The Information Distribution Technology:

LIBXMAP -
Application Program Interface

Kenneth G. Smith
Holly A. Ingham

ARL-TR-498
August 1994

DTIC ELECTED
AUG 24 1994

94-26928

DTIC QUALITY INSPECTED

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.
NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.
The Information Distribution Technology: LIBXML - Application Program Interface

Kenneth G. Smith and Holly A. Ingham

U.S. Army Research Laboratory
ATTN: AMSRL-CI-SB
Aberdeen Proving Ground, MD 21005-5067

U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066

Approved for public release; distribution is unlimited.

The Military Computer Science Branch (MCSB) of the Advanced Computational and Information Sciences Directorate (ACISD) of the Army Research Laboratory (ARL) has been developing concepts to provide enhanced information exchange capabilities to "fighting level" commanders and soldiers as part of an ongoing, long-term research effort known as the Information Distribution Technology (IDT). The IDT concepts have been successfully demonstrated on numerous occasions, most notably during the 1989 LABCOM Smart Weapons Cooperative Program Demonstration and the 1991 Second Counter-Air Symposium for Army Aviation and Air Defense. The report discusses a new software library created to facilitate the development of integrated application programs for testing, refining, and demonstrating existing and emerging IDT concepts.

application program interface, IDS, IDT, information systems, tactical data systems

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UL
PREFACE

The Military Computer Science Branch (MCSB) of the Advanced Computational and Informational Sciences Directorate (ACISD) of the Army Research Laboratory (ARL) has been developing concepts to provide enhanced information exchange capabilities to “fighting level” commanders and soldiers as part of an on-going, long term research effort known as the Information Distribution Technology (IDT). The IDT concepts have been successfully demonstrated on numerous occasions, most notably during the 1989 LABCOM Smart Weapons Systems Cooperative Program Demonstration and the 1991 Second Counter – Air Symposium for Army Aviation and Air Defense. This report discusses a new software library created to facilitate the development of integrated application programs for testing, refining, and demonstrating existing and emerging IDT concepts. The authors wish to thank Eric Heilman and Fred Brundick for their insightful contributions in reviewing this report.
INTENTIONALLY LEFT BLANK.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>2.1 Information Distribution Technology</td>
<td>1</td>
</tr>
<tr>
<td>2.2 Application Programs</td>
<td>2</td>
</tr>
<tr>
<td>3. LIBXMAP</td>
<td>3</td>
</tr>
<tr>
<td>3.1 General Overview</td>
<td>3</td>
</tr>
<tr>
<td>3.2 Functional Review</td>
<td>3</td>
</tr>
<tr>
<td>3.2.1 Objects</td>
<td>4</td>
</tr>
<tr>
<td>3.2.2 Events</td>
<td>4</td>
</tr>
<tr>
<td>3.2.3 Links and Associations</td>
<td>4</td>
</tr>
<tr>
<td>4. LIBXMAP Functions</td>
<td>5</td>
</tr>
<tr>
<td>4.1 Client-to-Xmap Commands</td>
<td>5</td>
</tr>
<tr>
<td>4.2 Xmap-to-Client Commands</td>
<td>6</td>
</tr>
<tr>
<td>5. CONCLUSION</td>
<td>6</td>
</tr>
<tr>
<td>6. REFERENCES</td>
<td>7</td>
</tr>
<tr>
<td>APPENDIX A: OBJECT TYPES and CLASSES</td>
<td>9</td>
</tr>
<tr>
<td>APPENDIX B: LIBXMAP FUNCTIONS</td>
<td>13</td>
</tr>
<tr>
<td>ACRONYMS</td>
<td>29</td>
</tr>
<tr>
<td>DISTRIBUTION LIST</td>
<td>31</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Information Distribution Technology.</td>
<td>2</td>
</tr>
<tr>
<td>2. Libxmap Architecture</td>
<td>3</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The Military Computer Science Branch (MCSB) of the Advanced Computational and Informational Sciences Directorate (ACISD) of the Army Research Laboratory (ARL) has been developing concepts to provide enhanced information exchange capabilities to "fighting level" commanders and soldiers as part of an on-going, long-term research effort known as the Information Distribution Technology (IDT). The goal of the IDT project is to facilitate the exchange of tactical information over standard combat net radios (CNR) at data rates as low as 1,200 bits per second. To satisfy this goal, the IDT seeks to improve information distribution by exchanging data as concisely as possible, only when absolutely necessary, and as efficiently as possible.

Paramount to the success of the IDT is the ability to test, evaluate, and demonstrate information distribution concepts being developed. For this purpose, several prototype experimental software application programs were developed. As the IDT matured, new application programs were needed that could more readily address the information distribution concepts. Hence, a new suite of demonstration application programs was developed. These new programs are designed to be more portable, more flexible, easily tailored, quick to develop, and integratable with each other. A key component of the new application programs was an interface library called libxmap which is the focus of this report.

2. OVERVIEW

2.1 Information Distribution Technology

The IDT is prototype software designed to provide enhanced information exchange capabilities to "fighting level" commanders and soldiers. The goal of the IDT project is to facilitate the exchange of tactical information over CNRs at data rates as low as 1200 bits per second. The development of the IDT is guided by three tenets for exchanging battlefield information (Chamberlain, 1990). Data is exchanged 1) as concisely as possible, 2) only when absolutely necessary, and 3) as efficiently as possible. Further, the IDT seeks to combine military science technology with state-of-the-art computer science technology to produce a powerful, effective, and flexible information exchange technology.

At the heart of the IDT is a free-form distributed factbase (DFB) that stores all of the information pertinent to a battlefield situation (Hartwig, 1991). A security control module (SCM) controls the flow of information into and out of the DFB. Data distribution rules provide the SCM with guidance for exchanging information with other DFBs. This information exchange is handled by the IDT's fact exchange protocol (FEP) (Kaste, 1990). "Triggers" within the DFB allow application programs to be notified when specific information is entered into the DFB that may be of interest to application programs. Lastly, an interface (I/F) exists that allows sophisticated application programs to extract, enter, and modify information stored in the DFB. See Figure 1.

Information stored in a DFB is an abstract representation of military concepts and current battlefield perceptions. Information is stored and manipulated in a manner consistent with the design tenets of the IDT. It is stored in the DFB in "facts" — logical groupings of data representing a unique item, event, or activity. Each fact is an instantiation of a pre-defined data structure called a facttype. A facttype is a template that defines the structure of a fact; i.e., the names of the fields in a fact and the type of data each field represents.
Data is stored in the DFB so as to be most manipulable by computers, not by humans. One of the functions of application programs is to enter and subsequently modify data in the DFB. Application programs provide the unique service of presenting DFB data in a form that is easily understood and manipulable by humans. Application program interfaces are the means for testing, evaluating, and demonstrating the IDT concepts.

2.2 Application Programs

To date, several application programs have been developed that, working in concert with each other, demonstrate these concepts. A viable, militarily sound and realistic IDT demonstration package has been developed to exercise the IDT concepts.

Leveraging off of earlier involvement in the 1989 Smart Weapons Systems (SWS) LABCOM Cooperative Program demonstration and the 1991 joint Human Engineering Laboratory—Ballistic Research Laboratory Counter—Air Program (HELCAP) demonstration, six application programs have been developed. These application programs are Xmap, Milmap, Org Chart, Vtracker, Tracker, and Extract. They provide the capability to demonstrate IDT concepts being used to exchange battlefield information based on both a ground and air war scenario consistent with a high-intensity European-based conflict in the area of the Fulda Gap region of Germany.

Extract and Tracker are the scenario driver programs providing the data depicting the ground and air war, respectively. Milmap, Vtracker, and Org Chart are user application programs that allow a commander to view/monitor the battle as it unfolds. Xmap is a general pur-

† The uniqueness of this service stems from the fact that data can be entered or modified in the DFB via the FEP. However, only application programs are capable of presenting data in a user-friendly format.
pose mapping application program that provides the common medium upon which unit and other overlay symbols are placed while viewing the battle. Milmap, Vtracker, and Org Chart use Xmap for all map-related functions (drawing unit symbols, range fans, routes, borders, areas, etc.) and can interact with each other through it. The means by which application programs communicate with Xmap is via the libxmap software library.

3. LIBXMAP

3.1 General Overview

Libxmap is a library of software routines used by Xmap and client application programs that communicate with it. Such application programs will be hereafter referred to interchangeably as "clients," "client programs," or "client application programs." Xmap and application programs that connect to Xmap follow a client-server paradigm, where Xmap is the server. Within this paradigm, the Xmap server provides the basic or generic capabilities for displaying a map and overlay symbols. For developmental purposes, the map used was digitized from a photograph of a 1:100,000 scale map of the Hunfeld region of Germany. Xmap is written in C and uses the X Window System. It provides map manipulation capabilities for zooming, panning, resizing the display window, turning on or off grid lines, and for creating and/or editing lines.

Client programs provide the link between Xmap and the military information to be displayed in Xmap. Clients decide when, where, and what information to display on the map in the form of map "overlays." This information is sent to Xmap via the standardized library, libxmap.

The client-server paradigm for application programs improves the IDT demonstration by 1) speeding up the development of client, or "military," application programs, 2) providing a single common map display program usable by all application programs, and 3) eliminating the need for client programs to concern themselves with windowing issues (e.g., resizing, exposures, drawing overlays that aren't in the field of view, etc.). This architecture allows client application programs to be developed quickly and reliably and does not require the developer to be an expert in developing graphical user interfaces. See Figure 2.

![Figure 2. Libxmap Architecture](image-url)
3.2 Functional Review

*Libxmap* is a collection of subroutines that are used to exchange information between *Xmap* and client programs. Client programs connect to *Xmap* and then command it to draw objects, modify existing objects, and remove objects. *Xmap* complies with these commands while providing a user window interface (e.g., mouse interaction, window resize capability, etc). Events that may be of interest to clients, such as mouse clicks on objects drawn by *Xmap*, are relayed to the client programs by *Xmap* through *libxmap*. In this fashion client programs maintain control over the objects that they commanded *Xmap* to display. There are two key concepts underlying the interaction between client programs and *Xmap*: objects and events.

3.2.1 Objects

Objects are the main component, or unit of information, on which *Xmap* and client programs interact. Client programs command *Xmap* to create, modify, or remove objects. To client programs, these objects take the form of unit symbols, map symbology, points, lines, routes, areas, etc. To *Xmap*, these are all just objects. To provide a means for client programs and *Xmap* to act on sets and/or subsets of objects, attributes are associated with each object that categorize them into object types and classes. These attributes are optionally set by client programs. As of the writing of this report, only a small number of types and classes have been defined and are explained in Appendix A.

3.2.2 Events

Events are actions that are performed by the user through *Xmap*’s user interface. Client programs can register their interest in specific events with *Xmap* or default to receive notification of all events. Typically, events occur on objects via mouse button presses. Other events are line modification and object relocation. Button press events generate a BUTTONPRESS event communication from *Xmap* to clients identifying which mouse button was pressed and on which object the mouse was pressed. The modification of lines (performed through *Xmap*’s user interface) generates a LINE CHANGE event that identifies the line that was modified and the new coordinates of the segments comprising the line.

Certain commands from client programs may have an indirect impact on other objects. For instance, moving one object may cause another object to move automatically. For such a case, *Xmap* generates a SYMBOL CHANGE event identifying the object that changed and its new location. Such indirect changes occur with “associated” objects.

3.2.3 Links and Associations

Objects may be linked together to form logical relationships to simplify client programming. *Libxmap* imposes no constraints on why objects are linked; it simply provides the capability to link them. There are no restrictions on the number or type (one-to-one, one-to-many, or many-to-one) of links between objects. *Libxmap* distinguishes between two kinds of links, referred to simply as “links” and “associations.”

“Links” are a one-to-one relationship used to visually bind two objects. *Xmap* draws a line between objects that are “linked.” If one of the objects move, then the line between them follows it automatically. If either object is removed, then the line (and link) between them is automatically removed.

Associations form a relationship between objects that governs the way objects respond to actions. Associations enable related objects to perform as a group. Unlike linked objects
associated objects have no line drawn between them in Xmap (although, associated objects can also be linked if a connecting line is desired). Associations are used to spatially bind objects. That is, when one object moves, any associated objects will also move to maintain their relative physical position to the moved object. There are two types of associations: MASTER_SLAVE and GROUP. In MASTER_SLAVE associations, if the master object moves, then any slave objects will follow it, but not vice versa. In a GROUP association, if either object moves, then all other objects follow it.

4. LIBXMAP Functions

Libxmap is divided into two sets of routines that provide for client-to-Xmap communication and Xmap-to-client communication. Client-to-Xmap commands, or client commands, are routines by which client programs connect and send object draw/modify requests to Xmap. Xmap-to-client commands, or Xmap commands, enable Xmap to send synchronous and asynchronous information to client programs.

Below is a brief description of each command. For readability, the commands are italicized and followed by open and closed parentheses, e.g., command_name(). Xmap-to-client commands are differentiated from client-to-Xmap commands by a preceding underscore character, "_", in the command name. More detailed information on each command can be found in Appendix B.

4.1 Client-to-Xmap Commands

Xmap_connect() is the command used by client programs to connect to Xmap. This command allows clients to request notification of event occurrences. Upon successful connection Xmap assigns a unique identification number to the client and returns that number to the client for use in all future exchanges. Clients terminate their connection to Xmap via the xmap_close() command.

Clients instruct Xmap to draw an object using the xmap_draw_symbol() command. The object’s location, foreground and background colors, class type and size are specified. For referencing purposes, Xmap returns an object identification number to the client. The object identification number and the client identification number uniquely identify an object. Attributes of objects can be modified via the xmap_change_symbol() command. Objects may be removed singularly using the xmap_remove_symbol() command. The xmap_remove_class() command removes all objects of the specified class type for the particular client.

There are two ways for clients to draw lines: xmap_add_line() and xmap_convert_line(). The xmap_add_line() command is used when the client supplies the end points of the segments defining the line to be drawn. The xmap_convert_line() command is used when a client converts a line created locally via Xmap to a line “owned” by the client. Lines are modified via the xmap_change_line() command. The xmap_remove_line() command is used to remove lines.

To show that two objects are linked together (illustrated by a line connecting the two objects in Xmap), the xmap_add_link() command is used. Links are removed via the xmap_remove_link() command. Objects can be associated (with no visible link) using the xmap_add_association() command. Associations are removed using the xmap_remove_association() command.

A somewhat specialized command exists to instruct Xmap to draw range fans. Range fans are used to show the area in which a weapon system is able to fire. To draw a range fan the
The *xmap_draw_rangefan()* command is used. Subsequently, range fans are removed using *xmap_remove_rangefans()* . Range fans originate from the center point of an object that is identified in the *xmap_draw_rangefan()* command. Range fans are automatically associated with the object from which they originate.

4.2 Xmap—Client Commands

* Xmap_connect() is the command used by Xmap to establish itself as the server in the client-server paradigm. Once initialized, it listens for new client connections and polls any existing connections for new client requests. When a new client connects, Xmap sends a unique number to the connecting client via the *xmap_send()* command to be used by the client in all future communications with Xmap. Connected clients are informed of an impending closure of the network server connection via the *xmap_close()* command. Errors are reported to clients via the *xmap_error()* command.

Clients are notified of any generic button press events via the *xmap_button_press()* command. A generic button press event, or “click,” is classified as a mouse button press occurring anywhere within the Xmap display window. Xmap sends clients the unique object identification number of the clicked object (or 0 to indicate the map background, not an object, was pressed), the mouse button number that caused the event, and the map grid location (easting and northing) of the button press.

When a user clicks on a line, Xmap responds by sending a line click message via the *xmap_line_click()* command. This command contains the corresponding easting and northing coordinates of each segment comprising the clicked line. *Xmap_line_change()* is a command used by Xmap to inform a client when a line has been edited by the user through Xmap’s line editing control panel.

Association of objects using a MASTER/SLAVE configuration can cause a change in location of one object when the other object’s location is changed. Clients are informed of the location of the “slaved” object via the *xmap_symbol_change()* command.

5. Conclusion

The libxmap library is an integral component of the application programs used to test, evaluate, and demonstrate the information distribution concepts being developed and explored as part of the IDT. It provides a uniform medium for linking together the various demonstration application programs and provides the utility to facilitate the development and integration of new application programs.
6. REFERENCES


INTENTIONALLY LEFT BLANK.
APPENDIX A:

OBJECT TYPES AND CLASSES
INTENTIONALLY LEFT BLANK.
Objects and Classes

Application programs instruct Xmap to create, move, modify, and remove objects. To facilitate the application program’s tasks of manipulating objects, they can be categorized into groups of classes. Application programs assign group and class attributes to each object they create. Groups and classes allow objects to be manipulated as a set so that operations can be performed en masse. For example, an application program can instruct Xmap to remove all objects of class “line.”

The definitions and uses assigned to each group and class are not rigidly stated. During prototype development, they were created on an “as needed” basis as determined by the current application programs. To date, the potential in creating and assigning group and class attributes to objects has not been fully exploited.

Object Groups

Below is a list of the current object groups along with a suggested use for each. Note that each group name starts with “OB” for Object group.

- OB_SYMBOL – object is typically a unit symbol, although symbols exist for other features such as bridges, buildings, etc.
- OB_TEXT – object is a text string
- OB_POINT – object is a military operations “point,” e.g., coordination point
- OB_LINE – object is a military operations “line,” e.g., phase line, FEBA, border
- OB_AREA – object delineates an area, e.g., no fire zone
- OB_RANGEFAN – object depicts a unit’s range fan (area in which its weapons can fire)
- OB_LINK – object “links” two other objects together with a visible line
- OB_ASSOCIATION – object “links” two other objects together spatially, i.e., if one object moves, the associated object will move with it, but with no visible line

Object Classes

Below is a list of the current object classes along with a suggested use for each. Xmap requires that all objects be assigned a group and class identifier. Note that each class starts with “CL” for object Class.

- CL_UNDEFINED – used by application program to specify no class attribute
- CL_UNIT – object is of group OB_SYMBOL and represents an actual unit
- CL_SENSING – object is of group OB_SYMBOL and represents the reported observation of a unit (typically an enemy unit)
- CL_LINE – object is of group OB_LINE. Possible future use is to have several different types (or classes) of lines
- CL_TRACK – object is of group OB_SYMBOL and represents an air defense track
- **CL_LINK** — object is of group OB_LINK (see OB_LINK above)
- **CL_RANGEFAN** — object is of group OB_RANGEFAN (see OB_RANGEFAN above)
- **CL_POINT** — object is of group OB_POINT (see OB_POINT above)
APPENDIX B:

LIBXMAP FUNCTIONS
Libxmap Functions

Libxmap is a software library of subroutines, or commands, used to exchange information between Xmap and client application programs. This appendix contains a listing of all of the commands along with a brief explanation of the arguments comprising each command and a short description. It is divided into two sections describing the commands available to client programs for communicating with Xmap and the commands available to Xmap for communicating with its clients.

Client Commands

Below is a list of the commands available to client programs for communicating to Xmap. For each command there is a brief explanation of its intended use as well as its syntax.

/*
 ~ xmap_connect ( )
 */

struct pkg_conn *
xmap_connect( host, switch_array, ev_button, ev_keyboard, ev_geometry, ev_object);

char *host;       /* hostname to connect to */
struct pkg_switch switch_array [ ]; /* array of switching routines */
int ev_button     /* application’s interest of button events */
      ev_keyboard,    /* application’s interest of keyboard events */
      ev_geometry,    /* application’s interest of geometry events */
      ev_object;      /* application’s interest of object events */

“Host” is the name of the machine on which Xmap is running. “Switch_array” is used for specifying the routines in the client program that are called when Xmap sends information through the pkg protocol. “Ev_button,” “ev_keyboard,” “ev_geometry,” and “ev_object” are flags used by the client to indicate its interest in X events of the named type that occur within Xmap’s window. For example, if an object is picked using the mouse and a client set “ev_object” to TRUE then Xmap would notify the client which object was picked and by which mouse button. In response, Xmap sends a message back to the client giving it its unique client identification number by which Xmap identifies it.

Return: A pointer to a pkg connection structure. Each client program gets a unique pkg connection structure that is used by the pkg routines for communicating to the client programs.

/*
 ~ xmap_close( )
 */

xmap_close( conn )
struct pkg_conn *conn; /*Application program’s connection to Xmap */
“Conn” is the pkg connection structure for this client (obtained when it first connected to Xmap). This routine informs Xmap that the client is closing its connection. Any objects that the client previously instructed Xmap to display are removed.

Return: Not used.

/*
~ xmap_send()
*
* Test routine for application program to send information to Xmap.
*/

xmap_send (msg_type, msg, conn)

int msg_type; /* Define (from libxmap.h) */
char *msg; /* Contents of message */
struct pkg_conn *conn; /* Application program’s connection to Xmap */

“msg_type” is a #define that specifies the type of message that is being sent to Xmap. “msg” is the actual message. “conn” is the client’s pkg connection structure.

Return: Not used.

/*
~xmap_add_symbol()
*
* Routine used by application programs to draw symbols on the map.
* This routine draws NEW symbols, or “objects” to Xmap, hence a handle id
* must be associated with the symbol (object).
*/

int xmap_add_symbol (symbol, fg_color, bg_color, scale, x, y, class, conn, fid, client_id)

char *symbol; /* Symbol string to draw. */
int fg_color, /* Foreground color of symbol */
bg_color, /* Background color of symbol */
x, y, /* Map grid coordinates (easting, northing) */
class; /* Class type of symbol object */
float scale; /* Scaling factor for symbol */
struct pkg_conn *conn; /* Application program’s connection to Xmap. */
dkb_factid_t fid; /* Fact id associated with this symbol. */
int client_id; /* Client identifier number. */

This routine is used for drawing unit symbols on the map. To Xmap, a unit symbol is just another overlay symbol, or object. Xmap identifies objects by their handle id and the client id of the client program that submitted the command to draw the object. The handle id must be unique to a specific client. That is, objects drawn by a client must never share the same handle id. To ensure this uniqueness, and to alleviate the client programs from the burden and task of assigning unique handle ids, libxmap assigns the handle id for all new objects. “symbol” is a string recognized by the symbol drawing library, symlib, that describes the unit symbol to be drawn. “fg color” and “bg color” are #defines that are described in sym-
bol.h used for specifying the foreground and background colors of the symbol. "x" and "y" are the UTM map coordinates for where the symbol should be drawn. The center of the symbol is drawn at this coordinate. "class" is a #define that further identifies the type of object being drawn. The class defines are in libxmap.h. Though not yet implemented, the idea was that Xmap could be instructed to act on whole classes of objects rather than single objects. "scale" is the factor by which symbols are scaled down when drawn by the symbol drawing library. Typical values are defined in symbol.h. "conn" is the client’s pkg connection structure. "fid" is the IDT fact id of the fact that is associated with the symbol being drawn. It is included so that other client programs that pick this symbol have a source of information pertaining to the symbol. "client_id" is the client identification number of the client that is drawing the symbol.

Return: Handle id of new object.

/*
~ xmap_change_symbol ( )
*
* Routine used by application programs to change attributes of a symbol drawn on the map. "handle" should have been previously assigned via xmap_add_symbol ( ). "trace" instructs Xmap whether or not to draw a (volatile) line from the old location to the new location showing the path of the symbol (if its location changed).
*/

void
xmap_change_symbol (handle, client_id, symbol, fg_color, bg_color, scale, x, y, class, trace, conn, fid)

char *symbol; /* Symbol string to draw. */
int handle; /* Handle id of object to change */
int client_id; /* Client identifier number. */
int fg_color, bg_color; /* Foreground color of symbol */
int class; /* Background color of symbol */
x, y; /* Map grid coordinates (easing, northing) */
float scale; /* Scaling factor for symbol */
int trace; /* Shows trail of moved symbol. */
struct pkg_conn *conn; /* Application program’s connection to Xmap. */
dkb_factid_t fid; /* Fact id associated with this symbol. */

This routine is used to change one or more attributes of objects that have been previously displayed (through xmap_add_symbol ( )). "symbol," "fg_color," "bg_color," "class," "x," "y," "scale," "trace," and "fid" can be changed. More than one attribute may be changed in a single call to xmap_change_symbol ( ). For details on each attribute, see xmap_add_symbol ( ).

Return: Not used.
/*
   ~ xmap_remove_symbol ( )
   *
   * Removes a previously added symbol.
   */

void
xmap_remove_symbol (handle, client_id, conn)

int handle;        /* Handle id of object to change */
int client_id;     /* Client identification number. */
struct pkg_conn *conn        /* Application program’s connection to Xmap. */

This routine is used to remove previously drawn symbols from Xmap. “handle_id” and “client_id” are the same as was used with xmap_add_symbol ( ). “conn” is the client’s pkg connection structure.

Return: Not used.

/*
   ~ xmap_remove_objects ( )
   *
   * Routine to remove all objects of a particular type, or all classes of a
   * particular type of object.
   */

void
xmap_remove_objects (type, class, conn)

int type,            /* Type of object to remove */
class;              /* Class of object type to remove */
struct pkg_conn *conn;    /* Application program’s connection to Xmap. */

This routine is used to remove a group of objects identified by their “class” or more specifically by a particular “type” of a “class.” “type” and “class” definitions can be found in libxmap.h. “conn” is the client’s pkg connection structure.

Return: Not used.

/*
   ~ xmap_add_line ( )
   *
   * Routine used by application programs to draw lines on the map.
   * This routine draws NEW lines, or “objects” to Xmap, hence a handle id
   * must be associated with the line (object).
   */

int
xmap_add_line (client_id, fid, coords, conn)

int client_id;    /* Client identifier number. */
dkb_factid_t fid;    /* Fact id associated with this line. */
char *coords;    /* List of coords defining line segments. */
struct pkg_conn *conn;    /* Application program’s connection to Xmap. */
This routine is used for drawing a line object in Xmap. Lines are defined to be a set of line segments identified by the coordinates of the endpoints of each segment. “client_id” is the client identification number of the client that is drawing the line. “fid” is the IDT factid of the fact that is associated with the line being drawn. “coords” is a list of UTM coordinates that specify the endpoints of the line segments that comprise the line. “conn” is the client’s pkg connection structure.

Return: Handle id of new object (line).

/*
 * xmap_change_line ()
 *
 * Routine used by application programs to change already drawn lines on the
 * map. The complete list of new coordinates is specified.
 */

void xmap_change_line (client_id,, handle, fid, coords, conn)

This routine is used to modify an existing line in Xmap. The line to be modified is identified by “client_id” and “handle” (the handle of the line was assigned when the line was created by xmap_add_line () or xmap_convert_to_line () ). “fid” is the IDT factid of the fact that is associated with the line. Lines can only change by taking on a new shape; that is, line segments are added, deleted, or modified in the original line. The entire list of segments comprising the line must be stated in “coords,” not just the new or different segments. “conn” is the client’s pkg connection structure.

Return: Not used.

/*
 * xmap_convert_to_line ()
 *
 * This routine is called when application programs want to instruct Xmap
 * to convert a local line to a global line. Global lines are lines that
 * application programs know about and have assigned fact ids to.
 */

int xmap_convert_to_line (old_handle, old_client_id, new_client_id, fid, conn)

Free-hand creation of lines (that is, drawing lines using the mouse) is a function of Xmap. Xmap provides the user interface to allow for the creation of lines using the mouse to input
the endpoints of the line segments comprising the line. When a line is created this way, it is considered to be a local line in Xmap; that is, it is an object that has a handle id and client id that identify it with Xmap, not with a client application program. To interact with lines, application programs must convert them from local lines to application-owned lines via this xmap_convert_to_line() command. “old_handle” and “old_client_id” are the handle and client ids of a local line. “new_client_id” is the client identification number of the application program converting the line. A new handle will be assigned to this line and returned to the client. The line is now identified by “new_client_id” and the new handle. Hence, it is no longer a local line. “conn” is the client’s pkg connection structure.

Return: Handle id of line.

/*
 ~ xmap_remove_line()
 */

* * Removes a previously added line.
 */

void
 xmap_remove_line (handle, client_id, conn)

int handle; /* Handle id of object to change */
int client_id; /* Client identification number. */
struct pkg_conn *conn; /* Application program’s connection to Xmap. */

This routine is used to remove from Xmap a line that was previously drawn. It does not work on local lines (see explanation in xmap_convert_to_line()). “handle” and “client_id” are used to uniquely identify the line being removed. “handle” was assigned to the line via xmap_convert_to_line() or xmap_add_line(). “conn” is the client’s pkg connection structure.

Return: Not used.

/*
 ~ xmap_add_link()
 *
 * Routine used by application programs to draw links on the map.
 * This routine draws NEW links, or “objects” to xmap, hence a handle id
 * must be associated with the link (object).
 */

int
 xmap_add_link (handle_1, client_id_1, handle_2, client_id_2, fid, client_id, conn)

int handle_1, client_id_1; /* Handle/client id of first (from) object */
int handle_2, client_id_2; /* Handle/client id of second (to) object */
dkb_factid_t fid; /* Fact id associated with this link. */
int client_id; /* Client identifier number of link */
struct pkg_conn *conn; /* Application program’s connection to Xmap. */

This routine is used to draw a line between two objects to show that the objects are linked together. This link is not a line in the sense that a line is an object composed of one or more line segments as created by xmap_add_line() or xmap_convert_to_line(). Rather it is
simply intended as a means for showing that two objects are linked together in some fashion. The link follows the objects in the event that one of the object’s location changes. Xmap removes the link if either of the objects is removed. “handle_1” and “client_id_1” uniquely define the first object being linked and “handle_2” and “client_id_2” uniquely describe the second object being linked. “fid” is the IDT factid of the fact associated with the link. “client_id” is the client identification number of the application program stating the link and “conn” is the client’s pkg connection structure.

Return: Handle id of the link.

/**
 * ~ xmap_remove_link ( )
 *
 * Routine used by application programs to remove links on the map.
 */

void xmap_remove_link (handle, client_id, conn)

int handle; /* Handle of first link to remove */
int client_id; /* Client identifier number of link */
struct pkg_conn *conn /* Application program’s connection to Xmap. */

This routine is used to remove a link between two objects. “handle” is the handle id of the link that was returned by xmap_add_link ( ) when the link was established. “client_id” is the client identification number of the application program that initially established the link. “conn” is the client’s pkg connection structure.

Return: Not used.

/**
 * ~ xmap_add_association ( )
 *
 * Routine used by application programs to “bind” objects to other objects. An association can be formed between two objects linking them physically, although the link itself is invisible. The bind in effect groups the two objects in such a way that if one object is physically moved then the other object will maintain its relative position to the moved object.
 *
 * Objects that are associated can be associated in a MASTER_SLAVE or a GROUP relationship. In a MASTER_SLAVE association object 1 is slaved to object 2. If object 2 moves then object 1 follows it, but not vice versa. In a GROUP association the objects are treated as equals whereby if either object is moved the other object will follow it.
 *
 */

int xmap_add_association (handle_1, client_id_1, handle_2, client_id_2, client_id, relate, conn)

int handle_1, client_id_1; /* Handle/client id of first (from) object */
int handle_2, client_id_2; /* Handle/client id of second (to) object */
int client_id; /* Client id of program forming association */
This routine invisibly binds two objects to form an association based on the location of the objects relative to one another. “handle_1” and “client_id_1” uniquely describe the first, or ‘from,’ object in the association. “handle_2” and “client_id_2” uniquely describe the second, or ‘to,’ object in the association. “client_id” is the client identification number of the application program forming the association. “relate” describes the relationship between the two objects as described in the description above. “conn” is the client’s pkg connection structure.

Return: Handle id of the association.

/*
~ xmap_remove_association ()
*
* Routine used by application programs to remove associations between two
* objects.
*/
void
xmap_remove_association (handle, client_id, conn)

This routine removes a previously formed association between two objects. “handle” and “client_id” uniquely identify the association. “conn” is the client’s pkg connection structure.

Return: Not used.

/*
~ xmap_add_rangefan ()
*
* Routine used by application programs to draw range fans on the map.
* This routine draws NEW range fans, or “objects” to Xmap, hence a handle id
* must be associated with the range fan (object).
* *
* Return: handle id of new range fan.
*/
int
xmap_add_rangefan (unit_handle, unit_client, min_range, max_range, azimuth, trav_limits, client_id, conn)
int tray_limits; /* Traversal limits of range fan (in mils) */
int client_id; /* Client identifier number of range fan */
struct pkg_conn *conn; /* Application program's connection to Xmap */

This routine instructs Xmap to draw a range fan based on the minimum and maximum ranges ("min_range" and "max_range," respectively), the "azimuth" (direction of center-line of range fan), and traversal limits (width of range fan in mils in either direction off of the azimuth). "min_range" and "max_range" are specified in meters. The origin of the range fan is based on the location of the object denoted by "unit_handle" and "unit_client". "client_id" is the client identification number of the application program stating the range fan. "conn" is the pkg connection structure of the client submitting the command.

Return: Handle id of the range fan.

/*
~ xmap_remove_rangefan ()
* 
* Routine used by application programs to remove range fans from Xmap.
* /

void xmap_remove_rangefan (handle, client_id, conn)

This routine removes range fans from Xmap that were previously created using xmap_add_rangefan(). "handle" is the handle id assigned to the range fan when it was created. "client_id" is the client identification number of the client that created the range fan. "conn" is the pkg connection structure of the client submitting the command.

Return: Not used.
Server Commands

Below is a list of commands available to the server program, Xmap, for communicating with client application programs. For each command there is a brief explanation of its intended use as well as its syntax.

`/*
~_xmap_connect ( )
* Command used by Xmap to start listening for client connections.
*/

int
_xmap_connect ( )

This is an initialization routine called by Xmap to create a network server for the indicated service. "XMAP_PORT" is a user defined number and should appear in the /etc/services file. Once the network server is initialized, the server starts listening for new connections and polls any existing connections using the 'select' system call.

Return: The file descriptor 'fd' for the network server is returned upon success, otherwise a -1 is returned.

`/*
~_xmap_close ( )
* Command used by Xmap to inform application programs of an impending closing of the package connection.
*/

void
_xmap_close ( conn )

struct pkg_conn *conn; /* Application program's connection to Xmap */

"Conn" is the pkg connection structure for a client that has connected to Xmap. Xmap is equipped with a 'quit' button and in the event this button is pushed xmap sends a message to any attached clients that it is terminating. This allows client applications to disconnect gracefully.

Return: Not used.

`/*
~_xmap_send ( )
* Routine for Xmap to send information to application program.
*/

void
_xmap_send (msg_type, msg, conn)

int msg_type; /* Define (from libxmap.h) */
char *msg; /* Contents of message */
struct pkg_conn *conn; /* Application program's connection to Xmap */
"Mag_type" is a #define that specifies the type of message that is being sent to the client application. "Mag" is the actual message. "Conn" is the client’s pkg connection structure.

Return: Not used.

/*
- _xmap_button_press()
 *
* Routine for Xmap to notify application program of a button press.
*/

void _xmap_button_press (handle, connection, button, map_x, map_y, fid, conn)

int handle; /* Handle id of affected object */
int connection; /* Connection number of client */
int button; /* Number of button that was pressed */
int map_x, /* X (East) map grid location of button press */
    map_y; /* Y (North) map grid location of button press */
dbk_factid_t fid; /* Fact id of object pressed */
struct pkg_conn *conn; /* Application program’s connection to Xmap */

This routine is used to notify appropriate client applications of generic button press events. An appropriate client is one that indicated upon connecting that it wanted to be informed of any button events. A generic button press event includes a button press anywhere on the background or on a unit symbol. "Handle" is the unique identifier of the object selected by a client. "Connection" is the unique identifier of the client. These two attributes are needed for Xmap to distinguish one object from another. "Button" is a #define from libxmap.h that specifies which button was pressed. "Map_x" and "map_y" are the easting and northing map coordinates of the object that was selected. "Fid" is the IDT factid of the fact that is associated with the object that is selected. Sometimes this value will be NULL. "Conn" is the pkg connection of the client to notify.

Return: Not used.

/*
- _xmap_line_click()
 *
* Routine for Xmap to notify application program of a line click.
*/

void _xmap_line_click (handle, connection, button, fid, coords, conn)

int handle; /* Handle id of affected object */
int connection; /* Connection number of client */
int button; /* Button that was pressed to generate this event */
dbk_factid_t fid; /* Fact id of object pressed */
char *coords; /* A string containing all the grid coordinates */
/* Application program’s connection to Xmap */

struct pkg_conn *conn;

This routine is used to inform an appropriate client of a line selection. An appropriate client
is the same as defined in _xmap_button_press (). “Handle,” “connection,” “button,” “fid,”
and “conn” are the same as described in _xmap_button_press (). “Coords” is a string con-
sisting of all the UTM coordinates of the line segments that make up the line.

Return: Not used.

/*
 * _xmap_line_change ()
 *
 * Routine used by Xmap to notify application programs of changes to already
 * drawn lines. The complete list of new coordinates is specified.
 */

void _xmap_line_change (client_id, handle, fid, coords, conn)

This routine is used to notify an appropriate client of a change to a line. An appropriate
client is the same as described in _xmap_send(). Xmap gives the user the ability to edit an
existing line. When this is done, the appropriate clients must be informed of the changes
so they can update their database if necessary. Note that it is Xmap which provides the capa-
bility to edit lines, not client application programs. “Handle,” “connection,” “button,”
“fid,” and “conn” are the same as described in _xmap_button_press (). “Coords” is the
same as described in _xmap_line_click (). The coords string setup looks like: “x y; x y; ....”.
This setup allows for ease of parsing.

Return: Not used.

/*
 * _xmap_symbol_change ()
 *
 * Routine for Xmap to notify application program of a change in location
 * of a symbol that is the SLAVE in an association.
 */

void _xmap_symbol_change (handle, connection, map_x, map_y, fid, conn)
This routine is used to inform a client application program of a change in location of the SLAVE in an association (relationship: MASTER–SLAVE). For a more detailed description of associations see the description for xmap_add_association(). "Handle," "connection," "map_x," "map_y," "fid," and "conn" are the same as described in _xmap_button_press().

Return: Not used.

/*
~ _xmap_error ()
*
* Routine for Xmap to notify application program of an error in an action
* that the application program sent Xmap.
*/

void _xmap_error (handle, msg.type, conn)
int handle; /* Handle id of affected object */
int msg.type; /* Message type from application program */
struct pkg_conn *conn; /* Application program's connection to Xmap */

This routine is used to inform the clients of errors made in an action that the client sent to Xmap. This could include mixing up the order of attributes, sending inappropriate data types, specifying unknown objects, etc. It is the client's responsibility to correct the error condition. "Msg_type" and "conn" are the same as described in _xmap_send(). "Handle" is the same as described in _xmap_button_press().

Return: Not used.
INTENTIONALLY LEFT BLANK.
<table>
<thead>
<tr>
<th>ACRONYM</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACISD</td>
<td>Advanced Computational and Informational Sciences Directorate</td>
</tr>
<tr>
<td>ARL</td>
<td>Army Research Laboratory</td>
</tr>
<tr>
<td>CNR</td>
<td>Combat Net Radios</td>
</tr>
<tr>
<td>HELCAP</td>
<td>Human Engineering Laboratory Counter—Air Program</td>
</tr>
<tr>
<td>IDT</td>
<td>Information Distribution Technology</td>
</tr>
<tr>
<td>MCSB</td>
<td>Military Computer Science Branch</td>
</tr>
<tr>
<td>SWS</td>
<td>Smart Weapons Systems LABCOM Cooperative Program</td>
</tr>
<tr>
<td>No. of Copies</td>
<td>Organization</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| 2            | Administrator  
Defense Technical Info Center  
ATTN: DTIC-DDA  
Cameron Station  
Alexandria, VA 22304-6145 |
| 1            | Commander  
U.S. Army Materiel Command  
ATTN: AMCAM  
5001 Eisenhower Ave.  
Alexandria, VA 22333-0001 |
| 1            | Director  
U.S. Army Research Laboratory  
ATTN: AMSRL-OP-CI-AD,  
Tech Publishing  
2800 Powder Mill Rd.  
Adelphi, MD 20783-1145 |
| 1            | Director  
U.S. Army Research Laboratory  
ATTN: AMSRL-OP-CI-AD,  
Records Management  
2800 Powder Mill Rd.  
Adelphi, MD 20783-1145 |
| 2            | Commander  
U.S. Army Armament Research,  
Development, and Engineering Center  
ATTN: SMCAR-TDC  
Picatinny Arsenal, NJ 07806-5000 |
| 1            | Director  
Battle Weapons Laboratory  
U.S. Army Armament Research,  
Development, and Engineering Center  
ATTN: SMCAR-CCB-TL  
Waterloo, NY 12189-4050 |
| 1            | Director  
U.S. Army Advanced Systems Research  
and Analysis Office (ATCOM)  
ATTN: AMSAT-R-NR, M/S 219-1  
Ames Research Center  
Moffett Field, CA 94035-1000 |
|              | Commander  
U.S. Army Missile Command  
ATTN: AMSMI-RD-CS-R (DOC)  
Redstone Arsenal, AL 35898-5010 |
|              | Commander  
U.S. Army Tank-Automotive Command  
ATTN: AMSTA-JSK (Armor Eng. Br.)  
Warren, MI 48397-5000 |
|              | Director  
U.S. Army TRADOC Analysis Command  
ATTN: ATRC-WSR  
White Sands Missile Range, NM 88002-5502 |
|              | Commandant  
U.S. Army Infantry School  
ATTN: ATSH-CD (Security Mgr.)  
Fort Benning, GA 31905-5660 |
|              | Commandant  
U.S. Army Infantry School  
ATTN: ATSH-WCB-O  
Fort Benning, GA 31905-5000 |

Aberdeen Proving Ground  
2 Dir, USAMSAA  
ATTN: AMXSY-D  
AMXSY-MP, H. Cohen  
1 Cdr, USATECOM  
ATTN: AMSTE-TC  
1 Dir, USAERDEC  
ATTN: SCBRD-RT  
1 Cdr, USACBDCOM  
ATTN: AMSCB-CII  
1 Dir, USARL  
ATTN: AMSRL-SL-1  
5 Dir, USARL  
ATTN: AMSRL-OP-AP-L
<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aberdeen Proving Ground</td>
</tr>
<tr>
<td>12</td>
<td>Dir, USARL</td>
</tr>
<tr>
<td></td>
<td>ATTN: AMSRL-CI-C, Walter B. Sturek</td>
</tr>
<tr>
<td></td>
<td>AMSRL-CI-CC,</td>
</tr>
<tr>
<td></td>
<td>Samuel C. Chamberlain</td>
</tr>
<tr>
<td></td>
<td>Douglas A. Gwyn</td>
</tr>
<tr>
<td></td>
<td>George W. Hartwig, Jr.</td>
</tr>
<tr>
<td></td>
<td>Maria C. Lopez</td>
</tr>
<tr>
<td></td>
<td>Barry R. Reichard</td>
</tr>
<tr>
<td></td>
<td>AMSRL-CI-S, Andrew Mark</td>
</tr>
<tr>
<td></td>
<td>AMSRL-CI-SA, Eric G. Heilman</td>
</tr>
<tr>
<td></td>
<td>AMSRL-CI-SB,</td>
</tr>
<tr>
<td></td>
<td>Virginia A. Kaste</td>
</tr>
<tr>
<td></td>
<td>Janet F. O'May</td>
</tr>
<tr>
<td></td>
<td>AMSRL-SL-BG, Frederick S. Brundick</td>
</tr>
<tr>
<td></td>
<td>AMSRL-SL-BS, Howell Caton</td>
</tr>
</tbody>
</table>
This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number: ARL-TN-498  
Date of Report: August 1994

2. Date Report Received:

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.)

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.)

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate.

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.)

Organization

CURRENT ADDRESS
Name
Street or P.O. Box No.
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

Organization

OLD ADDRESS
Name
Street or P.O. Box No.
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)