

UNCLASSIFIED

AD-A501168
Copy 16 of 42 copies

AD-A208 055

2

IDA DOCUMENT D-575

AN Ada/SQL UNITREP DEMONSTRATION

DTIC
ELECTE
MAY 22 1989
S D D

Bill R. Brykczynski
Fred Friedman

February 1989

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

Prepared for
WIS Joint Program Management Office



INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

UNCLASSIFIED

IDA Log No. HQ 89-34164

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Documents is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This IDA Document is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

DISCLAIMER OF WARRANTY AND LIABILITY

This is experimental prototype software. It is provided "as is" without warranty or representation of any kind. The Institute for Defense Analyses (IDA) does not warrant, guarantee, or make any representations regarding this software with respect to correctness, accuracy, reliability, merchantability, fitness for a particular purpose, or otherwise.

Users assume all risks in using this software. Neither IDA nor anyone else involved in the creation, production, or distribution of this software shall be liable for any damage, injury, or loss resulting from its use, whether such damage, injury, or loss is characterized as direct, indirect, consequential, incidental, special, or otherwise.

Approved for public release: distribution unlimited.

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, unlimited distribution.		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE					
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Document D-575			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses		6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION OUSDA, DIMO		
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311			7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311		
8a NAME OF FUNDING/SPONSORING ORGANIZATION WIS Joint Program Management Office		8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031		
8c ADDRESS (City, State, and Zip Code) Room 5B19, The Pentagon Washington, D.C. 20330-6600			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. T-W5-206
11 TITLE (Include Security Classification) An Ada/SQL UNITREP Demonstration (U)					
12 PERSONAL AUTHOR(S) Bill R. Brykczynski, Fred Friedman					
13a TYPE OF REPORT Final		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 1989 February	
15 PAGE COUNT 122					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Ada, Structured Query Language (SQL), Ada/SQL, Database Management System (DBMS), language interface.		
19 ABSTRACT (Continue on reverse if necessary and identify by block number)					
The purpose of IDA Document D-575, <i>An Ada/SQL UNITREP Demonstration</i> , is to demonstrate how the Ada/Structured Query Language (SQL) Ada-Database Management System (DBMS) interface can be used to implement a portion of the UNITREP message processing application.					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a NAME OF RESPONSIBLE INDIVIDUAL Mr. Bill R. Brykczynski			22b TELEPHONE (Include area code) (703) 824-5515		22c OFFICE SYMBOL IDA/CSED

IDA DOCUMENT D-575

AN Ada/SQL UNITREP DEMONSTRATION

Bill R. Brykczynski
Fred Friedman

February 1989



INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-W5-206

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

UNCLASSIFIED

CONTENTS

Preface	iii
1. Purpose	1
2. Experience	1
3. Database Interface, Program Structure, and Software Engineering	2
4. References	3
5. Source Files	3
6. Demonstration Output	4
7. Source Listings	7
7.1 DATABASE.ADA	7
7.2 TXTPRT.ADA	9
7.3 FUNCTION.ADS	19
7.4 FUNCTION.ADB	25
7.5 DBTYPES.ADS	45
7.6 GENPACK.ADA	47
7.7 DBVARS.ADA	79
7.8 DBVARS.ADS	81
7.9 DBCGEN.ADA	83
7.10 DBCNVRT.ADS	89
7.11 CARDA.ADS	91
7.12 CARDA.ADB	93
7.13 MAIN.ADA	97
7.14 Demonstration Output	98

UNCLASSIFIED

Preface

The purpose of IDA Document D-575, *An Ada/SQL UNITREP Demonstration*, is to demonstrate how the Ada/SQL Ada-DBMS interface can be used to implement a portion of the UNITREP message processing application.

The importance of this document is based on partial fulfillment of Task Order T-W5-206, WIS Application Software Study, which is to demonstrate the general mechanics of developing Ada/SQL applications. As a Document, D-575 is directed toward users who are concerned with how an Ada/SQL system is implemented and operates.

UNCLASSIFIED

1. Purpose

This latest demonstration of Ada/SQL technology was designed to take advantage of the following new features:

- Strongest typing of effective functions, as described in the latest Ada/SQL specification [M362], but not implemented in earlier releases. Additional typing includes:
 - Strong typing of Ada/SQL syntactic classes,
 - Strong typing by table for update,
 - All other strong typing was, of course, retained.
- Support for null values, not previously implemented
- Implementation on an IBM¹ PC workstation, not requiring special hardware (the Meridian AdaVantage compiler was used, and the demonstration can be compiled and run on a PC with 640K of RAM)

Additional references to published reports on Ada/SQL can be found in [M362, M459, M460].

2. Experience

It was originally desired to implement the entire UNITREP application on the IBM PC. However, it quickly became apparent that limitations of the Meridian compiler prevented rapid implementation. Although the complete implementation would probably ultimately be feasible, the effort required would not be practical merely for a demonstration of this nature.

The particular limitations of the Meridian compiler that caused problems are:

- Nested generics (and even some aspects of non-nested generics) either do not compile at all or produce erroneous code. The existing Ada/SQL runtime system relies heavily on nested generics for generating its required subprograms. Consequently, existing code could not be directly used for the demonstration.
- Neither internal compiler tables nor the generated code are compact. Consequently, compilation and linking memory become a problem for large programs.

The problem with generics was circumvented by recoding key Ada/SQL runtime routines to not be generic. Although this simplifies the underlying, application-independent routines, it complicates the code that must be generated for application-specific references to Ada/SQL entities (e.g., statements, table/column names, etc.) Furthermore, the existing Ada/SQL application scanner generates code for the highly generic underlying implementation. Therefore, the application scanner could not be used to generate the code required for the demonstration, and that code had to be hand-generated.

1. IBM is a registered trademark of International Business Machines Corp.

UNCLASSIFIED

The memory space problem was circumvented by only implementing processing for one of the UNITREP card types, and by omitting all the ancillary message processing (not germane to database interface). Implementing the other card types would have required a level of effort beyond that which is reasonable for a simple demonstration anyway, due to the requirement to hand-generate all application-specific Ada/SQL code. Nevertheless, important lessons about database interface, program structure, and software engineering can be learned from the card type coded.

3. Database Interface, Program Structure, and Software Engineering

The original SORTS (renamed UNITREP) code relevant to this demonstration is extracted in CARDA.OLD. There are many auxiliary packages and subprograms used by the code shown but not included, since their functions can be deduced from the names by which they are referenced. (Program files are included at the end of this document, identified by file name, when the document is in printed form. When the document is in machine-readable form, program files are included on the disk under the names given.)

First, the fashion of the database interface should be noted. (These comments are made concerning database interfaces in general, not to be critical of the SORTS code. The SORTS implementation was designed to demonstrate features other than the database interface, on the IBM PC. The developers of the code did not have an actual IBM PC database interface available, and so their interface is meant to be illustrative rather than operational.)

In the original code, database commands are assembled component by component, with successive clauses appended to a string. This string is then provided to the database interface. Such an interface is fine for non-transaction-oriented, low volume applications, but has serious performance ramifications for high-volume, transaction-oriented systems. This is because the database management system must parse the string each time a command is issued, and then decide on optimal data access strategy. For simple database operations such as those used in the demonstration, this processing represents a very major portion of the total processing required to perform the command.

In our Ada/SQL version of the same code, contained in CARDA.ADB, we have replaced the string interface with database commands that can be processed at compile time. This enables much time-consuming database processing to be performed before the program actually runs, enabling implementation of high-performance applications. (The implementation used for the demonstration does not actually perform this database processing at compile time, but the same application code could be used in a system that would.)

Of course, building database commands in strings does provide a degree of flexibility exceeding that which is available with compilable commands. In the original SORTS code, this flexibility allows optional insert and update of column values. But is this flexibility really required to perform the operations? No; we accomplish the same results using null values. (At the time the original SORTS code was written, no Ada/SQL implementation supported null values.)

When building a command to insert data, the original SORTS code simply omits a column for which no data is supplied. But the database contains such a column, and the column must be filled with something, so what gets placed in such a column? In ANSI standard SQL, an unspecified column is filled with a null value. Our database commands, being fixed at compile time, must provide for inserting data into all columns of a table. But, by using the

UNCLASSIFIED

INDICATOR_VARIABLE feature of Ada/SQL, we achieve the same effect of placing null values into columns for which no data has been supplied.

When building a command to update data, the original SORTS code omits those columns for which the data are to remain unchanged. Again, since our database commands are fixed at compile time, we must provide for updating all columns of a table. In order to achieve the effect of leaving some columns unchanged, we first retrieve the existing data, then set those values back into the columns that should not be changed, using new data only for those columns that should be updated. Because of the overhead involved in parsing and planning data access strategy, our retrieve/update done in a precompiled fashion should provide better performance than a single update done via a string interface.

The Ada/SQL interface provides other advantages as well, relating to program structure and software engineering. Much of the logic in the original SORTS code is devoted to building database command strings. The relationships between input format and database structure are somewhat obscured by all the command string logic. In the Ada/SQL version, each database command is a single Ada statement. That, coupled with the structure we have built for converting from input format to program variables, makes the relationship between input and database immediately apparent. This clarity of program operation has obvious value for reliability and maintainability.

In addition to the visual formatting advantage, the Ada/SQL approach provides strong type checking. In our demonstration, input data are converted from "card images". The converted data are placed in program variables which are appropriately typed. All database operations on those program variables then provide strong type checking. For example, a value of one type cannot be used to update a column of another. Contrariwise, there is no strong type control with a string interface. Once a value is placed in a command string, all type information is lost, and the database interface has no way of verifying correct typing.

4. References

- [M362] *Ada/SQL Binding Specifications*, F. Friedman and B. Brykczynski. IDA Memorandum Report M-362. June 1988. VA: Institute for Defense Analyses.
- [M459] *An Oracle - Ada/SQL Implementation*, F. Friedman, B. Brykczynski and K. Hilliard. IDA Memorandum Report M-459, April, 1988. VA: Institute for Defense Analyses.
- [M460] *An Ada/SQL Application Scanner*, F. Friedman, K. Heatwole, K. Hilliard and B. Brykczynski. IDA Memorandum Report M-460, March 1988. VA: Institute for Defense Analyses.

5. Source Files

The source files for the demonstration, listed in an order suitable for compilation, are:

DATABASE.ADA - type definitions used by the Ada/SQL system (as defined in the Ada/SQL

UNCLASSIFIED

specification, declares types indicating the maximum range of values supported by the database).

TXTPRT.ADA - utility routines for printing and formatting text.

FUNCTION.ADS and **FUNCTION.ADB** - the underlying application-independent Ada/SQL runtime implementation environment, adapted specifically for this demonstration by recoding without generics. Ada/SQL features unused by the demonstration are not included in these files, to save the unnecessary work of recoding unused items to also omit generics. Emulation of the specific database commands used in the demonstration is provided - values inserted and updated are retained and may be retrieved or deleted. This emulation applies only to the exact commands used in the demonstration, however, and will not work on other database commands. When commands are performed, their operation is also printed, so that the results of the demonstration can be seen.

DBTYPES.ADS - type definitions for UNITREP data used in the demonstration.

GENPACK.ADA - the "generated package" for CARDA.ADB (see below), defining the application-dependent Ada/SQL environment. In the existing Ada/SQL implementations, this is generated by the application scanner. For this demonstration, however, it was hand-generated, as already noted, to circumvent problems with the Meridian compiler.

DBVARS.ADA - the "generated package" for DBVARS.ADB (see below).

DBVARS.ADS - definitions of variables used for database communication in the demonstration (variables correspond to fields in the UNITREP input).

DBCGEN.ADA - generic routines to convert data from the UNITREP "card images" to variables of the appropriate type. Conversions are provided for discrete (integer and enumeration), string, and flag (blank or X on UNITREP "card image", converted to BOOLEAN for use by Ada program) data types. Two kinds of conversions are provided, according to the following rules: For inserting discrete and string data, a blank field causes a null value to be used in the database, otherwise the given data is used. For inserting flag data, a blank is FALSE and an X is TRUE. For updating discrete and string data, a pound sign (#) causes an update to a null value, a blank field causes the original data to not be updated, and anything else causes an update to the given value. For updating flag data, a pound sign causes an update to FALSE, a blank causes the original data to not be updated, and an X causes an update to TRUE.

DBCNVRT.ADS - instantiations of the conversion routines in DBCGEN.ADA for the types used in the UNITREP demonstration.

CARDA.ADS and **CARDA.ADB** - code to process a single UNITREP card type, based on the original SORTS code, but totally rewritten to use Ada/SQL.

MAIN.ADA - driver for the demonstration; contains several UNITREP "card images" which are echoed and fed to the processing routine in sequence.

6. Demonstration Output

MAIN.EXE contains the demonstration program, which can be run on virtually any IBM PC

UNCLASSIFIED

or compatible. The output produced by this program is in DEMO.OUT. (The demonstration normally produces output on the screen, but the output can be redirected to a file.) The output shows the database operations performed, in ANSI SQL, and the results retrieved by SELECs. Enumeration values show up as integers in the output, according to their 'POS. This is because Ada/SQL is designed to interface to existing database management systems which do not, in general, support enumeration types. Hence, enumeration values are converted to integers for interchange with the database management system. This conversion is performed by the interface; the application program treats enumeration values as it should (as enumeration values!).

UNCLASSIFIED

7. Source Listings

7.1 DATABASE.ADA

```
-- File: database.ads
-- DATABASE
-- 12/18/88
```

```
--
--
--          DISCLAIMER OF WARRANTY AND LIABILITY
--
--
```

```
-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
```

```
-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
```

```
-- Prepared for:
```

```
-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311
```

```
-- Prepared by:
```

```
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813      (703) 560-6799
```

```
--
with SYSTEM;
package DATABASE is
  type INT          is range SYSTEM.MIN_INT .. SYSTEM.MAX_INT;
  type DOUBLE_PRECISION is new STANDARD.FLOAT;
  type CHAR         is new STANDARD.STRING;
  type USER_AUTHORIZATION_IDENTIFIER is new STANDARD.STRING;
  type COLUMN_NUMBER is new STANDARD.INTEGER;
```

UNCLASSIFIED

end DATABASE;

UNCLASSIFIED

7.2 TXTPRT.ADA

```
-- File: txtprt.ada
-- TEXT_PRINT - specially tailored for UNITREF demo - not for other use
-- 12/18/88
```

```
--
--
--                DISCLAIMER OF WARRANTY AND LIABILITY
--
--
```

```
-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
--
```

```
-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
--
```

```
-- Prepared for:
```

```
-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311
--
```

```
-- Prepared by:
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813      (703) 560-6799
--
```

```
with DATABASE, TEXT_IO;
  use TEXT_IO;
package TEXT_PRINT is
```

```
  type LINE_TYPE is limited private;
```

```
  type BREAK_TYPE is (BREAK, NO_BREAK);
```

```
  type PHANTOM_TYPE is private;
```

UNCLASSIFIED

```

procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE);

procedure SET_LINE(LINE : in LINE_TYPE);

function CURRENT_LINE return LINE_TYPE;

procedure SET_INDENT(LINE : in LINE_TYPE; INDENT : in NATURAL);
procedure SET_INDENT(INDENT : in NATURAL);

procedure SET_CONTINUATION_INDENT(LINE : in LINE_TYPE;
                                   INDENT : in INTEGER);
procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER);

function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE;

procedure SET_PHANTOMS(LINE : in LINE_TYPE;
                      START_PHANTOM,
                      END_PHANTOM : in PHANTOM_TYPE);

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE);

procedure PRINT(FILE : in FILE_TYPE;
               LINE : in LINE_TYPE;
               ITEM : in STRING;
               BRK : in BREAK_TYPE := BREAK);
procedure PRINT(FILE : in FILE_TYPE;
               ITEM : in STRING;
               BRK : in BREAK_TYPE := BREAK);
procedure PRINT(LINE : in LINE_TYPE;
               ITEM : in STRING;
               BRK : in BREAK_TYPE := BREAK);
procedure PRINT(ITEM : in STRING;
               BRK : in BREAK_TYPE := BREAK);

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure PRINT_LINE(FILE : in FILE_TYPE);
procedure PRINT_LINE(LINE : in LINE_TYPE);
procedure PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure BLANK_LINE(FILE : in FILE_TYPE);
procedure BLANK_LINE(LINE : in LINE_TYPE);
procedure BLANK_LINE;

--generic ***** hard-wire for Meridian Ada
-- type NUM is range <>;
subtype NUM1 is DATABASE.INT;
package INTEGER_PRINT is

    procedure PRINT(FILE : in FILE_TYPE;
                   LINE : in LINE_TYPE;

```

UNCLASSIFIED

```

        ITEM : in NUMI;
        BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(FILE : in FILE_TYPE;
        ITEM : in NUMI;
        BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(LINE : in LINE_TYPE;
        ITEM : in NUMI;
        BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(ITEM : in NUMI;
        BRK  : in BREAK_TYPE := BREAK);

procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUMI);

end INTEGER_PRINT;

--generic ***** hard-wire for Meridian Ada
-- type NUM is digits <>;
subtype NUMF is DATABASE.DOUBLE_PRECISION;
package FLOAT_PRINT is

    procedure PRINT(FILE : in FILE_TYPE;
        LINE : in LINE_TYPE;
        ITEM : in NUMF;
        BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(FILE : in FILE_TYPE;
        ITEM : in NUMF;
        BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(LINE : in LINE_TYPE;
        ITEM : in NUMF;
        BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(ITEM : in NUMF;
        BRK  : in BREAK_TYPE := BREAK);

    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUMF);

end FLOAT_PRINT;

NULL_PHANTOM : constant PHANTOM_TYPE;

LAYOUT_ERROR : exception renames TEXT_IO.LAYOUT_ERROR;

private

type PHANTOM_TYPE is access STRING;

type LINE_REC(LENGTH : INTEGER) is
    record
        USED_YET          : BOOLEAN := FALSE;
        INDENT            : INTEGER := 0;
        CONTINUATION_INDENT : INTEGER := 2;
        BREAK             : INTEGER := 1;
    end record;

```

UNCLASSIFIED

```

INDEX          : INTEGER := 1;
DATA           : STRING(1..LENGTH);
START_PHANTOM,
END_PHANTOM    : PHANTOM_TYPE := NULL_PHANTOM;
end record;

type LINE_TYPE is access LINE_REC;

NULL_PHANTOM : constant PHANTOM_TYPE := new STRING('');

end TEXT_PRINT;

package body TEXT_PRINT is

    DEFAULT_LINE : LINE_TYPE;

    procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE) is
    begin
        LINE := new LINE_REC(LENGTH);
    end CREATE_LINE;

    procedure SET_LINE(LINE : in LINE_TYPE) is
    begin
        DEFAULT_LINE := LINE;
    end SET_LINE;

    function CURRENT_LINE return LINE_TYPE is
    begin
        return DEFAULT_LINE;
    end CURRENT_LINE;

    procedure SET_INDENT(LINE : in LINE_TYPE; INDENT : in NATURAL) is
    begin
        if INDENT >= LINE.LENGTH then
            raise TEXT_IO.LAYOUT_ERROR;
        end if;
        if LINE.INDEX = LINE.INDENT + 1 then
            for I in 1..INDENT loop
                LINE.DATA(I) := ' ';
            end loop;
            LINE.INDEX := INDENT + 1;
        end if;
        LINE.INDENT := INDENT;
    end SET_INDENT;

    procedure SET_INDENT(INDENT : in NATURAL) is
    begin
        SET_INDENT(DEFAULT_LINE, INDENT);
    end SET_INDENT;

    procedure SET_CONTINUATION_INDENT(LINE : in LINE_TYPE;

```

UNCLASSIFIED

```

                                INDENT : in INTEGER) is
begin
  if LINE.INDENT + INDENT >= LINE.LENGTH or else LINE.INDENT + INDENT < 0
  then
    raise TEXT_IO.LAYOUT_ERROR;
  end if;
  LINE.CONTINUATION_INDENT := INDENT;
end SET_CONTINUATION_INDENT;

procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER) is
begin
  SET_CONTINUATION_INDENT(DEFAULT_LINE,INDENT);
end SET_CONTINUATION_INDENT;

function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE is
begin
  return new STRING'(S);
end MAKE_PHANTOM;

procedure SET_PHANTOMS(LINE           : in LINE_TYPE;
                       START_PHANTOM,
                       END_PHANTOM   : in PHANTOM_TYPE) is
begin
  LINE.START_PHANTOM := START_PHANTOM;
  LINE.END_PHANTOM  := END_PHANTOM;
end SET_PHANTOMS;

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE) is
begin
  SET_PHANTOMS(DEFAULT_LINE,START_PHANTOM,END_PHANTOM);
end SET_PHANTOMS;

procedure PRINT(FILE : in FILE_TYPE;
                LINE  : in LINE_TYPE;
                ITEM   : in STRING;
                BRK    : in BREAK_TYPE := BREAK) is
  NEW_BREAK, NEW_INDEX : INTEGER;
begin
  if LINE.INDEX + ITEM'LENGTH + LINE.END_PHANTOM'LENGTH > LINE.LENGTH + 1
  then
    if LINE.INDENT + LINE.CONTINUATION_INDENT + LINE.START_PHANTOM'LENGTH +
      LINE.INDEX - LINE.BREAK + ITEM'LENGTH > LINE.LENGTH then
      raise TEXT_IO.LAYOUT_ERROR;
    end if;
    if ITEM = " " and then LINE.END_PHANTOM.all = " " then
      return;
    end if;
    PUT_LINE(FILE,LINE.DATA(1..LINE.BREAK-1) & LINE.END_PHANTOM.all);
    for I in 1..LINE.INDENT + LINE.CONTINUATION_INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
  end if;
end PRINT;

```

UNCLASSIFIED

```

NEW_BREAK := LINE.INDENT + LINE.CONTINUATION_INDENT + 1;
NEW_INDEX := NEW_BREAK + LINE.START_PHANTOM'LENGTH +
    LINE.INDEX - LINE.BREAK;
LINE.DATA(NEW_BREAK..NEW_INDEX-1) := LINE.START_PHANTOM.all &
    LINE.DATA(LINE.BREAK..LINE.INDEX-1);
LINE.BREAK := NEW_BREAK;
LINE.INDEX := NEW_INDEX;
end if;
NEW_INDEX := LINE.INDEX + ITEM'LENGTH;
LINE.DATA(LINE.INDEX..NEW_INDEX-1) := ITEM;
LINE.INDEX := NEW_INDEX;
if BRK = BREAK then
    LINE.BREAK := NEW_INDEX;
end if;
LINE.USED_YET := TRUE;
end PRINT;

procedure PRINT(FILE : in FILE_TYPE;
    ITEM : in STRING;
    BRK : in BREAK_TYPE := BREAK) is
begin
    PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(LINE : in LINE_TYPE;
    ITEM : in STRING;
    BRK : in BREAK_TYPE := BREAK) is
begin
    PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in STRING; BRK : in BREAK_TYPE := BREAK) is
begin
    PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
    if LINE.INDEX /= LINE.INDENT + 1 then
        PUT_LINE(FILE,LINE.DATA(1..LINE.INDEX-1));
    end if;
    for I in 1..LINE.INDENT loop
        LINE.DATA(I) := ' ';
    end loop;
    LINE.INDEX := LINE.INDENT + 1;
    LINE.BREAK := LINE.INDEX;
end PRINT_LINE;

procedure PRINT_LINE(FILE : in FILE_TYPE) is
begin
    PRINT_LINE(FILE,DEFAULT_LINE);

```

UNCLASSIFIED

```
end PRINT_LINE;

procedure PRINT_LINE(LINE : in LINE_TYPE) is
begin
  PRINT_LINE(CURRENT_OUTPUT, LINE);
end PRINT_LINE;

procedure PRINT_LINE is
begin
  PRINT_LINE(CURRENT_OUTPUT, DEFAULT_LINE);
end PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if LINE.USED_YET then
    NEW_LINE(FILE);
  end if;
end BLANK_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE) is
begin
  BLANK_LINE(FILE, DEFAULT_LINE);
end BLANK_LINE;

procedure BLANK_LINE(LINE : in LINE_TYPE) is
begin
  BLANK_LINE(CURRENT_OUTPUT, LINE);
end BLANK_LINE;

procedure BLANK_LINE is
begin
  BLANK_LINE(CURRENT_OUTPUT, DEFAULT_LINE);
end BLANK_LINE;

package body INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUMI;
                  BRK : in BREAK_TYPE := BREAK) is
    S : STRING(1..NUMI'WIDTH);
    L : NATURAL;
  begin
    PRINT(S, L, ITEM);
    PRINT(FILE, LINE, S(1..L), BRK);
  end PRINT;

  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUMI;
                  BRK : in BREAK_TYPE := BREAK) is
  begin
```

UNCLASSIFIED

```

    PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in NUMI;
                BRK : in BREAK_TYPE := BREAK) is
begin
    PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in NUMI;
                BRK : in BREAK_TYPE := BREAK) is
begin
    PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUMI) is
    S : constant STRING := NUMI'IMAGE(ITEM);
    F : NATURAL := S'FIRST; -- Bug in DG Compiler -- S'FIRST /= 1 ! ! ! ! !
    L : NATURAL;
begin
    if S(F) = ' ' then
        F := F + 1;
    end if;
    if TO'LENGTH < S'LAST - F + 1 then
        raise TEXT_IO.LAYOUT_ERROR;
    end if;
    L := TO'FIRST + S'LAST - F;
    TO(TO'FIRST..L) := S(F..S'LAST);
    LAST := L;
end PRINT;

end INTEGER_PRINT;

package body FLOAT_PRINT is

    package NUM_IO is new FLOAT_IO(NUMF);
    use NUM_IO;

    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUMF;
                    BRK : in BREAK_TYPE := BREAK) is
        S : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
        L : NATURAL;
    begin
        PRINT(S,L,ITEM);
        PRINT(FILE,LINE,S(1..L),BRK);
    end PRINT;

    procedure PRINT(FILE : in FILE_TYPE;

```

UNCLASSIFIED

```

ITEM : in NUMF;
BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(LINE : in LINE_TYPE;
ITEM : in NUMF;
BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in NUMF;
BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUMF) is
  S : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
  EXP : INTEGER;
  E_INDEX : NATURAL := S'LAST - DEFAULT_EXP;
  DOT_INDEX : NATURAL := DEFAULT_FORE + 1;
  L : NATURAL;
begin
  PUT(S,ITEM);
  EXP := INTEGER'VALUE(S(E_INDEX+1..S'LAST));
  if EXP >= 0 then
    if EXP <= DEFAULT_AFT-1 then
      S(DOT_INDEX..DOT_INDEX+EXP-1) := S(DOT_INDEX+1..DOT_INDEX+EXP);
      S(DOT_INDEX+EXP) := '.';
      for I in E_INDEX..S'LAST loop
        S(I) := ' ';
      end loop;
    end if;
  else -- EXP < 0
    if EXP >= - ( DEFAULT_EXP + 1 ) then
      S(DEFAULT_EXP+2..S'LAST) := S(1..S'LAST-DEFAULT_EXP-1);
      for I in 1..DEFAULT_EXP+1 loop
        S(I) := ' ';
      end loop;
      E_INDEX := S'LAST + 1;
      DOT_INDEX := DOT_INDEX + DEFAULT_EXP + 1;
      L := DOT_INDEX+EXP;
      for I in reverse L+1..DOT_INDEX loop
        case S(I-1) is
          when ' ' => S(I) := '0';
          when '-' => S(I-2) := '-'; S(I) := '0';
          when others => S(I) := S(I-1);
        end case;
      end loop;
    end if;
  end if;
end PRINT;

```

UNCLASSIFIED

```

end loop;
S(L) := '.';
case S(L-1) is
  when ' ' => S(L-1) := '0';
  when '-' => S(L-2) := '-'; S(L-1) := '0';
  when others => null;
end case;
end if;
end if;
for I in reverse 1..E_INDEX-1 loop
  exit when S(I) /= '0' or else S(I-1) = '.';
  S(I) := ' ';
end loop;
L := TO'FIRST - 1;
for I in S'RANGE loop
  if S(I) /= ' ' then
    L := L + 1;
    TO(L) := S(I);
  end if;
end loop;
LAST := L;
exception
  when CONSTRAINT_ERROR =>
    raise TEXT_IO.LAYOUT_ERROR;
end PRINT;

end FLOAT_PRINT;

end TEXT_PRINT;

```

UNCLASSIFIED

7.3 FUNCTION.ADS

```
-- File: function.ads
-- ADA_SQL_FUNCTIONS - specially tailored for UNITREP demo - not for other use
-- 12/18/88
```

```
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813      (703) 560-6799
```

```
with DATABASE;
package ADA_SQL_FUNCTIONS is
```

```
    procedure INITIATE_TEST;
```

```
    INTERNAL_ERROR,
    NO_DATA : exception;
```

```
    type EXTENDED_INDICATOR is ( NO_INDICATOR , NULL_VALUE , NOT_NULL );
```

```
    subtype INDICATOR_VARIABLE is EXTENDED_INDICATOR
    range NULL_VALUE .. NOT_NULL;
```

```
    type SQL_OPERATION is
```

```
    ( O_AVG           , O_MAX           , O_MIN           , O_SUM           ,
      O_UNARY_PLUS     , O_UNARY_MINUS , O_PLUS           , O_MINUS         ,
      O_TIMES          , O_DIVIDE        , O_EQ             , O_NE            ,
      O_LT             , O_GT            , O_LE             , O_GE            ,
      O_BETWEEN        , O_AND           , O_IS_IN          , O_OR            ,
      O_NOT             , O LIKE          , O_AMPERSAND      , O_SELEC         ,
      O_SELECT_DISTINCT , O_ASC           , O_DESC           , O_TABLE_COLUMN_LIST ,
      O_COUNT_STAR     , O_NULL_OP       , O_STAR           , O_NOT_IN        ,
      O_VALUES         , O_DECLAR );
```

```
    type TYPED_SQL_OBJECT      is private;
    type SQL_OBJECT             is private;
    type TABLE_NAME           is private;
    type FROM_CLAUSE           is private;
    type INSERT_VALUE_LIST      is private;
    type INSERT_VALUE_LIST_STARTER is private;
    type SEARCH_CONDITION       is private;
    type SELECT_LIST            is private;
    type VALUE_EXPRESSION       is private;
    type VALUE_SPECIFICATION    is private;
    type CURSOR_NAME            is private;
```

```
    NULL_SQL_OBJECT      : constant SQL_OBJECT;
```

UNCLASSIFIED

```
NULL_SEARCH_CONDITION : constant SEARCH_CONDITION;
NULL_CURSOR_NAME      : constant CURSOR_NAME;

-- conversion routines for user types

function INTEGER_AND_ENUMERATION_CONVERT ( VAR : DATABASE.INT )
return SQL_OBJECT;

function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : STANDARD.STRING )
return SQL_OBJECT;

-- column and table name routines

function COLUMN_OR_TABLE_NAME ( GIVEN_NAME : STANDARD.STRING )
return SQL_OBJECT;

-- value specification routines

function INDICATOR_FUNCTION
( VALUE : SQL_OBJECT;
  IND   : INDICATOR_VARIABLE := NOT_NULL ) return SQL_OBJECT;

-- operation routines

function BINARY_OPERATION
( GIVEN_OPERATION : SQL_OPERATION;
  L                : SQL_OBJECT;
  R                : SQL_OBJECT )
return SQL_OBJECT;

-- delete routines

procedure DELETE
( FROM   : in TABLE_NAME;
  WHERE  : in SEARCH_CONDITION := NULL_SEARCH_CONDITION );

-- into routines

procedure INTEGER_AND_ENUMERATION_INTO
( TARGET      : out DATABASE.INT;
  INDICATOR    : out INDICATOR_VARIABLE;
  CURSOR       : in CURSOR_NAME := NULL_CURSOR_NAME );

procedure INTEGER_AND_ENUMERATION_INTO
( TARGET      : out DATABASE.INT;
  CURSOR       : in CURSOR_NAME := NULL_CURSOR_NAME );

procedure CONSTRAINED_STRING_INTO
( TARGET      : out STANDARD.STRING;
  LAST        : out STANDARD.NATURAL;
  INDICATOR    : out INDICATOR_VARIABLE;
```

UNCLASSIFIED

```

        CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME );

procedure CONSTRAINED_STRING_INTO
( TARGET      : out STANDARD.STRING;
  LAST        : out STANDARD.NATURAL;
  CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME );

-- insert into routines

procedure INSERT_INTO
( TABLE      : in TABLE_NAME;
  VALUES     : in INSERT_VALUE_LIST );

function VALUES return INSERT_VALUE_LIST_STARTER;

-- select statement routines

-- see above for into routines

procedure SELECT_LIST_SELECT
( SELECT_TYPE : in SQL_OPERATION;
  WHAT        : in SQL_OBJECT;
  FROM        : in FROM_CLAUSE;
  WHERE       : in SEARCH_CONDITION := NULL_SEARCH_CONDITION );

-- update routines

procedure UPDATE_PROCEDURE
( TABLE : in SQL_OBJECT;
  SET     : in SQL_OBJECT;
  WHERE   : in SEARCH_CONDITION := NULL_SEARCH_CONDITION );

-- conversion routines for SQL objects

function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT;

function R_CONVERT ( R : TYPED_SQL_OBJECT ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT;

package CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT;

    function R_CONVERT ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT;

```

UNCLASSIFIED

```
function R_CONVERT ( R : TABLE_NAME ) return SQL_OBJECT renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME;

function L_CONVERT ( L : FROM_CLAUSE ) return SQL_OBJECT;

function R_CONVERT ( R : FROM_CLAUSE ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return FROM_CLAUSE;

function L_CONVERT ( L : INSERT_VALUE_LIST ) return SQL_OBJECT;

function R_CONVERT ( R : INSERT_VALUE_LIST ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return INSERT_VALUE_LIST;

function L_CONVERT ( L : INSERT_VALUE_LIST_STARTER ) return SQL_OBJECT;

function R_CONVERT ( R : INSERT_VALUE_LIST_STARTER ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return INSERT_VALUE_LIST_STARTER;

function L_CONVERT ( L : SEARCH_CONDITION ) return SQL_OBJECT;

function R_CONVERT ( R : SEARCH_CONDITION ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return SEARCH_CONDITION;

function L_CONVERT ( L : SELECT_LIST ) return SQL_OBJECT;

function R_CONVERT ( R : SELECT_LIST ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return SELECT_LIST;

function L_CONVERT ( L : VALUE_EXPRESSION ) return SQL_OBJECT;

function R_CONVERT ( R : VALUE_EXPRESSION ) return SQL_OBJECT
renames L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return VALUE_EXPRESSION;

function L_CONVERT ( L : VALUE_SPECIFICATION ) return SQL_OBJECT;

function R_CONVERT ( R : VALUE_SPECIFICATION ) return SQL_OBJECT
renames L_CONVERT;
```

UNCLASSIFIED

```

function CONVERT_R ( R : SQL_OBJECT ) return VALUE_SPECIFICATION;

end CONVERT;

private

type DATABASE_NAME is access STANDARD.STRING;
type ACCESS_STRING is access STANDARD.STRING;

type SQL_VALUE_KIND is ( K_INTEGER , K_FLOAT , K_STRING );

type SQL_VALUE ( KIND : SQL_VALUE_KIND := K_INTEGER ) is
  record
    case KIND is
      when K_INTEGER =>
        INTEGER : DATABASE.INT;
      when K_FLOAT =>
        FLOAT : DATABASE.DOUBLE_PRECISION;
      when K_STRING =>
        STRING : ACCESS_STRING;
    end case;
  end record;

type SQL_OBJECT_KIND is ( NAME , VALUE , OPERATION );

type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND );
type TYPED_SQL_OBJECT is access SQL_OBJECT_RECORD;
type SQL_OBJECT          is new TYPED_SQL_OBJECT;
type TABLE_NAME         is new TYPED_SQL_OBJECT;
type FROM_CLAUSE         is new TYPED_SQL_OBJECT;
type INSERT_VALUE_LIST   is new TYPED_SQL_OBJECT;
type INSERT_VALUE_LIST_STARTER is new TYPED_SQL_OBJECT;
type SEARCH_CONDITION     is new TYPED_SQL_OBJECT;
type SELECT_LIST         is new TYPED_SQL_OBJECT;
type VALUE_EXPRESSION     is new TYPED_SQL_OBJECT;
type VALUE_SPECIFICATION is new TYPED_SQL_OBJECT;

type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND ) is
  record
    ACROSS : SQL_OBJECT;
    case KIND is
      when NAME =>
        NAME : DATABASE_NAME;
      when VALUE =>
        INDICATOR : EXTENDED_INDICATOR;
        VALUE : SQL_VALUE;
      when OPERATION =>
        OPERATION : SQL_OPERATION;
        OPERANDS : SQL_OBJECT;
    end case;
  end record;

```

UNCLASSIFIED

```
type CURSOR_NAME is new TYPED_SQL_OBJECT;

NULL_SQL_OBJECT      : constant SQL_OBJECT      := null;
NULL_SEARCH_CONDITION : constant SEARCH_CONDITION := null;
NULL_CURSOR_NAME     : constant CURSOR_NAME     := null;

end ADA_SQL_FUNCTIONS;
```

UNCLASSIFIED

7.4 FUNCTION.ADB

```
-- File: function.adb
-- ADA_SQL_FUNCTIONS - specially tailored for UNITREP demo - not for other use
-- 12/18/88
```

```
--
--
--          DISCLAIMER OF WARRANTY AND LIABILITY
--
--
```

```
-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
--
```

```
-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
--
```

```
-- Prepared for:
```

```
--
-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311
--
```

```
-- Prepared by:
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813      (703) 560-6799
--
```

```
with TEXT_PRINT;
use TEXT_PRINT;
package body ADA_SQL_FUNCTIONS is
```

```
    BIDE_TABLE,
    COMMAND_TABLE : SQL_OBJECT :=
        new SQL_OBJECT_RECORD' ( OPERATION , null , O_VALUES , null );
```

```
    INTO_COLUMN : SQL_OBJECT;
```

UNCLASSIFIED

```

INDENT : STANDARD.INTEGER;

package INT_PRINT renames INTEGER_PRINT;

use INT_PRINT;

LINE : LINE_TYPE;

-- declarations for print routines (since some are recursive and mutually
-- recursive)

procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT );
procedure SHOW_ALL_SET_FUNCTION    ( S : in SQL_OBJECT );
procedure SHOW_VALUE_EXPRESSION    ( S : in SQL_OBJECT );
procedure SHOW_BETWEEN_PREDICATE   ( S : in SQL_OBJECT );
procedure SHOW_IN_VALUE_LIST       ( S : in SQL_OBJECT );
procedure SHOW_LIKE_PREDICATE      ( S : in SQL_OBJECT );
procedure SHOW_SEARCH_CONDITION    ( S : in SQL_OBJECT );
procedure SHOW_TABLE_EXPRESSION    ( S : in SQL_OBJECT );
procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT );
procedure SHOW_SELECT_LIST         ( S : in SQL_OBJECT );
procedure SHOW_ORDER_BY_CLAUSE     ( S : in SQL_OBJECT );
procedure SHOW_INSERT_VALUE_LIST   ( S : in SQL_OBJECT );
procedure SHOW_SET_CLAUSES         ( S : in SQL_OBJECT );
procedure SHOW_COMPARISON_PREDICATE
    ( S : in SQL_OBJECT ; P : in STANDARD.STRING );
procedure SHOW_IN_PREDICATE
    ( S : in SQL_OBJECT ; P : in STANDARD.STRING );

procedure INITIATE_TEST is
begin
    CREATE_LINE ( LINE , 79 );
    SET_LINE ( LINE );
    SET_CONTINUATION_INDENT ( 7 );
end INITIATE_TEST;

-- conversion routines for user types

function INTEGER_AND_ENUMERATION_CONVERT ( VAR : DATABASE.INT )
return SQL_OBJECT is
begin
    return new
        SQL_OBJECT_RECORD' ( VALUE , null , NO_INDICATOR , ( K_INTEGER , VAR ) );
end INTEGER_AND_ENUMERATION_CONVERT;

function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : STANDARD.STRING )
return SQL_OBJECT is
begin
    return new
        SQL_OBJECT_RECORD'
        ( VALUE , null , NO_INDICATOR,

```

UNCLASSIFIED

```

        ( K_STRING , new STANDARD.STRING' ( VAR ) ) );
end CONSTRAINED_CHARACTER_STRING_CONVERT;

-- column and table name routines

function COLUMN_OR_TABLE_NAME ( GIVEN_NAME : STANDARD.STRING )
return SQL_OBJECT is
begin
    return new
        SQL_OBJECT_RECORD'
        ( NAME , null , new STANDARD.STRING' ( GIVEN_NAME ) );
end COLUMN_OR_TABLE_NAME;

-- value specification routines

function INDICATOR_FUNCTION
( VALUE : SQL_OBJECT;
  IND   : INDICATOR_VARIABLE := NOT_NULL ) return SQL_OBJECT is
begin
    VALUE.INDICATOR := IND;
    return VALUE;
end INDICATOR_FUNCTION;

function COPY_NAME ( OBJECT : SQL_OBJECT ) return SQL_OBJECT is
begin
    if OBJECT /= null and then OBJECT.KIND = NAME then
        return new SQL_OBJECT_RECORD' ( OBJECT.all );
    else
        return OBJECT;
    end if;
end COPY_NAME;

function BINARY_OPERATION
( GIVEN_OPERATION : SQL_OPERATION;
  L                : SQL_OBJECT;
  R                : SQL_OBJECT )
return SQL_OBJECT is
    LEFT : SQL_OBJECT := COPY_NAME ( L );
begin
    LEFT.ACROSS := R;
    return
        new SQL_OBJECT_RECORD' ( OPERATION , null , GIVEN_OPERATION , LEFT );
end BINARY_OPERATION;

-- subquery routines

function NEW_TAIL ( L , R : SQL_OBJECT ) return SQL_OBJECT is
begin
    if R = null then
        L.ACROSS :=
            new SQL_OBJECT_RECORD' ( OPERATION , null , O_NULL_OP , null );
    end if;
end NEW_TAIL;

```

UNCLASSIFIED

```

else
    L.ACROSS := R;
end if;
return L.ACROSS;
end NEW_TAIL;

function BUILD_SELECT
    ( SELECT_TYPE          : SQL_OPERATION;
      WHAT                 : SQL_OBJECT;
      FROM                 : FROM_CLAUSE;
      WHERE , GROUP_BY , HAVING : SQL_OBJECT )
return SQL_OBJECT is
    F   : SQL_OBJECT := COPY_NAME ( SQL_OBJECT ( FROM ) );
    W   : SQL_OBJECT := COPY_NAME ( WHAT );
    TAIL : SQL_OBJECT :=
        NEW_TAIL
        ( NEW_TAIL
          ( NEW_TAIL ( F , WHERE ) , COPY_NAME ( GROUP_BY ) ) , HAVING );
begin
    W.ACROSS := F;
    return new SQL_OBJECT_RECORD' ( OPERATION , null , SELECT_TYPE , W );
end BUILD_SELECT;

-- print routines

-- 5.6.1 <value specification>

procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT ) is
begin
    case S.VALUE.KIND is
        when K_INTEGER => PRINT ( S.VALUE.INTEGER );
        when K_FLOAT   => null; -- PRINT ( S.VALUE.FLOAT );
        when K_STRING  => PRINT ( "'" & S.VALUE.STRING.all & "'" );
    end case;
    if S.INDICATOR in INDICATOR_VARIABLE then
        PRINT ( " " & INDICATOR_VARIABLE'IMAGE ( S.INDICATOR ) );
    end if;
end SHOW_VALUE_SPECIFICATION;

-- 5.8.3 <all set function>

procedure SHOW_ALL_SET_FUNCTION ( S : in SQL_OBJECT ) is
begin
    case S.OPERATION is
        when O_AVG  => PRINT ( "AVG( " );
        when O_MAX  => PRINT ( "MAX( " );
        when O_MIN  => PRINT ( "MIN( " );
        when O_SUM  => PRINT ( "SUM( " );
        when others => raise INTERNAL_ERROR;
    end case;
    SHOW_VALUE_EXPRESSION ( S.OPERANDS );

```

UNCLASSIFIED

```
PRINT ( " )" );
end SHOW_ALL_SET_FUNCTION;
```

-- 5.9.1 <value expression>

```
procedure PARENTHESIZE_ADDING_OPERANDS
    ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
    SHOW_VALUE_EXPRESSION ( S );
    PRINT ( P );
    if S.ACROSS.KIND = OPERATION then
        case S.ACROSS.OPERATION is
            when O_UNARY_MINUS | O_PLUS | O_MINUS =>
                PRINT ( "(" );
                SHOW_VALUE_EXPRESSION ( S.ACROSS );
                PRINT ( " )" );
            when others =>
                SHOW_VALUE_EXPRESSION ( S.ACROSS );
        end case;
    else
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end if;
end PARENTHESIZE_ADDING_OPERANDS;
```

```
procedure PARENTHESIZE_MULTIPLYING_OPERANDS
    ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
    if S.KIND = OPERATION then
        case S.OPERATION is
            when O_UNARY_MINUS | O_PLUS | O_MINUS =>
                PRINT ( "(" );
                SHOW_VALUE_EXPRESSION ( S );
                PRINT ( " )" );
            when others =>
                SHOW_VALUE_EXPRESSION ( S );
        end case;
    else
        SHOW_VALUE_EXPRESSION ( S );
    end if;
    PRINT ( P );
    if S.ACROSS.KIND = OPERATION then
        case S.ACROSS.OPERATION is
            when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
                PRINT ( "(" );
                SHOW_VALUE_EXPRESSION ( S.ACROSS );
                PRINT ( " )" );
            when others =>
                SHOW_VALUE_EXPRESSION ( S.ACROSS );
        end case;
    else
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end if;
end PARENTHESIZE_MULTIPLYING_OPERANDS;
```

UNCLASSIFIED

```

end if;
end PARENTHESIZE_MULTIPLYING_OPERANDS;

procedure SHOW_VALUE_EXPRESSION ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when NAME =>
      PRINT ( S.NAME.all );
    when OPERATION =>
      case S.OPERATION is
        when O_AVG | O_MAX | O_MIN | O_SUM =>
          SHOW_ALL_SET_FUNCTION ( S );
        when O_COUNT_STAR =>
          PRINT ( "COUNT(*)" );
        when O_UNARY_PLUS =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
        when O_UNARY_MINUS =>
          PRINT ( " - " );
          if S.OPERANDS.KIND = OPERATION then
            case S.OPERANDS.OPERATION is
              when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
                PRINT ( "(" );
                SHOW_VALUE_EXPRESSION ( S.OPERANDS );
                PRINT ( ")" );
              when others => SHOW_VALUE_EXPRESSION ( S.OPERANDS );
            end case;
          else
            SHOW_VALUE_EXPRESSION ( S.OPERANDS );
          end if;
        when O_PLUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " + " );
        when O_MINUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " - " );
        when O_TIMES =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " * " );
        when O_DIVIDE =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " / " );
        when others => raise INTERNAL_ERROR;
      end case;
    end case;
end SHOW_VALUE_EXPRESSION;

```

-- 5.11.1 <comparison predicate>

```

procedure SHOW_COMPARISON_PREDICATE
  ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( P );

```

UNCLASSIFIED

```

if S.ACROSS.KIND = OPERATION then
  case S.ACROSS.OPERATION is
    when O_SELEC | O_SELECT_DISTINCT =>
      SHOW_QUERY_SPECIFICATION ( S.ACROSS );
    when others =>
      SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end case;
else
  SHOW_VALUE_EXPRESSION ( S.ACROSS );
end if;
end SHOW_COMPARISON_PREDICATE;

-- 5.12.1 <between predicate>

procedure SHOW_BETWEEN_PREDICATE ( S : in SQL_OBJECT ) is
  OPERAND : SQL_OBJECT := S.ACROSS.OPERANDS; -- first operand of AND
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " BETWEEN " );
  SHOW_VALUE_EXPRESSION ( OPERAND );
  PRINT ( " AND " );
  SHOW_VALUE_EXPRESSION ( OPERAND.ACROSS );
end SHOW_BETWEEN_PREDICATE;

-- 5.13.1 <in predicate>

procedure SHOW_IN_PREDICATE
  ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  PRINT ( P );
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " IN " );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_SELEC | O_SELECT_DISTINCT =>
        SHOW_QUERY_SPECIFICATION ( S.ACROSS );
        return;
      when others =>
        null;
    end case;
  end if;
  PRINT ( "(" ); SHOW_IN_VALUE_LIST ( S.ACROSS ); PRINT ( ")" );
end SHOW_IN_PREDICATE;

-- 5.13.2 <in value list>

procedure SHOW_IN_VALUE_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
  end case;
end SHOW_IN_VALUE_LIST;

```

UNCLASSIFIED

```

when OPERATION =>
  if S.OPERATION /= O_OR then
    raise INTERNAL_ERROR;
  end if;
  SHOW_IN_VALUE_LIST ( S.OPERANDS );
  PRINT ( " , " );
  SHOW_IN_VALUE_LIST ( S.OPERANDS.ACROSS );
when others =>
  raise INTERNAL_ERROR;
end case;
end SHOW_IN_VALUE_LIST;

-- 5.14.1 <like predicate>

procedure SHOW_LIKE_PREDICATE ( S : in SQL_OBJECT ) is
  P : ACCESS_STRING := S.ACROSS.VALUE.STRING; -- must be of right type
begin
  PRINT ( S.NAME.all ); PRINT ( " LIKE " );
  -- for I in P'RANGE loop
  --   case P(I) is
  --     when '_' => P(I) := '?';
  --     when '%' => P(I) := '*';
  --     when others => null;
  --   end case;
  -- end loop;
  SHOW_VALUE_SPECIFICATION ( S.ACROSS );
end SHOW_LIKE_PREDICATE;

-- 5.18.1 <search condition>

procedure PARENTHESE_RELATIONAL_OPERATORS
  ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
  OPERAND : SQL_OBJECT := S.OPERANDS;
begin
  case OPERAND.OPERATION is -- must be operation
    when O_AND | O_OR =>
      if OPERAND.OPERATION /= S.OPERATION then
        PRINT ( "(" ); SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( " )" );
      else
        SHOW_SEARCH_CONDITION ( OPERAND );
      end if;
    when others => SHOW_SEARCH_CONDITION ( OPERAND );
  end case;
  PRINT_LINE; PRINT ( P );
  OPERAND := OPERAND.ACROSS;
  case OPERAND.OPERATION is -- again, must be operation
    when O_AND | O_OR =>
      PRINT ( "(" ); SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( " )" );
    when others =>
      SHOW_SEARCH_CONDITION ( OPERAND );
  end case;

```

UNCLASSIFIED

end PARENTHESIZE_RELATIONAL_OPERATORS;

procedure SHOW_SEARCH_CONDITION (S : in SQL_OBJECT) is
begin

```

    case S.OPERATION is
        when O_EQ      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " = " );
        when O_NE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " ^= " );
        when O_LT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " < " );
        when O_GT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " > " );
        when O_LE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " <= " );
        when O_GE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " >= " );
        when O_BETWEEN => SHOW_BETWEEN_PREDICATE   ( S.OPERANDS );
        when O_IS_IN   => SHOW_IN_PREDICATE         ( S.OPERANDS , " " );
        when O_NOT_IN  => SHOW_IN_PREDICATE         ( S.OPERANDS , "NOT" );
        when O_LIKE     => SHOW_LIKE_PREDICATE       ( S.OPERANDS );
        when O_AND      => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "AND"   " );
        when O_OR       => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "OR"    " );
        when O_NOT =>
            PRINT ( "NOT " );
            case S.OPERANDS.OPERATION is -- must be operation
                when O_AND | O_OR =>
                    PRINT ( "(" );
                    SHOW_SEARCH_CONDITION ( S.OPERANDS );
                    PRINT ( " )" );
                when others =>
                    SHOW_SEARCH_CONDITION ( S.OPERANDS );
            end case;
        when others => raise INTERNAL_ERROR;
    end case;
end SHOW_SEARCH_CONDITION;
```

-- 5.19.1 <table expression>

```

procedure SHOW_TABLE_EXPRESSION ( S : in SQL_OBJECT ) is
    CLAUSE : SQL_OBJECT := S.ACROSS;
begin
    PRINT ( "FROM " ); SHOW_SELECT_LIST ( S );
    if CLAUSE.OPERATION /= O_NULL_OP then -- WHERE must have operation on top
        PRINT_LINE; PRINT ( "WHERE " ); SHOW_SEARCH_CONDITION ( CLAUSE );
    end if;
    CLAUSE := CLAUSE.ACROSS;
    if CLAUSE.KIND /= OPERATION or else CLAUSE.OPERATION /= O_NULL_OP then
        PRINT_LINE; PRINT ( "GROUP BY " ); SHOW_SELECT_LIST ( CLAUSE );
    end if;
    CLAUSE := CLAUSE.ACROSS;
    if CLAUSE.OPERATION /= O_NULL_OP then -- same as WHERE
        PRINT_LINE; PRINT ( "HAVING " ); SHOW_SEARCH_CONDITION ( CLAUSE );
    end if;
end SHOW_TABLE_EXPRESSION;
```

-- 5.25.1 <query specification>

UNCLASSIFIED

```

procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT ) is
  CLAUSE : SQL_OBJECT := S.OPERANDS;
begin
  INDENT := INDENT + 5;
  if INDENT >= 0 then
    SET_INDENT ( INDENT );
  end if;
  PRINT_LINE;
  INDENT := INDENT + 2;
  if INDENT > 0 then
    PRINT ( "( SELECT " );
  else
    PRINT ( "SELECT " );
  end if;
  SET_INDENT ( INDENT );
  case S.OPERATION is
    when O_SELEC          => null;
    when O_SELECT_DISTINCT => PRINT ( "DISTINCT " );
    when others           => raise INTERNAL_ERROR;
  end case;
  SHOW_SELECT_LIST ( CLAUSE );
  PRINT_LINE;
  SHOW_TABLE_EXPRESSION ( CLAUSE.ACROSS );
  INDENT := INDENT - 7;
  if INDENT >= 0 then
    PRINT ( " )" );
    SET_INDENT ( INDENT );
  end if;
end SHOW_QUERY_SPECIFICATION;

```

-- 5.25.2 <select list>

```

procedure SHOW_SELECT_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when NAME | VALUE =>
      SHOW_VALUE_EXPRESSION ( S );
    when OPERATION =>
      case S.OPERATION is
        when O_STAR =>
          PRINT ( "*" );
        when O_AMPERSAND =>
          SHOW_SELECT_LIST ( S.OPERANDS );
          PRINT ( ", " );
          SHOW_SELECT_LIST ( S.OPERANDS.ACROSS );
        when others =>
          SHOW_VALUE_EXPRESSION ( S );
      end case;
    end case;
end SHOW_SELECT_LIST;

```

UNCLASSIFIED

-- 8.3.5 <order by clause>

```
procedure SHOW_ORDER_BY_CLAUSE ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when NAME | VALUE =>
      SHOW_VALUE_EXPRESSION ( S );
    when OPERATION =>
      case S.OPERATION is
        when O_AMPERSAND =>
          SHOW_ORDER_BY_CLAUSE ( S.OPERANDS );
          PRINT ( ", " );
          SHOW_ORDER_BY_CLAUSE ( S.OPERANDS.ACROSS );
        when O_ASC =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
        when O_DESC =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
          PRINT ( " DESC" );
        when others =>
          raise INTERNAL_ERROR;
      end case;
    end case;
  end SHOW_ORDER_BY_CLAUSE;
```

-- 8.7.3 <insert value list>

```
procedure SHOW_INSERT_VALUE_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when OPERATION =>
      case S.OPERATION is
        when O_AND =>
          SHOW_INSERT_VALUE_LIST ( S.OPERANDS );
          PRINT ( ", " );
        when O_LE =>
          null;
        when others =>
          raise INTERNAL_ERROR;
      end case;
      SHOW_INSERT_VALUE_LIST ( S.OPERANDS.ACROSS );
    when others =>
      raise INTERNAL_ERROR;
    end case;
  end SHOW_INSERT_VALUE_LIST;
```

-- 8.11.2 <set clause>

```
procedure SHOW_SET_CLAUSES ( S : in SQL_OBJECT ) is
begin
```

UNCLASSIFIED

```

case S.OPERATION is -- must be operation
  when O_AND =>
    SHOW_SET_CLAUSES ( S.OPERANDS ); PRINT ( "," ); PRINT_LINE;
    SHOW_SET_CLAUSES ( S.OPERANDS.ACROSS );
  when O_LE =>
    PRINT ( S.OPERANDS.NAME.all & " = " );
    SHOW_VALUE_EXPRESSION ( S.OPERANDS.ACROSS );
  when others =>
    raise INTERNAL_ERROR;
end case;
end SHOW_SET_CLAUSES;

-- delete routines

function DEMO_TABLE ( TABLE : TABLE_NAME ) return SQL_OBJECT is
begin
  if TABLE.NAME.all = "BIDE" then
    return BIDE_TABLE;
  else
    return COMMAND_TABLE;
  end if;
end DEMO_TABLE;

function ROW_BEFORE_DESIRED_ONE
  ( TABLE : TABLE_NAME;
    WHERE : SEARCH_CONDITION ) return SQL_OBJECT is
  KEY      : ACCESS_STRING := WHERE.OPERANDS.ACROSS.VALUE.STRING;
  PREVIOUS_ROW : SQL_OBJECT := DEMO_TABLE ( TABLE );
  CURRENT_ROW  : SQL_OBJECT;
begin
  loop
    CURRENT_ROW := PREVIOUS_ROW.OPERANDS;
    if CURRENT_ROW = null then
      PRINT ( "***** NO DATA *****" ); PRINT_LINE;
      raise NO_DATA;
    elsif CURRENT_ROW.ACROSS.VALUE.STRING.all = KEY.all then
      return PREVIOUS_ROW;
    end if;
    PREVIOUS_ROW := CURRENT_ROW;
  end loop;
end ROW_BEFORE_DESIRED_ONE;

procedure DELETE
  ( FROM : in TABLE_NAME;
    WHERE : in SEARCH_CONDITION := NULL_SEARCH_CONDITION ) is
  PREVIOUS_ROW : SQL_OBJECT;
begin
  BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "DELETE " & FROM.NAME.all );
  if WHERE /= null then
    INDENT := 0; PRINT_LINE; PRINT ( "WHERE " );
    SHOW_SEARCH_CONDITION ( SQL_OBJECT ( WHERE ) );
  end if;
end DELETE;

```

UNCLASSIFIED

```

    PRINT_LINE;
    PREVIOUS_ROW := ROW_BEFORE_DESIRED_ONE ( FROM , WHERE );
    PREVIOUS_ROW.OPERANDS := PREVIOUS_ROW.OPERANDS.OPERANDS;
else
    PRINT_LINE;
end if;
exception
    when NO_DATA => null;
    when others => raise INTERNAL_ERROR;
end DELETE;

-- into routines

function INDICATOR_VALUE ( VALUE : SQL_OBJECT ) return INDICATOR_VARIABLE is
begin
    if VALUE.INDICATOR = NO_INDICATOR then
        return NOT_NULL;
    else
        return VALUE.INDICATOR;
    end if;
end INDICATOR_VALUE;

procedure SHOW_INT0 is
begin
    PRINT ( "INT0 returning " ); SHOW_VALUE_SPECIFICATION ( INTO_COLUMN );
    PRINT_LINE; INTO_COLUMN := INTO_COLUMN.ACROSS;
end SHOW_INT0;

procedure INTEGER_AND_ENUMERATION_INT0
    ( TARGET      : out DATABASE.INT;
      INDICATOR   : out INDICATOR_VARIABLE;
      CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME ) is
begin
    TARGET      := INTO_COLUMN.VALUE.INTEGER;
    INDICATOR   := INDICATOR_VALUE ( INTO_COLUMN );
    SHOW_INT0;
end INTEGER_AND_ENUMERATION_INT0;

procedure INTEGER_AND_ENUMERATION_INT0
    ( TARGET      : out DATABASE.INT;
      CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME ) is
begin
    TARGET := INTO_COLUMN.VALUE.INTEGER;
    SHOW_INT0;
end INTEGER_AND_ENUMERATION_INT0;

procedure CONSTRAINED_STRING_INT0
    ( TARGET      : out STANDARD.STRING;
      LAST        : out STANDARD.NATURAL;
      INDICATOR   : out INDICATOR_VARIABLE;
      CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME ) is

```

UNCLASSIFIED

```

begin
    TARGET      := INTO_COLUMN.VALUE.STRING.all;
    LAST        := TARGET'LAST;
    INDICATOR   := INDICATOR_VALUE ( INTO_COLUMN );
    SHOW_INT0;
end CONSTRAINED_STRING_INT0;

procedure CONSTRAINED_STRING_INT0
    ( TARGET      : out STANDARD.STRING;
      LAST        : out STANDARD.NATURAL;
      CURSOR      : in  CURSOR_NAME := NULL_CURSOR_NAME ) is
begin
    TARGET := INTO_COLUMN.VALUE.STRING.all;
    LAST   := TARGET'LAST;
    SHOW_INT0;
end CONSTRAINED_STRING_INT0;

-- insert into routines

procedure RECURSIVELY_ADD_COLUMN
    ( PREVIOUS_LAST_COLUMN : in out SQL_OBJECT;
      VALUES               : in   SQL_OBJECT ) is
begin
    case VALUES.KIND is
        when VALUE =>
            PREVIOUS_LAST_COLUMN.ACROSS := new
                SQL_OBJECT_RECORD'
                ( VALUE , null , VALUES.INDICATOR , VALUES.VALUE );
            PREVIOUS_LAST_COLUMN := PREVIOUS_LAST_COLUMN.ACROSS;
        when OPERATION =>
            case VALUES.OPERATION is
                when O_AND =>
                    RECURSIVELY_ADD_COLUMN ( PREVIOUS_LAST_COLUMN , VALUES.OPERANDS );
                when O_LE =>
                    null;
                when others =>
                    raise INTERNAL_ERROR;
            end case;
            RECURSIVELY_ADD_COLUMN
                ( PREVIOUS_LAST_COLUMN , VALUES.OPERANDS.ACROSS );
        when others =>
            raise INTERNAL_ERROR;
    end case;
end RECURSIVELY_ADD_COLUMN;

procedure MAKE_DATABASE_ROW
    ( VALUES : in INSERT_VALUE_LIST;
      TABLE  : in SQL_OBJECT ) is
    LAST_COLUMN : SQL_OBJECT :=
        new SQL_OBJECT_RECORD' ( OPERATION , null , O_VALUES , TABLE.OPERANDS );
begin

```

UNCLASSIFIED

```
TABLE.OPERANDS := LAST_COLUMN;
RECURSIVELY_ADD_COLUMN ( LAST_COLUMN , SQL_OBJECT ( VALUES ) );
end MAKE_DATABASE_ROW;
```

```
procedure INSERT_INT0
    ( TABLE : in TABLE_NAME;
      VALUES : in INSERT_VALUE_LIST ) is
begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "INSERT INTO " );
    if TABLE.KIND = NAME then
        PRINT ( TABLE.NAME.all );
    else -- must be O_TABLE_COLUMN_LIST
        PRINT ( TABLE.OPERANDS.NAME.all );
        PRINT ( "( " );
        SHOW_SELECT_LIST ( TABLE.OPERANDS.ACROSS );
        PRINT ( " )" );
    end if;
    PRINT_LINE;
    case VALUES.OPERATION is -- must be an operation
        when O_SELEC | O_SELECT_DISTINCT =>
            INDENT := -7; SHOW_QUERY_SPECIFICATION ( SQL_OBJECT ( VALUES ) );
        when O_LE | O_AND =>
            PRINT ( "VALUES ( " );
            SHOW_INSERT_VALUE_LIST ( SQL_OBJECT ( VALUES ) );
            PRINT ( " )" );
            MAKE_DATABASE_ROW ( VALUES , DEMO_TABLE ( TABLE ) );
        when others =>
            raise INTERNAL_ERROR;
    end case;
    PRINT_LINE;
exception
    when others => raise INTERNAL_ERROR;
end INSERT_INT0;
```

```
function VALUES return INSERT_VALUE_LIST_STARTER is
begin
    return new SQL_OBJECT_RECORD' ( OPERATION , null , O_VALUES , null );
end VALUES;
```

-- select statement routines

```
function SECOND_COLUMN ( TABLE : TABLE_NAME ; WHERE : SEARCH_CONDITION )
    return SQL_OBJECT is
begin
    return
        ROW_BEFORE_DESIRED_ONE ( TABLE , WHERE ) . OPERANDS . ACROSS . ACROSS;
end SECOND_COLUMN;
```

```
procedure SHOW_SELECT ( S : in SQL_OBJECT ) is
begin
    BLANK_LINE; INDENT := -7;
```

UNCLASSIFIED

```

SHOW_QUERY_SPECIFICATION ( S );
PRINT_LINE;
exception
  when others => raise INTERNAL_ERROR;
end SHOW_SELECT;

procedure SELECT_LIST_SELECT
  ( SELECT_TYPE : in SQL_OPERATION;
    WHAT        : in SQL_OBJECT;
    FROM        : in FROM_CLAUSE;
    WHERE       : in SEARCH_CONDITION := NULL_SEARCH_CONDITION ) is
begin
  SHOW_SELECT
  ( BUILD_SELECT
    ( SELECT_TYPE , WHAT , FROM , SQL_OBJECT ( WHERE ) , null , null ) );
  INTO_COLUMN := SECOND_COLUMN ( TABLE_NAME ( FROM ) , WHERE );
end SELECT_LIST_SELECT;

-- update routines

procedure RECURSIVELY_SET_VALUES
  ( COLUMN : in out SQL_OBJECT ; SET : in SQL_OBJECT ) is
begin
  case SET.OPERATION is -- must be operation
    when O_AND =>
      RECURSIVELY_SET_VALUES ( COLUMN , SET.OPERANDS );
      RECURSIVELY_SET_VALUES ( COLUMN , SET.OPERANDS.ACROSS );
    when O_LE =>
      COLUMN.INDICATOR := SET.OPERANDS.ACROSS.INDICATOR;
      COLUMN.VALUE     := SET.OPERANDS.ACROSS.VALUE;
      COLUMN            := COLUMN.ACROSS;
    when others =>
      raise INTERNAL_ERROR;
  end case;
end RECURSIVELY_SET_VALUES;

procedure UPDATE_KEYED_ROW
  ( TABLE : in SQL_OBJECT;
    WHERE   : in SEARCH_CONDITION;
    SET     : in SQL_OBJECT ) is
  CURRENT_COLUMN : SQL_OBJECT :=
    SECOND_COLUMN ( TABLE_NAME ( TABLE ) , WHERE );
begin
  RECURSIVELY_SET_VALUES ( CURRENT_COLUMN , SET );
end UPDATE_KEYED_ROW;

procedure UPDATE_PROCEDURE
  ( TABLE : in SQL_OBJECT;
    SET     : in SQL_OBJECT;
    WHERE   : in SEARCH_CONDITION := NULL_SEARCH_CONDITION ) is
begin

```

UNCLASSIFIED

```

BLANK_LINE; SET_INDENT ( 0 );
PRINT ( "UPDATE " & TABLE.NAME.all );
PRINT_LINE; PRINT ( "SET " ); SET_INDENT ( 4 );
SHOW_SET_CLAUSES ( SET );
if WHERE /= null then
    INDENT := 0; SET_INDENT ( 0 ); PRINT_LINE; PRINT ( "WHERE " );
    SHOW_SEARCH_CONDITION ( SQL_OBJECT ( WHERE ) ); PRINT_LINE;
    UPDATE_KEYED_ROW ( TABLE , WHERE , SET );
else
    PRINT_LINE;
end if;
exception
    when NO_DATA => null;
    when others => raise INTERNAL_ERROR;
end UPDATE_PROCEDURE;

package body CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT is
    begin
        return L;
    end L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT is
    begin
        return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME is
    begin
        return TABLE_NAME ( R );
    end CONVERT_R;

    function L_CONVERT ( L : FROM_CLAUSE ) return SQL_OBJECT is
    begin
        return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return FROM_CLAUSE is
    begin
        return FROM_CLAUSE ( R );
    end CONVERT_R;

    function L_CONVERT ( L : INSERT_VALUE_LIST ) return SQL_OBJECT is
    begin
        return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return INSERT_VALUE_LIST is
    begin
        return INSERT_VALUE_LIST ( R );
    end CONVERT_R;

```

UNCLASSIFIED

end CONVERT_R;

```
function L_CONVERT ( L : INSERT_VALUE_LIST_STARTER ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;
```

```
function CONVERT_R ( R : SQL_OBJECT ) return INSERT_VALUE_LIST_STARTER is
begin
    return INSERT_VALUE_LIST_STARTER ( R );
end CONVERT_R;
```

```
function L_CONVERT ( L : SEARCH_CONDITION ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;
```

```
function CONVERT_R ( R : SQL_OBJECT ) return SEARCH_CONDITION is
begin
    return SEARCH_CONDITION ( R );
end CONVERT_R;
```

```
function L_CONVERT ( L : SELECT_LIST ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;
```

```
function CONVERT_R ( R : SQL_OBJECT ) return SELECT_LIST is
begin
    return SELECT_LIST ( R );
end CONVERT_R;
```

```
function L_CONVERT ( L : VALUE_EXPRESSION ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;
```

```
function CONVERT_R ( R : SQL_OBJECT ) return VALUE_EXPRESSION is
begin
    return VALUE_EXPRESSION ( R );
end CONVERT_R;
```

```
function L_CONVERT ( L : VALUE_SPECIFICATION ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;
```

```
function CONVERT_R ( R : SQL_OBJECT ) return VALUE_SPECIFICATION is
begin
    return VALUE_SPECIFICATION ( R );
end CONVERT_R;
```

UNCLASSIFIED

```
end CONVERT;

-- conversion routines for SQL objects

function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT is
begin
    return SQL_OBJECT ( L );
end L_CONVERT;

function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT is
begin
    return TYPED_SQL_OBJECT ( R );
end CONVERT_R;

end ADA_SQL_FUNCTIONS;
```

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

7.5 DBTYPES.ADS

-- File: dbtypes.ads
-- DATABASE_TYPES
-- 12/18/88

DISCLAIMER OF WARRANTY AND LIABILITY

THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE, OR OTHERWISE.

USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
SPECIAL, OR OTHERWISE.

Prepared for:

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Prepared by:

Fred J. Friedman
RACOM Computer Professionals
P.O. Box 576
Annandale, VA 22003-1620
(703) 560-6813 (703) 560-6799

package DATABASE_TYPES is
package ADA_SQL is

subtype X_DATE is INTEGER range 1 .. 8;
subtype X_REVAL is INTEGER range 1 .. 1;
subtype X_SHORT_NAME is INTEGER range 1 .. 30;
subtype X_UIC is INTEGER range 1 .. 6;
subtype X_ULC is INTEGER range 1 .. 3;
subtype X_UTC is INTEGER range 1 .. 5;

UNCLASSIFIED

```
type T_DATE      is array ( X_DATE ) of CHARACTER;
type T_REVAL     is array ( X_REVAL ) of CHARACTER;
type T_SECUR     is ( U , C , S , T );
type T_SHORT_NAME is array ( X_SHORT_NAME ) of CHARACTER;
type T_TPSN      is range 0 .. 9_999_999;
type T_UIC       is array ( X_UIC ) of CHARACTER;
type T_ULC       is array ( X_ULC ) of CHARACTER;
type T_UTC       is array ( X_UTC ) of CHARACTER;

end ADA_SQL;
end DATABASE_TYPES;
```

UNCLASSIFIED

7.6 GENPACK.ADA

-- File: genpack.ads
-- DATABASE_CARD_A_ADA_SQL
-- 12/18/88

DISCLAIMER OF WARRANTY AND LIABILITY

THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE, OR OTHERWISE.

USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
SPECIAL, OR OTHERWISE.

Prepared for:

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Prepared by:

Fred J. Friedman
RACOM Computer Professionals
P.O. Box 576
Annandale, VA 22003-1620
(703) 560-6813 (703) 560-6799

with ADA_SQL_FUNCTIONS, DATABASE_TYPES;
package DATABASE_CARD_A_ADA_SQL is

NO_DATA : exception renames ADA_SQL_FUNCTIONS.NO_DATA;

procedure INITIATE_TEST renames ADA_SQL_FUNCTIONS.INITIATE_TEST;

package ADA_SQL is

package DATABASE_DEFINITION_PACKAGE is

UNCLASSIFIED

```
package BIDE_PACKAGE is

  type TABLE_NAME is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

  type SET_CLAUSE is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

  package DATABASE_TYPES is

    type COLUMN_NAME_T_DATE is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_REVAL is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_REVAL is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_SECUR is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_SECUR is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_SHORT_NAME is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_SHORT_NAME is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_TPSN is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_TPSN is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_ULC is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_ULC is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type COLUMN_NAME_T_UTC is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type VALUE_EXPRESSION_T_UTC is new
      ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

  end DATABASE_TYPES;

end BIDE_PACKAGE;
```

UNCLASSIFIED

```
package COMMAND_PACKAGE is

    type TABLE_NAME is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    type SET_CLAUSE is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    package DATABASE_TYPES is

        type COLUMN_NAME_T_DATE is new
            ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

        type COLUMN_NAME_T_SECUR is new
            ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

        type COLUMN_NAME_T_UIC is new
            ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

        type VALUE_EXPRESSION_T_UIC is new
            ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    end DATABASE_TYPES;

    package STANDARD is

        type COLUMN_NAME_BOOLEAN is new
            ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

    end STANDARD;

end COMMAND_PACKAGE;

end DATABASE_DEFINITION_PACKAGE;

package DATABASE_TYPES_TYPE_PACKAGE is

    type VALUE_EXPRESSION_T_UIC is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

end DATABASE_TYPES_TYPE_PACKAGE;

end ADA_SQL;

function CONVERT_COMPONENT_TO_CHARACTER
    ( C : STANDARD.CHARACTER ) return STANDARD.CHARACTER;

function L_CONVERT ( L : STANDARD.BOOLEAN )
    return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : STANDARD.BOOLEAN )
    return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_DATE )
```

UNCLASSIFIED

```
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_DATE )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_REVAL )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_REVAL )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_SECUR )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_SECUR )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_TPSN )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_TPSN )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_UIC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_UIC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_ULC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_ULC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_UTC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function R_CONVERT ( R : DATABASE_TYPES.ADA_SQL.T_UTC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

procedure DELETE
( FROM : in ADA_SQL_FUNCTIONS.TABLE_NAME;
  WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
    ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION )
```

UNCLASSIFIED

```
renames ADA_SQL_FUNCTIONS.DELETE;

procedure INSERT_INT0
( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
  VALUES : in ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST )
renames ADA_SQL_FUNCTIONS.INSERT_INT0;

procedure SELEC
( WHAT : in ADA_SQL_FUNCTIONS.SELECT_LIST;
  FROM : in ADA_SQL_FUNCTIONS.FROM_CLAUSE;
  WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
    ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION );

procedure UPDATE
( TABLE : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
  TABLE_NAME;
  SET : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
  SET_CLAUSE;
  WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
    ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION );

procedure UPDATE
( TABLE : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
  TABLE_NAME;
  SET : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
  SET_CLAUSE;
  WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
    ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION );

function BIDE return ADA_SQL_FUNCTIONS.FROM_CLAUSE;

function BIDE return ADA_SQL_FUNCTIONS.TABLE_NAME;

function BIDE
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.TABLE_NAME;

function COMMAND return ADA_SQL_FUNCTIONS.FROM_CLAUSE;

function COMMAND return ADA_SQL_FUNCTIONS.TABLE_NAME;

function COMMAND
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.TABLE_NAME;

function IS_MAJOR return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function IS_MAJOR
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.
COLUMN_NAME_BOOLEAN;

function MAJCOM return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;
```

UNCLASSIFIED

```
function MAJCOM
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_UIC;

function ORG_SCLASS return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function ORG_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR;

function REC_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR;

function REC_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR;

function REPORT_DATE
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_DATE;

function REPORT_DATE
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_DATE;

function REVAL return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function REVAL
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_REVAL;

function SHORT_NAME return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function SHORT_NAME
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SHORT_NAME;

function TPSN return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function TPSN
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
```

UNCLASSIFIED

```
COLUMN_NAME_T_TPSN;

function UIC
return ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.VALUE_EXPRESSION_T_UIC;

function ULC return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function ULC
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_ULC;

function UPDATE_DATE
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_DATE;

function UPDATE_DATE
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_DATE;

function UTC return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;

function UTC
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_UTC;

function VALUES return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST_STARTER
renames ADA_SQL_FUNCTIONS.VALUES;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_REVAL;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_REVAL;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
VALUE_EXPRESSION_T_REVAL;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_SECUR;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;
```

UNCLASSIFIED

```

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SECUR;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
    VALUE_EXPRESSION_T_SECUR;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
    VALUE_EXPRESSION_T_SHORT_NAME;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_TPSN;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_TPSN;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
    VALUE_EXPRESSION_T_TPSN;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_UIC;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_UIC;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
    VALUE_EXPRESSION_T_UIC;

function INDICATOR

```

UNCLASSIFIED

```

    ( VALUE : DATABASE_TYPES.ADA_SQL.T_ULC;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
  return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_ULC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
      VALUE_EXPRESSION_T_ULC;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_UTC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
  return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_UTC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
      VALUE_EXPRESSION_T_UTC;

procedure INTO
  ( TARGET      : out STANDARD.BOOLEAN;
    CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
      ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );

procedure INTO
  ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_REVAL;
    LAST        : out DATABASE_TYPES.ADA_SQL.X_REVAL;
    INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
    CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
      ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );

procedure INTO
  ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_SECUR;
    INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
    CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
      ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );

procedure INTO
  ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
    LAST        : out DATABASE_TYPES.ADA_SQL.X_SHORT_NAME;
    INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
    CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
      ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );

```

UNCLASSIFIED

```
procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_TPSN;
  INDICATOR    : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR       : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );
```

```
procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_UIC;
  LAST        : out DATABASE_TYPES.ADA_SQL.X_UIC;
  INDICATOR    : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR       : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );
```

```
procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_ULC;
  LAST        : out DATABASE_TYPES.ADA_SQL.X_ULC;
  INDICATOR    : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR       : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );
```

```
procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_UTC;
  LAST        : out DATABASE_TYPES.ADA_SQL.X_UTC;
  INDICATOR    : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR       : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME );
```

```
function "and"
( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
  R : ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
```

```
function "and"
( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
  R : DATABASE_TYPES.ADA_SQL.T_DATE )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
```

```
function "and"
( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
  R : DATABASE_TYPES.ADA_SQL.T_SECUR )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
```

```
function "and"
( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
  R : STANDARD.BOOLEAN )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
```

```
function "and"
( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;
  R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;
```

UNCLASSIFIED

```

function "and"
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      SET_CLAUSE;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      SET_CLAUSE )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE;

function EQ
    ( L : ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.VALUE_EXPRESSION_T_UIC;
      R : DATABASE_TYPES.ADA_SQL.T_UIC )
    return ADA_SQL_FUNCTIONS.SEARCH_CONDITION;

function "&"
    ( L : ADA_SQL_FUNCTIONS.SELECT_LIST;
      R : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION )
    return ADA_SQL_FUNCTIONS.SELECT_LIST;

function "&"
    ( L : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;
      R : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION )
    return ADA_SQL_FUNCTIONS.SELECT_LIST;

function "<="
    ( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST_STARTER;
      R : DATABASE_TYPES.ADA_SQL.T_UIC )
    return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.
      COLUMN_NAME_BOOLEAN;
      R : STANDARD.BOOLEAN )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_DATE;
      R : DATABASE_TYPES.ADA_SQL.T_DATE )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_DATE;
      R : DATABASE_TYPES.ADA_SQL.T_DATE )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_REVAL;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.VALUE_EXPRESSION_T_REVAL )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

```

UNCLASSIFIED

```

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_SECUR;
      R : DATABASE_TYPES.ADA_SQL.T_SECUR )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_SECUR;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
        DATABASE_TYPES.VALUE_EXPRESSION_T_SECUR )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_SECUR;
      R : DATABASE_TYPES.ADA_SQL.T_SECUR )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_SHORT_NAME;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
        DATABASE_TYPES.VALUE_EXPRESSION_T_SHORT_NAME )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_TPSN;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
        DATABASE_TYPES.VALUE_EXPRESSION_T_TPSN )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_UIC;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
        DATABASE_TYPES.VALUE_EXPRESSION_T_UIC )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_ULC;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
        DATABASE_TYPES.VALUE_EXPRESSION_T_ULC )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

function "<="
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
      DATABASE_TYPES.COLUMN_NAME_T_UTC;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.

```

UNCLASSIFIED

```
        DATABASE_TYPES.VALUE_EXPRESSION_T.UTC )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;

end DATABASE_CARD_A_ADA_SQL;

with DATABASE;
package body DATABASE_CARD_A_ADA_SQL is

    function CONVERT_COMPONENT_TO_CHARACTER
        ( C : STANDARD.CHARACTER ) return STANDARD.CHARACTER is
    begin
        return C;
    end CONVERT_COMPONENT_TO_CHARACTER;

    function L_CONVERT ( L : STANDARD.BOOLEAN )
        return ADA_SQL_FUNCTIONS.SQL_OBJECT is
    begin
        return
            ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT
            ( DATABASE.INT ( STANDARD.BOOLEAN'POS ( L ) ) );
    end L_CONVERT;

    function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_DATE )
        return ADA_SQL_FUNCTIONS.SQL_OBJECT is
    begin
        return
            ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
            ( STANDARD.STRING ( L ) );
    end L_CONVERT;

    function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_REVAL )
        return ADA_SQL_FUNCTIONS.SQL_OBJECT is
    begin
        return
            ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
            ( STANDARD.STRING ( L ) );
    end L_CONVERT;

    function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_SECUR )
        return ADA_SQL_FUNCTIONS.SQL_OBJECT is
    begin
        return
            ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT
            ( DATABASE.INT ( DATABASE_TYPES.ADA_SQL.T_SECUR'POS ( L ) ) );
    end L_CONVERT;

    function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME )
        return ADA_SQL_FUNCTIONS.SQL_OBJECT is
    begin
        return
            ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
```

UNCLASSIFIED

```

    ( STANDARD.STRING ( L ) );
end L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_TPSN )
return ADA_SQL_FUNCTIONS.SQL_OBJECT is
begin
    return
        ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT
        ( DATABASE.INT ( DATABASE_TYPES.ADA_SQL.T_TPSN'POS ( L ) ) );
end L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_UIC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT is
begin
    return
        ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
        ( STANDARD.STRING ( L ) );
end L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_ULC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT is
begin
    return
        ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
        ( STANDARD.STRING ( L ) );
end L_CONVERT;

function L_CONVERT ( L : DATABASE_TYPES.ADA_SQL.T_UTC )
return ADA_SQL_FUNCTIONS.SQL_OBJECT is
begin
    return
        ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
        ( STANDARD.STRING ( L ) );
end L_CONVERT;

procedure SELEC
( WHAT : in ADA_SQL_FUNCTIONS.SELECT_LIST;
  FROM : in ADA_SQL_FUNCTIONS.FROM_CLAUSE;
  WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
    ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION ) is
begin
    ADA_SQL_FUNCTIONS.SELECT_LIST_SELECT
    ( ADA_SQL_FUNCTIONS.O_SELEC,
      ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( WHAT ) , FROM , WHERE );
end SELEC;

procedure UPDATE
( TABLE : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
  TABLE_NAME;
  SET : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
  SET_CLAUSE;

```

UNCLASSIFIED

```

        WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
            ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION ) is
begin
    ADA_SQL_FUNCTIONS.UPDATE_PROCEDURE
    ( ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.L_CONVERT ( TABLE ),
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.L_CONVERT ( SET ),
      WHERE );
end UPDATE;

procedure UPDATE
    ( TABLE : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      TABLE_NAME;
      SET : in ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
      SET_CLAUSE;
      WHERE : in ADA_SQL_FUNCTIONS.SEARCH_CONDITION :=
        ADA_SQL_FUNCTIONS.NULL_SEARCH_CONDITION ) is
begin
    ADA_SQL_FUNCTIONS.UPDATE_PROCEDURE
    ( ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.L_CONVERT ( TABLE ),
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.L_CONVERT ( SET ),
      WHERE );
end UPDATE;

function BIDE return ADA_SQL_FUNCTIONS.FROM_CLAUSE is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "BIDE" ) );
end BIDE;

function BIDE return ADA_SQL_FUNCTIONS.TABLE_NAME is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "BIDE" ) );
end BIDE;

function BIDE
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.TABLE_NAME is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "BIDE" ) );
end BIDE;

function COMMAND return ADA_SQL_FUNCTIONS.FROM_CLAUSE is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "COMMAND" ) );
end COMMAND;

```

UNCLASSIFIED

```
function COMMAND return ADA_SQL_FUNCTIONS.TABLE_NAME is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "COMMAND" ) );
end COMMAND;

function COMMAND
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.TABLE_NAME is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "COMMAND" ) );
end COMMAND;

function IS_MAJOR return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "IS_MAJOR" ) );
end IS_MAJOR;

function IS_MAJOR
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.
    COLUMN_NAME_BOOLEAN is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "IS_MAJOR" ) );
end IS_MAJOR;

function MAJCOM return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "MAJCOM" ) );
end MAJCOM;

function MAJCOM
return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
    COLUMN_NAME_T_UIC is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
        CONVERT_R ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "MAJCOM" ) );
end MAJCOM;

function ORG_SCLASS return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is
begin
```

UNCLASSIFIED

```

return
  ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
  ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "ORG_SCLASS" ) );
end ORG_SCLASS;

function ORG_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "ORG_SCLASS" ) );
end ORG_SCLASS;

function REC_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REC_SCLASS" ) );
end REC_SCLASS;

function REC_SCLASS
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_SECUR is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
    CONVERT_R ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REC_SCLASS" ) );
end REC_SCLASS;

function REPORT_DATE
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_DATE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REPORT_DATE" ) );
end REPORT_DATE;

function REPORT_DATE
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_DATE is
begin
  return

```

UNCLASSIFIED

```
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.  
  CONVERT_R ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REPORT_DATE" ) );  
end REPORT_DATE;
```

```
function REVAL return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is  
begin  
  return  
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R  
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REVAL" ) );  
end REVAL;
```

```
function REVAL  
return  
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.  
  COLUMN_NAME_T_REVAL is  
begin  
  return  
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R  
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "REVAL" ) );  
end REVAL;
```

```
function SHORT_NAME return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is  
begin  
  return  
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R  
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "SHORT_NAME" ) );  
end SHORT_NAME;
```

```
function SHORT_NAME  
return  
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.  
  COLUMN_NAME_T_SHORT_NAME is  
begin  
  return  
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R  
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "SHORT_NAME" ) );  
end SHORT_NAME;
```

```
function TPSN return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is  
begin  
  return  
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R  
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "TPSN" ) );  
end TPSN;
```

```
function TPSN  
return  
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.  
  COLUMN_NAME_T_TPSN is  
begin  
  return
```

UNCLASSIFIED

```
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "TPSN" ) );
end TPSN;
```

```
function UIC
return ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.VALUE_EXPRESSION_T_UIC is
begin
return
ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.CONVERT_R
( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "UIC" ) );
end UIC;
```

```
function ULC return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is
begin
return
ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "ULC" ) );
end ULC;
```

```
function ULC
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_ULC is
begin
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "ULC" ) );
end ULC;
```

```
function UPDATE_DATE
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_DATE is
begin
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "UPDATE_DATE" ) );
end UPDATE_DATE;
```

```
function UPDATE_DATE
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
COLUMN_NAME_T_DATE is
begin
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
CONVERT_R ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "UPDATE_DATE" ) );
end UPDATE_DATE;
```

```
function UTC return ADA_SQL_FUNCTIONS.VALUE_EXPRESSION is
begin
```

UNCLASSIFIED

```

return
  ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
  ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "UTC" ) );
end UTC;

function UTC
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  COLUMN_NAME_T_UTC is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.COLUMN_OR_TABLE_NAME ( "UTC" ) );
end UTC;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_REVAL;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
  return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_REVAL;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  VALUE_EXPRESSION_T_REVAL is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_SECUR;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
  return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );

```

UNCLASSIFIED

end INDICATOR;

```
function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SECUR;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
    return
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
        VALUE_EXPRESSION_T_SECUR is
begin
    return
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
        ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
          ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;
```

```
function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
    return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
    return
      ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
          ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;
```

```
function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
    return
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
        VALUE_EXPRESSION_T_SHOPT_NAME is
begin
    return
      ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
        ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
          ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;
```

```
function INDICATOR
    ( VALUE : DATABASE_TYPES.ADA_SQL.T_TPSN;
      IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
        ADA_SQL_FUNCTIONS.NOT_NULL )
    return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
    return
      ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
```

UNCLASSIFIED

```

        ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_TPSN;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
    ADA_SQL_FUNCTIONS.NOT_NULL )
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  VALUE_EXPRESSION_T_TPSN is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
      ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
        ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_UIC;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
    ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
  return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
      ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
        ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_UIC;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
    ADA_SQL_FUNCTIONS.NOT_NULL )
return
ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
  VALUE_EXPRESSION_T_UIC is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
      CONVERT_R
        ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
( VALUE : DATABASE_TYPES.ADA_SQL.T_ULC;
  IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
    ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
  return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R

```

UNCLASSIFIED

```

    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_ULC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  VALUE_EXPRESSION_T_ULC is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_UTC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION is
begin
  return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

function INDICATOR
  ( VALUE : DATABASE_TYPES.ADA_SQL.T_UTC;
    IND   : ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE :=
      ADA_SQL_FUNCTIONS.NOT_NULL )
return
  ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  VALUE_EXPRESSION_T_UTC is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.CONVERT_R
    ( ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
      ( L_CONVERT ( VALUE ) , IND ) );
end INDICATOR;

procedure INTO
  ( TARGET      : out STANDARD.BOOLEAN;
    CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
      ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
  OUR_TARGET : DATABASE.INT;
begin
  ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INT0

```

UNCLASSIFIED

```

( OUR_TARGET , CURSOR );
TARGET := STANDARD.BOOLEAN'VAL ( OUR_TARGET );
end INTO;

procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_REVAL;
  LAST        : out DATABASE_TYPES.ADA_SQL.X_REVAL;
  INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
begin
  ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INT0
  ( STANDARD.STRING ( TARGET ) , STANDARD.NATURAL ( LAST ) , INDICATOR,
    CURSOR );
end INTO;

procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_SECUR;
  INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
  OUR_TARGET : DATABASE.INT;
begin
  ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INT0
  ( OUR_TARGET , INDICATOR , CURSOR );
  TARGET := DATABASE_TYPES.ADA_SQL.T_SECUR'VAL ( OUR_TARGET );
end INTO;

procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_SHORT_NAME;
  LAST        : out DATABASE_TYPES.ADA_SQL.X_SHORT_NAME;
  INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
begin
  ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INT0
  ( STANDARD.STRING ( TARGET ) , STANDARD.NATURAL ( LAST ) , INDICATOR,
    CURSOR );
end INTO;

procedure INTO
( TARGET      : out DATABASE_TYPES.ADA_SQL.T_TPSN;
  INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
    ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
  OUR_TARGET : DATABASE.INT;
begin
  ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INT0
  ( OUR_TARGET , INDICATOR , CURSOR );
  TARGET := DATABASE_TYPES.ADA_SQL.T_TPSN'VAL ( OUR_TARGET );
end INTO;

```

UNCLASSIFIED

```

procedure INTO
    ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_UIC;
      LAST        : out DATABASE_TYPES.ADA_SQL.X_UIC;
      INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
      CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
        ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
begin
    ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INT0
    ( STANDARD.STRING ( TARGET ) , STANDARD.NATURAL ( LAST ) , INDICATOR,
      CURSOR );
end INTO;

```

```

procedure INTO
    ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_ULC;
      LAST        : out DATABASE_TYPES.ADA_SQL.X_ULC;
      INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
      CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
        ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
begin
    ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INT0
    ( STANDARD.STRING ( TARGET ) , STANDARD.NATURAL ( LAST ) , INDICATOR,
      CURSOR );
end INTO;

```

```

procedure INTO
    ( TARGET      : out DATABASE_TYPES.ADA_SQL.T_UTC;
      LAST        : out DATABASE_TYPES.ADA_SQL.X_UTC;
      INDICATOR   : out ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
      CURSOR      : in  ADA_SQL_FUNCTIONS.CURSOR_NAME :=
        ADA_SQL_FUNCTIONS.NULL_CURSOR_NAME ) is
begin
    ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INT0
    ( STANDARD.STRING ( TARGET ) , STANDARD.NATURAL ( LAST ) , INDICATOR,
      CURSOR );
end INTO;

```

```

function "and"
    ( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
      R : ADA_SQL_FUNCTIONS.VALUE_SPECIFICATION )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST is
begin
    return
    ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_AND,
        ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
        ADA_SQL_FUNCTIONS.CONVERT.R_CONVERT ( R ) ) );
end "and";

```

```

function "and"
    ( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;

```

UNCLASSIFIED

```

        R : DATABASE_TYPES.ADA_SQL.T_DATE )
    return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AND,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "and";

function "and"
    ( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
      R : DATABASE_TYPES.ADA_SQL.T_SECUR )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AND,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "and";

function "and"
    ( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST;
      R : STANDARD.BOOLEAN )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AND,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "and";

function "and"
    ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE;
      R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AND,
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.L_CONVERT ( L ),
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.R_CONVERT ( R ) ) );
end "and";

function "and"

```

UNCLASSIFIED

```

        ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
          SET_CLAUSE;
          R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
          SET_CLAUSE )
    return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AND,
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.L_CONVERT ( L ),
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.R_CONVERT
              ( R ) ) ) );
end "and";

function EQ
    ( L : ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.VALUE_EXPRESSION_T_UIC;
      R : DATABASE_TYPES.ADA_SQL.T_UIC )
return ADA_SQL_FUNCTIONS.SEARCH_CONDITION is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_EQ,
            ADA_SQL.DATABASE_TYPES_TYPE_PACKAGE.L_CONVERT ( L ),
            R_CONVERT ( R ) ) ) );
end EQ;

function "&"
    ( L : ADA_SQL_FUNCTIONS.SELECT_LIST;
      R : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION )
return ADA_SQL_FUNCTIONS.SELECT_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AMPERSAND,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
            ADA_SQL_FUNCTIONS.CONVERT.R_CONVERT ( R ) ) ) );
end "&";

function "&"
    ( L : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION;
      R : ADA_SQL_FUNCTIONS.VALUE_EXPRESSION )
return ADA_SQL_FUNCTIONS.SELECT_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_AMPERSAND,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),

```

UNCLASSIFIED

```

        ADA_SQL_FUNCTIONS.CONVERT.R_CONVERT ( R ) ) );
end "&";

function "<="
( L : ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST_STARTER;
  R : DATABASE_TYPES.ADA_SQL.T_UIC )
return ADA_SQL_FUNCTIONS.INSERT_VALUE_LIST is
begin
    return
        ADA_SQL_FUNCTIONS.CONVERT.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_LE,
            ADA_SQL_FUNCTIONS.CONVERT.L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "<=";

function "<="
( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.
  COLUMN_NAME_BOOLEAN;
  R : STANDARD.BOOLEAN )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_LE,
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.STANDARD.
              L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "<=";

function "<="
( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
  DATABASE_TYPES.COLUMN_NAME_T_DATE;
  R : DATABASE_TYPES.ADA_SQL.T_DATE )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
    return
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
        ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
          ( ADA_SQL_FUNCTIONS.O_LE,
            ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
              L_CONVERT ( L ),
            R_CONVERT ( R ) ) );
end "<=";

function "<="
( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
  DATABASE_TYPES.COLUMN_NAME_T_DATE;
  R : DATABASE_TYPES.ADA_SQL.T_DATE )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE is

```

UNCLASSIFIED

```

begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_REVAL;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_REVAL )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_SECUR;
    R : DATABASE_TYPES.ADA_SQL.T_SECUR )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_SECUR;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_SECUR )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R

```

UNCLASSIFIED

```

( ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_LE,
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
      L_CONVERT ( L ),
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
      R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_SECUR;
    R : DATABASE_TYPES.ADA_SQL.T_SECUR )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_SHORT_NAME;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_SHORT_NAME )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_TPSN;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_TPSN )
  return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,

```

UNCLASSIFIED

```

ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  L_CONVERT ( L ),
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
  R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_UIC;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_UIC )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.COMMAND_PACKAGE.DATABASE_TYPES.
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_ULC;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_ULC )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          L_CONVERT ( L ),
        ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.
          R_CONVERT ( R ) ) );
end "<=";

function "<="
  ( L : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.COLUMN_NAME_T_UTC;
    R : ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.
    DATABASE_TYPES.VALUE_EXPRESSION_T_UTC )
return ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.SET_CLAUSE is
begin
  return
    ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.CONVERT_R
    ( ADA_SQL_FUNCTIONS.BINARY_OPERATION
      ( ADA_SQL_FUNCTIONS.O_LE,

```

UNCLASSIFIED

```
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.  
  L_CONVERT ( L ),  
ADA_SQL.DATABASE_DEFINITION_PACKAGE.BIDE_PACKAGE.DATABASE_TYPES.  
  R_CONVERT ( R ) ) );  
end "<=";  
  
end DATABASE_CARD_A_ADA_SQL;
```

UNCLASSIFIED

7.7 DBVARS.ADA

```
-- File: dbvars.ada
-- DATABASE_VARIABLES_ADA_SQL
-- 12/18/88
```

```
--
--
```

DISCLAIMER OF WARRANTY AND LIABILITY

```
--
```

```
-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
```

```
--
```

```
-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
```

```
--
```

```
-- Prepared for:
```

```
--
```

```
-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311
```

```
--
```

```
-- Prepared by:
```

```
--
```

```
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813 (703) 560-6799
```

```
--
```

```
with ADA_SQL_FUNCTIONS;
package DATABASE_VARIABLES_ADA_SQL is
  subtype INDICATOR_VARIABLE is ADA_SQL_FUNCTIONS.INDICATOR_VARIABLE;
  function NULL_VALUE return INDICATOR_VARIABLE
    renames ADA_SQL_FUNCTIONS.NULL_VALUE;
  function NOT_NULL return INDICATOR_VARIABLE
    renames ADA_SQL_FUNCTIONS.NOT_NULL;
end DATABASE_VARIABLES_ADA_SQL;
```

UNCLASSIFIED

UNCLASSIFIED

7.8 DBVARS.ADS

```
-- File: dbvars.ads
-- DATABASE_VARIABLES
-- 12/18/88

-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813      (703) 560-6799

with DATABASE_TYPES, DATABASE_VARIABLES_ADA_SQL;
  use DATABASE_VARIABLES_ADA_SQL;
package DATABASE_VARIABLES is

  use DATABASE_TYPES.ADA_SQL;

  V_IS_MAJOR      : BOOLEAN;

  I_MAJCOM        : INDICATOR_VARIABLE;
  V_MAJCOM        : T_UIC;
  L_MAJCOM        : X_UIC;

  I_ORG_SCLASS    : INDICATOR_VARIABLE;
  V_ORG_SCLASS    : T_SECUR;

  I_REC_SCLASS    : INDICATOR_VARIABLE range NOT_NULL .. NOT_NULL;
  V_REC_SCLASS    : T_SECUR;

  V_REPORT_DATE   : T_DATE;
  L_REPORT_DATE   : X_DATE;

  I_REVAL         : INDICATOR_VARIABLE;
  V_REVAL         : T_REVAL;
  L_REVAL         : X_REVAL;

  I_SHORT_NAME    : INDICATOR_VARIABLE;
  V_SHORT_NAME    : T_SHORT_NAME;
  L_SHORT_NAME    : X_SHORT_NAME;

  I_TPSN          : INDICATOR_VARIABLE;
  V_TPSN          : T_TPSN;

  I_UIC           : INDICATOR_VARIABLE range NOT_NULL .. NOT_NULL;
  V_UIC           : T_UIC;
  L_UIC           : X_UIC;

  I_ULC           : INDICATOR_VARIABLE;
```

UNCLASSIFIED

V_ULC : T_ULC;
L_ULC : X_ULC;

V_UPDATE_DATE : T_DATE;
L_UPDATE_DATE : X_DATE;

I_UTC : INDICATOR_VARIABLE;
V_UTC : T_UTC;
L_UTC : X_UTC;

end DATABASE_VARIABLES;

UNCLASSIFIED

7.9 DBCGEN.ADA

-- File: dbcgen.ada
-- GENERIC_DATABASE_CONVERSIONS
-- 12/18/88

DISCLAIMER OF WARRANTY AND LIABILITY

THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE, OR OTHERWISE.

USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
SPECIAL, OR OTHERWISE.

Prepared for:

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Prepared by:

Fred J. Friedman
RACOM Computer Professionals
P.O. Box 576
Annandale, VA 22003-1620
(703) 560-6813 (703) 5 0-6799

with DATABASE_VARIABLES_ADA_SQL;
use DATABASE_VARIABLES_ADA_SQL;
package GENERIC_DATABASE_CONVERSIONS is

generic

type INDEX_TYPE is range <>;

type DATABASE_TYPE is array (INDEX_TYPE) of CHARACTER;

package STRING_CONVERT is

procedure INSERT_CONVERT

UNCLASSIFIED

```

        ( SOURCE      : in  STRING;
          RESULT      : out DATABASE_TYPE;
          INDICATOR   : out INDICATOR_VARIABLE );

    procedure UPDATE_CONVERT
        ( SOURCE      : in      STRING;
          RESULT      : in out DATABASE_TYPE;
          INDICATOR   : in out INDICATOR_VARIABLE );

end STRING_CONVERT;

generic
    type DATABASE_TYPE is ( <> );
package DISCRETE_CONVERT is

    procedure INSERT_CONVERT
        ( SOURCE      : in  STRING;
          RESULT      : out DATABASE_TYPE;
          INDICATOR   : out INDICATOR_VARIABLE );

    procedure UPDATE_CONVERT
        ( SOURCE      : in      STRING;
          RESULT      : in out DATABASE_TYPE;
          INDICATOR   : in out INDICATOR_VARIABLE );

end DISCRETE_CONVERT;

procedure INSERT_CONVERT
    ( SOURCE      : in  STRING;
      RESULT      : out BOOLEAN );

procedure UPDATE_CONVERT
    ( SOURCE      : in      STRING;
      RESULT      : in out BOOLEAN );

type FIELD_STATUS is ( BLANK , POUND , DATA );
-- would go in body if not required for next function spec

function FIELD_CONTENTS ( S : STRING ) return FIELD_STATUS;
-- Meridian Ada bug shows up if this spec is not here

end GENERIC_DATABASE_CONVERSIONS;

package body GENERIC_DATABASE_CONVERSIONS is

    function FIELD_CONTENTS ( S : STRING ) return FIELD_STATUS is
    begin
        for I in S'RANGE loop
            if S(I) = '#' then
                return POUND;
            elsif S(I) /= ' ' then

```

UNCLASSIFIED

```
        return DATA;
    end if;
end loop;
return BLANK;
end FIELD_CONTENTS;
```

package body STRING_CONVERT is

```
procedure INSERT_CONVERT
( SOURCE      : in  STRING;
  RESULT      : out DATABASE_TYPE;
  INDICATOR   : out INDICATOR_VARIABLE ) is
```

```
begin
    RESULT := DATABASE_TYPE ( SOURCE );
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
            INDICATOR := NULL_VALUE;
        when POUND =>
            raise CONSTRAINT_ERROR; -- "cannot happen"
        when DATA =>
            INDICATOR := NOT_NULL;
    end case;
end INSERT_CONVERT;
```

```
procedure UPDATE_CONVERT
( SOURCE      : in  STRING;
  RESULT      : in out DATABASE_TYPE;
  INDICATOR   : in out INDICATOR_VARIABLE ) is
```

```
begin
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
            null;
        when POUND =>
            INDICATOR := NULL_VALUE;
        when DATA =>
            INDICATOR := NOT_NULL;
            RESULT := DATABASE_TYPE ( SOURCE );
    end case;
end UPDATE_CONVERT;
```

end STRING_CONVERT;

package body DISCRETE_CONVERT is

```
procedure INSERT_CONVERT
( SOURCE      : in  STRING;
  RESULT      : out DATABASE_TYPE;
  INDICATOR   : out INDICATOR_VARIABLE ) is
```

```
begin
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
```

UNCLASSIFIED

```

        INDICATOR := NULL_VALUE;
        RESULT    := DATABASE_TYPE'FIRST;
    when POUND =>
        raise CONSTRAINT_ERROR; -- "cannot happen"
    when DATA =>
        INDICATOR := NOT_NULL;
        RESULT    := DATABASE_TYPE'VALUE ( SOURCE );
    end case;
end INSERT_CONVERT;

procedure UPDATE_CONVERT
    ( SOURCE      : in    STRING;
      RESULT      : in out DATABASE_TYPE;
      INDICATOR   : in out INDICATOR_VARIABLE ) is
begin
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
            null;
        when POUND =>
            INDICATOR := NULL_VALUE;
        when DATA =>
            INDICATOR := NOT_NULL;
            RESULT    := DATABASE_TYPE'VALUE ( SOURCE );
        end case;
    end UPDATE_CONVERT;

end DISCRETE_CONVERT;

procedure INSERT_CONVERT
    ( SOURCE      : in    STRING;
      RESULT      : out BOOLEAN ) is
begin
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
            RESULT := FALSE;
        when POUND =>
            raise CONSTRAINT_ERROR; -- "cannot happen"
        when DATA =>
            RESULT := TRUE;
        end case;
    end INSERT_CONVERT;

procedure UPDATE_CONVERT
    ( SOURCE      : in    STRING;
      RESULT      : in out BOOLEAN ) is
begin
    case FIELD_CONTENTS ( SOURCE ) is
        when BLANK =>
            null;
        when POUND =>
            RESULT := FALSE;
    end case;
end UPDATE_CONVERT;

```

UNCLASSIFIED

```
      when DATA =>  
        RESULT := TRUE;  
      end case;  
    end UPDATE_CONVERT;  
  end GENERIC_DATABASE_CONVERSIONS;
```

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

7.10 DBCNVRT.ADS

-- File: dbcnvrt.ads
-- DATABASE_CONVERSIONS
-- 12/18/88

--

--

--

--

--

DISCLAIMER OF WARRANTY AND LIABILITY

--

--

--

--

--

--

THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE, OR OTHERWISE.

--

--

--

--

--

--

--

USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
SPECIAL, OR OTHERWISE.

--

--

Prepared for:

--

--

--

--

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

--

--

--

--

--

--

--

Prepared by:
Fred J. Friedman
RACOM Computer Professionals
P.O. Box 576
Annandale, VA 22003-1620
(703) 560-6813 (703) 560-6799

--

with DATABASE_TYPES, GENERIC_DATABASE_CONVERSIONS;
use GENERIC_DATABASE_CONVERSIONS;
package DATABASE_CONVERSIONS is

use DATABASE_TYPES.ADA_SQL;

package DATE_CONVERT is new STRING_CONVERT (X_DATE , T_DATE);

package REVAL_CONVERT is new STRING_CONVERT (X_REVAL , T_REVAL);

UNCLASSIFIED

```
package SECUR_CONVERT is new DISCRETE_CONVERT ( T_SECUR );

package SHORT_NAME_CONVERT is
  new STRING_CONVERT ( X_SHORT_NAME , T_SHORT_NAME );

package TPSN_CONVERT is new DISCRETE_CONVERT ( T_TPSN );

package UIC_CONVERT is new STRING_CONVERT ( X_UIC , T_UIC );

package ULC_CONVERT is new STRING_CONVERT ( X_ULC , T_ULC );

package UTC_CONVERT is new STRING_CONVERT ( X_UTC , T_UTC );

end DATABASE_CONVERSIONS;
```

UNCLASSIFIED

7.11 CARDA.ADS

-- File: carda.ads
-- DATABASE_CARD_A
-- 12/18/88

--
--

DISCLAIMER OF WARRANTY AND LIABILITY

-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.

-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.

-- Prepared for:

-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311

-- Prepared by:

-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813 (703) 560-6799

--
package DATABASE_CARD_A is

 subtype CARD_TYPE is STRING (1 .. 80);

 procedure Process_Card_A (Card : in Card_Type);

end DATABASE_CARD_A;

UNCLASSIFIED

UNCLASSIFIED

7.12 CARDA.ADB

-- File: carda.adb
-- DATABASE_CARD_A
-- 12/18/88

DISCLAIMER OF WARRANTY AND LIABILITY

THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE, OR OTHERWISE.

USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
SPECIAL, OR OTHERWISE.

Prepared for:

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Prepared by:

Fred J. Friedman
RACOM Computer Professionals
P.O. Box 576
Annandale, VA 22003-1620
(703) 560-6813 (703) 560-6799

with DATABASE_CONVERSIONS, DATABASE_VARIABLES,
DATABASE_CARD_A_ADA_SQL, GENERIC_DATABASE_CONVERSIONS;
use DATABASE_CONVERSIONS, DATABASE_VARIABLES,
DATABASE_CARD_A_ADA_SQL, GENERIC_DATABASE_CONVERSIONS;
package body DATABASE_CARD_A is

use DATE_CONVERT, REVAL_CONVERT, SECUR_CONVERT, SHORT_NAME_CONVERT,
TPSN_CONVERT, UIC_CONVERT, ULC_CONVERT, UTC_CONVERT;

procedure Process_Card_A(Card : in Card_Type) is

UNCLASSIFIED

```
function Transaction_Code ( Card : in Card_Type ) return String is
begin
    return Card ( 5 .. 5 );
end Transaction_Code;
```

```
begin
```

```
INSERT_CONVERT ( CARD ( 9 .. 14 ) , V_UIC , I_UIC );
```

```
if Transaction_Code(Card) = "A" then
```

```
INSERT_CONVERT ( CARD ( 4 .. 4 ) , V_REC_SCLASS , I_REC_SCLASS );
INSERT_CONVERT ( CARD ( 16 .. 45 ) , V_SHORT_NAME , I_SHORT_NAME );
INSERT_CONVERT ( CARD ( 46 .. 50 ) , V_UTC , I_UTC );
INSERT_CONVERT ( CARD ( 51 .. 53 ) , V_ULC , I_ULC );
INSERT_CONVERT ( CARD ( 54 .. 59 ) , V_MAJCOM , I_MAJCOM );
INSERT_CONVERT ( CARD ( 60 .. 60 ) , V_IS_MAJOR );
INSERT_CONVERT ( CARD ( 61 .. 61 ) , V_REVAL , I_REVAL );
INSERT_CONVERT ( CARD ( 62 .. 68 ) , V_TPSN , I_TPSN );
INSERT_CONVERT ( CARD ( 69 .. 69 ) , V_ORG_SCLASS , I_ORG_SCLASS );
```

```
INSERT_INTO ( BIDE ,
VALUES <=
```

```

    V_UIC and
INDICATOR ( V_SHORT_NAME , I_SHORT_NAME ) and
INDICATOR ( V_UTC , I_UTC ) and
INDICATOR ( V_ULC , I_ULC ) and
INDICATOR ( V_REVAL , I_REVAL ) and
INDICATOR ( V_TPSN , I_TPSN ) and
INDICATOR ( V_ORG_SCLASS , I_ORG_SCLASS ) and
    V_REC_SCLASS and
    V_REPORT_DATE and
    V_UPDATE_DATE );
```

```
INSERT_INTO ( COMMAND,
VALUES <=
```

```

    V_UIC and
INDICATOR ( V_MAJCOM , I_MAJCOM ) and
    V_IS_MAJOR and
    V_REC_SCLASS and
    V_REPORT_DATE and
    V_UPDATE_DATE );
```

```
elsif Transaction_Code(Card) = "D" then
```

```
DELETE ( FROM => BIDE,
WHERE => EQ ( UIC , V_UIC ) );
```

```
DELETE ( FROM => COMMAND,
WHERE => EQ ( UIC , V_UIC ) );
```

UNCLASSIFIED

elseif Transaction_Code(Card) = "C" then

```

SELEC ( SHORT_NAME & UTC & ULC & REVAL & TPSN & ORG_SCLASS,
FROM => BIDE,
WHERE => EQ ( UIC , V_UIC ) );
INTO ( V_SHORT_NAME , L_SHORT_NAME , I_SHORT_NAME );
INTO ( V_UTC , L_UTC , I_UTC );
INTO ( V_ULC , L_ULC , I_ULC );
INTO ( V_REVAL , L_REVAL , I_REVAL );
INTO ( V_TPSN , I_TPSN );
INTO ( V_ORG_SCLASS , I_ORG_SCLASS );

SELEC ( MAJCOM & IS_MAJOR,
FROM => COMMAND,
WHERE => EQ ( UIC , V_UIC ) );
INTO ( V_MAJCOM , L_MAJCOM , I_MAJCOM );
INTO ( V_IS_MAJOR );

INSERT_CONVERT ( CARD ( 4 .. 4 ) , V_REC_SCLASS , I_REC_SCLASS );
UPDATE_CONVERT ( CARD ( 16 .. 45 ) , V_SHORT_NAME , I_SHORT_NAME );
UPDATE_CONVERT ( CARD ( 46 .. 50 ) , V_UTC , I_UTC );
UPDATE_CONVERT ( CARD ( 51 .. 53 ) , V_ULC , I_ULC );
UPDATE_CONVERT ( CARD ( 54 .. 59 ) , V_MAJCOM , I_MAJCOM );
UPDATE_CONVERT ( CARD ( 60 .. 60 ) , V_IS_MAJOR );
UPDATE_CONVERT ( CARD ( 61 .. 61 ) , V_REVAL , I_REVAL );
UPDATE_CONVERT ( CARD ( 62 .. 68 ) , V_TPSN , I_TPSN );
UPDATE_CONVERT ( CARD ( 69 .. 69 ) , V_ORG_SCLASS , I_ORG_SCLASS );

UPDATE ( BIDE,
SET => SHORT_NAME <= INDICATOR ( V_SHORT_NAME , I_SHORT_NAME )
and UTC <= INDICATOR ( V_UTC , I_UTC )
and ULC <= INDICATOR ( V_ULC , I_ULC )
and REVAL <= INDICATOR ( V_REVAL , I_REVAL )
and TPSN <= INDICATOR ( V_TPSN , I_TPSN )
and ORG_SCLASS <= INDICATOR ( V_ORG_SCLASS , I_ORG_SCLASS )
and REC_SCLASS <= V_REC_SCLASS
and REPORT_DATE <= V_REPORT_DATE
and UPDATE_DATE <= V_UPDATE_DATE,
WHERE => EQ ( UIC , V_UIC ) );

UPDATE ( COMMAND,
SET => MAJCOM <= INDICATOR ( V_MAJCOM , I_MAJCOM )
and IS_MAJOR <= V_IS_MAJOR
and REC_SCLASS <= V_REC_SCLASS
and REPORT_DATE <= V_REPORT_DATE
and UPDATE_DATE <= V_UPDATE_DATE,
WHERE => EQ ( UIC , V_UIC ) );

```

else

null;

UNCLASSIFIED

```
end if;  
end Process_Card_A;  
end DATABASE_CARD_A;
```

UNCLASSIFIED

7.13 MAIN.ADA

-- File: main.ada
-- MAIN
-- 12/18/88

--
--

-- DISCLAIMER OF WARRANTY AND LIABILITY
--

-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
--

-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
--

-- Prepared for:
--

-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311
--

-- Prepared by:
-- Fred J. Friedman
-- RACOM Computer Professionals
-- P.O. Box 576
-- Annandale, VA 22003-1620
-- (703) 560-6813 (703) 560-6799
--

with DATABASE_CARD_A, DATABASE_CARD_A_ADA_SQL, DATABASE_VARIABLES, TEXT_PRINT;
use DATABASE_CARD_A, DATABASE_CARD_A_ADA_SQL, DATABASE_VARIABLES, TEXT_PRINT;
procedure MAIN is

subtype SHORT_CARD_INDEX is POSITIVE range 1 .. 69;

subtype SHORT_CARD is STRING (SHORT_CARD_INDEX);

type TEST_CARDS is array (POSITIVE range <>) of SHORT_CARD;

UNCLASSIFIED

```

CARDS : constant TEST_CARDS :=
( " UA   UIC001 Test Card 1 (add)           UTC01LC1MJCOM1 10000001U",
  " UC   UIC002 Change non-existent card      U",
  " CC   UIC001 Change with explicit fields   UTC02LC2MJCOM2X20000002C",
  " SC   UIC001 Change with blank fields      S",
  " TC   UIC001 Change with # fields          # # #   ###   T",
  " UC   UIC001 Change to show results        UTC03LC3MJCOM3X30000003U",
  " CA   UIC004 Test Card 4 (add null fields) MJCOM4      C",
  " SC   UIC004 Change with blank fields      S",
  " TC   UIC004 Change with # fields          # # #   ###   T",
  " SC   UIC004 Change to show results        UTC05LC5MJCOM5X50000005S",
  " UD   UIC004 Delete Card 4                 U",
  " UD   UIC004 Delete Card 4 again (error)    U",
  " UC   UIC004 Change Card 4 (also error)     U",
  " TC   UIC001 Change Card 1 (ok)            UTC06LC6MJCOM6X60000006T",
  " UD   UIC001 Delete Card 1 - end of demo   U" );

CARD : CARD_TYPE := ( others => ' ' );

begin

V_REPORT_DATE := "ReportDa";
V_UPDATE_DATE := "UpdateDa";

INITIATE_TEST;

for I in CARDS'RANGE loop
  BLANK_LINE; BLANK_LINE;
  PRINT ( "Card: " ); PRINT ( CARDS ( I ) ); PRINT_LINE;
  CARD ( SHORT_CARD_INDEX'FIRST .. SHORT_CARD_INDEX'LAST ) := CARDS ( I );
  begin
    PROCESS_CARD_A ( CARD );
  exception
    when NO_DATA => null;
  end;
end loop;

end MAIN;

```

7.14 Demonstration Output

```

Card:    UA   UIC001 Test Card 1 (add)           UTC01LC1MJCOM1 10000001U

INSERT INTO BIDE
VALUES ( 'UIC001', 'Test Card 1 (add)'           ' NOT_NULL, 'UTC01' NOT_NULL
        , 'LC1' NOT_NULL, '1' NOT_NULL, 1 NOT_NULL, 0 NOT_NULL, 0, 'ReportDa'
        , 'UpdateDa' )

INSERT INTO COMMAND

```

UNCLASSIFIED

VALUES ('UIC001', 'MJCOM1' NOT_NULL, 0, 0, 'ReportDa', 'UpdateDa')

Card: UC UIC002 Change non-existent card

U

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC002'
***** NO DATA *****
```

Card: CC UIC001 Change with explicit fields UTC02LC2MJCOM2X20000002C

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC001'
INTO returning 'Test Card 1 (add)' NOT_NULL
INTO returning 'UTC01' NOT_NULL
INTO returning 'LC1' NOT_NULL
INTO returning '1' NOT_NULL
INTO returning 1 NOT_NULL
INTO returning 0 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC001'
INTO returning 'MJCOM1' NOT_NULL
INTO returning 0
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change with explicit fields' NOT_NULL,
    UTC = 'UTC02' NOT_NULL,
    ULC = 'LC2' NOT_NULL,
    REVAL = '2' NOT_NULL,
    TPSN = 2 NOT_NULL,
    ORG_SCLASS = 1 NOT_NULL,
    REC_SCLASS = 1,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM2' NOT_NULL,
    IS_MAJOR = 1,
    REC_SCLASS = 1,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

Card: SC UIC001 Change with blank fields

S

UNCLASSIFIED

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC001'
INTO returning 'Change with explicit fields' NOT_NULL
INTO returning 'UTC02' NOT_NULL
INTO returning 'LC2' NOT_NULL
INTO returning '2' NOT_NULL
INTO returning 2 NOT_NULL
INTO returning 1 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC001'
INTO returning 'MJCOM2' NOT_NULL
INTO returning 1
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change with blank fields' NOT_NULL,
    UTC = 'UTC02' NOT_NULL,
    ULC = 'LC2' NOT_NULL,
    REVAL = '2' NOT_NULL,
    TPSN = 2 NOT_NULL,
    ORG_SCLASS = 2 NOT_NULL,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM2' NOT_NULL,
    IS_MAJOR = 1,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

Card: TC UIC001 Change with # fields # # # ### T

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC001'
INTO returning 'Change with blank fields' NOT_NULL
INTO returning 'UTC02' NOT_NULL
INTO returning 'LC2' NOT_NULL
INTO returning '2' NOT_NULL
INTO returning 2 NOT_NULL
INTO returning 2 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
```

UNCLASSIFIED

```
WHERE UIC = 'UIC001'
INTO returning 'MJCOM2' NOT_NULL
INTO returning 1
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change with # fields' NOT_NULL,
  UTC = 'UTC02' NULL_VALUE,
  ULC = 'LC2' NULL_VALUE,
  REVAL = '2' NULL_VALUE,
  TPSN = 2 NULL_VALUE,
  ORG_SCLASS = 3 NOT_NULL,
  REC_SCLASS = 3,
  REPORT_DATE = 'ReportDa',
  UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM2' NULL_VALUE,
  IS_MAJOR = 0,
  REC_SCLASS = 3,
  REPORT_DATE = 'ReportDa',
  UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

Card. UC UIC001 Change to show results UTC03LC3MJCOM3X30000003U

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC001'
INTO returning 'Change with # fields' NOT_NULL
INTO returning 'UTC02' NULL_VALUE
INTO returning 'LC2' NULL_VALUE
INTO returning '2' NULL_VALUE
INTO returning 2 NULL_VALUE
INTO returning 3 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC001'
INTO returning 'MJCOM2' NULL_VALUE
INTO returning 0
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change to show results' NOT_NULL,
  UTC = 'UTC03' NOT_NULL,
  ULC = 'LC3' NOT_NULL,
  REVAL = '3' NOT_NULL,
  TPSN = 3 NOT_NULL,
  ORG_SCLASS = 0 NOT_NULL,
  REC_SCLASS = 0,
```

UNCLASSIFIED

```
REPORT_DATE = 'ReportDa',
UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM3' NOT_NULL,
    IS_MAJOR = 1,
    REC_SCLASS = 0,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'
```

Card: CA UIC004 Test Card 4 (add null fields) MJCOM4 C

```
INSERT INTO BIDE
VALUES ( 'UIC004', 'Test Card 4 (add null fields)' NOT_NULL, '
        NULL_VALUE, ' NULL_VALUE, ' NULL_VALUE, 0 NULL_VALUE, 1
        NOT_NULL, 1, 'ReportDa', 'UpdateDa' )
```

```
INSERT INTO COMMAND
VALUES ( 'UIC004', 'MJCOM4' NOT_NULL, 0, 1, 'ReportDa', 'UpdateDa' )
```

Card: SC UIC004 Change with blank fields S

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC004'
INTO returning 'Test Card 4 (add null fields)' NOT_NULL
INTO returning ' NULL_VALUE
INTO returning ' NULL_VALUE
INTO returning ' NULL_VALUE
INTO returning 0 NULL_VALUE
INTO returning 1 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC004'
INTO returning 'MJCOM4' NOT_NULL
INTO returning 0
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change with blank fields' NOT_NULL,
    UTC = ' NULL_VALUE,
    ULC = ' NULL_VALUE,
    REVAL = ' NULL_VALUE,
    TPSN = 0 NULL_VALUE,
    ORG_SCLASS = 2 NOT_NULL,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
```

UNCLASSIFIED

```
UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM4' NOT_NULL,
    IS_MAJOR = 0,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

```
Card:    TC    UIC004 Change with # fields      #    #    #    ###    T
```

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC004'
INTO returning 'Change with blank fields' NOT_NULL
INTO returning ' ' NULL_VALUE
INTO returning ' ' NULL_VALUE
INTO returning ' ' NULL_VALUE
INTO returning 0 NULL_VALUE
INTO returning 2 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC004'
INTO returning 'MJCOM4' NOT_NULL
INTO returning 0
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change with # fields' NOT_NULL,
    UTC = ' ' NULL_VALUE,
    ULC = ' ' NULL_VALUE,
    REVAL = ' ' NULL_VALUE,
    TPSN = 0 NULL_VALUE,
    ORG_SCLASS = 3 NOT_NULL,
    REC_SCLASS = 3,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM4' NULL_VALUE,
    IS_MAJOR = 0,
    REC_SCLASS = 3,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

UNCLASSIFIED

Card: SC UIC004 Change to show results UTC05LC5MJCOM5X50000005S

```
SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
FROM BIDE
WHERE UIC = 'UIC004'
INTO returning 'Change with # fields' NOT_NULL
INTO returning ' ' NULL_VALUE
INTO returning ' ' NULL_VALUE
INTO returning ' ' NULL_VALUE
INTO returning 0 NULL_VALUE
INTO returning 3 NOT_NULL
```

```
SELECT MAJCOM, IS_MAJOR
FROM COMMAND
WHERE UIC = 'UIC004'
INTO returning 'MJCOM4' NULL_VALUE
INTO returning 0
```

```
UPDATE BIDE
SET SHORT_NAME = 'Change to show results' NOT_NULL,
    UTC = 'UTC05' NOT_NULL,
    ULC = 'LC5' NOT_NULL,
    REVAL = '5' NOT_NULL,
    TPSN = 5 NOT_NULL,
    ORG_SCLASS = 2 NOT_NULL,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

```
UPDATE COMMAND
SET MAJCOM = 'MJCOM5' NOT_NULL,
    IS_MAJOR = 1,
    REC_SCLASS = 2,
    REPORT_DATE = 'ReportDa',
    UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC004'
```

Card: UD UIC004 Delete Card 4 U

```
DELETE BIDE
WHERE UIC = 'UIC004'
```

```
DELETE COMMAND
WHERE UIC = 'UIC004'
```

Card: UD UIC004 Delete Card 4 again (error) U

```
DELETE BIDE
```

UNCLASSIFIED

WHERE UIC = 'UIC004'
 ***** NO DATA *****

DELETE COMMAND
 WHERE UIC = 'UIC004'
 ***** NO DATA *****

Card: UC UIC004 Change Card 4 (also error) U

SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
 FROM BIDE
 WHERE UIC = 'UIC004'
 ***** NO DATA *****

Card: TC UIC001 Change Card 1 (ok) UTC06LC6MJCOM6X60000006T

SELECT SHORT_NAME, UTC, ULC, REVAL, TPSN, ORG_SCLASS
 FROM BIDE
 WHERE UIC = 'UIC001'
 INTO returning 'Change to show results' NOT_NULL
 INTO returning 'UTC03' NOT_NULL
 INTO returning 'LC3' NOT_NULL
 INTO returning '3' NOT_NULL
 INTO returning 3 NOT_NULL
 INTO returning 0 NOT_NULL

SELECT MAJCOM, IS_MAJOR
 FROM COMMAND
 WHERE UIC = 'UIC001'
 INTO returning 'MJCOM3' NOT_NULL
 INTO returning 1

UPDATE BIDE
 SET SHORT_NAME = 'Change Card 1 (ok)' NOT_NULL,
 UTC = 'UTC06' NOT_NULL,
 ULC = 'LC6' NOT_NULL,
 REVAL = '6' NOT_NULL,
 TPSN = 6 NOT_NULL,
 ORG_SCLASS = 3 NOT_NULL,
 REC_SCLASS = 3,
 REPORT_DATE = 'ReportDa',
 UPDATE_DATE = 'UpdateDa'
 WHERE UIC = 'UIC001'

UPDATE COMMAND
 SET MAJCOM = 'MJCOM6' NOT_NULL,
 IS_MAJOR = 1,
 REC_SCLASS = 3,
 REPORT_DATE = 'ReportDa',

UNCLASSIFIED

UPDATE_DATE = 'UpdateDa'
WHERE UIC = 'UIC001'

Card: UD UIC001 Delete Card 1 - end of demo

U

DELETE BIDE
WHERE UIC = 'UIC001'

DELETE COMMAND
WHERE UIC = 'UIC001'

UNCLASSIFIED

Distribution List for IDA Document D-575

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
CPT Stephen Myatt WIS JPMO/DXP Room 5B19, The Pentagon Washington, D.C. 20330-6600	5
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Mr. Bill Allen SAIC Corporation 311 Park Place Blvd Suite 360 Clearwater, FL 34619	1
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22003	1
Ms. Kerry Hilliard 7321 Franklin Rd. Annandale, VA 22003	1
Mr. Kevin Heatwole 5124 Harford Lane Burke, VA 22015	3
Ms. Linn Roller General Dynamics P.O. Box 748 M-2 1786 Ft. Worth, TX 76101	1
Mr. Eugen Vasilescu 35 Chestnut St. Malverne, Long Island, NY 11565	3

UNCLASSIFIED

NAME AND ADDRESS	NUMBER OF COPIES
CSED Review Panel	
Dr. Dan Alpert, Director Program in Science, Technology & Society University of Illinois Room 201 912-1/2 West Illinois Street Urbana, Illinois 61801	1
Dr. Barry W. Boehm TRW Defense Systems Group MS R2-1094 One Space Park Redondo Beach, CA 90278	1
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1
Dr. John M. Palms, Vice President Academic Affairs & Professor of Physics Emory University Atlanta, GA 30322	1
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1

UNCLASSIFIED

NAME AND ADDRESS	NUMBER OF COPIES
Mr. Keith Uncapher University of Southern California Olin Hall 330A University Park Los Angeles, CA 90089-1454 IDA	1
General W.Y. Smith, HQ	1
Mr. Philip L. Major, HQ	1
Dr. Robert E. Roberts, HQ	1
Ms. Anne Douville, CSED	1
Dr. John F. Kramer, CSED	1
Mr. Terry Mayfield, CSED	1
Ms. Katydean Price, CSED	2
Mr. Bill R. Brykczynski, CSED	5
IDA Control & Distribution Vault	3