9. FORTRAN Interface

The GDI FORTRAN interface provides access from FORTRAN 77 applications to a database. To use it, the user must know:

- The query language of the target database.
- The FORTRAN 77 Language.
- How to use the GDI functions and subroutines described in this section.

The software components listed below are referenced throughout this section. Contact your local system or database administrator to determine the actual location on your system:

libraries	The main GDI library is named *libgdi.a*. Each database has its own additional library, named *libgdipg.a* for POSTGRES, *libgdiora.a* for ORACLE, and *libgdisyb.a* for SYBASE. Each database also has its own link file, named *pg_link.o* for POSTGRES, *ora_link.o* for ORACLE, and *syb_link.o* for SYBASE.
include files	The GDI FORTRAN include file is named *gdi_f77_refdefs.h* and must be included in all FORTRAN source code that executes GDI calls. It establishes a labelled common that contains standard codes for data types and error handling.
sample code	Sample code is available in the GDI source code tree. For its exact location, contact your local system or database administrator. The Makefiles in this directory will be configured correctly for your installation.

9.1 Document Organization

This section is organized as follows:

Section 9.2	Summary of all GDI functions and subroutines
Section 9.3	Database connection
Section 9.4	Query execution
Section 9.5	Error handling
Section 9.6	Complete sample program
Section 9.7	Problem tracking
Section 9.8	Known problems and restrictions

9.2 Subroutine & Function Calls

This section summarizes FORTRAN function and subroutine calls, sorted alphabetically by name.

The data type of each argument is listed in the right hand column. Character variables are of an arbitrary length.

Name	*Description*	*Type*
HEADER Variables	These header variables are defined in *gdi_f77_refdefs.h.* *GDI DATA TYPES:* GDI_INT2 GDI_INT4 GDI_REAL4 GDI_REAL8 GDI_CHAR GDI_STRING GDI_UNDEFINED *ERROR HANDLING & DEBUGGING:* GDI_SUCCESS GDI_FAILURE GDI_NOMAP GDI_NOCONN GDI_DEBUG_OFF GDI_DEBUG_ON GDI_DEBUG_VERBOSE	 *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer* *integer*
GDI_ADD_MAP_FIELD	INTEGER FUNCTION GDI_ADD_MAP_FIELD(DBCONN, MAP_ID, DB_NAME, PGM_NAME, DATA_TYPE, STR_LEN, ARRAY_LEN) *PURPOSE:* Execute a database query. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). MAP_ID Query map ID (see GDI_OPEN_MAP). DB_NAME Name of the database column in the retrieve/select list. PGM_NAME Name of the FORTRAN variable. DATA_TYPE GDI data type of PGM_NAME. STR_LEN The length if DATA_TYPE is a GDI_STRING. ARRAY_LEN If DATA_TYPE is an array, the number of elements in the array. This will always be 0 for ORACLE and SYBASE. *RETURN:* GDI_SUCCESS or GDI_FAILURE.	 *integer* *integer* *char* *char* *integer* *integer* *integer* *integer*

Name	*Description*	*Type*
GDI_CLOSE	INTEGER FUNCTION GDI_CLOSE(DBCONN) *PURPOSE:* Close the specified database connection. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). *RETURN:* GDI_SUCCESS or GDI_FAILURE.	 *integer* *integer*
GDI_CLOSE_MAP	SUBROUTINE GDI_CLOSE_MAP(DBCONN, MAP_ID) *PURPOSE:* Ends definition for a query mapping. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). MAP_ID Query map ID (see GDI_OPEN_MAP).	 *integer* *integer*
GDI_DESTROY_MAP	SUBROUTINE GDI_DESTROY_MAP(DBCONN, MAP_ID) *PURPOSE:* Destroys mapping. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). MAP_ID Query map ID (see GDI_OPEN_MAP).	 *integer* *integer*
GDI_ERROR_GET	SUBROUTINE GDI_ERROR_GET (DBCONN, ERRCODE, ERRTEXT, MAXTEXT, STATUS, SEVERITY) *PURPOSE:* Retrieve the error from the GDI error handler. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). MAXTEXT Length of ERRTEXT variable. Database message text longer than this will be truncated. *OUTPUT ARGUMENTS:* ERRCODE Error code. ERRTEXT Error message. STATUS GDI error status (GDI_SUCCESS or GDI_FAILURE). SEVERITY GDI severity level (GDI_NOERROR, GDI_WARNING, or GDI_FATAL).	 *integer* *integer* *integer* *char* *integer* *integer*

Name	*Description*	*Type*
GDI_ERROR_INIT	SUBROUTINE GDI_ERROR_INIT (DBCONN, DEBUG, THRESHOLD, RESERVED1, RESERVED2) *PURPOSE:* Initialize error handling flags. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). DEBUG Default setting is GDI_DEBUG_OFF. GDI_DEBUG_ON causes error messages to be output to *stderr*. GDI_DEBUG_VERBOSE may cause additional messages to be output. THRESHOLD Controls how severe an error must be in order to cause failure. The default setting is GDI_WARNING, which means that warning and fatal errors both return GDI_FAILURE to the calling routine. If set to GDI_FATAL, then only fatal errors return GDI_FAILURE; warnings return GDI_SUCCESS. RESERVED1 Currently not used. RESERVED2 Currently not used.	 *integer* *integer* *integer* *integer* *integer*
GDI_INIT	INTEGER FUNCTION GDI_INIT(APPNAME) *PURPOSE:* Initialize the GDI. *INPUT ARGUMENTS:* APPNAME: Program name. *RETURN:* GDI_SUCCESS or GDI_FAILURE	 *char* *integer*
GDI_OPEN	INTEGER FUNCTION GDI_OPEN (VENDOR, ACCOUNT, PASSWORD, DATABASE, SERVER, APPNAME) *PURPOSE:* Open a connection to a database. *INPUT ARGUMENTS:* VENDOR Database vendor name; currently includes *oracle* or *postgres*. ACCOUNT Database account or user name. PASSWORD Password for the account. DATABASE Database name. SERVER Server name (Sybase & Postgres only). APPNAME Program name. *RETURN:* Database connection ID. GDI_NOCONN means it failed.	 *char* *char* *char* *char* *char* *char* *integer*

Name	*Description*	*Type*
GDI_OPEN_MAP	INTEGER FUNCTION GDI_OPEN_MAP(DBCONN) *PURPOSE:* Establishes the relationship between database query columns and FORTRAN variables. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). *RETURN:* Query map id. GDI_NOMAP means it failed.	 *integer* *integer*
GDI_SUBMIT	INTEGER FUNCTION GDI_SUBMIT(DBCONN, MAP_ID, QUERY, MAXRECS, RETRIEVED, AFFECTED, MORE_DATA) *PURPOSE:* Execute a database query. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). MAP_ID Query map ID (see GDI_OPEN_MAP). QUERY Character string containing a complete database query. MAXRECS Controls how many instances are retrieved. Should be set to the maximum number of records that can fit into the FORTRAN variable. *OUTPUT ARGUMENTS:* RETRIEVED Records the number of records retrieved. AFFECTED Records the number of records affected by the query. MORE_DATA If the data available is greater than MAXRECS, MORE_DATA will be set to TRUE. *RETURN:* GDI_SUCCESS or GDI_FAILURE.	 *integer* *integer* *char* *integer* *integer* *integer* *logical*4* *integer*
GDI_TRACE	SUBROUTINE GDI_TRACE (DBCONN, STATE, FILENAME) *PURPOSE:* Turns database-specific debug on/off. *INPUT ARGUMENTS:* DBCONN Database connect ID (see GDI_OPEN). STATE TRUE turns trace on, FALSE turns it off. FILENAME Output filename (SYBASE only).	 *integer* *integer* *char*

9.3 Connecting to a Database

This section describes how to initialize the GDI with *GDI_INIT*, connect to a database with *GDI_OPEN* and disconnect from a database with *GDI_CLOSE*.

GDI_INIT initializes the GDI to communicate with the database(s) to which a program will connect. *GDI_OPEN* establishes a connection to the database. *GDI_OPEN* arguments were described in detail in Section 9.2. But since not all databases use all arguments, Table 14 summarizes which databases use each parameter.

Table 14: GDI_OPEN Parameters

Parameter	*ORACLE*	*POSTGRES*	*SYBASE*
vendor	yes	yes	yes
account	yes	no	yes
password	optional	no	yes
database	optional	optional	yes
server	no	optional	yes
appname	no	no	yes

Some *GDI_OPEN* parameters are optional.

For ORACLE, *password* is not applicable to ops$ logins. Also the entire account/password connect string may be sent in *via* the *database* parameter.

For POSTGRES, if *database* is not set, the connection will be set from the PGDATABASE environmental variable. If *server* is not set, it will be set from the PGHOST environmental variable.

GDI_OPEN returns an integer database connect handle that is used by other GDI calls; its main purpose is to store error information. If it is equal to GDI_NOCONN, it means that the connection failed. Example 1 initializes the GDI and establishes a connection to a POSTGRES database.

Example 1:

```
c  === Initialize the GDI and connect to POSTGRES database 'demo' ===
      program sample
      include '../../include/gdi_f77_refdefs.h
      character*30  VENDOR, DBNAME, DBHOST, na
      integer       DBCONN, STATUS

C     === Initialize program variables ===
      VENDOR = 'postgres'
      DBNAME = 'demo'
      DBHOST = 'heel.s2k.berkeley.edu'
      NA = ' '

C     === Initialize GDI ===
      STATUS = GDI_INIT ('sample')

C     === OPEN DATABASE CONNECTION ===
      DBCONN = GDI_OPEN (VENDOR, NA, NA, DBNAME, DBHOST, NA)
      IF (DBCONN .EQ. GDI_NOCONN) THEN
           ... handle error, described in Section 9.5...
      END IF
```

If the *database* and *server* parameters are set in the PGDATABASE and PGHOST environmental variables, all parameters to GDI_OPEN, except for *vendor*, can be blank.

GDI_CLOSE disconnects an application from the database, demonstrated in Example 2.

Example 2:

```
c  === Disconnect from the database ===
      STATUS = GDI_CLOSE (DBCONN)
```

9.4 Executing Queries

GDI_SUBMIT executes a database query, returning GDI_SUCCESS if the query succeeded and GDI_FAILURE if it did not.

The GDI distinguishes between queries that return data, as with a POSTQUEL retrieve or an SQL select, and queries that do not return data. First we will look at queries that do not return data results.

9.4.1 Queries that Do Not Return Data

Example 3 creates two classes in a POSTGRES database.[1]

Example 3:

```
      character*100 QUERY
C     This is not a retrieve so set MAP_ID and MAXRECS to 0.
      integer       MAP_ID=0, MAXRECS=0
      integer       ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA

C     ========== CREATE cnsierra CLASS ==========
      QUERY=   'create cnsierra (year=int4, julday=int4, precip=int4,' //
     &         'tmax=float4, 'tmin=float4, tmean=float4)'
      STATUS=GDI_SUBMIT (DBCONN, MAP_ID, QUERY, MAX_RECS,
     &              ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA)

C     ========== CREATE sst CLASS ==========
      QUERY=   'create sst (lat=float4, long=float4, time=float8,' //
     &         'temp=float4[6]'
      STATUS=GDI_SUBMIT (DBCONN, MAP_ID, QUERY, MAX_RECS,
     &              ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA)
```

GDI_SUBMIT executes any query. Example 4 loads data into *cnsierra*, then updates one of its attributes.

Example 4:

```
      QUERY=   'copy cnsierra from /usr/data/cnsierra.dat'
      STATUS=GDI_SUBMIT (DBCONN, MAP_ID, QUERY, MAX_RECS)

      QUERY=   'replace cnsierra (cnsierra.precip= -9.99) ' //
     &         'where cnsierra.precip=0'
      STATUS=GDI_SUBMIT (DBCONN, MAP_ID, QUERY, MAX_RECS,
     &              ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA)
```

After an update, ROWS_AFFECTED should report the number of rows that were updated. Currently this does not work for POSTGRES databases.

1. Example queries are from the *Introductory Guide to POSTGRES* by Emelia C. Villaros-Bainto.

9.4.2 Queries That Return Data

A query that returns data from the database has two steps:

1. Map each column in the query's retrieve list to a FORTRAN variable.
2. Execute the query with *GDI_SUBMIT*.

GDI_CREATE_MAP, demonstrated in Example 5, allocates a mapping to establish relationships between a query column and FORTRAN variables. It returns a MAP_ID, which is used in the other mapping calls.

Example 5:

```
C    ============ Create a query mapping ==============
     INTEGER MAP_ID

     MAP_ID = GDI_OPEN_MAP (DBCONN)
     IF (MAP_ID .EQ. GDI_NOMAP) THEN
          WRITE (6,*) 'GDI_OPEN_MAP failed.'
     END IF
```

GDI_ADD_MAP_FIELD, demonstrated in Example 6, matches a database result column to a FORTRAN variable. Each column in a query must have a corresponding map.

Example 6:

```
C    =========== Map Database Columns to FORTRAN variables ====
     REAL      LATITUDE(100), TEMP(6,100)
     REAL*8    TIME(100)
     CHAR*80   QUERY

     QUERY = 'retrieve s.latitude, s.temp, s.time) from s in sst'

     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID, 'latitude',
    &     LATITUDE, GDI_REAL4, 0, 0)

     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID, 'temp',
    &     TEMP, GDI_REAL4, 0, 6)

     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID, 'time',
    &     TIME, GDI_REAL8, 0, 0)
```

Note that the *temp* attribute in Example 6 is an array attribute in POSTGRES containing 6 values. This syntax is only valid for a POSTGRES database. Currently array support is limited to 2 dimensional arrays, and variables must be declared carefully. The size of the POSTGRES array must be first, as in *TEMP(6, 100)*.

GDI_CLOSE_MAP, demonstrated in Example 7, ends the definition for a mapping.

Example 7:

```
C    ========= End Query Mapping =================
     CALL GDI_CLOSE_MAP(MAP_ID)
```

GDI_DESTROY_MAP, demonstrated in Example 8, drops the mapping relationship, freeing all local memory allocated.

Example 8:

```
C    =============== Drop Query Map =====================
     CALL GDI_DESTROY_MAP ( DBCONN, MAP_ID)
```

The MAP_ID does not have to be destroyed after executing a query. It may be reused in subsequent queries so long as the number of columns do not change or the data types of the columns do not change.

Once the mapping has been established, the query may be executed with GDI_SUBMIT, demonstrated in Example 9.

Example 9:

```
C    ============== Execute the Query ===================
     integer       MAXRECS, ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA

     MAXRECS = 100

     STATUS=GDI_SUBMIT (DBCONN, MAP_ID, QUERY, MAXRECS,
    &              ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA)
```

MAXRECS indicates the maximum number of instances or rows of data that should be returned. It must not be set higher than the size of the FORTRAN variables. The number of rows actually retrieved will be be stored in ROWS_RETRIEVED. If more data are available than could fit into MAXRECS, the MORE_DATA flag will be set to TRUE.

9.5 Handling Errors

Some GDI functions, such as *GDI_OPEN* and *GDI_OPEN_MAP* return an integer handle that should be greater than 0 if the call succeeded. All other GDI functions return GDI_SUCCESS or GDI_FAILURE.

GDI_ERROR_GET retrieves specific error information. Example 10 calls *GDI_ERROR_GET* after detecting an error.

Example 10:

```
      character*80  ERRTXT
      integer       DBCONN, DBERR, SEVERITY

      DBCONN = GDI_OPEN (VENDOR, na, na, DBNAME, na, na)
      IF (DBCONN .EQ. GDI_NOCONN) THEN
           CALL GDI_ERROR_GET (DBCONN, DBERR, ERRTXT, 80, STATUS,
     &          SEVERITY)
           WRITE(0, *) ERRTXT
           ....... handle error .....
      END IF
```

GDI_ERROR_INIT initializes two error handling flags, *debug* and *threshold*. *debug* and *threshold* may be changed at any time. Example 11 sets *debug* to GDI_DEBUG_VERBOSE and *threshold* to GDI_WARNING.

Example 11:

```
c     === Output verbose debug messages & treat warnings as fatal ===
      CALL GDI_ERROR_INIT (DBCONN, GDI_DEBUG_VERBOSE, GDI_WARNING)
```

GDI_TRACE turns database vendor-specific tracing on and off and may be called at any time. Example 12 turns trace on.

Example 12:

```
c     === Turn database tracing on ===
      CALL GDI_TRACE (DBCONN, TRUE, FILENAME)
```

9.6 Sample Programs

This section includes complete sample FORTRAN programs. Example 13 is a POSTGRES example.

Example 13:

```
C    ========== Sample POSTGRES program ================
include '../../include/gdi_f77_refdefs.h'
C    define local variables
C    ========== Connect to database ====================
     CHARACTER*10       VENDOR, DATABASE, NA
     CHARACTER*16       PRGNAM
     INTEGER            DBCONN
C    ========== Error handling variables ===============
     CHARACTER*80       ERRTXT
     INTEGER            MAXTXT, STATUS, SEVERITY, ERRCDE
c    ========== Query variables ========================
     INTEGER*4          MAP_ID
     CHARACTER*80       QUERY
     INTEGER            MAXRECS, ROWS_RETRIEVED, ROWS_AFFECTED, ROWS_LEFT
     LOGICAL            MORE_DATA
C   ============ Output Variables ======================
     REAL*8             TIME(20)
     INTEGER            NSAMP(20)
     CHARACTER*16       STA(20)
     INTEGER            I

     VENDOR = 'postgres'
     DATABSE = 'geodemo'
     PRGNAM = 'gdi_f77_pg_test'

C   Some GDI_OPEN arguments are Not Applicable (NA) to POSTGRES
     NA = ' '

     MAXRECS = 20
     MAXTXT = 80

C    ================== Initialize the GDI.=====================
     STATUS = GDI_INIT(PRGNAM)
     IF (STATUS .NE. GDI_SUCCESS) THEN
          WRITE (6,*) 'GDI_INIT Failed. Program exiting.'
          GOTO 999
     END IF

C    ================== Open a connection to the database.======
     DBCONN = GDI_OPEN(VENDOR, NA, NA, DATABASE, NA, PRGNAM)
     IF (DBCONN .EQ. GDI_NOCONN) THEN
          CALL GDI_ERROR_GET(DBCONN, ERRCDE, ERRTXT, MAXTXT,
   +           STATUS, SEVERITY)
          WRITE (6,*) 'GDI_OPEN Failed: Error Code ', ERRCDE
          WRITE (6,*) ERRTXT
          GOTO 999
     END IF
```

```
C    Setting GDI_DEBUG_ON prints errors to the screen.
     CALL GDI_ERROR_INIT(DBCONN, GDI_DEBUG_ON,GDI_WARNING,
    +     RESERVED1, RESERVED2)

C    ================= Build a query.============================
     QUERY = 'retrieve (w.time, w.nsamp, w.sta) from w in wfdisc'

C    ================= Create query mapping.===================
     MAP_ID = GDI_OPEN_MAP(DBCONN)
     IF (MAP_ID .EQ. GDI_NOMAP) THEN
          GOTO 999
     END IF

C    Map each attribute being retrieved to a FORTRAN variable.
     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID,
    +     'time', TIME, GDI_REAL8, 0, 0)
     IF (STATUS .NE. GDI_SUCCESS) THEN
          GOTO 999
     END IF

     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID,
    +     'nsamp', NSAMP, GDI_INT4, 0, 0)
     IF (STATUS .NE. GDI_SUCCESS) THEN
          GOTO 999
     END IF

     STATUS = GDI_ADD_MAP_FIELD(DBCONN, MAP_ID,
    +     'sta', STA, GDI_STRING, 16, 0)
     IF (STATUS .NE. GDI_SUCCESS) THEN
          GOTO 999
     END IF

     CALL GDI_CLOSE_MAP(DBCONN, MAP_ID)

C    ================ Execute the query ============================
     STATUS = GDI_SUBMIT(DBCONN, MAP_ID, QUERY, MAXRECS,
    +     ROWS_RETRIEVED, ROWS_AFFECTED, MORE_DATA)
     IF (STATUS .NE. GDI_SUCCESS) THEN
          GOTO 999
     END IF

C    ================ Print out retrieved data. ===================
     WRITE (6,*) ROWS_AFFECTED, ' rows satisfied the query.'
     WRITE (6,*) ROWS_RETRIEVED, ' rows were retrieved.'
     DO 10 I = 1, ROWS_RETRIEVED
          WRITE (6,*) STA(I), TIME(I), NSAMP(I)
10   CONTINUE

     IF (MORE_DATA) THEN
          ROWS_LEFT = ROWS_AFFECTED - ROWS_RETRIEVED
```

```
          WRITE (6,*) ROWS_LEFT, ' more rows are available.'
     ELSE
          WRITE (6,*) 'No more data exists in the database.'
     END IF

C    Destroy query mapping.
     CALL GDI_DESTROY_MAP(DBCONN, MAP_ID)

999  STATUS = GDI_CLOSE(DBCONN)
     END
```

When run on a database containing seismic data, output looks like this:

```
% gdi_f77_pg_test
   63 rows satisfied the query.
   20 rows were successfully retrieved from the database.
BLA      636710425.00000   14280
MOX      636710786.05000   1180
MO       636710786.05000   1180
MO       636710786.05000   1180
WRA      636710849.49200   2400
WRA      636710849.49200   2400
ASAR     636710887.89900   2400
ASAR     636710887.89900   2400
ARA0     636711023.70900   4797
ARA0     636711023.70900   4800
LTX      636711827.00000   10320
GRF      636713180.00000   2400
GRF      636713559.00000   2400
KBA      636713564.00200   12000
ASAR     636713609.66400   2400
ASAR     636713609.66400   2400
NRA0     636713630.60300   4792
NRA0     636713630.60300   4800
GRF      636713920.00000   2400
GAR      636713921.69900   2400
   43 more rows are available.
```

9.7 Troubleshooting Tips

Here are a few tips for when things do not work as expected:

- Test database queries interactively before putting them into a program.
- *GDI_ERROR_INIT* with the debug flag set to *GDI_DEBUG_ON* outputs errors to the screen.
- *GDI_ERROR_INIT* with the debug flag set to *GDI_DEBUG_VERBOSE* outputs debug messages to the screen.
- *GDI_TRACE* set to *TRUE* outputs database-specific debugging messages.

9.8 Current Restrictions

<u>POSTGRES</u>

- GDI_SUBMIT:
  ROWS_AFFECTED will not be set unless the command was an APPEND.
- BUILTIN TYPES:
  The following builtin types are not directly supported yet:

  large objects
  types composed of a structure, such as box and polygon

  The GDI will return these types as a string to the application.
- USER-DEFINED TYPES:
  The following SEQUIOA types are handled:
  char2
  char4
  char8

  Adding new types requires changing source code and recompiling. We are working on a new strategy to dynamically manage types.
- DATABASE NULLS:
  If a database attribute is NULL (*i.e.,* it does not have a value), the output variable will be assigned a value as follows:

GDI_INT2, GDI_INT4:	0
GDI_REAL4, GDI_REAL8:	0.0
GDI_STRING:	blank padded to the size of the FORTRAN variable
GDI_CHAR:	blank

- NAMING COLUMNS IN QUERIES:
  The GDI cannot determine the type of some named columns.

  Instead of this: retrieve (**my_name=**p.name) from p in foo
  Do this: retrieve(p.name) from p in foo