtb_vce5_mainwindow_initialize(&dtb_vce5_mainwindow, dtb_get_toplevel_widget());
/* Map any initially-visible windows */

save_yourself_atom = XmInternAtom(XtDisplay(toplevel), 
"WM_SAVE_YOURSELF", False);
XmAddWMProtocolCallback(toplevel, save_yourself_atom, 
save_yourself, (XtPointer)NULL);

/* Initialize all global variables. */
dtb_vce5_mainwindow_info_clear(&dtb_vce5_mainwindow);
/* Set up the application's root window. */
dtb_vce5_mainwindow = toplevel;
XtVaSetValues(dtb_vce5_mainwindow, 
XmNallowShellResize, True, 
XmNtitle, "VCE Receiving Window", 
XmNinitialState, NormalState, 
XmNbackground, dtb_cv_string_to_pixel(dtb_vce5_mainwindow, "white"), 
NULL);
dtb_vce5_mainwindow_initialize(&dtb_vce5_mainwindow, dtb_get_toplevel_widget());
/* Map any initially-visible windows */

save_yourself_atom = XmInternAtom(XtDisplay(toplevel), 
"WM_SAVE_YOURSELF", False);
XmAddWMProtocolCallback(toplevel, save_yourself_atom, 
save_yourself, (XtPointer)NULL);
XtRealizeWidget(toplevel);

#ifndef DKB_USER_CODE_START

The initially-mapped widgets have all been realized, and
the Xt main loop is about to be entered.

#endif

#ifndef DTB_USER_CODE_END

End of user code section

#endif

住房公积

Encer event loop

XtAppMainLoop(app);
return 0;

#ifndef DKB_USER_CODE_START

All automatically-generated data and functions have been defined.
Add new functions here, or at the top of the file.

#endif

#ifndef DTB_USER_CODE_END

End of user code section

#endif
/** Add file header below vvv ***/ 
/* Add file header above *** */

/* *
Pile: project5.h
*
Contains: object data structures and callback declarations
*
This file was generated by dtcodegen, from project project5
*
Any text may be added between the DTB_USER_CODE_START and
DTB_USER_CODE_END comments (even non-C code). Descriptive comments
are provided only as an aid.
*
DIV ** EDIT ONLY WITHIN SECTIONS MARKED WITH DTB_USER_CODE COMMENTS. **
DIV ** ALL OTHER MODIFICATIONS WILL BE OVERWRITTEN. DO NOT MODIFY OR **
DIV ** DELETE THE GENERATED COMMENTS! **
*/
 ifndef _PROJECTS_H_
define _PROJECTS_H_

#include <stdlib.h>
#include <Xll/Intrinsic.h>
/* *
Structure to store values for Application Resources
*/
t ypedef struct {
  char *session_file;
  
} Dt bAppResourceRec;

extern DtbAppResourceRec dtb_app_resource_rec;

/*****************************/

/** *
Add types, macros, and externs here
**/

/*****************************/

#endif /* _PROJECTS_H_ */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/uio.h>
#include <stropts.h>
#include <sys/con:.h>
#include <errno .h>
#include <sys/fcntl.h>
#include <stdio. h>

/* *
File: v-5-stubs.c
Contains: Module callbacks and connection functions
This file was generated by dtcgen, from module vce5
Any text may be added between the DTB-USER-CODE-START and
DTB-USER-CODE-END comments (even non-C code). Descriptive comments
are provided only as an aid.
** EDIT ONLY WITHIN SECTIONS MARKED WITH DTB-USER-CODE коммент.
** ALL OTHER MODIFICATIONS WILL BE OVERWRITTEN. DO NOT MODIFY OR
** DELETE THE GENERATED COMMENTS!
*/
#include <stdio.h>
#include <Xm/Xm.h>
#include "dtb-uti1s.h"
#include "vces-ui.h"
/* *
Header files for cross-module connections
*/
#include <Xm/Rowcolumn.h>
#include <Xm/Label.h>
#include <Xm/TextF.h>

/** *
** All necessary header files have been included.
** Add include files, types, macros, externs, and user functions here.
**
 extern XtAppContext appusr;
check-data(

DtbVceSMainwindowInfo clientmta,
XtIntervalId callData
void
vce5 textField CBli-
Widget widgec.
XtPointer clienc3ata.
XtPointer callData

if (!(!(dtbTarget->inirialized))
XtPopdown (instance-mainwindow)

DtbVce5MainwindowInfo instance - dtbTarget; /* obsolete */
if (!(!dtbTarget->initialized))

DcbVce5MainwindowInfo instance = dtbTarget; /* obsolete */
if (!(!dtbTarget->initialized))

if (!(!dtbTarget->initialized))

void
vce5_button_CBli(
 Widget widget,
 XtPointer clientData,
 XtPointer callData
}
```c
void exit_func(Widget widget, XtPointer clientData, XtPointer callData)
{
    /* *** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/
    exit(0);
    /* *** DTB_USER_CODE_END >>> Add C variables and code above <<< */
}

void activate_CB(Widget widget, XtPointer clientData, XtPointer callData)
{
    /* *** DTB_USER_CODE_START vvv Add C variables and code below vvv ***/
    XtIntervalId timer_id;
    DtbVceSMainwindowInfo dtbTarget = (DtbVceSMainwindowInfo)clientData;
    DtbVceSMainwindowInfo instance = dtbTarget; /* obsolete */
    if (!dtbTarget->initialized)
    {
        dtb_vce5_mainwindow_initialize(dtbTarget, dtb_get_toplevel_widget());
    }
    /* timer set for every 3 seconds (3000) */
    timer_id = XtAppAddTimeOut(app_usr, 3000, check-data, instance);
    /* *** DTB_USER_CODE_END >>> Add C variables and code above <<< */
    /* *** DTB_USER_CODE_START vvv Add C code below vvv ***/
    /* *** DTB_USER_CODE_END >>> Add C code above <<< */
}

/***
DTB_USER_CODE-START
All automatically-generated data and functions have been defined.
Add new functions here, or at the top of the file.
***/

void check_data(DtbVceSMainwindowInfo clientData, XtIntervalId callData)
{
    XtIntervalId timer_id;
    int count_id;
    if ((fd = open("test_file", 0_RDWR)) < 0)
    {
        printf("VCE module inactive
");
        timer_id = XtAppAddTimeOut(app_usr, 5000, check_data, clientData);
        return;
    }
    close(fd);
    printf("VCE module found\n");
    XtManageChild(clientData->textfield_rowcolumn);
    XtManageChild(clientData->textfield3_rowcolumn);
    XtVaSetValues(clientData->textfield, XmNvalue, "VCE_text_or_file_name", NULL); /* unencrypted */
    XtVaSetValues(clientData->textfield3, XmNvalue, "VCE_text_or_file_name", NULL); /* encrypted */
}
```
*/
* File: vce5_ui.c
* Contains: user module object initialize and create functions
* 
* This file was generated by dtcodegen, from module vce5
* 
* ** DO NOT MODIFY BY HAND - ALL MODIFICATIONS WILL BE LOST **
*/

#include <stdio.h>
#include <stdlib.h>
#include <Xll/Intrinsic.h>
#include <Xll/Shell.h>
#include <Xm/PanedW.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/NewColumn.h>
#include <Xm/TextF.h>
#include <Xm/Label.h>
#include "dtb_utils.h"
#include "project5.h"
#include "vce5-ui.h"

DtbVce5MainwindowInfoRec    dtb_vce5_mainwindow =
{
    False /* initialized */
};

/* Widget create procedure decls */
static int dtb_vce5_mainwindow_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_controlpane_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_button_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_button2_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_textfield_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_button3_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_textfield3_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

static int dtb_vce5_textfield4_create(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
);

int dtbVce5MainwindowInfo_clear(DtbVce5MainwindowInfo instance)
{
    memset((void *)(instance), 0, sizeof(*instance));
    return 0;
}

int dtb_vce5_mainwindow_initialize(    
    DtbVce5MainwindowInfo instance,    
    Widget parent
)
{
    WidgetList children = NULL;

    int numChildren = 0;

    if (instance->initialized)
    {
        return 0;
    }

    instance->initialized = True;

    dtb_vce5_mainwindow_create(instance, parent);
    dtb_vce5_controlpane_create(instance, instance->mainwindow_form);
    dtb_vce5_button_create(instance, instance->controlpane);
    dtb_vce5_button2_create(instance, instance->controlpane);
    dtb_vce5_textfield_create(instance, instance->controlpane);
    dtb_vce5_button3_create(instance, instance->controlpane);
    dtb_vce5_textfield3_create(instance, instance->controlpane);
    dtb_vce5_textfield4_create(instance, instance->controlpane);

    dtb_vce5_mainwindow_destroy(instance, parent);
}

59
/* Add widget-reference resources. */
XtVaSetValues(instance-mainwindow-mainwin, XmNworkWindow, instance->mainwindow-form, NULL);

/* Call utility functions to do group layout */

/* Manage the tree, from the bottom up. */
XtVaGetValues(instance->textfield_rowcolumn, XmNchildren, &children, XmNnumChildren, &numChildren, NULL);
XtManageChildren(children, numchildren);
XtVaGetValues(instance->textfield3_rowcolumn, XmNchildren, &children, XmNnumChildren, &numChildren, NULL);
XtManageChildren(children, numchildren);
XtManageChild(instance->button);
XtManageChild(instance->textfield4);
XtVaGetValues(instance->mainwindow-form, XmNchildren, &children, XmNnumChildren, &numChildren, NULL);
XtManageChildren(children, numchildren);
XtManageChild(instance->mainwindow-form);
XtVaGetValues(instance-mainwindow, XmNchildren, &children, XmNnumChildren, &numChildren, NULL);
XtManageChildren(children, numchildren);
XtManageChild(instance->mainwindow-form);

/* Add User and Connection callbacks */
XtAddCallback(instance->button, XmNactivateCallback, vce5-button-CBl.
(XtPointer)&(*instance:));
XtAddCallback(instance->button2, XmNactivateCallback, activate-CB,
(XtPointer)&(*instance));
XtAddCallback(instance->textfield, XmNvalueChangeCallback, vce5-textfield-CB1,
(XtPointer)&(*instance));
XtAddCallback(instance->button3, XmNactivateCallback, exit_func,
(XtPointer)&(*instance));

return 0;

static int
dtb_vce5_mainwindow_create(instance, widget parent);
{
    XmString  label_xmstring = NULL;
    Display  *display = (parent == NULL ? NULL : XtDisplay(parent));

    if (instance->mainwindow == NULL) {
        instance->a.window =
        XtVaCreatePopupShell("mainwindow",
            XtAppShellWidgetClass, parent);
        XtVaSetValues(instance->mainwindow, XmNborderWidth, 1,
            XmNinnerBorderWidth, 1, XmNresistive, True,
            XmNtitle "TCI ReceiveWindow",
            XmNnormalState, NormalState,
            XtVaFromId, db_cv5_string_to_pixel(parent, "white"), NULL);
    }
    if (instance->a.window == NULL) return -1;

    if (instance->mainwindow_mainwin == NULL) {
        instance->mainwindow_mainwin =
        XtVaCreateWidget("db_cv5_mainwindow_mainwin",
            XmMainWindowClass, instance->mainwindow, NULL);
    }
    if (instance->mainwindow_mainwin == NULL) return -1;

    if (instance->mainwindow_form == NULL) {
        instance->mainwindow_form =
        XtVaCreateWidget("mainwindow_form",
            XmFormWidgetClass, instance->mainwindow_mainwin, NULL);
    }
    if (instance->mainwindow_form == NULL) return -1;

    return 0;
}
static int
dtb_vce5_controlpane_create(
  DtbVce5MainWindowInfo instance,
  Widget parent)
{
  if (instance->controlpane == NULL)
    return -1;
  return 0;
}

static int
dtb_vce5_button_create(
  DtbVce5MainWindowInfo instance,
  Widget parent)
{
  XmString label_xmstring = NULL;
  label_xmstring = XmStringCreateLocalized("Clear");
  if (instance->button == NULL)
    instance->button =
      XVaCreateWidget("button",
        xmPushButtonClass, parent,
        XmNbottomAttachment, XmATTACH_NONE,
        XmNrightAttachment, XmATTACH_NONE,
        XmNleftAttachment, XmATTACH_NONE,
        XmNrightOffset, 3, 3,
        XmNleftOffset, 3, 3,
        XmNtopAttachment, XmATTACH_FORM,
        XmNtopOffset, 247, 247,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNrecomputeSize, True,
        XmNalignment, XmALIGNMENT_CENTER,
        XmNlabel, label_xmstring,
        NULL);
  XmStringFree(label_xmstring);
  label_xmstring = NULL;
  if (instance->button == NULL)
    return -1;
  return 0;
}

static int
dtb_vce5_button2_create(
  DtbVce5MainWindowInfo instance,
  Widget parent)
{
  XmString label_xmstring = NULL;
  label_xmstring = XmStringCreateLocalized("Activate");
  if (instance->button2 == NULL)
    instance->button2 =
      XVaCreateWidget("button2",
        xmPushButtonClass, parent,
        XmNbottomAttachment, XmATTACH_NONE,
XmNrightAttachment, XmATTACH_NONE,
XmNleftAttachment, XmATTACH_FORM,
XmNtopOffset, 99,
XmNtopAttachment, XmATTACH_FORM,
XmNrecomputeSize, True,
XmNalignment, XmALIGNMENT_CENTER,
XmNheight, 99,
XmNwidth, 170,
XmNbackgro un, dtb cvt_string_to_pixel(parent, "Yellow"),
XmNlabelString, label_xmstring,
NULL);
XmStringFree(label_xmstring);
label_xmstring = NULL;
if (instance->button2 == NULL)
    return -1;
return 0;

static int
dtb_vce5_textfield_create(
    DtbVce5MainWindowInfo instance,
    Widget parent)
{
    XmString label_xmstring = NULL;
    Arg arqs[47]; /* need 42 args (add 5 to be safe) */
    int n = 0;
    if (instance->textfield_rowcolumn == NULL) {
        instance->textfield_rowcolumn =
            XmCreateWidget("textfield_rowcolumn",
            xRowColumnWidgetClass,
            parent,
            XmNrightAttachment, XmATTACH_NONE,
            XmNleftAttachment, XmATTACH_NONE,
            XmNleftOffset, 45,
            XmNleftAttachment, XmATTACH-FORM,
            XmNtopOffset, 159,
            XmNtopAttachment, XmATTACH-FORM,
            XmNadjustLast, True,
            XmNpadding, 0,
            XmNmarginWidth, 0,
            XmNmarginHeight, 0,
            XmNtryAlignment, XmALIGNMENT_END,
            XmNorientation, XmHORIZONTAL,
            XmNheight, 159,
            XmNwidth, 45,
            NULL);
    }
    if (instance->textfield_rowcolumn == NULL)
        return -1;
    label_xmstring = XmStringCreateLocalized("Uncrypted Text:");
    if (instance->textfield_label == NULL) {
        instance->textfield_label =
            XmCreateWidget("textfield_label",
            xLabelWidgetClass,
            instance->textfield_rowcolumn,
            XmNlabelString, label_xmstring,
            NULL);
        XmStringFree(label_xmstring);
        label_xmstring = NULL;
    }
    if (instance->textfield_label == NULL)
        return -1;
    if (instance->textfield == NULL) {
        n = 0;
        XmSetArg(args[n], XmNmaxLength, 80); ++n;
        XmSetArg(args[n], XmNeditable, True); ++n;
        XmSetArg(args[n], XmNcursorPositionVisible, True); ++n;
        instance->textfield =
            XmCreateTextField(instance->textfield_rowcolumn,
            "textfield", args, n);
    }
    if (instance->textfield == NULL)
        return -1;
    return 0;

static int
dtb_vce5_button3_create(
    DtbVce5MainWindowInfo instance,
    Widget parent)
{
    XmString label_xmstring = NULL;
    label_xmstring = XmStringCreateLocalized("Exit");
    if (instance->button3 == NULL) {
        instance->button3 =
            XtVaCreateWidget("button3",
            NULL);
        if (instance->button3 == NULL)
            return -1;
        return 0;
    }
}
xmPushButtonWidgetClass,
parent,
XmNbottomAttachment, XmATTACH-NONE,
XmNrightAttachment, XmATTACH-NONE,
XmNleftOffset, 273,
XmNleftAttachment, XmATTACH-FORM,
XmNtopOffset, 100,
XmNtopAttachment, XmATTACH-FORM,
XmNrecomputeSize, False,
XmNalignment, XmALIGNMENT_CENTER,
XmNheight, 23,
XmNwidth, 50,
XmNbackgound, "Yellow",
NULL);
XmStringFree(labe1-xmstrinq);
labe1-xmstring = NULL;
if (instance->button3 == NULL)
return -1;
return 0;

static int
dtb_vce5_textfield3_create(
DtbVce5MainwindowInfo instance,
Widget parent
)
{
XmString label-xmstrinq = NULL;
Arg args[50]; /* need 45 args (add 5 to be safe) */
in n = 0;
if (instance->textfield3_rowcolumn == NULL) {
instance->textfield3_rowcolumn = 
XtVaCreateWidget("textfield3_rowcolumn",
XmRowColumnWidgetClass,
parent,
XmNbottomAttachment, XmATTACH-NONE,
XmNrightAttachment, XmATTACH-NONE,
XmNleftOffset, 62,
XmNleftAttachment, XmATTACH-FORM,
XmNtopOffset, 196,
XmNtopAttachment, XmATTACH-FORM,
XmNadjustLast, True,
XmNspacing, 0,
XmNmarginWidth, 0,
XmNmarginHeight, 0,
XmNentryAlignment, XmALIGNMENT_END,
XmNorientation, XmHORIZONTAL,
XmNvy, 196,
XmNx, 62,
XmNforegound, "Red",
NULL);
if (instance->textfield3_rowcolumn == NULL)
return -1;
label_xmstring = XmStringCreateLocalized("Encrypted Text: ");
if (instance->textfield3_label == NULL) {
instance->textfield3_label = 
XtVaCreateWidget("textfield3_label",
XmLabelWidgetClass,
instance->textfield3_rowcolumn,
XmNforeground, dtb_cvstring_to_pixel(instance->textfield3_rowcolumn,
"Red"),
XmNlabelString, label-xmstrinq,
NULL);
XmStringFree(label_xmstring);
label_xmstring = NULL;
} if (instance->textfield3_label == NULL)
return -1;
if (instance->textfield3 == NULL) {
	c = 0;
XtSetArg(args[c], XmNmaxLength, 80); ++c;
XtSetArg(args[c], XmNeditable, True); ++c;
XtSetArg(args[c], XmNcursorPositionVisible, True); ++c;
XtSetArg(args[c], XmNColumns, 42); ++c;
XtSetArg(args[c], XmNforeground, dtb_cvstring_to_pixel(instance->textfield3_rowcolumn,
"Red")); ++c;
instance->textfield3 = 
XmCreateTextField(instance->textfield3_rowcolumn,
"textfield3", args, c);
} if (instance->textfield3 == NULL)
return -1;
return 0;
}

static int
dtb_vce5_textfield4_create(
DtbVce5MainwindowInfo instance,
Widget parent
)
Arg args[19]; /* need 14 args (add 5 to be safe) */
int n = 0;

if (instance->textfield4 == NULL) {
  n = 0;
  XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); ++n;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); ++n;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); ++n;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); ++n;
  XtSetArg(args[n], XmNtopOffset, 46); ++n;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); ++n;
  XtSetArg(args[n], XmNvalue, "Transmission Functions"); ++n;
  XtSetArg(args[n], XmNmaxLength, 80); ++n;
  XtSetArg(args[n], XmNx, 157); ++n;
  XtSetArg(args[n], XmNy, 46); ++n;
  XtSetArg(args[n], XmNeditable, False); ++n;
  XtSetArg(args[n], XmNcursorPositionVisible, False); ++n;
  XtSetArg(args[n], XmNcolumns, 23); ++n;
  instance->textfield4 = XmCreateTextField(parent, "textfield4", args, n);
}

if (instance->textfield4 == NULL)
  return -1;

return 0;
/* File: vce5-ui.h
 * Contains: Declarations of module objects, user create procedures 
 * and callbacks.
 * This file was generated by dtcodegen, from module vce5
 * ** DO NOT MODIFY BY HAND - ALL MODIFICATIONS WILL BE LOST ***/

#ifndef _VCE5_UI_H_
#define _VCE5_UI_H_

#include <stdlib.h>
#include <Xll/Intrinsic.h>
#include "dtb-utils.h"

/* User Interface Objects */

/* Shared data structures */

typedef struct {
  Boolean initialized;
  Widget mainwindow; /* object "mainwindow" */
  Widget mainwindow_mainwin;
  Widget mainwindow_form;
  Widget controlpane;
  Widget button1;
  Widget button2;
  Widget textfield_rowcolumn; /* object "textfield" */
  Widget textfield_label;
  Widget textfield;
  Widget button3;
  Widget textfield3_rowcolumn; /* object "textfield3" */
  Widget textfield3_label;
  Widget textfield3;
  Widget textfield4;
  DtbVce5MainwindowInfoRec dtb_vce5_mainwindow_info;
} DtbVce5MainwindowInfoRec, *DtbVce5MainwindowInfo;

extern DtbVce5MainwindowInfoRec dtb_vce5_mainwindow;

/* Structure Clear Procedures: These set the fields to NULL */
int dtbVce5MainwindowInfo_clear(DtbVce5MainwindowInfo instance);

/* Structure Initialization Procedures: These create the widgets */
int dtb_vce5_mainwindow_initialize(DtbVce5MainwindowInfo instance, Widget parent);

/* User Callbacks */
void exit_func(Widget widget, XtPointer clientData, XtPointer callData);
void activate_CB(Widget widget, XtPointer clientData, XtPointer callData);

/* Connections */
void vce5_textfield_CB(Widget widget, XtPointer clientData, XtPointer callData);
void vce5_button_CB(Widget widget, XtPointer clientData, XtPointer callData);

#endif /* _VCE5_UI_H_ */
/* File: dtb_util.c
   CDE Application Builder General Utility Functions
   * This file was generated by dtcodegen, from project(project)
   ** DO NOT MODIFY BY HAND - ALL MODIFICATIONS WILL BE LOST **
*/

#include <unistd.h>
#include <stdlib.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/MessageB.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Dt/Help.h>
#include <Dt/HelpDialog.h>
#include <Dt/HelpQuickD.h>
#include <Dt/Session.h>
#include <Dt/Dnd.h>
#include "dtb-utils.h"

ifndef min
#define min(a,b) ((a)<(b)? (a):(b))
endif

ifndef max
#define max(a,b) ((a)>(b)? (a):(b))
endif

ifndef ABS
#define ABS(x) ((x)>0? (x):(-(x)))
endif

typedef struct
{
    Widget widget;
    DtDndProtocol protocol;
    unsigned char operations;
    Boolean bufferIsText;
    Boolean allowDropOnRootWindow;
    Widget
    DtDndDragCallback callback;
    XtCallbackRec convertCBRec[2];
    XtCallbackRec dragToRootCBRec[2];
    XtCallbackRec dragFinishCBRec[2];
} DtDragSiteRec, *DtDragSite;

typedef struct
{
    Widget widget;
    DtDndProtocol protocol;
    unsigned char operations;
    Boolean textIsBuffer;
    Boolean dropsOnChildren;
    Boolean preservePreviousRegistration;
    DtDndDragCallback callback;
    XtCallbackRec animateCBRec[2];
    XtCallbackRec transferCBRec[2];
} DtDropSiteRec, *DtDropSite;

/* This structure keeps track of widget/menu pairs */
typedef struct
{
    Widget widget;
    Widget menu;
} DtMenuRefRec, *DtMenuRef;

/* Private functions used for dynamic centering of objects */
static void center_widget;
    Widget form_child,
    DDB_CENTERING_TYPES type
};
static void uncenter_widget(
    Widget form_child,
    DDB_CENTERING_TYPES type
);
static void centering_handler;
    Widget widget,
    XPointer client_data,
    XEvent *event,
    Boolean *cant_dispatch
);

/* Static functions used for dynamic aligning of group objects */
static Widget get_label_widget;
    Widget widget
};
static Position get_offset_from_ancestor(Widget ancestor, Widget w);
static Dimension get_label_width(Widget widget);
static void get_widest_label(WidgetList list, int count, Widget *child_widget, Dimension *label_width);
static void get_widest_value(WidgetList list, int count, Widget *child_widget, Dimension *value_width);
static void get_widest_rect(Widget widget, XRectangle *rect);
static void get_greatest_size(WidgetList list, int count, Widget *child_widget, Dimension *tallest, Dimension *widest);
static void get_group_widest_cell_size(Widget parent, DtbGroupInfo *group_info, int *cell_width, int *cell_height);
static void get_group_row_col(Widget parent, DtbGroupInfo *group_info, int *rows, int *cols);
static Widget get_group_child(Widget parent, DtbGroupInfo *group_info, int xgos, int yogos);
static void align_children(Widget parent, DtbGroupInfo *group_info, XRectangle *rect, XRectangle *call_data);
static void align_handler(Widget widget, XtPointer client_data, XEvent *event, Boolean *continue_dispatch);
static void expose_handler(Widget widget, XtPointer client_data, XEvent *event, Boolean *continue_dispatch);
static void free_group_info(DtbGroupInfo *group_info);
static void align_rows(Widget parent, DtbGroupInfo *group_info, int *rows);
static void align_cols(Widget parent, DtbGroupInfo *group_info, int *cols);
static void align_left(Widget parent, DtbGroupInfo *group_info);
static void align_right(Widget parent, DtbGroupInfo *group_info);
static void align_labels(Widget parent, DtbGroupInfo *group_info);
static void align_center(Widget parent, DtbGroupInfo *group_info, Boolean init);
static void align_top(Widget parent, DtbGroupInfo *group_info, Boolean init);
static void align_bottom(  
    Widget  
    DtbGroupInfo *group_info  
);  
  
static void align_hcenter(  
    Widget  
    DtbGroupInfo *group_info  
    Boolean init  
);  
  
/* Private functions used for finding paths */  
static int determine_exe_dir(  
    char *argv0,  
    char *buf,  
    int bufSize  
);  
  
static int determine_exe_dir_from_argv(  
    char *argv0,  
    char *buf,  
    int bufSize  
);  
  
static int determine_exe_dir_from_path(  
    char *argv0,  
    char *buf,  
    int bufSize  
);  
  
static Boolean path_is_executable(  
    char *path,  
    uid_t euid,  
    gid_t egid  
);  
  
static void dtb_popup_menu(  
    Widget widget,  
    XEvent *event,  
    String *params,  
    Cardinal *num_params  
);  
  
static void dtb_popup_menu_destroyCB(  
    Widget widget,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
static int dtb_drag_terminate(DtbDragSite dragSite);  
  
static void dtb_drag_button1_motion_handler(  
    Widget dragInitiator,  
    XtPointer clientData,  
    XEvent *event  
);  
  
static void dtb_drag_button2_event_handler(  
    Widget dragInitiator,  
    XtPointer clientData,  
    XEvent *event  
);  
  
static int dtb_drag_start(DtbDragSite dragSite, XEvent *event);  
  
static void dtb_drag_convertCB(  
    Widget dragContext,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
static void dtb_drag_to_rootCB(  
    Widget dragContext,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
static void dtb_drag_finishCB(  
    Widget dragContext,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
static void dtb_drop_animateCB(  
    Widget dragContext,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
static void dtb_drop_transferCB(  
    Widget dragContext,  
    XtPointer clientData,  
    XtPointer callData  
);  
  
/* Variable for storing client session save callback */
static DtbClientSessionSaveCB  dtb_client_session_saveCB = NULL;

/*
 * Variable for storing top level widget
 */
static Widget  dtb_project_toplevel_widget = (Widget)NULL;

/*
 * Variable for storing command used to invoke application
 */
static char *  dtb_save_command_str = (char *)NULL;

/*
 * Variables that keep track of which menus go with which widgets
 */
static DkbMenuRef  popupMenus = NULL;
static int  numPopupMenus = 0;
#include <Dt/Dnd.h>
#define DRAG_THRESHOLD 4
static Boolean  dragInProgress = False;
static int  dragInitialX = -1;
static rnt  dragInitialY = -1;

/*
 * Directory where the binary for this process where loaded from
 */
static char *  dtb_exe_dir = (char *)NULL;

/* Application Builder utility funs */

/*
 * Create/load a Pixmap given an XPM or Bitmap files
 * NOTE: this allocates a server Pixmap; it is the responsibility
 * of the caller to free the Pixmap
 */

int dtb_cv_file_toPixmap()
{
    String  fileName, widget, pixmap;
    Pixmap *pixmapReturnPtr;

#define pixmapReturn (*pixmapReturnPtr)

   Pixmap pixmap = NULL;
    Screen *screen = NULL;
    Pixel fgPixel = 0;
    Pixel bgPixel = 0;
    char image_path[MAXPATHLEN+1];
    Boolean pixmap_found = False;

    /* Get default values */
    screen = XtScreenOfObject(widget);
    fgPixel = WhitePixelOfScreen(screen);
    bgPixel = BlackPixelOfScreen(screen);

    /* Get proper colors for widget */
    XtVaGetValues(widget, XmNforeground, &fgPixel,
                  XmNbackground, bgPixel,
                  NULL);

    /* In CDE, XmGetPixmap handles .xpm files, as well. */
    if ((pixmap_found))
    {
        pixmap = XmGetPixmap(screen, imageName, fgPixel, bgPixel);
    } else
    { (pixmap = XmUNSPECIFIED_PIXMAP); }

    pixmap_found = ((pixmap != NULL) && (pixmap != XmUNSPECIFIED_PIXMAP));
    if (!pixmap_found)
    {
        sprintf(image_path, "%s/%s", dtb_get_exe_dir(), imageName);
        pixmap = XmGetPixmap(screen, image_path, fgPixel, bgPixel);
    } else
    { (pixmap = XmUNSPECIFIED_PIXMAP); }

    pixmap_found = ((pixmap != NULL) && (pixmap != XmUNSPECIFIED_PIXMAP));
    if (!pixmap_found)
    {
        sprintf(image_path, "%s/bitmaps/%s", dtb_get_exe_dir(), imageName);
        pixmap = XmGetPixmap(screen, image_path, fgPixel, bgPixel);
    } else
    { (pixmap = XmUNSPECIFIED_PIXMAP); }

    if (!pixmap_found)
    {
        return -1;
    }

    pixmapReturn = pixmap;
    return 0;
}
}
Sets both the sensitive and insensitive pixmaps

```c
int dtb_set_label_from_bitmap_data(
    Widget widget,
    int width,
    int height,
    unsigned char *bitmapData)
{
    Display *display = NULL;
    Screen *Screen = NULL;
    Drawable window = NULL;
    long bgPixel = 0;
    long fgPixel = 0;
    unsigned int depth = 0;
    Pixmap labelpixmap = NULL;

    if ((widget == NULL) || (width < 1) || (height < 1) || (bitmapData == NULL))
        return -1;

    /* Get a whole slew of information X needs */
    Pixel widgetBg = 0;
    Pixel widgetFg = 0;
    int widgetDepth = 0;

    display = XDisplay(widget);
    screen = XtScreen(widget);
    window = XWindow(widget);
    if (window == NULL)
        /* Widget has not been realized, yet */
        window = RootWindowOfScreen(screen);

    XtVaGetValues(widget == XmGadgetClass ? XtParent(widget) : widget,
        XmNbackground, &widgetBg,
        XmNforeground, &widgetFg,
        XmNdepth, &widgetDepth,
        &bgPixel, &fgPixel, &depth);  // Get background, foreground and depth

    /* Create the pixmap */
    labelpixmap = XCreatePixmapFromBitmapData(
        display, window,
        (char *) bitmapData,
        width, height,
        fgPixel, bgPixel, depth);

    if (labelpixmap == NULL)
        return -1;

    dtb_set_label_pixmaps(widget, labelpixmap, NULL);
    return 0;
}
```

Sets the label and insensitive label pixmaps of the widget.
* If either (or both) pixmap is NULL, it is ignored.

```c
int dtb_set_label_pixmaps(
    Widget widget,
    Pixmap labelPixmap,
    Pixmap labelInsensitivePixmap)
{
    if ((widget == NULL) || (labelPixmap == NULL) || labelInsensitivePixmap == NULL))
        return -1;

    /* Set the appropriate resources. */
    XtVaSetValues(widget, XmNlabelPixmap, labelPixmap, NULL);
    if (labelInsensitivePixmap != NULL)
        XtVaSetValues(widget, XmNInsensitivePixmap, labelInsensitivePixmap, NULL);
    return 0;
}
```
XtVaSetValues(widget, XmNlabelPixmap, labelPixmap, NULL);
}
if (labelInsensitivePixmap != NULL)
{
    XtVaSetValues(widget, XmNLabelInsensitivePixmap, labelInsensitivePixmap, NULL);
}
return 0;

/* Returns True if the fileName has the extension */

Boolean dtb_file_has_extension(
    String fileName,
    String extension
)
{
    Boolean hasExt = False;
    if (extension == NULL)
    {
        hasExt = ( (fileName == NULL) || (strlen(fileName) == 0) );
    }
    else
    {
        if (fileName == NULL)
            hasExt = False;
        else
        {
            char *dotPtr = strchr(fileName, '.');
            if (dotPtr == NULL)
                hasExt = False;
            else if (strcmp(dotPtr+1, extension) == 0)
                hasExt = True;
        }
    }
    return hasExt;
}

/* Appends the extension to fileBase and attempts to load in the Pixmap */

int dtb_cvtp_filebase_to_pixmap(
    Widget widget,
    String fileBase,
    String extension,
    Pixmap *pixmap)
{
    char fileName[512];
    int rc = 0;
    strcpy(fileName, fileBase);
    strcat(fileName, extension);
    rc = dtb_cvtp_file_to_pixmap(fileName, widget, pixmap);
    return rc;
}

int dtb_cvtp_image_file_to_pixmap(
    Widget widget,
    String fileName,
    Pixmap *pixmap)
{
    int rc = 0; /* return code */
    Pixmap tmpPixmap = NULL; /* return code */
    if (dtb_file_has_extension(fileName, "pm") ||
        dtb_file_has_extension(fileName, "xpm") ||
        dtb_file_has_extension(fileName, "bm") ||
        dtb_file_has_extension(fileName, "xbm"))
    {
        /* If explicit filename requested, use it directly */
        rc = dtb_cvtp_file_to_pixmap(fileName, widget, &tmpPixmap);
    }
    else /* Append extensions to locate best graphic match */
    {
        XtVaSetValues(XtIsSubclass(widget, XmGadgetClass)? XtParent(widget) :
            widget, XmNdepth, addepth, NULL);
        if (depth > 1) /* Look for Color Graphics First */
        {
            rc = dtb_cvtp_filebase_to_pixmap(widget, fileName, ".pm", &tmpPixmap);
            if (rc < 0)
            {
                rc = dtb_cvtp_filebase_to_pixmap(widget, fileName, ".xpm", &tmpPixmap);
                if (rc < 0)
                {
                    rc = dtb_cvtp_filebase_to_pixmap(widget, fileName, ".bm", &tmpPixmap);
                    if (rc < 0)
                    {
                        rc = dtb_cvtp_filebase_to_pixmap(widget, fileName, ".xbm", &tmpPixmap);
                    }
                }
            }
        }
        else /* Look for Monochrome First */
    }
Irc - dtb-cvt-filebase-to-pixmap(widget, fileName, "bm", &tmpPixmap);
if (rc < 0)
  irc - dtb-cvt-filebase-to-pixmap(widget, fileName, "xbm", &tmpPixmap):
if (rc < 0)
  irc - dtb-cvt-filebase-to-pixmap(widget, fileName, "pm", &tmpPixmap):
if (rc < 0)
  irc - dtb-cvt-filebase-to-pixmap(widget, fileName, "xpm", &tmpPixmap):
}

if (rc < 0)
  *pixmap = NULL;
  return rc;
*pixmap = tmpPixmap;
return 0;

/* Sets the XmNlabel from the image file [either xbitmap or x pixmap format].
* returns negative on error.
*/
int dtb_set_label_from_image_file(
  Widget widget,
  String fileName)
{
  int rc = 0;  /* return code */
 Pixmap labelPixmap = NULL;
  Pixmap insensitivePixmap = NULL;
  int depth;
  rc = dtb_cvI_image_file_toPixmap(widget, fileName, &labelPixmap);
  if (rc < 0)
  {
    return rc;
  }
  insensitivePixmap = dtb_create_greyedPixmap(widget, labelPixmap);
  rc = dtb_set_label_pixmap(widget, labelPixmap, insensitivePixmap);
  if (rc < 0)
  {
    return rc;
  }
  return 0;
}

unsigned long
dtb_cvt_resource_from_string(
  Widget parent,
  String res_type,
  unsigned int size_of_type,
  String res_str_value,
  unsigned long error_value)
{
  unsigned long cvt_value_return = error_value;
  unsigned char cvt_value1 = 0;
  unsigned short cvt_value2 = 0;
  unsigned int cvt_value3 = 0;
  unsigned long cvt_value4 = 0;
  XtPointer cvt_value_ptr = NULL;
  int which_cvt_value = -1;
  XrmValue source;
  XrmValue dest;
  if (size_of_type > sizeof(cvt_value_return))
  {
    /* Type we are converting to is too large */
    return cvt_value_return;
  }
  /* Get a data object of the appropriate size */
  if (size_of_type == sizeof(cvt_value1))
  {
    which_cvt_value = 1;
    cvt_value_ptr = (XtPointer)&cvt_value1;
  }
  else if (size_of_type == sizeof(cvt_value2))
  {
    which_cvt_value = 2;
    cvt_value_ptr = (XtPointer)&cvt_value2;
  }
  else if (size_of_type == sizeof(cvt_value3))
  {
    which_cvt_value = 3;
    cvt_value_ptr = (XtPointer)&cvt_value3;
  }
  else if (size_of_type == sizeof(cvt_value4))
  {
    which_cvt_value = 4;
  }
/* Actually do the conversion */
source.size = strlen(res_str_value) + 1;
src_addr = res_str_value;
dest.size = size_of_type;
dest_addr = (char *)cvt_value_ptr;
if (XtConvertAndStore(parent, XtRString, &source, res_type, &dest) != 0)
{
    switch (which_cvt_value)
    {
    case 1:
        cvt_value_return = (unsigned long)cvt_value1;
        break;
    case 2:
        cvt_value_return = (unsigned long)cvt_value2;
        break;
    case 3:
        cvt_value_return = (unsigned long)cvt_value3;
        break;
    case 4:
        cvt_value_return = (unsigned long)cvt_value4;
        break;
    }
    return cvt_value_return;
}

/* For a given pixmap, create a 50% greyed version. Most likely this will be used where the source pixmap is the labelPixmap for a widget and an insensitivePixmap is needed so the widget will look right when it is "not sensitive" ("grayed out" or "inactive"). */

NOTE: This routine creates a Pixmap, which is an X server resource. The created pixmap must be freed by the caller when it is no longer needed.

Pixmap
dtb_create_greyedPixmap(
    Widget    widget,
    Pixmap    pixmap
    )
{
    Display   *dpy;
    Window     root;
    Pixmap     insensitivePixmap;
    Pixel      background;
    unsigned int width, height, depth, bw;
    int        x, y;
    XGCValues  gcv;
    GC          gc;
    dpy = XtDisplayOfObject(widget);
    if(pixmap == XmUNSPECIFIED_PIXMAP || pixmap == (Pixmap)NULL) { return((Pixmap)NULL); }
    XtVaGetValues(widget, XmBackground, &background, XmNoBackground, &background, NULL);
    /* Get width/height of source pixmap */
    if (!XGetGeometry(dpy, pixmap, &root, &x, &y, &width, &height, &depth)) { return((Pixmap)NULL); }
    gc.foreground = background;
    gc.fill_style = FillStippled;
    gc.stipple = XmGetPixmapByDepth(XtScreenOfObject(widget), "50_foreground", 1.0, 1);
    gc = XtGetGC(widget, gc, &gcv);
    /* Create insensitive pixmap */
    insensitivePixmap = XCreatePixmap(dpy, pixmap, width, height, depth);
    XCopyArea(dpy, pixmap, insensitivePixmap, gc, 0, 0, width, height, 0, 0);
    XFillRectangle(dpy, insensitivePixmap, gc, 0, 0, width, height);
    XPutImage(dpy, insensitivePixmap, gc, 0, 0, &gcv);
    return((Pixmap)NULL);
}  

73
/* ** Routines to save and access the toplevel widget for an application. ** This is useful in dtb convenience functions, and also probably by ** developers in routines they provide in their _stubs.c files. ** static Widget dtb_project_toplevel_widget = (Widget) NULL; */

void dtb_save_toplevel_widget(
    Widget toplevel
) {
    dtb_project_toplevel_widget = toplevel;
}

Widget dtb_get_toplevel_widget()
    return(dtb_project_toplevel_widget);

/* ** Function to turn off traversal on the invisible sash within a ** PanedWindow. This is primarily used for the PanedWindow within ** a Custom Dialog object. */

void dtb_remove_sash_focus(
    Widget widget
) {
    WidgetList children;
    int numChildren, i;
    if (widget != NULL && XtIsSubclass(widget, xmPanedWindowWidgetClass))
        return;
    XtVaGetValues(widget, XmNchildren, &children, XmNnumChildren, &numChildren, NULL);
    for(i = 0; i < numChildren; i++)
        if (XtIsSubclass(children[i], xmSashWidgetClass))
            XtVaSetValues(children[i], XmNtraversalOn, False, NULL);
}

/* ** Routines to save and access the command used to invoke the application. ** */

void dtb_save_command(
    char *argv0
) {
    char exe_dir[MAXPATHLEN+1];
    dtb_save_command_str = argv0;
    /* * Save the path to the executable */
    if (determine_exe_dir(argv0, exe_dir, MAXPATHLEN+1) >= 0)
        dtb_exe_dir = (char *)malloc(strlen(exe_dir)+1);
        if (dtb_exe_dir != NULL)
            strcpy(dtb_exe_dir, exe_dir);
    return;
}

char * dtb_get_command();
    return(dtb_save_command_str);

/* ** Generic callback function to be attached as XmHelpCallback and ** provide support for on-object and Help-key help. The help text to ** be displayed is provided via a specialized data structure passed in ** as client data. */

void dtb_help_dispatch(
    Widget widget, 
    XtPointer clientData, 
    XtPointer callData 
) {
    DtbObjectHelpData help_data = (DtbObjectHelpData)clientData;
    int Arg;
    char help[] = buffer[100];
    Widget back_button;
    static Widget Quick_help_dialog = (Widget)NULL;

74
static Widget MoreButton;

/*
 ** In order to save the more-help info (help volume & location ID) as part
 ** of the quick help dialog's backtrack mechanism, we have to splice the
 ** volume & ID strings together and save them as the help volume field.
 ** If there isn't supplemental help information, we save a null string.
 ** Checking the status of the more-help info also lets us decide whether
 ** the 'More...' button should be enabled on the dialog.
 */
if( help_data->help_volume == 0 || *(help_data->help_volume) == 'N' || help_data->help_locationID == 0 || *(help_data->help_locationID) == 'N' ){
  buffer[0] = '\0';
} else {
  sprintf(buffer, "%s/%s", help_data->help_volume, help_data->help_locationID);
}
/*
 ** If this is our first time to post help, create the proper dialog and
 ** set its attributes to suit the current object. If not, then just
 ** update the attributes.
 ** (You have to be careful about gratuitous SetValues on the dialog because
 ** its internal stack mechanism takes repeated settings as separate items
 ** and updates the stack for each.)
 */
if(Quick-help-dialog == (Widget)NULL) {
  /* Create shared help dialog */
  1 = 0;
  XtSetArg(wargs[i], XmNtitle, "Application Help"); i++;
  XtSetArg(wargs[i], XmNhelpType, DtHELP-TYPE-DYNAMIC_STRING); i++;
  XtSetArg(wargs[i], XmNhelpVolume, buffer); i++;
  Quick_help_dialog = DtCreateHelpQuickDialog(db-get-toplevel-widget(),
              "Help", wargs, i);
  /* Fetch out the Dialog's More button child and hook the 'more help'
 ** handler to its activateCallback. Set its current status to
 ** indicate whether this object has supplemental help data.
 **/ MoreButton = DtHelpQuickDialogGetChild(Quick_help_dialog,
                                                  DtHELP-QUICK_MORE_BUTTON);
  XtManageChild(MoreButton);
  XtAddCallback(MoreButton, XmNactivateCallback, dtb_more_help_dispatch,
                (XtPointer)Quick_help_dialog);
  if(buffer[0] == '\0') XtSetSensitive(MoreButton, False);
  /*
 ** Fetch out the Dialog's Backtrack button child and hook a callback
 ** that will control button sensitivity based on the presence of more
 ** help data.
 **/ back_button = DtHelpQuickDialogGetChild(Quick_help_dialog,
                                                DtHELP-QUICK_BACK_BUTTON);
  XtAddCallback(back_button, XmNactivateCallback, dtb-help-back-hdlr,
                (XtPointer)Quick_help_dialog);
  /* Otherwise the dialog already exists so we just set the attributes. */
  else {
    /*
    ** If we have supplemental help info, enable the more button.
    ** Also save this info for later use in the backtrack handler.
    */
    if(buffer[0] == '\0') {
      XtSetSensitive(MoreButton, False);
    } else {
      XtSetSensitive(MoreButton, True);
    }
    XtVaSetValues(Quick_help_dialog, 
                  XmNhelpType, DtHELP-TYPE-DYNAMIC_STRING, 
                  DtNhelpVolume, buffer, 
                  DtNstringData, help_data->help_text, 
                  NULL);
  }
  /* Now display the help dialog */
  XtManageChild(Quick_help_dialog);
}
/*
 ** This callback is invoked when the user presses 'More...' on the
 ** QuickHelpDialog. It figures out whether a help volume entry is associated
 ** with the displayed help text, and if so it brings up a GeneralHelpDialog
 ** to display the appropriate help volume information.
 */
void dtb_more_help_dispatch(
  Widget widget,
  XtPointer clientData,
  XtPointer callData
){
  int i;
  Arg wargs[10];
  String buffer, vol, loc;
/* Fetch the saved volume/locationID information from the dialog widget */
XtVaGetValues(help.Dialog,
  DtNhelpVolume,dbufuffer,
  DtNstringData,dtext,
  NULL);

/* ** Parse the combined volume/locationID string. If that fails there ** must be no data, so don’t bother displaying the GeneralHelpDialog. ** (We shouldn’t be in this callback routine if that happens, though...)
*/
if( (cp=strchr(buffer,'/')) != (char *)NULL) {
  *cp+ = 0;
  vol = buffer;
  loc = cp;
}

if(GeneralHelpDialog == (Widget)NULL) {
  /* Create General Help Dialog */
  i = 0;
  XtSetArg(wargs[i],XmNtitle, "Application Help"); i++;
  XtSetArg(wargs[i],DtNhelpType, DtHELP-TYPE-TOPIC); i++;
  XtSetArg(wargs[i],DtNhelpVolume, vol); i++;
  XtSetArg(wargs[i],DtNlocationId, loc); i++;
  GeneralHelpDialog = XmCreateHelpDialog(dtb-qet-toplevel-widget(),
    "GeneralHelp",wargs, i);
}
else {
  i = 0;
  XtSetArg(wargs[i],DtNhelpType, DtHELP-TYPE-TOPIC); i++;
  XtSetArg(wargs[i],DtNhelpVolume, vol); i++;
  XtSetArg(wargs[i],DtNlocationId, loc); i++;
  XtSetValueS (GeneralHelpDialog, wargs, i);
  i++;
  i++;
  i++;
}

/* Now take down the quick help dialog and display the full help one */
XtManageChild(GeneralHelpDialog);
XtUnmanageChild(help-dialog);

/** Callback that is added to the QuickHelpDialog widget’s "Backtrack" button ** and is used to control the "More..." button. At each step in the backtrack, ** this routine checks to see if there is help volume & location info stored ** in the dialog’s helpvolume resource. If so, then the "More..." button is ** enabled. If not, then it is disabled. */

void
dtb-help-back-hdlr(Widget widget, ClientData callData{
  char *cp;
  /* Fetch the saved volume/locationID information from the dialog widget */
  XtVaGetValues(help.Dialog,
    DtNhelpVolume,dbufuffer,
    DtNstringData,dtext,
    NULL);

  /* Get a handle to the "More..." button */
  more_button = XmGetDialogContentChildId(help Dialog, 
    DHELP_QUICK_MORE_BUTTON);

  /* Parse the combined volume/locationID string. Disable the "More..." ** button if there isn’t any help info, and enable it if there is. 
*/
  if( (buffer == 0) || (buffer == NULL) ) { 
    (cp=strchr(buffer,'/')) == (char *)NULL) {
      XtSetSensitive(more_button,False);
    }
    else {
      XtSetSensitive(more_button,True);
    }
  }

  /* Utility function used to provide support for on-item help. ** It is typically invoked via a callback on the "On Item" item in the 
** main menuBar’s "Help" menu. */

void
dtb-do-onitem-help(Widget target;

  /* Call the DHelp routine that supports interactive on-item help. */
  if(DHelpReturnSelectedWidgetId(dtb-qet-toplevel-widget(),
    (Cursor)NULL,target) != DHELP_SELECT_VALID) return;
** Starting at the target widget, wander up the widget tree looking
** for one that has an XmNhelpCallback, and call the first one we
** find.
*/

while(target != (Wid9et)NULL)
{
    if( XtHasCallbacks(target,XmNhelpCallback) == XtCallbackHasSome)
        XtCallCallbacks(target,XmNhelpCallback,(XtPointer)NULL);
        return;
    else
        target = XtParent(target);
}

return;

/**
** Utility function called to display help volume information.
** It needs the name of the help volume and the location ID (both as
** strings) so it can configure the full help dialog widget properly.
*/

int dtb_show_help_volume_info(char *volume_name, char *location_id)
{
    int i;
    Arg wargs[10];
    static Widget GeneralHelpDialog = (Widget) NULL;
    if(Genera1HelpDialog == (Wid9et)NULL) {
        /* Create General Help Dialog */
        i = 0;
        XtSetArg(wargs[i], XmNtitle, "Application Help"); i++;
        XtSetArg(wargs[i], XmNhelpType, XmHELP_TYPE_TOPIC); i++;
        XtSetArg(wargs[i], XmNhelpVolume, volume_name); i++;
        XtSetArg(wargs[i], XmNlocationId, location_id); i++;
        GeneralHelpDialog = XtCreateHelpDialog(dtb_get_toplevel_widget(),
                                              "GeneralHelp", wargs, i);
    } else {
        i = 0;
        XtSetArg(wargs[i], XmNhelpType, XmHELP_TYPE_TOPIC); i++;
        XtSetArg(wargs[i], XmNhelpVolume, volume_name); i++;
        XtSetArg(wargs[i], XmNlocationId, location_id); i++;
        XtSetValues(GeneralHelpDialog, wargs, i);
    }

    /* Now display the full help dialog */
    XtManageChild(GeneralHelpDialog);
    return(0);
}

/**
** dtb_call_help_callback()
** Utility routine to call the help callbacks on a target widget. This
** is predominantly used to display help data on a dialog by having this
** function as the activate callback on the dialog's help button.
*/

void dtb_call_help_callback(Widget widget, XtPointer clientData, XtPointer callData)
{
    Widget target = (Widget) clientData;
    XtCallCallbacks(target, XmNhelpCallback, (XtPointer)NULL);
}

/**
** dtb_session_save()
** Callback that is called when the application (top level
** widget of application) gets a WM_SAVE_YOURSELF ClientMessage
** This callback will call the client/application's session
** save callback.
*/

void dtb_session_save(Widget widget, XtPointer clientData, XtPointer callData)
{
    int new_argc, client_argc = 0, new_argc_counter, i;
    char **new_argv;
    **new_argv = NULL;
    *session_file_path, *session_file_name,
fapp-name = NULL;
Boolean status = False;
DrbClientSessionSaveCB session-SaveCB:

/* Return if no widget passed in. */
if (!widget)
    return;

/* Get session file path/name to store application’s state */
if (DtSessionSavePath(widget, &session_file_path, &session_file_name) == False)
    return;

/* Get client session save callback */
session-saveCB = dtb_get_client_session_saveCB();

/* Call client session save callback */
if (session-saveCB)
    /* client_argv and client_argc are the variables that */
    /* will contain any extra command line options */
    /* that need to be used when invoking the application */
    /* to bring it to the current state. */
    status = session-saveCB(widget, session_file_path,
                            client_argv, client_argc);

    /* Generate the reinvoking command and add it as the property value */
    /* Fetch command used to invoke application */
    app_name = dtb_get_command();

    /* new_argc and new_argc are the variables used to reconstruct */
    /* the command to re-invoking the application */

    /* Start new_argc with: */
    /* 1 for argv[0], normally the application */
    /* client_argc any extra command line options as */
    /* returned from client session save */
    /* callback */
    new_argc = 1 + client_argc;

    /* If the status returned from session save callback is 'True', */
    /* the session file was actually used. This means we need to */
    /* add: */
    /* -session <session file name> */
    /* to the command saved, which is 2 more strings. */
    if (status)
        new_argc += 2;

    /* Allocate vector */
    new_argv = (char **) Xmalloc(sizeof(char **) * new_argc);

    /* Set new_argv[] to be the application name */
    new_argc_counter = 0;
    new_argv[new_argc_counter] = app_name;
    new_argc_counter++;

    /* Proceed to copy every command line option from */
    /* client_argv. Skip -session, if found. */
    for (i = 0; i < client_argc;)
        if (strcmp(client_argv[i], "-session"))
            new_argv[new_argc_counter] = client_argv[i];
            new_argc_counter++;
        else
            /* Skip -session */
            /* The next increment below will skip the session file. */
            i++;
        i;

    **i;
/* If session file used, add */
if (status)
{
    new_argv[new_argc_counter] = "-session";
    new_argc_counter++;
    new_argv[new_argc_counter] = session_file_name;
}
else
{
    /* otherwise, destroy session file */
    (void)unlink(session_file_path);
}
/* set WM_COMMAND property with vector constructed */
XSetCommand(XtDisplay(widget), XtWindow(widget),
             new_argv, new_argc);
/* Free argument vector */
XtFree ((char *)new_argv);
/* CDE Sessioning API states that the path/name 
* strings have to be free'd by the application. */
XtFree ((char *)session_file_path);
XtFree ((char *)session_file_name);

/* dtb_get_client_session_saveCB(); */
DtbClientSessionSaveCB
dtb_get_client_session_saveCB()
{
    return(dtb_client_session_saveCB);
}

/* This function will center all the passed form's children. 
* The type of centering depends on what 'type' is. */
void
dtb_children_center(
    Widget form,
    DTB_CENTERING_TYPES type
)
{
    WidgetList children_list;
    int i, num_children;
    if (!form || (type == DTB_CENTER_NONE))
        return;
    /* Get children list */
    XtVaGetValues(form, 
                   XmNnumChildren, &num_children, 
                   XmNchildren, &children_list, NULL);
    /* Center all children */
    for (i=0; i < num_children; ++i)
        dtb_center(children_list[i], type);
}

/* This function 'uncenters' the children of the passed 
* form widget. */
void
dtb_children_uncenter(
    Widget form,
    DTB_CENTERING_TYPES type
)
{
    WidgetList children_list;
    int i, num_children;
    if (!form || (type == DTB_CENTER_NONE))
/* Get children list */
XtVaGetValues(form,
XmNnumChildren, &num_children,
XmNchildren, &children_list,
NULL);

/* Center all children */
for (i=0; i < num_children; +1) {
    dtb_uncenter(children_list[i], type);
}

/* This function centers the passed widget.
* This is done by setting the proper offsets.
* Dynamic centering is accomplished by attaching an event handler
* which detects resizes and recomputes and sets the appropriate offset.
*/
void
dtb_center(
    Widget form_child,
    DTB_CENTERING_TYPES type)
{
    if (!form_child || (type == DTB_CENTER_NONE))
        return;
    center_widget(form_child, type);
    XtAddEventHandler(form_child,
        StructureNotifyMask, False,
        centering_handler, (XtPointer)type);
}

/* This function 'uncenters' the passed widget.
* This involves resetting the attachment offsets
* and removing the resize event handler.
*/
void
dtb_uncenter(
    Widget form_child,
    DTB_CENTERING_TYPES type)
{
    if (!form_child || (type == DTB_CENTER_NONE))
        return;
    uncenter_widget(form_child, type);
    XtRemoveEventHandler(form_child,
        StructureNotifyMask, False,
        centering_handler, (XtPointer)type);
}

/* This function centers the passed widget.
* This is done by making the appropriate offset equal
* to the negative half of its width/height (depending
* on whether horizontal or vertical centering was chosen.
*/
static void
center_widget(
    Widget form_child,
    DTB_CENTERING_TYPES type)
{
    Widget parent;
    Dimension width = 0,
    height = 0;
    int center_offset;
    unsigned char left_attach = XmATTACH_NONE,
    top_attach = XmATTACH_NONE;
    if (!form_child || !XtIsManaged(form_child) || !XtIsRealized(form_child))
        return;
    parent = XtParent(form_child);
    if (!parent || !XtIsClassInstance(parent, XmFormWidgetClass))
        return;
    XtVaGetValues(form_child,
        XmNwidth, &width,
        XmNheight, &height,
        XmNleftAttachment, &left_attach,
        XmNtopAttachment, &top_attach,
        NULL);
    switch (type) {
        case DTB_CENTER_POSITION_VERT:
            /* This function centers the passed widget.
             * This is done by making the appropriate offset equal
             * to the negative half of its width/height (depending
             * on whether horizontal or vertical centering was chosen.
             */
            break;
        case DTB_CENTER_POSITION_HORIZ:
            /* This function centers the passed widget.
             * This is done by making the appropriate offset equal
             * to the negative half of its width/height (depending
             * on whether horizontal or vertical centering was chosen.
             */
            break;
        case DTB_CENTER_NONE:
            /* This function 'uncenters' the passed widget.
             * This involves resetting the attachment offsets
             * and removing the resize event handler.
             */
            break;
    }
    if (left_attach == XmATTACH_NONE)
        left_attach = XmATTACH_LEFT;
    if (top_attach == XmATTACH_NONE)
        top_attach = XmATTACH_TOP;
    center_offset = (width - left_attach) / 2;
    if (left_attach == XmATTACH_LEFT)
        center_offset = width / 2;
    if (top_attach == XmATTACH_TOP)
        center_offset = height / 2;
    XtVaSetValues(form_child,
        XmNleftAttachment, left_attach,
        XmNtopAttachment, top_attach,
        NULL);
    return;
}

/* This function centers the passed widget.
* This is done by making the appropriate offset equal
* to the negative half of it's width/height (depending
* on whether horizontal or vertical centering was chosen.
*/
static void
center_widget(
    Widget form_child,
    DTB_CENTERING_TYPES type)
{
    Widget parent;
    Dimension width = 0,
    height = 0;
    int center_offset;
    unsigned char left_attach = XmATTACH_NONE,
    top_attach = XmATTACH_NONE;
    if (!form_child || !XtIsManaged(form_child) || !XtIsRealized(form_child))
        return;
    parent = XtParent(form_child);
    if (!parent || !XtIsClassInstance(parent, XmFormWidgetClass))
        return;
    XtVaGetValues(form_child,
        XmNwidth, &width,
        XmNheight, &height,
        XmNleftAttachment, &left_attach,
        XmNtopAttachment, &top_attach,
        NULL);
    switch (type) {
        case DTB_CENTER_POSITION_VERT:
            /* This function centers the passed widget.
             * This is done by making the appropriate offset equal
             * to the negative half of its width/height (depending
             * on whether horizontal or vertical centering was chosen.
             */
            break;
        case DTB_CENTER_POSITION_HORIZ:
            /* This function centers the passed widget.
             * This is done by making the appropriate offset equal
             * to the negative half of it's width/height (depending
             * on whether horizontal or vertical centering was chosen.
             */
            break;
        case DTB_CENTER_NONE:
            /* This function 'uncenters' the passed widget.
             * This involves resetting the attachment offsets
             * and removing the resize event handler.
             */
            break;
    }
    if (left_attach == XmATTACH_NONE)
        left_attach = XmATTACH_LEFT;
    if (top_attach == XmATTACH_NONE)
        top_attach = XmATTACH_TOP;
    center_offset = (width - left_attach) / 2;
    if (left_attach == XmATTACH_LEFT)
        center_offset = width / 2;
    if (top_attach == XmATTACH_TOP)
        center_offset = height / 2;
    XtVaSetValues(form_child,
        XmNleftAttachment, left_attach,
        XmNtopAttachment, top_attach,
        NULL);
    return;
}
if (left_attach == XmATTACH_POSITION)
    return;
center_offset = -(width/2);
 XtVaSetValues(form_child, 
    XmNleftOffset, center_offset, 
    NULL);
break;
case DTB_CENTER_POSITION_HORIZ:
    if (top_attach == XmATTACH_POSITION)
        return;
    center_offset = -(height/2);
    XtVaSetValues(form_child, 
        XmNtopOffset, center_offset, 
        NULL);
break;
case DTB_CENTER_POSITION_BOTH:
{
int left_offset, top_offset;
if (((left_attach == XmATTACH_POSITION) &&
    (top_attach == XmATTACH_POSITION))
    return;
    left_offset = -(width/2);
    top_offset = -(height/2);
    XtVaSetValues(form_child, 
        XmNleftOffset, left_offset, 
        XmNtopOffset, top_offset, 
        NULL);
    /*
    * This function 'uncenters' the passed widget.
    * It merely resets the offsets of the top/left attachments to 0.
    */
    static void
uncenter_widget( 
    Widget form_child, 
    MB_CENTERING_TYPES type) 
{ 
    Widget parent;
    unsigned char left_attach = XMATTACH_NONE, 
        top_attach = XMATTACH_NONE;
    if (!form_child || !XtIsManaged(form_child) || !XtIsRealized(form_child))
        return;
    parent = XtParent(form_child);
    if (!parent || !XtIsSubclass(parent, xmFormWidgetClass))
        return;
    XtVaGetValues(form_child, 
        XmNleftAttachment, &left_attach, 
        XmNtopAttachment, &top_attach, 
        NULL);
    switch (type) {
    case DTB_CENTER_POSITION_VERT:
        if (left_attach != XMATTACH_POSITION)
            return;
        XtVaSetValues(form_child, 
            XmNleftOffset, 0, 
            NULL);
        break;
    case DTB_CENTER_POSITION_HORIZ:
        if (top_attach != XMATTACH_POSITION)
            return;
        XtVaSetValues(form_child, 
            XmNtopOffset, 0, 
            NULL);
        break;
    case DTB_CENTER_POSITION_BOTH:
        if (((left_attach != XMATTACH_POSITION) &&
            (top_attach != XMATTACH_POSITION))
            return;
Event handler to center a widget
The type of centering needed is passed in as client_data

/*
 * static void
centering-handler(
    Widget widget,
    XtPointer client_data,
    XEvent *event,
    Boolean *cont-dispatch
)
*/
XConfigureEvent *xcon = &event->xconfigure;
MB-CENTERING-TYPES = (DTB_CENTERING_TYPES)client_data:
if (event->type != ConfigureNotify) && (event->type != MapNotify)
    return;
    resized_child = XWindowToWidget(XIDisplay(widget), xcon->window);
if (!resized_child)
    return;
center_widget(resized_child, type);

/*
 * Given a widget, return it's label widget.
 */
static Widget
get_label_widget(
    Widget widget
)
    WidgetList children_list;
    Widget label_widget = NULL;
    int i;
    char *subobj_name = NULL, *label_name = NULL;
    char *underscore_ptr = NULL;
    if (XtIsSubclass(widget, xmLabelWidgetClass)) {
        num_children = 0;
        return(widget);
    }
    subobj_name = XtName(widget);
    label_name = (char *)XtMalloc(strlen(subobj_name) + strlen("label") + 5);
    label_name[0] = '*';
    strcpy(label_name, subobj_name);
    if ((underscore_ptr = strrchr(label_name, '_')) != NULL)
        strcpy(underscore_ptr, "Label");
    label_widget = XtNameToWidget(widget, label_name);
}
if (label_widget == NULL)
    if (label_widget = XtNameToWidget(widget, ".label"))
        XtFree((char *)label_name);
if (label_widget)
    return(label_widget);

/*
 * How to look for 1st child of group object ??
 * How do we know if 'widget' is a group object ??
 * For now, just check if it is a form
 */
if (XtIsSubclass(widget, xmFormWidgetClass) ||
    XtIsSubclass(widget, xmFrameWidgetClass))
    XtVaGetValues(widget,
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);
if (num_children > 0)
    return(get_label_widget(children_list[0]));
return(NULL);

/*
 * How to look for 1st child of group object ??
 * How do we know if 'widget' is a group object ??
 * For now, just check if it is a form
 */
if (XtIsSubclass(widget, xmFormWidgetClass) ||
    XtIsSubclass(widget, xmFrameWidgetClass))
    XtVaGetValues(widget,
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);
if (num_children > 0)
    return(get_label_widget(children_list[0]));
return(NULL);

static Position
get_offset_from_ancestor(
    Widget ancestor,

82
Widget W

if (!ancestor || !w || (w == ancestor))
    return (0);

XtVaGetValues(cur, XmNmax, offset, NULL);
cur_parent = XtParent(cur);
while (cur_parent != ancestor)
{
    Position tmp_offset = 0;
    XtVaGetValues(cur, XmNmax, &tmp_offset, NULL);
    offset += tmp_offset;
    cur_parent = XtParent(cur);
}
return (offset);

static Dimension get_label_width(
    Widget widget
)
{
    WidgetList children_list;
    Widget lbl_widget = NULL;
    Dimension lbl_width = 0;
    lbl_widget = get_label_widget(widget);
    if (lbl_widget)
    {
        Position offset = 0;
        XtVaGetValues(lbl_widget, XmNwidth, &lbl_width, NULL);
        offset = get_offset_from_ancestor(widget, lbl_widget);
        lbl_width += (Dimension)offset;
    }
    return (lbl_width);
}

static void get_widest_label(
    WidgetList list,
    int count,
    Widget *child_widget,
    Dimension *label_width
)
{
    Widget cur_widest = NULL;
    Dimension cur_width = 0;
    int i;
    for (i = 0; i < count; ++i)
    {
        Dimension tmp;
        tmp = get_widest_width(list[i]);
        if (tmp > cur_width)
        {
            cur_width = tmp;
            cur_widest = list[i];
        }
    }
    *child_widget = cur_widest;
    *label_width = cur_width;
}

static void get_widest_value(
    WidgetList list,
    int count,
    Widget *child_widget,
    Dimension *value_width
)
{
    Widget cur_widest = NULL;
    Dimension cur_width = 0;
    int i;
    for (i = 0; i < count; ++i)
    {
Dimension tmp, label_width, obj_width = 0;

label_width = get_label_width(list[1]);
XtVaGetValues(list[1], XmNwidth, &obj_width, NULL);

tmp = obj_width - label_width;

if (tmp > cur_width)
{
    cur_width = tmp;
    cur_widest = list[1];
}

*child_widget = cur_widest;
*value_width = cur_width;

static void
get_widget_rect(
    Widget widget,
    XRectangle *rect)
{
    if (!rect)
        return;

    XtVaGetValues(widget,
        XtNwidth, (XtArgVal)&(rect->width),
        XtNheight, (XtArgVal)&(rect->height),
        XtNx, (XtArgVal)&(rect->x),
        XtNy, (XtArgVal)&(rect->y),
        NULL);
}

static void
greatest_size(
    Widget *list,
    int count,
    int *width,
    Widget *tallest,
    Widget *widest)
{
    XRectangle w_rect;

    if (!list || (count < 0))
        return;

    get_widget_rect(list[0], &w_rect);
    *width = w_rect.width;
    previous_width = *width;
    *height = w_rect.height;
    previous_height = *height;

    if (tallest != NULL)
        *tallest = list[0];
    if (widest != NULL)
        *widest = list[0];

    for (i = 0; i < count; i++)
    {
        get_widget_rect(list[i], &w_rect);
        *width = max((int) w_rect.width, (int) *width);
        previous_width = *width;
        if (widest != NULL && *width > previous_width)
            *widest = list[i];
        *height = max((int) w_rect.height, (int)*height);
        if (tallest != NULL && *height > previous_height)
            *tallest = list[i];
    }
}

static void
group_maxsize(
    Widget parent,
    DtbGroupInfo *group_info,
    int *cell_width,
    int *cell_height)
{
    WidgetList children_list = NULL;
    int i,
    num_children = 0;

    /* Get children list */
    XtVaGetValues(parent,
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);

    if (num_children > 0)
    {
        get_greatest_size(&children_list, num_children, cell_width, tallest, widest);
        if (tallest != NULL)
            XtVaGetValues(tallest,
                XmNwidth, (XtArgVal)&(cell_width),
                XmNheight, (XtArgVal)&(cell_height),
                NULL);
        if (widest != NULL)
            XtVaGetValues(widest,
                XmNwidth, (XtArgVal)&(cell_width),
                XmNheight, (XtArgVal)&(cell_height),
                NULL);
    }
}
static void
get_greacest_size(children_list, num_children,
cell_width, cell_height,
(Widget *)NULL, (Widget *)NULL);}

static void
get_group_row_col(Widget
parent, DbGroupInfo *group_info,
int *rows, int *cols)

/**
XtVaGetValues(parent, XmNnumChildren, num_children,
XmNchildren, &children_list, NULL);

num_rows = group_info->num_rows;
um_cols = group_info->num_cols;
if ((num_rows <= 0) && (num_cols <= 0))
{ *rows = *cols = -1;
return;
}
*/

if (num-cols <= 0)
num_cols = (num_children/num_rows) + ((num_children % num_rows) ? 1 : 0);
if (num_rows <= 0)
num_rows = (num_children/num_cols) + ((num_children % num_cols) ? 1 : 0);
*rows = num_rows;
*cols = num_cols;

static Widget
get_group_child(Widget
parent, DbGroupInfo *group_info, int x_pos, int y_pos)

/**
XtVaGetValues(parent, XmNnumChildren, num_children,
XmNchildren, &children_list, NULL);

if (num_children <= 0)
return (NULL);
*/

switch (group_type)
{ case DTA_GROUP_NONE:
break;
case DTA_GROUP_ROWS:
/* * rows = 1
 * y_pos is ignored
*/

85
/*
1 = x_pos;
break;

case DTB_GROUP_COLUMNS:
    /*
    * num_columns = 1
    * x_pos is ignored
    */
    i = y_pos;
break;

case DTB_GROUP_ROWS_COLUMNS:
    if (num_rows <= num_columns)
        break;
    if (num_rows > 0)
    {
        /*
        ROWFIRST
        */
        if (y_pos < num_rows)
        {
            i = (x_pos * num_rows) + y_pos;
        }
        else
        {
            /* COLFIRST */
            if (x_pos < num_columns)
                i = x_pos * num_columns;
        }
    break;
    default:
        break;
    }

if ((i >= 0) && (i < num_children))
    {
    ret_child = children_list[i];
    }

return (ret_child);
}

void
group_info = (DtbGroupInfo *)XtMalloc(sizeof(DtbGroupInfo));
group_info->group_type = group_type;
group_info->row_align = row_align;
group_info->col_align = col_align;
group_info->margin = margin;
group_info->num_rows = num_rows;
group_info->num_cols = num_cols;
group_info->hoffset = hoffset;
group_info->voffset = voffset;
group_info->ref_widget = NULL;
align_children(parent, group_info, True);
/*
 * Register expose handler
 * Some group objects depend on it’s members’ sizes for their layout.
 * Unfortunately, some group members have invalid sizes prior to
 * XtRealize( ), so the group layout has to be recalculated after the
 * group is realized or exposed in this case, since there is no realize
 * callback.
 */
switch(group_info->group_type)
{
case DTB_GROUP_NONE:
*/

86
break;

case DTB_GROUP_ROWS:
    if (group_info->row_align == DTB_ALIGN_HCENTER)
        XtAddEventHandler(parent, ExposureMask, False,
            expose_handler, (XtPointer)group_info);
    break;

case DTB_GROUP_COLUMNS:
    if ((group_info->col_align == DTB_ALIGN_LABELS) ||
        (group_info->col_align == DTB_ALIGN_VCENTER))
        XtAddEventHandler(parent, ExposureMask, False,
            expose_handler, (XtPointer)group_info);
    break;

case DTB_GROUP_ROWS_COLUMNS:
    if ((group_info->row_align == DTB_ALIGN_HCENTER) ||
        (group_info->col_align == DTB_ALIGN_LABELS))
        XtAddEventHandler(parent, ExposureMask, False,
            expose_handler, (XtPointer)group_info);
    break;

    XtAddCallback(parent, XtNdestroyCallback,
        free_group_info, (XtPointer)group_info);

static void
align_children(   Widget parent,
    DtbGroupInfo *group_info,
    Boolean init
){
    if (!parent || !group_info)
        return;

    switch(group_info->group_type)
    {   case DTB_GROUP_NONE:
            break;

        case DTB_GROUP_ROWS:
            align_rows(parent, group_info, init);
            align_left(parent, group_info);
            break;

        case DTB_GROUP_COLUMNS:
            align_als(parent, group_info, init);
            align_top(parent, group_info);
            break;

        case DTB_GROUP_ROWSCOLUMNS:
            align_rows(parent, group_info, init);
            align_cols(parent, group_info, init);
            break;
    }
}

static void
align_handler(   Widget widget,
    XtPointer client_data,
    XEvent event,
    Boolean *event_dispatch
){
    DtbGroupInfo *group_info = (DtbGroupInfo *)client_data;
    WidgetList children_list;
    int i;
    num_children = 0;
    Boolean relayout_all = False;

    /*
    * Get children list
    */
    XtVaGetValues(widget,
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);

    if (num_children <= 0)
        return;

    XtRemoveEventHandler(widget,
        SubstructureNotifyMask, False,
        align_handler, (XtPointer)client_data);

    if (event->type == ConfigureNotify) {
        XConfigureEvent *xcon = event->xconfigure;
        Widget resized_child;
if (xcon->window != xcon->event)
    resized_child = XtWindowToWidget(XtDisplay(widget), xcon->window);

switch(group_info->group_type)
{
    case DTB_GROUP_NONE:
        break;
    case DTB_GROUP_ROWS:
        if (group_info->row_align == DTB_ALIGN_HCENTER)
            relayout_all = True;
        break;
    case DTB_GROUP_COLUMNS:
        if (group_info->col_align == DTB_ALIGN_HCENTER)
            relayout_all = True;
        break;
    case DTB_GROUP_ROWS_COLUMNS:
        if (group_info->row_align == DTB_ALIGN_HCENTER)
            relayout_all = True;
        break;
    case DTB_GROUP_NONE_COLUMNS:
        if (group_info->col_align == DTB_ALIGN_HCENTER)
            relayout_all = True;
        break;
}

/* Relayout when new widgets are created */
if (event->type == CreateNotify)
{
    XCreateWindowEvent *xcreate = &event->xcreatewindow;
    relayout_all = True;
}

/* Relayout when widgets are destroyed */
if (event->type == DestroyNotify)
{
    XDestroyWindowEvent *xdestroy = &event->xdestroywindow;
    Widget destroyed_child = XtWindowToWidget(XtDisplay(widget), xdestroy->window);
    relayout_all = True;

    if (relayout_all)
    {
        align_children(widget, group_info, False);
        XtAddEventHandler(widget, SubstructureNotifyMask, False, align_handler, (xtPointer)client_data);
    }
}

static void expose_handler()
{
    Widget widget,
    XtPointer client_data,
    *event, *
    Boolean *canDispatch 

    DtbGroupInfo *group_info = (DtbGroupInfo *)client_data;
    WidgetList children_list;
    int num_children = 0;
    Boolean relayout_all = False,
    register_align_handler = False;

    if (event->type != Expose)
        return;

    if (!group_info)
        return;

    /* Get children list */
    XtVaGetValues(widget, XmNnumChildren, &num_children, XmNchildren, children_list, NULL);

    if (num_children < 0)
        return;

    XtRemoveEventHandler(widget, ExposuresMask, False, expose_handler, (XtPointer)client_data);
switch(group_info->group_type)
{
    case DTB_GROUP_NONE:
        break;
    case DTB_GROUP_ROWS:
        if (group_info->row_align == DTB_ALIGN_HCENTER)
            { relayout_all = True;  
              register_align_handler = True;
            }
        break;
    case DTB_GROUP_COLUMNS:
        if (group_info->col_align == DTB_ALIGN_LABELS)
        { relayout_all = True;
          register_align_handler = True;
        }
        break;
    case DTB_GROUP_ROWSCOLUMNS:
        if (group_info->row_align == DTB_ALIGN_HCENTER)
        { relayout_all = True;
          register_align_handler = True;
        }
        break;
    case DTB_GROUP_NONE:
        if (relayout_all)
        {  if (register_align_handler)
            { align_children(widget, group_info, False);
              XtAddEventHandler(widget, SubstructureNotifyMask, False, align_handler, (XtPointer)group_info);
            }
        }
}

static void
free_group_info(
    Widget      widget,         
    XtPointer   client_data,    
    XtPointer   call_data)
{
    DtbGroupInfo *group_info = (DtbGroupInfo *)client_data;
    XtFree((char *)group_info);
}

static void
align_row(
    Widget      parent,  
    DtbGroupInfo *group_info,  
    Boolean     init)
{
    if (parent .group_info || (group_info->group_type == DTB_GROUP_COLUMNS))
    return;

    switch (group_info->row_align)
    {
    case DTB_ALIGN_TOP:
        align_top(parent, group_info);
        break;
    case DTB_ALIGN_HCENTER:
        align_hcenter(parent, group_info, init);
        break;
    case DTB_ALIGN_BOTTOM:
        align_bottom(parent, group_info);
        break;
    default:
        break;
    }
}

static void
align_cols(
    Widget      parent,  
    DtbGroupInfo *group_info,  
    Boolean     init)
DtbgpGroupInfo *group-info,
Boolean init

if (!parent || !group-info || (group-info->group-type == DTB_GROUP_ROWS))
    return;

switch (group-info->col_align)
{
    case DTB_ALIGN_LEFT:
        align_left(parent, group-info);
        break;
    case DTB_ALIGN_LABELS:
        align_labels(parent, group-info);
        break;
    case DTB_ALIGN_VCENTER:
        align_vcenter(parent, group-info, init);
        break;
    case DTB_ALIGN_RIGHT:
        align_right(parent, group-info);
        break;
    default:
        break;
}

static void
align_left(Widget parent, DtbgpGroupInfo *group-info)

WidgetList children_list;
Widget child,
int num_children = 0,
previous_child;
int num_columns, num_rows,
cell_width, cell_height,
i, j;

if (!parent || !group-info)
    return;

XtVaGetValues(parent, XtNnumChildren, &num_children,
               XmNchildren, &children_list, NULL);
if (num_children < 0)
    return;

get_group_cell_size(parent, group-info, &cell_width, &cell_height);
get_group_row_col(parent, group-info, &num_rows, &num_columns);
for (j = 0; j < num_rows; j++)
{
    for (i = 0; i < num_columns; i++)
    {
        Arg args[12];
        int n = 0;

        child = get_group_child(parent, group-info, i, j);
        if (!child)
            continue;

        if (i == 0 && j == 0)
            {s++;
                XtSetArg(args[n], XmLeftAttachment, XmATTACH_FORM);
                n++;
                XtSetArg(args[n], XmLeftOffset, 0);
                n++;
                XtSetArg(args[n], XmRightAttachment, XmATTACH_NONE);
                n++;
                XtSetValues(child, args, n);
                continue;
            }

        if (j == 0)
        {
            int DTB_GROUP_TYPES
            offset = group-info->hoffset;
            group_type = group-info->group_type;
            previous_child = get_group_child(parent, group-info, i-1, j);
            if (!previous_child)
                continue;

            if (group_type == DTB_GROUP_ROWS_COLUMNS)
                Dimension width = 0;

70
static void align_right(Widget parent, DtbGroupInfo *group_info, WidgetList children_list, int first_child, int last_child, int num_children, int num_columns, int cell_width, int cell_height, int offset, int i, int j)
{
    if (!group_info)
        return;

    /* Get children list */
    XtVaGetValues(parent, XmNnumChildren, &num_children, 
                   XmNchildren, &children_list, NULL);

    if (num_children < 0)
        return;

    get_group_cell_size(parent, group_info, &cell_width, &cell_height);
    get_group_row_col(parent, group_info, &num_rows, &num_columns);

    for (j = 0; j < num_rows; j++)
    {
        for (i = 0; i < num_columns; i++)
        {
            Arg arg[] = { Arg, args[12];
            int n = 0;

            child = get_group_child(parent, group_info, i, j);
            if (!child)
                continue;

            if ((i == 0) && (j == 0))
            {
                Dimension width = 0;
                XtVaGetValues(child, XmNwidth, &width, NULL);

                offset += (cell_width - (int)width);
            }

            XtSetArg(arg[n++], XmNpointerAttachment, XmATTACH_MOTION);
            XtSetArg(arg[n++], XmNleftWidget, previous_child);
            XtSetArg(arg[n++], XmNleftOffset, offset);
            XtSetArg(arg[n++], XmNrightAttachment, XmATTACH_NONE);
            XtSetValues(child, arg, n);
            continue;
        }
    }
}

91
static void align_labels(Widget parent, DbgGroupInfo *group_info, WidgetList *children_list = NULL, one_col = NULL, ref_widget = NULL, Dimension ref_width = NULL, num_children = 0, num_rows = 0, num_columns = 0, cell_width = 0, cell_height = 0, offset = 0, ref_x = 0, ref_y = 0, num_labels = 0, num_values = 0, cell_width = 0, cell_height = 0)

if (!parent || !group_info)
    return;

/* Get children list */
XmVaGetValues(parent, XmNnumChildren, &num_children,
              XmNchildren, &children_list, NULL);
if (num_children <= 0)
    return;

get_group_row_col(parent, group_info, &num_rows, &num_columns);
if (num_rows > 0)
    one_col = (WidgetList)XtMalloc(num_rows * sizeof(WidgetList));
for (i = 0; i < num_rows; i++)
    one_col[i] = get_group_child(parent, group_info, 1, i);

for (; i < num_rows; i++)
    one_col[i] = get_group_child(parent, group_info, 1, i);
get_widest_label(one_col, num_rows, &ref_widget, &ref_width);
if (!ref_widget)
    continue;
if (previous_ref_widget)
else
    offset -= (i * (group_info->hoffset + cell_width));
    offset = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
XtSetArg(args[n], XmNIeftOf fset, offset);
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE);
    offset = 0;
XtSetValues(ref_widget, args, n);
for (j = 0; j < num_rows; j++)
    {
        child = get_group_child(parent, group_info, i, j);
        if (!child || (child == ref_widget))
            continue;
        offset = (i * (group_info->hoffset + cell_width));
        offset += (int)(ref_width - get_label_width(child));
        n = 0;
        XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
        XtSetArg(args[n], XmNIeftOf fset, offset);
        XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE);
        XtSetValues(child, args, n);
    }
previous_ref_widget = ref_widget;
}
if (num_rows > 0)
    XtFree((char*)one_col);
}

static void
align_vcenter(
    Widget parent,
    DtDgroupInfo *group_info,
    Boolean init
)
{
    WidgetList children_list;
    Widget child, previous_child;
    DtD_GROUP_TYPES group_type;
    int num_children = 0,
        num_columns,
        num_rows,
        cell_width,
        cell_height,
        group_width,
        group_height,
        offset,
        gridline,
        i,
        j;
    if (!parent || !group_info)
        return;
    /*
     * Get children list
     */
    XtVaGetValues(parent,
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);
    if (num_children <= 0)
        return;
    get_group_cell_size(parent, group_info, &cell_width, &cell_height);
    get_group_row_col(parent, group_info, &num_rows, &num_columns);
    offset = group_info->hoffset;
    group_type = group_info->group_type;
    if (group_type == DTB_GROUP_ROWSCOLUMNS)
        group_width = (num_columns * cell_width) + ((num_columns-1) * offset);
    for (i = 0; i < num_columns; i++)
        
            if (group_type == DTB_GROUP_ROWSCOLUMNS)
                gridline = (((i * (cell_width + offset)) + (cell_width/2)) * 100)/group_width;
                gridline = 50;
                for (j = 0; j < num_rows; j++)
                    
            else
                gridline = 50;
                for (j = 0; j < num_rows; j++)
                    
}
child = get_group_child(parent, group_info, i, j);
if (!child)
    continue;

 XtVaGetValues(child, XmNwidth, &width, NULL);
if (i != 0)
    offset = 0;
if ( !(XtIsSubclass(child, compositeWidgetClass))
    offset = (cell_width - (int)width)/2;
if (group_type == DTB_GROUP_NONCOLLUMNS)
    offset += (i * (cell_width + group_info->hoffset));
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
XtSetArg(args[n], XmNleftOffset, offset);
    n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE);
    n++;
}
else
{        n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
XtSetArg(args[n], XmNleftPosition, gridline);
XtSetArg(args[n], XmNleftOffset, (int)(width/2));
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE);
    n++;
}
XtSetValues(child, args, n);
}
}

static void align_top(
    Widget parent,
    DtbGroupInfo *group_info
){
    WidgetList children_list;
    Widget previous_child = NULL,
    child,
    cur_child;
    int num_children = 0,
    num_columns,
    num_rows,
    cell_width,
    cell_height,
    i,
    j;
    if (parent == NULL || group_info)
        return;
        /*
        
        */
    XtVaGetValues(parent, 
        XmNnumChildren, &num_children,
        XmNchildren, &children_list,
        NULL);
    if (num_children <= 0)
        return;
    get_group_cell_size(parent, group_info, &cell_width, &cell_height);
    get_group_row_col(parent, group_info, &num_rows, &num_columns);
    for (j = 0; j < num_rows; j++)
    {        for (i = 0; i < num_columns; i++)
    {        Arg args[12];
        int n = 0;
        child = get_group_child(parent, group_info, i, j);
        if (!child)
            continue;
        if (i == 0 && (j == 0))
        {
            XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
            n++;
            XtSetArg(args[n], XmNtopOffset, 0);
            n++;
            XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE);
            n++;
            XtSetValues(child, args, n);
            continue;
        }
    }
}
if (i == 0)
    previous_child = get_group_child(parent, group_info, 0, j-1);
if (previous_child)
    { 
        DTB_GROUP_TYPES group_type = group_info->group_type;
        int offset = group_info->voffset;
        if (group_type == DTB_GROUP_ROWS_COLUMNS)
            { 
                Dimension height = 0;
                XtVaGetValues(previous_child, XmNheight, &height, NULL);
                offset += cell_height - (int)(height);
            
                XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET);
                ++n;
                XtSetArg(args[n], XmNtopWidget, previous_child);
                ++n;
                XtSetArg(args[n], XmNtopOffset, offset);
                ++n;
                XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE);
                ++n;
                XtSetValues(child, args, n);
        } 
        continue;
    }
previous_child = get_group_child(parent, group_info, i-1, j);
if (previous_child)
    { 
        XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET);
        XtSetArg(args[n], XmNtopWidget, previous_child);
        n++;
        XtSetArg(args[n], XmNtopOffset, 0);
        XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE);
        ++n;
        XtSetValues(child, args, n);
    }
}

static void
align_bottom(
    Widget parent,
    DtBGInfo *group_info
)

    WidgetList children_list;
    Widget child;
    DTB_GROUP_TYPES group_type;
    int num_children = 0,
    num_columns,
    num_rows,
    cell_height,
    cell_width,
    offset,
    i, j;
if (!parent || !group_info)
    return;
/*
 * Get children list
 */
XtVaGetValues(parent,
    XmNnumChildren, &num_children,
    XmNchildren, &children_list,
    XmNchildrenList, NULL);
if (num_children <= 0)
    return;
get_group_cell_size(parent, group_info, &cell_width, &cell_height);
get_group_row_col(parent, group_info, &num_rows, &num_columns);
for (i = 0; i < num_rows; i++)
    { 
        for (j = 0; i < num_columns; i++)
            { 
                child = get_group_child(parent, group_info, i, j);
                if (!child)
                    continue;
                if ((i == 0) && (j == 0))
                    { 
                        Dimension height = 0;
                        XtVaGetValues(child, XmNheight, &height, NULL);
                        offset = cell_height - height;
                    
                        }
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopOffset, offset); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(child, args, n);

continue;
}

if (i == 0)
{
    previous_child = get_group_child(parent, group_info, 0, j-1);
    if (previous_child)
    {
        Dimension height = 0;
        XtVaGetValues(child, XmNheight, &height, NULL);
        offset = group_info->voffset;
        if (group_info->group_type == DTB_GROUP_ROWS_COLUMNS)
            offset += (cell_height - height);
        XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
        XtsetArg(args[n], XmNtopWidget, previous_child);
        XtSetArg(args[n], XmNtopOffset, offset); n++;
        XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
        XtSetValues(child, args, n);
    }
    continue;
}

previous_child = get_group_child(parent, group_info, i-1, j);
if (child && previous_child)
{
    XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
    XtSetArg(args[n], XmNbottomWidget, previous_child); n++;
    XtSetArg(args[n], XmNbottomOffset, 0); n++;
    XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
    XtSetValues(child, args, n);
}

static void
align_hcenter(
    Widget parent,
    DtBGInfo *group_info,
    int *n,
    int i,
    int j)
{
    WidgetList children_list = NULL;
    Widget child, previous_child;
    DTB_GROUP_TYPES group_type;
    int num_children = 0,
    num_columns,
    num_rows,
    cell_width,
    cell_height,
    group_width,
    group_height,
    offset,
    gridline,
    i;

    if (!parent || !group_info)
        return;

    /*
    * Get children list
    */
    XtVaGetValues(parent, XmNnumChildren, &num_children, NULL);
    if (num_children <= 0)
        return;
    group_type = group_info->group_type;
    get_group_cell_size(parent, group_info, &cell_width, &cell_height);
    get_group_row_col(parent, group_info, &num_rows, &num_columns);
    offset = group_info->voffset;
    if (group_type == DTB_GROUP_ROWS_COLUMNS)
        group_height = (num_rows * cell_height) + ((num_rows-1) * offset);
    for (j = 0; j < num_rows; j++)
if (group_type == DTB_GROUP_ROWS_COLUMNS)
    gridline = (((i * (cell_height + offset)) + (cell_height/2)) * 100)/group_height;
else
    gridline = 50;
for (i = 0; i < num_columns; i++)
{
    Arg args[12];
    int n = 0;
    Dimension height = 0;
    child = get_group_child(parent, group_info, i, j);
    if (!child)
        continue;
    XtVaSetValues(child, XmNheight, &height, NULL);
    int init = 0;
    if (init)
    {
        init_offset = 0;
        if (!XtIsSubclass(child, compositeWidgetClass))
            init_offset = (cell_height - (int)height)/2;
        if (group_type == DTB_GROUP_ROWS_COLUMNS)
            init_offset += (j * (cell_height + group_info->voffset));
        XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
        XtSetArg(args[n], XmNtopOffset, init_offset); n++;
        XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
    } else
    {
        XtSetArg(args[n], XmNtopAttachment, XmATTACH_POSITION); n++;
        XtSetArg(args[n], XmNtopPosition, gridline); n++;
        XtSetArg(args[n], XmNtopOffset, (int)(-(height/2))); n++;
        XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
    }
    XtSetValues(child, args, n);
}

/*
 * Returns the directory that the executable for this process was loaded
 * from.
 * String dtb_get_exe_dir(void)
 * return dtb_exe_dir;
 */

/* Determines the directory the executable for this process was loaded from.
 */
static int determine_exe_dir(char *argv0, char *buf, int bufsize)
{
    Boolean foundDir = False;
    if ((buf == NULL) || (bufsize < 1))
        return -1;
    if (determine_exe_dir_from_argv(argv0, buf, bufsize) >= 0)
        foundDir = True;
    if (!foundDir)
    {
        if (determine_exe_dir_from_path(argv0, buf, bufsize) >= 0)
        {
            foundDir = True;
        }
    }
    return foundDir;
}

/*
 * Convert relative path to absolute, so that directory changes will
 * not affect us.
 */
if (foundDir && (buf[0] != '/'))
{
    char cwd[MAXPATHLEN+1];
    char *argv_pwd = NULL;
    char *path_prefix = NULL;
}
char abs_exe_dir[MAXPATHLEN+1];
if (getcwd(cwd, MAXPATHLEN+1) == NULL)
    path_prefix = cwd;
else
    env_pwd = getenv("PWD");
    if (env_pwd != NULL)
        path_prefix = env_pwd;

if (path_prefix == NULL)
    strcpy(abs_exe_dir, path_prefix);
    if (strcmp(buf, ".") != 0)
        strcat(abs_exe_dir, "/");
    strcat(abs_exe_dir, buf);
    strcpy(buf, abs_exe_dir);
}
return foundDir? 0:-1;

/* Looks for absolute path in arv[0]. */
static int determine_exe_dir_from_argv(char *argv0, char *buf, int bufsize)
{
    int i = 0;
    Boolean foundit = False;
    for (i = strlen(argv0)-1; (i >= 0) && (argv0[i] != '/'); i--)
    {
    }
    if (i > 0)
    {
        int maxStringSize = min(i, bufsize);
        strncpy(buf, argv0, maxStringSize);
        buf[maxStringSize-1] = 0;
        foundit = True;
    }
    return foundit? 0:-1;
}

/* Assumes: bufsize > 0 */
static int determine_exe_dir_from_path(char *argv0, char *buf, int bufsize)
{
    Boolean foundDir = False;
    Boolean moreDirs = True;
    char *sxExeName = argv0;
    int iExeNameLen = strlen(sxExeName);
    char *szTemp= NULL;
    char szPathVar[MAXPATHLEN+1];
    int iPathVarLen = 0;
    char szCurrentPath[MAXPATHLEN+1];
    int iCurrentPathLen = 0;
    int iCurrentPathStart = 0;
    uid_t euid = geteuid();
    gid_t egid = getegid();
    szTemp= getenv("PATH");
    if (szTemp == NULL)
    {
        moreDirs = False;
    }

    while ((szPathVar[0] == ' ') || (szPathVar[0] == '.'))
    {
        if (szPathVar[0] == '.');

    while (((iCurrentPathStart < iPathVarLen) && (szPathVar[i] != ':'))
    {  
    }
    szPathVar[i] = 0;
    }  
    }  

    return foundDir? 0:-1;
}
++i;

}  
iCurrentPathLen = iCurrentPathStart;
if (((iCurrentPathLen + iExeNameLen + 2) > MAXPATHLEN))
{
    iCurrentPathLen = MAXPATHLEN - (iExeNameLen + 2);
}

/* create a possible path to the executable */
strcpy(szCurrentPath, aszPathVar[iCurrentPathStart], iCurrentPathLen);
szCurrentPath[iCurrentPathLen] = 0;
strcat(szCurrentPath, "/");
strcat(szCurrentPath, aszExeName);

/* see if the executable exists and we can execute it */
if (path_is_executable(szCurrentPath, uid, egid))
{
    foundDir = True;
}

*/ skip past the current directory name */
if (!foundDir)
{
    iCurrentPathLen = iCurrentPathStart - iExeNameLen - 2;
    foundDir = True;
    iCurrentPathStart = iCurrentPathLen;
    while ((iCurrentPathStart < iPathVarLen)
        && (aszPathVar[iCurrentPathStart] != ':'))
    {
        ++iCurrentPathStart;  /* find : */
    }

    if (iCurrentPathStart < iPathVarLen)
    {
        ++iCurrentPathStart;  /* skip : */
    }
    if (iCurrentPathStart >= iPathVarLen)
    {
        moreDir = False;
    }
}

}  /* while !foundDir */
if (foundDir)
{
    szCurrentPath[iCurrentPathLen] = 0;
    strcpy(buf, szCurrentPath, bufsize);
    buf[bufSize - 1] = 0;
    return foundDir ? 0 : -1;
}

/* returns False is path does not exist or is not executable */
static Boolean path_is_executable(
    char *path,
    uid_t uid,
    gid_t gid)
{
    Boolean bExecutable = False;
    struct stat sStat;
    if (stat(path, &sStat) == C)
    {
        Boolean bDetermined = False;
        if (bDetermined)
        {
            if ((S_ISREG(sStat.st_mode))
            {
                /* not a regular file */
                bDetermined = True;
                bExecutable = False;
            }
            if (bDetermined)
            {
                if (uid == 0)
                {
                    if ((sStat.st_mode & S_IXOTH) == 0)  
                    |
                    ((sStat.st_mode & S_IXGRP) == 0)
                    |
                    ((sStat.st_mode & S_IXUSR) == 0)
                }
                bDetermined = True;
                bExecutable = True;
            }
        }
        if (bDetermined)
        {
            if ((sStat.st_mode & S_IXOTH) == 0)  
            |
            ((sStat.st_mode & S_IXGRP) == 0) && (sStat.st_gid == egid)
            |
            ((sStat.st_mode & S_IXUSR) == 0) && (sStat.st_gid == uid)
        }
        bDetermined = True;
        bExecutable = True;
    }

    return bExecutable;
}
/* if stat */

return bExecutable;

/* Registers a popup menu to be brought by button three on the parent */
dtb_popup_menu_register(Widget popupMenu, Widget parent)
{
    static XtTranslations popupMenuTrans = NULL;
    static XtActionsRec menuActions[] = {
        "DtDbPopupMenu", (XtActionProc) dtb_popup_menu
    };
    int i = 0;
    Boolean foundEntry = False;
    if (popupMenuTrans == NULL)
    {
        XtAppContext appContext = XtWidgetToApplicationContext(dtb_qetoplevel_widger());
        XtAppAddActions(appContext, menuActions, XtNumber(menuActions));
    }
    popupMenuTrans = XtParseTranslationTable(XtWidgetToApplicationContext(dtb_qetoplevel_widger()); "<Btn3Downz: DtDbPopupMenu(");
    XtOverrideTranslations(parent, popupMenuTrans);

    /* Save the reference from this widget to the menu */
    /* see if an entry already exists */
    for (i = 0; i < numPopupMenus; +i)
    {
        if (popupMenus[i].widget == parent)
            { if (popupMenus[i].menu == popupMenu)
                { foundEntry = True; break; } }
    }
    /* look for an empty slot */
    if (!foundEntry)
    {
        /* look for an empty slot */
        for (i = 0; i < numPopupMenus; +i)
        {
            if (popupMenus[i].widget == NULL)
                { foundEntry = True; break; } }
    } /* make a new slot, if necessary */
    if (!foundEntry)
    {
        /* It's not in the list - add it */
        DbMenuRef newPopupMenu = (DbMenuRef)
        realloc(popupMenus, sizeof(DbMenuRef)*(numPopupMenus+1));
        if (newPopupMenu != NULL)
        {
            popupMenus = newPopupMenu;
            +numPopupMenus;
            i = numPopupMenus-1;
            foundEntry = True;
        }
    } /* we have a slot: fill it in */
    if (foundEntry)
    {
        popupMenus[i].widget = parent;
        popupMenus[i].menu = popupMenu;
        XtAddCallback(popupMenus[i].widget, XmNdestrOyCallback, dtb_popup_menu_destroyCB, (XtPointer)NULL);
        XtAddCallback(popupMenus[i].menu, XmNdestrOyCallback, dtb_popup_menu_destroyCB, (XtPointer)NULL);
    }
    return 0;
}

static void
dtb_popup_menu:
{ Widget widget, XEvent *event, String *params, Cardinal *num_params }
{ int i = 0;
    Widget menu = NULL;
    if (event->type == buttonPress)
for (i = 0; i < numPopupMenu + 1; i++)
{
    if (popupMenus[i].widget == widget)
        menu = popupMenus[i].menu;
}
}
}

if (menu != NULL)
{
    XmMenutPosition(menu, (XButtonPressedEvent*)event);
    XtManageChild(menu);
}

/* This keeps the list of popup menus up-to-date, if widgets are destroyed */
static void
dtb_popup_menu_destroyCB(Widget widget,
    XtPointer clientData,
    XtPointer callData)
{
    int i = 0;
    for (i = 0; i < numPopupMenu + 1; i++)
    {
        if (popupMenus[i].widget == widget)
        {
            if (popupMenus[i].widget != NULL && popupMenus[i].menu != NULL)
                break;
        }
    }
}

/* Returns non-negative if successful. */
int
dtb_drag_site_register(Widget widget,
    DtbDragCallback callback,
    DtbDragProtocol protocol,
    unsigned char* bufferIsText,
    bool allowDropOnRootWindow,
    Pixmap cursor,
    Pixmap cursorMask,
    DtbDragSiteHandle* dragSiteHandleOut)
{
    DtbDragSite dragSite = {DtbDragSiteWidget(widget),
        XtAlloc(1, sizeof(DtbDragSiteRec))};
    Widget sourceIcon = NULL;
    if (dragSite == NULL)
    {
        dragSite->widget = widget;
        dragSite->protocol = protocol;
        dragSite->bufferIsText = bufferIsText;
        dragSite->allowDropOnRootWindow = allowDropOnRootWindow;
        if ((cursor != NULL) && (cursorMask != NULL))
        {
            dragSite->sourceIcon =
                DtbCreateSourceIcon(widget, cursor, cursorMask);
            dragSite->callback = callback;
            dragSite->convertCBRec[0].callback = dtb_drag_convertCB;
            dragSite->finishCBRec[0].callback = dtb_drag_finishCB;
            XtAddEventHandler(widget, ButtonMotionMask, False,
                (XEventHandler)dtb_drag_button1_motion_handler,
                (XtPointer)dragSite);
            XtAddEventHandler(widget, ButtonPressMask, False,
                (XtEventHandler)dtb_drag_button1_event_handler,
                (XtPointer)dragSite);
        }
        return 0;
    }
    /* Pass back a handle, so that this can be freed, later. Unregistering
     * drag sites is not currently implemented, but this gives the ability
     * to provide this functionality in the future. */
    if (dragSiteHandleOut != NULL)
    {
        *dragSiteHandleOut = (DtbDragSiteHandle)dragSite;
    }
    return 0;
}

int
dtb_drop_site_register(
Widget widget,
DtDndDropCallback callback,
DtDndProtocol protocols,
unsigned char operations,
Boolean dropsonChildren,
Boolean preservePreviousRegistration,
DtDndSiteHandle *dropSiteHandleOut
}

DtDndSite dropSite = (DtDndSite)NULL;
if ((callback != NULL)
  && ((dropSite = (DtDndSite)XtCalloc(sizeof(DtDndSiteRec)),
       (= NULL))

  DtDndRegisterInfoRec regInfo;

  /* initialize the data */
dropSite->widget = widget;
dropSite->callback = callback;
dropSite->protocols = protocols;
dropSite->operations = operations;
dropSite->dropsonChildren = dropsonChildren;
dropSite->preservePreviousRegistration = preservePreviousRegistration;
dropSite->animateCBRec[0].close = XtPointer(dropSite);
dropSite->transferCBRec[0].close = XtPointer(dropSite);

  /* Let the client modify the drop site initialization */
  regInfo.protocols = dropSite->protocols;
  regInfo.operations = dropSite->operations;
  regInfo.textIsBuffer = dropSite->textIsBuffer;
  regInfo.preservePreviousRegistration = dropSite->preservePreviousRegistration;

  /* actually register it: */
  DtDndVaDropRegister(
    widget, regInfo.protocols, regInfo.operations,
    dropSite->transferCBRec, DtNdropAnimateCallback, dropSite->animateCBRec,
    NULL)

tifdef DtNpreserveRegistration
  DtNSaveDropRegistration, regInfo.preservePreviousRegistration,
#endif

  if (dropSiteHandleOut != NULL)
    *dropSiteHandleOut = (DtDndSiteHandle)dropSite;
return ((dropSite == NULL)? -1:0);

static int
dtb_drag_terminate(DtbDragSite dragSite)
{
  dragInProgress = False;
dragInitialX = -1;
dragInitialY = -1;
dragSite->convertCBRec[0].close = NULL;
dragSite->dragToRootCBRec[0].close = NULL;
dragSite->dragFinishCBRec[0].close = NULL;
return 0;
}

/*
 * dragMotionHandler
 *
 * Determine if the pointer has moved beyond the drag threshold while button 1
 * was being held down.
 */
static void
dtb_drag_button1_motion_handler(
  Widget dragInitiator,
  XtPointer clientData,
  XEvent *event)
{
  int diffX, diffY;
  DtbDragSite dragSite = (DtbDragSite)clientData;

  if (dragInProgress) {

    if (dragInitialX == -1)

      if (dragInitialY == -1) {

        dragInitialX = event->xmotion.x;
        dragInitialY = event->xmotion.y;

      }

    /* Find out how far pointer has moved since button press

    */
diffX = dragInitialX - event->xmotion.x;
diffY = dragInitialY - event->xmotion.y;
if ((ABS(diffX) > DRAG_THRESHOLD) ||
    (ABS(diffY) > DRAG_THRESHOLD));
{
    dtb_drag_start(dragSite, event);
    dragInitialX = -1;
    dragInitialY = -1;
}

/* * Starts a drag if Button2 is pressed (if the system is configured for * Button2 transfers). */
static void dtb_drag_button2_event_handler(
    Widget dragInitiator,
    XPointer clientData,
    XEvent *event)
{
    if ((event->type == ButtonPress) && (event->xbutton.button == 2))
    {
        Boolean enableBtnlTransfer = (Boolean)0;
        Boolean mouseButton2IsTransfer = False;
        /* * We need to determine whether mouse button 2 is adjust or * transfer. Although the resource value is of type Boolean, it * can actually have integer values. This code is from page 52 * of "Common Desktop Environment: Programmer's Guide." */
        XtVaGetValues((Widget)XmGetDisplay(dragInitiator),
            "enableBtnlTransfer", enableBtnlTransfer, NULL);
        mouseButton2IsTransfer = (enableBtnlTransfer != 1);
        if (mouseButton2IsTransfer)
        {
            if (dragInProgress)
            {
                dtb_drag_start(dragSite, event);
            }
        }
    }
}

static int dtb_dndDragStart(DtbDragSite dragSite, XEvent *event)
{
    DtbDndDragStartInfoRec dragStart;
    Arg args[3];
    int n = 0;
    dragInProgress = True;
    dragSite->convertCBRec[0].closure = (XtPointer)dragSite;
    dragSite->dragToRootCBRec[0].closure = (XtPointer)dragSite;
    dragSite->dragFinishCBRec[0].closure = (XtPointer)dragSite;
    /* * Call the client callback to update values for drag start */
    memset((void *)&dragStart, 0, sizeof(DtbDndDragStartInfoRec));
    dragStart.protocol = dragSite->protocol;
    dragStart.operations = dragSite->operations;
    dragStart.cursor = dragSite->sourceIcon;
    dragStart.bufferIsText = dragSite->bufferIsText;
    dragStart.allowDropOnRootWindow = dragSite->allowDropOnRootWindow;
    dragStart.numItems = 1;
    if (dragSite->callback != NULL)
    {
        dragSite->callback(DTB_DND_DRAG_START, &dragStart, NULL, NULL, NULL, NULL);
    }
    n = 0;
    if (dragStart.cursor != NULL)
    {
        XtSetArg(args[n], DTSourceIcon, dragStart.cursor); ++n;
    }
    if ( (dragStart.protocol == DT_DND_BUFFER_TRANSFER) )
    {
        XtSetArg(args[n], DTBufferIsText, True); ++n;
    }
    if (dragStart.allowDropOnRootWindow)
    {
        XtSetArg(args[n], DTAllowDropOnRootCallback, (XtPointer)dragSite->dragToRootCBRec); ++n;
    }
    if (DtbDndDragStart(dragSite->widget, event, dragStart.protocol, dragStart.numItems,
dragStart operations, dragSite->convertCBRec, dragSite->dragFinishCBRec, 
arg. 0) == NULL;
/* drag start failed */
dtb_drag_terminate(dragSite);
return -1;
return 0;

static void
dtb_drag_convertCB(
Widget dragContext,
 XtPointer clientData,
 XtPointer callData
)
{
DtbDragSite dragSite = (DtbDragSite)clientData;
DtdDndConvertCallback convert = (DtDndConvertCallback)callData;
int steroids;
switch (convert->reason).
{
case DtCR-DND-CONVERT-DATA:
dragSite->callback(DTB_DND_CONVERT, NULL, convert, NULL, NULL, NULL);
break;
case DtCR-DND-CONVERT-DELETE:
dragSite->callback(DTB_DND_DELETE, NULL, NULL, convert, NULL);
break;
}

static void
dtb_drag_to_rootCB(
Widget dragContext,
 XtPointer clientData,
 XtPointer callData
)
{
DtbDragSite dragSite = (DtbDragSite)clientData;
DtdDndDroppedOnRootWindowInfoRec dropInfo;
memset(&dropInfo, 0, sizeof(DtdDndDroppedOnRootWindowInfoRec));
dropInfo.droppedOnRootWindow = True;
dragSite->callback(DTB_DND_DROPPED_ON_ROOT_WINDOW,
NULL, NULL, &dropInfo, NULL, NULL);

static void
dtb_drag_terminateCB(
Widget dragContext,
 XtPointer clientData,
 XtPointer callData
)
{
DtbDragSite dragSite = (DtbDragSite)clientData;
DtdDndFinishCallback finish = (DtDndFinishCallback)callData;
dragSite->callback(DTB_DND_FINISH, NULL, NULL, NULL, NULL);
dtb_drag_terminate((DtbDragSite)clientData);
}

static void
dtb_drop_animateCB(
Widget dragContext,
 XtPointer clientData,
 XtPointer callData
)
{
DtbDropSite dropSite = (DtbDropSite)clientData;
DtdDndAnimateCallback animateInfo = (DtDndAnimateCallback)callData;
dropSite->callback(dropSite, dtdDndAnimateCallback, NULL, NULL, animateInfo);

static void
dtb_drop_transferCB(
Widget dragContext,
 XtPointer clientData,
 XtPointer callData
)
{
DtbDropSite dropSite = (DtbDropSite)clientData;
DtdDndTransferCallback transferInfo = (DtDndTransferCallback)callData;
dropSite->callback(dropSite, dtdDndTransferCallback, NULL, NULL, transferInfo, NULL);
DISTRIBUTION:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MS 0431</td>
<td>S. G. Varnado</td>
</tr>
<tr>
<td>1</td>
<td>MS 0451</td>
<td>J. H. Moore</td>
</tr>
<tr>
<td>5</td>
<td>MS 0451</td>
<td>M. R. Sjulin</td>
</tr>
<tr>
<td>2</td>
<td>MS 0100</td>
<td>Document Processing for DOE/OSTI, 7613-2</td>
</tr>
<tr>
<td>1</td>
<td>MS 0619</td>
<td>Print Media, 12615</td>
</tr>
<tr>
<td>5</td>
<td>MS 0899</td>
<td>Technical Library, 4414</td>
</tr>
<tr>
<td>1</td>
<td>MS 9018</td>
<td>Central Technical Files, 8523-2</td>
</tr>
<tr>
<td>3</td>
<td>MS 0449</td>
<td>R. A. Sarfaty</td>
</tr>
</tbody>
</table>