NAME

blr_assignment −assign value

SYNTAX

blr_assignment *from-value to-value*

DESCRIPTION

The **blr_assignment** statement assigns a supplied value to a database field or message parameter.

You can assign values in the substatements of:

- **blr_modify** or **blr_store**. The assignment to a database field can reference only the context established by that modify or store statement.
- **blr_send**. The assignment to a message parameter must occur within a substatement of a **blr_send** statement of the appropriate message type.

PARAMETERS

from-value A valid value expression. The access method copies this value to *to-value*. If *from-value* specifies a missing value, *to-value* will also be missing.

to-value A database field or message parameter that is the target of the assignment.

EXAMPLE

Examples of the **blr_assignment** statement appear at the vertical line at the right margin:

```
static char
         gds_$21 [89] = {
           blr_version4,
           blr_begin,
             /* message number is 0, contains 3 params */
             blr_message, 0, 3,0,
               blr_date, /* param 0: date */
               blr_long, 0, /* param 1: long scale 0 */
               blr_cstring, 6,0,
               /* param 2: null-terminated text 6 char */
             blr_receive, 0, /* receive message 0 */
               /* store a record into ORDER_ITEMS cxt = 0 */
               blr_store,
                 blr_relation, 11,
          'O','R','D','E','R','_','I','T','E','M','S', 0,
                 blr_begin,
                   blr_assignment,
                     blr_parameter, 0, 1,0,
                     blr_field, 0, 12,
            'O','R','D','E','R','_',
            'N','U','M','B','E','R',
                   blr_assignment,
                     blr_parameter, 0, 2,0,
                     /* msg 0, param 2 goes into */
                     blr_field, 0, 11,
            'I','T','E','M','_',
            'N','U','M','B','E','R',
                   blr_assignment,
                     blr_parameter, 0, 0,0,
                     blr_field, 0, 9,
            'S','H','I','P','_',
            'D','A','T','E',
                   blr_end,
           blr_end,
         blr_eoc
          };
```

See Chapter 6 for other examples of this statement.

SEE ALSO

See the entries in this chapter for:

- **blr_modify**
- **blr_send**
- **blr_store**
- **blr_store2**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_boolean −relationship between value-expressions

SYNTAX

boolean-expression ::= { *relational-operator value-1 value-2* |
blr_not *boolean-expression* |
blr_and *boolean-1 boolean-2* |
blr_or *boolean-1 boolean-2* |
blr_missing *value-expression* |
blr_any *rse* |
blr_unique *rse* |
blr_between *value-1 value-2 value-3*}

relational-operator ::= { **blr_containing** |
blr_eql |
blr_geq |
blr_gtr |
blr_leq |
blr_lss |
blr_matching |
blr_neq |
blr_starting }

DESCRIPTION

A *boolean-expression* evaluates to true, false, or missing. It describes the characteristics of a single value expression (for example, a missing value) or the relationship between two value expressions (for example, *x* is greater than *y*).

The order of precedence for evaluating compound Boolean expressions is not—and—or.

Boolean expressions are constructed in reverse Polish notation. That is, *blr_and value-1 value-2* is legal, while *value-1 blr_and value_2* is not.

PARAMETERS

relational-operator Specifies the relationship between two value expressions, *value-1* and *value-2*. The following table lists all supported relational operators.

Table: Relational Operators

Value	Relationship	Case-Sensitive?
blr_containing	*value-1* contains *value-2*	no
blr_eql	*value-1* equals *value-2*	yes
blr_geq	*value-1* greater than or equal to *value-2*	yes
blr_gtr	*value-1* greater than *value-2*	yes
blr_leq	*value-1* less than or equal to *value-2*	yes
blr_lss	*value-1* less than *value-2*	yes
blr_matching	*value-1* matches *value-2*	no
blr_neq	*value-1* does not equal *value-2*	yes
blr_starting	*value-1* starts with *value-2*	yes

In all cases, if the value expressed by *value-1* or *value-2* is missing, the result of the test is missing.

Blr_containing, **blr_starting**, and **blr_matching** all conduct substring searches, but have different rules for case sensitivity. This disparity results in different search optimizations. Generally speaking, you may have some conventions for data entry that result in a predictable pattern of uppercase and lowercase characters. Because **blr_starting** looks only at the first characters in a field, it can utilize indexes on that field. The access method conducts a less optimized search for **blr_containing**.

The following table describes logical and other operators, and their meaning in Boolean expressions.

Table: Logical Operators

Operator	Meaning
blr_not	Negates the specified Boolean expression.
blr_and	‘‘Ands’’ the truth values of two Boolean expressions.
blr_or	‘‘Ors’’ the truth values of two Boolean expressions.
blr_missing	Tests for the absence of a value in the field specified by the value expression. It is true if the value of the expression is missing. If you add **blr_not**, the expression is true if the specified value expression does not evaluate to missing.
blr_any	Tests for the presence of any record in a stream. This expression is true if the record stream specified by *rse* includes at least one record. If you add **blr_not**, the expression is true if there are no records in the record stream.
blr_unique	Tests for the presence of a single record in a record stream. This expression is true if the record stream specified by *rse* consists of only one record. If you add **blr_not**, the condition is true if there is more than one record in the record stream or if the record stream is empty.

The following table provides truth values for compound Boolean expressions.

Values of A and B		**NOT Condition**	**AND Condition**	**OR Condition**
A	**B**	**NOT A**	**A AND B**	**A OR B**
True	True	False	True	True
True	False	False	False	True
True	Missing	False	Missing	True
False	True	True	False	True
False	False	True	False	False
False	Missing	True	False	Missing
Missing	Missing	Missing	Missing	Missing

EXAMPLE

Consider the following query in **qli**:

```
QLI> print last_name of customers with credit_rating missing
```

This query generates the relatively simple request that follows, in which the Boolean expression appears at the vertical line at the right margin:

```
blr_version4,
       blr_begin,
         blr_message, 1, 3,0,/* message 1, three parameters */
           blr_varying, 20,0, /* parameter 0: last_name */
           blr_short, 0, /* parameter 1: missing parameter for p0 */
           blr_short, 0, /* parameter 2: EOS indicator */
         blr_begin, /* Make block of FOR loop and terminating SEND */
           blr_for,  /* begin FOR loop */
             blr_rse, 1, /* rse is composed of one record stream */
               /* relation id for customers, context 0 */
               blr_rid, 12,0, 0,
               blr_boolean, /* restriction expression */
                 blr_missing, /* true for a missing value */
                   blr_fid, 0, 0,0, /* of field id 0 in context 0 */
             blr_end,                    /* end of the rse */
             blr_begin,
               blr_send, 1,
                 blr_begin,
                   /* value of last name or mis
```

NAME

blr_erase −erase record

SYNTAX

blr_erase *context-variable*

DESCRIPTION

The **blr_erase** statement removes a record from a relation in the database.

The access method does not update the database if the request unwinds or if some error occurs.

IF you accepted the default dbkey scope (that is, the transaction) on the call to **gds_$attach_database** or **gds_$create_database** and then refer to an erase record in the same transaction that erased it, the access method returns an error. See the entries in Chapter 10 for **gds_$attach_database** or **gds_$create_database** for more information.

PARAMETERS

context-variable Identifies the record stream from which the current record is to be erased. The notion of ‘‘current record’’ refers only to the record currently being processed in a stream.

The context variable cannot refer to a record from a view or union.

EXAMPLE

```
/*
        * This (unlikely) statement erases all records in the ORDERS
        * relation.
        */

       blr_version4,
       blr_begin,
         blr_for,            /* establish a FOR loop */
           blr_rse, 1,     /* comprised of one relation, context 0 */
             blr_relation, 6, ’O’,’R’,’D’,’E’,’R’,’S’, 0,
           blr_erase, 0,   /* erase records from context 0 */
                blr_end,
       blr_eoc
```

SEE ALSO

See the entry in this chapter for **blr_for**. See the entries in Chapter 10 for:

- **gds_$attach_database**
- **gds_$create_database**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_fetch −retrieve record by database key

SYNTAX

> **blr_fetch** *relation-name context-variable*
> *value-expression blr-statement*

DESCRIPTION

The **blr_fetch** statement evaluates a value expression as a database key (dbkey), retrieves the corresponding record, and then performs a request language statement.

Once you have fetched the record, you can read, modify, or erase it with request language statements that reference the context variable.

The **blr_fetch** statement can be used to establish collections of records that you can process.

PARAMETERS

relation-name Identifies the relation from which the record is to be retrieved. You can use either **blr_relation** or **blr_relation_id** in the record selection expression to specify *relation-name*.

context-variable Identifies record stream from which the record is to be retrieved.

value-expression A value expression that the access method interprets as a database key (dbkey).

blr-statement Any valid request language statement.

EXAMPLE

An example of the **blr_fetch** statement appears in bold print in the following request:

```
.
        .
        .
        .
        .
        .
        .
        .
        .
        .
        DATABASE    DB = FILENAME "test_0.gdb";
        static int
          *req_0 = 0;

        static char
          fetch_0 [] = {
            blr_version4,
            blr_begin,
              blr_message, 0, 1,0,
                blr_cstring, 31,0,
              blr_message, 1, 1, 0,
                blr_text, 8, 0,
            blr_begin,
              blr_receive, 1,
              blr_fetch,
                blr_relation, 9,
                  'C','U','S','T','O','M','E','R','S', 0,
                blr_parameter, 1,0,0,
                blr_send, 0,
                  blr_assignment,
                    blr_field, 0, 9,
             'F','U','L','L','_',
             'N','A','M','E',
                    blr_parameter, 0, 0,0,
            blr_end,
          blr_end,
        blr_eoc
        };

        main()
        {

        ready;
        start_transaction;

        for c0 in customers sorted by c0.last_name, c0.first_name
          {
          printf ("Name from for loop: %s0, c0.full_name);
          fetch (c0.rdb$db_key, sizeof (c0.rdb$db_key));
          }
        end_for;
        commit;
        finish;
        }

        fetch (db_key, key_length)
          char db_key [];
          int key_length;
        {
        struct {
          char  name [31]; /* full_name */
          } msg_1;

        if (!req_0)
          gds_$compile_request (
            *gds_$null, DB, req_0,
            (short) sizeof(fetch_0), fetch_0);
```

```
gds_$start_and_send (
  *gds_$null, req_0, gds_$trans,
  0, (short) key_length, *db_key, 0);

gds_$receive (*gds_$null, req_0, 1, sizeof(msg_1), msg_1 , 0);
printf ("Name from FETCH: %s0, msg_1.name);
}
```

SEE ALSO

See the entries in this chapter for:

- **blr_for**
- **blr_erase**
- **blr_modify**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_for −repeating loop

SYNTAX

blr_for *rse blr-statement*

DESCRIPTION

The **blr_for** statement evaluates a record selection expression and executes a substatement for each record in the resulting record stream.

PARAMETERS

rse Record selection expression. If you do not specify a sort order in the record selection expression, the access method returns the record in an undefined order.

blr-statement Any valid request language statement. This statement is typically a **blr_begin/blr_end**, thus letting you include as many request actions as you need.

EXAMPLE

Consider the following query in **qli**:

QLI> *print last_name of customers with credit_rating missing*

This query generates the relatively simple request that follows, in which the for loop can be found at the vertical line at the right margin:

```
blr_version4,
       blr_begin,
         blr_message, 1, 3,0,/* message 1, three parameters */
           blr_varying, 20,0, /* parameter 0: last_name */
           blr_short, 0, /* parameter 1: missing parameter for p0 */
           blr_short, 0, /* parameter 2: EOS indicator */
         blr_begin, /* Make block of FOR loop and terminating SEND */
           blr_for,  /* begin FOR loop */
             blr_rse, 1, /* rse is composed of one record stream */
               /* relation id for customers, context 0 */
               blr_rid, 12,0, 0,
               blr_boolean, /* restriction expression */
                 blr_missing, /* true for a missing value */
                   blr_fid, 0, 0,0, /* of field id 0 in context 0 */
             blr_end,                    /* end of the rse */
             blr_begin,
               blr_send, 1,
                 blr_begin,
                   /* value of last name or missing flag */
                   blr_assignment,
                     blr_fid, 0, 4,0,
                     blr_parameter2, 1, 0,0, 1,0,
                   blr_assignment, /* EOS is not true */
                     blr_literal, blr_short, 0, 0,0,
                     blr_parameter, 1, 2,0,
                   blr_end, /* end of assignment list */
           blr_end,            /* end of FOR loop action */
           blr_send, 1,              /* send an EOS message back */
             blr_assignment,
               blr_literal, blr_short, 0, 1,0,
               blr_parameter, 1, 2,0,
           blr_end,               /* end the block of FOR and SEND */
         blr_end,
       blr_eoc
```

See Chapter 6 for other examples.

SEE ALSO

See the entries in this chapter for:

- **blr_rse**
- **blr_loop**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_handler −error handling

SYNTAX

blr_handler *blr-statement*

DESCRIPTION

The **blr_handler** statement lets you exert control over program flow in the event of errors or other abnormal conditions.

Ordinarily, an error causes the entire request to unwind. For some errors, such as validation failures in interactive systems, it would be more appropriate to unwind just the failed statement and retry it.

The **blr_handler** statement is often used with a **blr_modify** statement or a **blr_store** statement in a for loop. If the error is correctable, such as in a validation error, the **blr_handler** statement lets you retry the update without unwinding the entire loop.

PARAMETERS

blr-statement Any valid request language statement.

EXAMPLE

See Chapter 8 for an example of **blr_handler**.

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_if −conditional branching

SYNTAX

blr_if *boolean-expression blr-statement*
{ **blr_end** | *blr-statement...* }

DESCRIPTION

The **blr_if** statement evaluates a Boolean expression and executes a statement depending on the result. If the expression evaluates to:

- ‘‘True,’’ the access method executes the required *blr-statement* and then terminates the branch with **blr_end**.
- ‘‘False’’ or ‘‘missing,’’ the access method executes the optional *blr-statement* (if there is one). Otherwise, it terminates the branch with **blr_end**.

If there is only the required *blr-statement*, the logic is *if-then-end*. If you include the optional *blr-statement* to handle Boolean expressions that evaluate to false or missing, the logic is *if-then-else*.

PARAMETERS

boolean-expression A valid Boolean expression.

EXAMPLE

The following example returns *1* if there is at least one customer. Note that the main action of the routine is a **blr_send**, and that the **blr_if** is the action under the send.

```
blr_begin,
         blr_message, 1, 1,0,
           blr_short, 0, /* message for TRUE/FALSE */
         blr_send, 1,
           blr_if,
             blr_any,
               blr_rse, 1,  /* RSE section of IF */
                 blr_rid, 12,0, 0, /* relation id of customers */
             blr_end,
           blr_assignment,  /* action if condition true */
             blr_literal, blr_short, 0, 1,0,
             blr_parameter, 1, 0,0,
           blr_assignment,  /* action if condition FALSE */
             blr_literal, blr_short, 0, 0,0,
             blr_parameter, 1, 0,0,
         blr_end,
       blr_eoc
```

SEE ALSO

See the entry in this chapter for **boolean-expression**.

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_label −name program section

SYNTAX

blr_label *label-name* *blr-statement*

label-name ::= byte

DESCRIPTION

The **blr_label** declaration names a program section or module.

This declaration lets you name a program section with a numeric label. You can then use the **blr_leave** statement to exit from a loop, return to an outer routine, and continue.

PARAMETERS

label-name User-supplied identifier for a program module.

blr-statement Any valid request language statement This statement is typically a **blr_begin** / **blr_end**.

EXAMPLE

An example of the **blr_label** declaration appears at the vertical line at the right margin:

```
static char
         blr_version4,
         blr_begin,
           blr_message, 2, 1,0,
             blr_short, 0,  /* utility message */
           blr_message, 1, 1,0,
             blr_long, 0,  /* new value for credit rating */
           blr_message, 0, 3,0,
             blr_long, 0,  /* credit rating */
             blr_short, 0,  /* EOS indicator */
             blr_cstring, 31,0,  /* name          */
           blr_begin,
             blr_for,
               blr_rse, 1,
                 blr_relation, 9,
           'C','U','S','T','O','M','E','R','S', 0,
               blr_end,
               blr_begin,
                 blr_send, 0,  /* database sends name and */
                 blr_begin,  /* credit rating to program */
                 blr_assignment,
                   blr_field, 0, 13,
            'C','R','E','D','I','T',
            '_','R','A','T','I','N','G',
                   blr_parameter, 0, 0,0,
                 blr_assignment,
                   blr_literal, blr_short, 0, 1,0,
                   blr_parameter, 0, 1,0,
                 blr_assignment,
                   blr_field, 0, 9,
            ’F’,’U’,’L’,’L’,’_’,’N’,’A’,’M’,’E’,
                   blr_parameter, 0, 2,0,
               blr_end,
               /* database enters a select/retrieve loop */
               blr_label, 0,
                 blr_loop,
                   blr_select,
                     /* message 2 indicates this record is done */
                     /* so exit from loop 0                     */
                     blr_receive, 2,
                       blr_leave, 0,
                     /* message 1 is a new value for credit rating */
                     blr_receive, 1,
                       blr_modify, 0, 1,
                         blr_begin,
                           blr_assignment,
                             blr_parameter, 1, 0,0,
                             blr_field, 1, 13,
                             ’C’,’R’,’E’,’D’,’I’,’T’,’_’,
                             ’R’,’A’,’T’,’I’,’N’,’G’,
                           blr_end,
                 blr_end,  /* end of loop */
             blr_end,              /* end of actions for FOR */
           blr_send, 0,           /* send back EOS */
             blr_assignment,
             blr_literal, blr_short, 0, 0,0,
             blr_parameter, 0, 1,0,
           blr_end,         /* end of block of FOR/SEND */
         blr_end,        /* end of entire request */
       blr_eoc
          };
```

See Chapter 8 for other examples.

SEE ALSO

See the entries in this chapter for:

- **blr_leave**
- **blr_begin**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_leave −redirecting program flow

SYNTAX

blr_leave *label-name*

label-name ::= byte

DESCRIPTION

The **blr_leave** statement lets a program exit a loop, return to an outer level identified by a label, and continue.

PARAMETERS

label-name User-supplied identifier for a program module assigned in a **blr_label** statement.

EXAMPLE

An example of the **blr_leave** statement appears at the vertical line at the right margin:

```
static char
         blr_version4,
         blr_begin,
           blr_message, 2, 1,0,
             blr_short, 0,  /* utility message */
           blr_message, 1, 1,0,
             blr_long, 0,  /* new value for credit rating */
           blr_message, 0, 3,0,
             blr_long, 0,  /* credit rating */
             blr_short, 0,  /* EOS indicator */
             blr_cstring, 31,0,  /* name          */
           blr_begin,
             blr_for,
               blr_rse, 1,
                 blr_relation, 9,
           'C','U','S','T','O','M','E','R','S', 0,
               blr_end,
               blr_begin,
                 blr_send, 0,  /* database sends name and */
                 blr_begin,  /* credit rating to program */
                 blr_assignment,
                   blr_field, 0, 13,
            'C','R','E','D','I','T',
            '_','R','A','T','I','N','G',
                   blr_parameter, 0, 0,0,
                 blr_assignment,
                   blr_literal, blr_short, 0, 1,0,
                   blr_parameter, 0, 1,0,
                 blr_assignment,
                   blr_field, 0, 9,
            ’F’,’U’,’L’,’L’,’_’,’N’,’A’,’M’,’E’,
                   blr_parameter, 0, 2,0,
               blr_end,
               /* database enters a select/retrieve loop */
               blr_label, 0,
                 blr_loop,
                   blr_select,
           /* message 2 indicates this record is done */
           /* so exit from loop 0                     */
                     blr_receive, 2,
                       **blr_leave, 0,**
           /* message 1 is a new value for credit rating */
                     blr_receive, 1,
                       blr_modify, 0, 1,
                         blr_begin,
                           blr_assignment,
                             blr_parameter, 1, 0,0,
                             blr_field, 1, 13,
               ’C’,’R’,’E’,’D’,’I’,’T’,’_’,
               ’R’,’A’,’T’,’I’,’N’,’G’,
                           blr_end,
                 blr_end,           /* end of loop */
             blr_end,              /* end of actions for FOR */
           blr_send, 0,           /* send back EOS */
             blr_assignment,
             blr_literal, blr_short, 0, 0,0,
             blr_parameter, 0, 1,0,
           blr_end,         /* end of block of FOR/SEND */
         blr_end,        /* end of entire request */
       blr_eoc
          };
```

See Chapter 8 for other examples.

SEE ALSO

See the entry in this chapter for **blr_label**.

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_loop −loop

SYNTAX

blr_loop *blr-statement*

DESCRIPTION

The **blr_loop** executes a statement until some event unwinds the request. The following events cause a request to unwind:

- An error
- A **blr_leave** statement
- Restart of the request by **gds_$start_request**
- Release of the request by **gds_$release_request**
- Unwinding of the request by **gds_$unwind_request**
- Ending a transaction by **gds_$commit_transaction** or **gds_$rollback_transaction**
- Detaching the database by **gds_$detach_database**
- Powerdown, hardware fault, operating system crash, or other system failure

PARAMETERS

blr-statement Any valid request language statement.

EXAMPLE

An example of the **blr_loop** statement appears at the vertical line at the right margin:

```
static char
          blr_version4,
          blr_begin,
            blr_message, 2, 1,0,
              blr_short, 0,  /* utility message */
            blr_message, 1, 1,0,
              blr_long, 0,  /* new value for credit rating */
            blr_message, 0, 3,0,
              blr_long, 0,  /* credit rating */
              blr_short, 0,  /* EOS indicator */
              blr_cstring, 31,0, /* name */
            blr_begin,
              blr_for,
                blr_rse, 1,
                  blr_relation, 9,
           'C','U','S','T','O','M','E','R','S', 0,
                blr_end,
                blr_begin,
                  blr_send, 0,  /* database sends name and */
                  blr_begin,  /* credit rating to program */
                  blr_assignment,
                    blr_field, 0, 13,
            ’C’,’R’,’E’,’D’,’I’,’T’,
            ’_’,’R’,’A’,’T’,’I’,’N’,’G’,
                    blr_parameter, 0, 0,0,
                  blr_assignment,
                    blr_literal, blr_short, 0, 1,0,
                    blr_parameter, 0, 1,0,
                  blr_assignment,
                    blr_field, 0, 9,
            ’F’,’U’,’L’,’L’,’_’,’N’,’A’,’M’,’E’,
                    blr_parameter, 0, 2,0,
                blr_end,
                /* database enters a select/retrieve loop */
                blr_label, 0,
                  blr_loop,
                    blr_select,
                      /* message 2 indicates this record is done */
                      /* so exit from loop 0                     */
                      blr_receive, 2,
                        blr_leave, 0,
                      /* message 1 is a new value for credit rating */
                      blr_receive, 1,
                        blr_modify, 0, 1,
                          blr_begin,
                            blr_assignment,
                              blr_parameter, 1, 0,0,
                              blr_field, 1, 13,
                              ’C’,’R’,’E’,’D’,’I’,’T’,’_’,
                              ’R’,’A’,’T’,’I’,’N’,’G’,
                            blr_end,
                  blr_end,           /* end of loop */
              blr_end,              /* end of actions for FOR */
            blr_send, 0,           /* send back EOS */
              blr_assignment,
              blr_literal, blr_short, 0, 0,0,
              blr_parameter, 0, 1,0,
            blr_end,         /* end of block of FOR/SEND */
          blr_end,        /* end of entire request */
        blr_eoc
           };
```

See Chapter 8 for other examples.

SEE ALSO

See the entry in this chapter for **blr_leave**. See also the entries in Chapter 10 for:

- **gds_$start_request**
- **gds_$release_request**
- **gds_$unwind_request**
- **gds_$commit_transaction**
- **gds_$rollback_transaction**
- **gds_$detach_database**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_message −define message type

SYNTAX

blr_message *message-type field-count [field-desc...]*

message-type ::= byte
field-count ::= word
field-desc ::= vector_byte...

DESCRIPTION

The **blr_message** declaration defines a message type and its template.

The fields in a message are densely packed. Therefore, the offset of each field is the sum of the lengths of preceding fields. When alignment is required, you must declare dummy filler fields to round out the message to the required boundary.

For each message, you must construct a buffer in your program, and be able to reference each field.

The following table lists the logical datatype values and their corresponding datatype by language:

Table: Datatype Support by Language

Datatype	BASIC	C	COBOL	FORTRAN	Pascal	PL/I
blr_short	word	short	s9(4) comp	I*2	integer	fixed binary(15)
blr_long	long	long	s9(9) comp	I*4	integer32	fixed
blr_float	single	float	comp-1	real	real	float binary(24)
blr_double	double	double	comp-2	double_ precision	double	float binary(53)
blr_text	string	char[n]	pic x (n)	character dimension(n)	array[1...n] of char	character(n)
blr_cstring	NA	char	NA	NA	NA	NA
blr_varying	string	char[n]	pic x (n)	character dimension(n)	array[1...n] of char	character(n)
blr_date	gds_$quad	gds_$quad	gds_$quad	gds_$quad	gds_$quad	gds_$quad
blr_blob	gds_$quad	gds_$quad	gds_$quad	gds_$quad	gds_$quad	gds_$quad

This table lists the datatypes by size and range/precision.

Datatype	Size	Range/Precision
blr_short	16 bits	-32768 to 32767
blr_long	32 bits	-2**31 to (2**31)-1
blr_float	32 bits	approx. 7 decimal digits
blr_double	64 bits	approx. 15 decimal digits
blr_text	*n* bytes	0 to 32767 characters
blr_cstring	?	?
blr_varying	varies	0 to 32767 characters
blr_date	64 bits	1 January 100 to 11 December 5941
blr_blob	varies	none

PARAMETERS

message-type Identifier that is unique within a request. When you define message types, it is recommended that you assign types densely starting at zero (that is, 0, 1, 2, 3...).

field-count Number of fields in the message. Zero is a valid value for *field-count*; you may want to use a message with no fields for synchronizing communication between your program and the access method. Filler fields must be included in the count.

field-desc Description of the field(s) in the message.

EXAMPLE

An example of the **blr_message** declaration appears at the vertical line at the right margin:

```
static char
         gds_$21 [89] = {
           blr_version4,
           blr_begin,
             /* message number is 0, contains 3 params */
             blr_message, 0, 3,0,
               blr_date, /* param 0: date */
               blr_long, 0, /* param 1: long scale 0 */
               blr_cstring, 6,0,
               /* param 2: null terminated text 6 char */
             blr_receive, 0,          /* receive message 0 */
               /* store a record into ORDER_ITEMS cxt = 0 */
               blr_store,
                 blr_relation, 11,
           'O','R','D','E','R','_','I','T','E','M','S', 0,
                 blr_begin,
                   blr_assignment,
                     blr_parameter, 0, 1,0,
                     blr_field, 0, 12,
            ’O’,’R’,’D’,’E’,’R’,’_’,
            ’N’,’U’,’M’,’B’,’E’,’R’,
                   blr_assignment,
                     /* msg 0, param 2 goes into */
                     blr_parameter, 0, 2,0,
                     blr_field, 0, 11,
            ’I’,’T’,’E’,’M’,’_’,’N’,’U’,’M’,’B’,’E’,’R’,
                   blr_assignment,
                     blr_parameter, 0, 0,0,
                     blr_field, 0, 9,
            ’S’,’H’,’I’,’P’,’_’,’D’,’A’,’T’,’E’,
                   blr_end,
           blr_end,
         blr_eoc
          };
```

See Chapter 5 and 6 for more examples.

SEE ALSO

See the entries in this chapter for:

- *value-expression*
- **blr_send**
- **blr_receive**

See also the entries in Chapter 10 for:

- **gds_$send**
- **gds_$receive**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_modify −modify field value

SYNTAX

blr_modify *old-context new-context blr-statement*

DESCRIPTION

The **blr_modify** statement updates the values of one or more fields in an existing record.

The access method does not update the database if the request unwinds or if some error occurs.

PARAMETERS

old-context Context variable that identifies the record to be modified. An outer *rse* establishes this context.

new-context Context variable that identifies the record stream in which your program provides new field values with **blr_assignment** statements. You can reference *new-context* only in the substatements of the **blr_modify** statement.

Once your program assigns values to *new-context*, the access method copies those values to *old-context* and updates the database. The updates will be undone if the transaction does not complete.

blr-statement Any valid request language statement except:

- **blr_receive**
- **blr_send**
- A statement containing **blr_receive** or **blr_send**

If you want to include more than one *blr-statement*, you must use **blr_begin/blr_end**.

EXAMPLE

The example below demonstrates the request logic associated with a **modify** statement, while Chapter 6 shows a less complex field update based on an program. A modify request for a statement is more complicated than an update for two reasons:

- The **modify** statement allows host language assignments, so it must provide for the chance that the user needs the old value.
- The **on_error** clause allows the user to retry a **modify** statement. Even when there is no **on_error** clause, a **modify** request includes most of the structure to retry a modify operation.

Consider the following simple **modify** statement:

```
for c0 in customers
          {
          credit_rating = get_credit (c0.full_name);
          modify c0 using
            c0.credit_rating = credit_rating;
          end_modify;
          }
        end_for;
```

The request generated for this **modify** statement and the program code that controls the request follow below. An example of the **blr_modify** statement appears at the vertical line at the right margin:

```
static char
          blr_version4,
          blr_begin,
            blr_message, 2, 1,0,
              blr_short, 0,  /* utility message */
            blr_message, 1, 1,0,
              blr_long, 0,  /* new value for credit rating */
            blr_message, 0, 3,0,
              blr_long, 0,  /* credit rating */
              blr_short, 0,  /* EOF indicator */
              blr_cstring, 31,0,  /* name          */
            blr_begin,
              blr_for,
                blr_rse, 1,
                  blr_relation, 9,
           'C','U','S','T','O','M','E','R','S', 0,
                blr_end,
                blr_begin,
                  blr_send, 0,  /* database sends name and */
                  blr_begin,  /* credit rating to program */
                  blr_assignment,
                    blr_field, 0, 13,
            'C','R','E','D','I','T',
            '_','R','A','T','I','N','G',
                    blr_parameter, 0, 0,0,
                  blr_assignment,
                    blr_literal, blr_short, 0, 1,0,
                    blr_parameter, 0, 1,0,
                  blr_assignment,
                    blr_field, 0, 9,
            'F','U','L','L','_','N','A','M','E',
                    blr_parameter, 0, 2,0,
                blr_end,
                /* database enters a select/retrieve loop */
                blr_label, 0,
                  blr_loop,
                    blr_select,
            /* message 2 indicates this record is done */
            /* so exit from loop 0                     */
                      blr_receive, 2,
                        blr_leave, 0,
            /* message 1 is a new value for credit rating */
                      blr_receive, 1,
                        blr_modify, 0, 1,
                          blr_begin,
                            blr_assignment,
                              blr_parameter, 1, 0,0,
                              blr_field, 1, 13,
                 'C','R','E','D','I','T','_',
                 'R','A','T','I','N','G',
                            blr_end,
                  blr_end, /* end of loop */
              blr_end,  /* end of actions for FOR */
            blr_send, 0,    /* send back EOF */
              blr_assignment,
              blr_literal, blr_short, 0, 0,0,
              blr_parameter, 0, 1,0,
            blr_end,         /* end of block of FOR/SEND */
          blr_end,        /* end of entire request */
        blr_eoc
           };
```

The program side code reflects the request structure. The while loop executes until the database sends back the EOS indicator (*0* value for EOS flag). The program first sends *message_1* which directs the database to modify the current record. Next, it sends *message_2*, which instructs the database to advance to the next record.

```
while (1)
          {
          gds_$receive (*gds_$null, gds_$0, 0, (short) 37, gds_$2, 0);
          if (!gds_$2.gds_$4)
            break;
          credit_rating = get_credit (gds_$2.gds_$5);
          gds_$2.gds_$3 = credit_rating;
          gds_$6.gds_$7 = gds_$2.gds_$3;
          gds_$send (*gds_$null, gds_$0, 1, (short) 4, gds_$6, 0);
          gds_$send (*gds_$null, gds_$0, 2, (short) 2, gds_$8, 0);
          }
```

See Chapter 6 for more examples.

SEE ALSO

See the entries in this chapter for:

- **blr_for**
- **blr_assignment**
- **blr_begin**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_rse −create a record stream

SYNTAX

rse ::= **blr_rse** *stream-count stream-clause...*
[*selection-clause...*]
blr_end

DESCRIPTION

The **blr_rse** expression specifies a record stream by naming the input streams and other clauses that determine the content and order of the stream.

The **blr_rse** expression builds a record stream from one or more input streams. The input streams can be simple relations, views, or complex streams created with the union or aggregate operators. All streams listed in a record selection expression (RSE) are joined, and subjected to the screening and ordering described in the selection clause.

PARAMETERS

stream-count Specifies the number of input streams in this rse.

stream-clause Specifies a record source used as input for this record selection expression. The record source can be a simple relation, or it can be an artificial source created by a union or aggregate. Each input stream in an RSE has a context variable associated with it. Actions based on an RSE can reference only contexts of the RSE input streams.

selection-clause Specifies the restriction and ordering characteristics of the RSE. Selection clauses often reference fields from the input streams. Field references must be qualified with the context variable of an input stream.

The following pages describe the syntax and semantics of the **blr_rse** clauses in more detail.

SEE ALSO

See the entries in this chapter for:

- *boolean-expression*
- *value-expression*

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: stream-clause

stream-clause ::= { **blr_relation** *name context-variable* |
blr_rid *id context-variable* |
aggregate-expression |
union-expression }

name ::= count byte, ascii name

context-variable ::= byte

id ::= value of RDB$RELATION_ID from RDB$RELATIONS

Syntax: stream-clause

The *stream-clause* of the record selection expression specifies a record source for input to an RSE. A stream is anything that has a context variable and a set of named fields. Most commonly, a stream consists of a relation or view, and the fields are defined in the database. However, a stream can also be an aggregate or union for which the ‘‘fields’’ are defined in mapping statements.

The forms of the *stream-clause* are:

- **blr_relation** provides the name of the source relation.
- **blr_rid** provides the relation identifier of the source relation as stored in the system relations.
- An *aggregate-expression* or *union-expression* that specifies a stream that is not a relation. The following pages discuss *aggregate-expression* and *union-expression*.

Example: stream-clause

An example of **blr_rse** appears at the vertical line at the right margin:

```
blr_version4,
       blr_begin,
         blr_message, 1, 3,0,/* message 1, three parameters */
           blr_varying, 20,0, /* parameter 0: last_name */
           blr_short, 0, /* parameter 1: missing parameter for p0 */
           blr_short, 0, /* parameter 2: EOS indicator */
         blr_begin, /* Make block of FOR loop and terminating SEND */
           blr_for,  /* begin FOR loop */
             blr_rse, 1, /* rse is composed of one record stream */
               /* relation id for customers, context 0 */
               blr_rid, 12,0, 0,
               blr_boolean, /* restriction expression */
                 blr_missing, /* true for a missing value */
                   blr_fid, 0, 0,0, /* of field id 0 in context 0 */
             blr_end,                    /* end of the rse */
             blr_begin,
               blr_send, 1,
                 blr_begin,
                   /* value of last name or missing flag */
                   blr_assignment,
                     blr_fid, 0, 4,0,
                     blr_parameter2, 1, 0,0, 1,0,
                   blr_assignment, /* EOS is not true */
                     blr_literal, blr_short, 0, 0,0,
                     blr_parameter, 1, 2,0,
                   blr_end, /* end of assignment list */
           blr_end,      /* end of FOR loop action */
           blr_send, 1,  /* send an EOS message back */
             blr_assignment,
               blr_literal, blr_short, 0, 1,0,
               blr_parameter, 1, 2,0,
           blr_end,               /* end the block of FOR and SEND */
         blr_end,
       blr_eoc
```

See Chapter 6 for other examples of record selection.

Syntax: aggregate-expression

blr_aggregate aggregate-stream

aggregate-stream ::= *context-variable*, **blr_rse**, *sub-rse*,
blr_group_by, *group-expression*, **blr_map**, *map-expression*

group-expression ::= *count, field_list*

count ::= number of fields in group-expression
field_list ::= {*field-value-expression*} ...

map-expression ::= *count (map_entry...)*

count ::= long
map-entry ::= { *mapped-id*, *value-expression* |
mapped-id, *agg-value-expression* }

mapped-id ::= long

agg-value-expression ::= { **blr_agg_count** |
agg-operator, value-expression }

agg_operator ::= { **blr_agg_max** |
blr_agg_min |
blr_agg_total |
blr_agg_average }

sub-rse ::= *rse*

Description: aggregate-expression

The *blr_aggregate* clause of **blr_rse** establishes a stream by sorting and grouping records from a sub-rse, and performing statistical operations on each group.

Blr_aggregate creates a record stream that is used in an RSE. Like a relation, an aggregate produces records composed of field values. Unlike a relation, neither the records nor the fields are actual values stored in the database. Instead, the aggregate stream definition includes a formula that establishes a sub-rse as input, groups records in the input, and computes values for each group. Effectively, each group from the sub-rse generates one ‘‘record’’ from the aggregate.

Outside the **blr_aggregate** clause, you can only access fields from the sub-rse through the aggregate context and field identifiers established in the mapping expression.

Parameters: aggregate-expression

context_variable Establishes context for the result of aggregate operations.

sub-rse Specifies a record stream for input to the aggregate operation.

group-expression Specifies a list of fields in the sub-rse used to group input records. Whenever the value of any field changes, a new group is formed. The first field is the primary control break and subsequent fields are i

NAME

blr_select −selective message reception

SYNTAX

blr_select
[*receive-statement...*]
blr_end

DESCRIPTION

The **blr_select** statement blocks execution of the request until it receives an instance of *any* of the message types specified by the nested **blr_receive** statements.

When the access method receives a message, it searches the list of **blr_receive** statements in the **blr_select** statement. If the received message matches one of the list of expected messages, the access method executes the appropriate **blr_receive** statements. Otherwise, it returns a synchronization error.

PARAMETERS

receive-statement A valid **blr_receive** statement. The definition of the message type in a **blr_message** statement must precede any **blr_receive** statements that reference that message.

EXAMPLE

An example of the **blr_select** statement appears at the vertical line at the right margin:

```
static char
         blr_version4,
         blr_begin,
           blr_message, 2, 1,0,
             blr_short, 0,  /* utility message */
           blr_message, 1, 1,0,
             blr_long, 0,  /* new value for credit rating */
           blr_message, 0, 3,0,
             blr_long, 0,  /* credit rating */
             blr_short, 0,  /* EOS indicator */
             blr_cstring, 31,0,  /* name          */
           blr_begin,
             blr_for,
               blr_rse, 1,
                 blr_relation, 9,
          'C','U','S','T','O','M','E','R','S', 0,
               blr_end,
               blr_begin,
                 blr_send, 0,  /* database sends name and */
                 blr_begin,  /* credit rating to program */
                 blr_assignment,
                   blr_field, 0, 13,
           'C','R','E','D','I','T',
           '_','R','A','T','I','N','G',
                   blr_parameter, 0, 0,0,
                 blr_assignment,
                   blr_literal, blr_short, 0, 1,0,
                   blr_parameter, 0, 1,0,
                 blr_assignment,
                   blr_field, 0, 9,
            'F','U','L','L','_','N','A','M','E',
                   blr_parameter, 0, 2,0,
               blr_end,
               /* database enters a select/retrieve loop */
               blr_label, 0,
                 blr_loop,
                   blr_select,
                     /* message 2 indicates this record is done */
                     /* so exit from loop 0                     */
                     blr_receive, 2,
                       blr_leave, 0,
                     /* message 1 is a new value for credit rating */
                     blr_receive, 1,
                       blr_modify, 0, 1,
                         blr_begin,
                           blr_assignment,
                             blr_parameter, 1, 0,0,
                             blr_field, 1, 13,
                'C','R','E','D','I','T','_',
                'R','A','T','I','N','G',
                           blr_end,
                 blr_end,           /* end of loop */
             blr_end,              /* end of actions for FOR */
           blr_send, 0,           /* send back EOS */
             blr_assignment,
             blr_literal, blr_short, 0, 0,0,
             blr_parameter, 0, 1,0,
           blr_end,         /* end of block of FOR/SEND */
         blr_end,        /* end of entire request */
       blr_eoc
          };
```

See Chapter 6 for other examples.

SEE ALSO

See the entry in this chapter for **blr_receive**.

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_send −send message

SYNTAX

blr_send *message-type blr-statement*

DESCRIPTION

The **blr_send** statement evaluates a substatement and sends a formatted message to the calling program.

PARAMETERS

message-type Identifies the message to be sent.

blr-statement Any valid request language statement. For example, the statement may be a **blr_assignment** statement that assigns values to message fields.

EXAMPLE

An example of the **blr_send** statement appears at the vertical line at the right margin:

```
blr_version4,
       blr_begin,
         blr_message, 1, 3,0,
         /* message 1, three parameters */
           blr_varying, 20,0, /* parameter 0: last_name */
           blr_short, 0, /* parameter 1: missing parameter for p0 */
           blr_short, 0, /* parameter 2 is the EOS indicator */
         blr_begin, /* Make block of FOR loop and terminating SEND */
           blr_for,  /* begin FOR loop */
             blr_rse, 1, /* rse is composed of one record stream */
               /* relation id for customers, context 0 */
               blr_rid, 12,0, 0,
               blr_boolean, /* restriction expression */
                 blr_missing,
                 /* true for a missing value */
                   blr_fid, 0, 0,0,
                 /* of field id 0 in context 0 */
             blr_end,  /* end of the rse */
             blr_begin,
               blr_send, 1,
                 blr_begin,
                   /* value of last name or missing flag */
                   blr_assignment,
                     blr_fid, 0, 4,0,
                     blr_parameter2, 1, 0,0, 1,0,
                   blr_assignment, /* EOS is not true */
                     blr_literal, blr_short, 0, 0,0,
                     blr_parameter, 1, 2,0,
                   blr_end, /* end of assignment list */
           blr_end,            /* end of FOR loop action */
           blr_send, 1,              /* send an EOS message back */
             blr_assignment,
               blr_literal, blr_short, 0, 1,0,
               blr_parameter, 1, 2,0,
           blr_end,               /* end the block of FOR and SEND */
         blr_end,
       blr_eoc
```

See Chapter 6 for other examples.

SEE ALSO

See the entries in this chapter for:

- **blr_message**
- **blr_receive**

See also the entries in Chapter 10 for:

- **gds_$send**
- **gds_$receive**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

blr_store −store record

SYNTAX

blr_store *relation-name context-variable blr-statement*

DESCRIPTION

The **blr_store** statement stores a new record in a relation. This statement differs from **blr_store2** in that **blr_store2** permits a second request language statement in the context of the store operation.

Use **blr_assignment** to supply field values. The access method sets a field to missing if you do not supply a value for that field.

The access method does not update the database if the request unwinds or if some error occurs.

PARAMETERS

relation-name Identifies the relation into which the record will be stored.

context-name Identifies the record stream that includes the relation into which the record will be stored.

blr-statement Any valid request language statement except:

- **blr_receive**
- **blr_send**
- A statement containing **blr_receive** or **blr_send**

If you want to include more than one *blr-statement*, you must use a **blr_begin** statement.

EXAMPLE

An example of the **blr_store** statement appears at the vertical line at the right margin:

```
static char
         gds_$21 [89] = {
           blr_version4,
           blr_begin,
             blr_message, 0, 3,0,
             /* message number is 0, contains 3 params */
               blr_date, /* param 0: date */
               blr_long, 0, /* param 1: long scale 0 */
               blr_cstring, 6,0,
               /* param 2: null-terminated text 6 char */
             blr_receive, 0, /* receive message 0 */
               /* store a record into ORDER_ITEMS cxt = 0 */
               blr_store,
                 blr_relation, 11,
           'O','R','D','E','R','_','I','T','E','M','S', 0,
                 blr_begin,
                  blr_assignment,
                    blr_parameter, 0, 1,0,
                    blr_field, 0, 12,
           'O','R','D','E','R','_',
           'N','U','M','B','E','R',
                  blr_assignment,
                    /* msg 0, param 2 goes into */
                    blr_parameter, 0, 2,0,
                    blr_field, 0, 11,
           'I','T','E','M','_','N','U','M','B','E','R',
                  blr_assignment,
                    blr_parameter, 0, 0,0,
                    blr_field, 0, 9,
           'S','H','I','P','_','D','A','T','E',
                  blr_end,
          blr_end,
        blr_eoc
         };
```

SEE ALSO

See the entries in this chapter for:

- **blr_begin**
- **blr_assignment**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

NAME

value-expression −calculating value

SYNTAX

```
value-expression ::= { arithmetic-expression |
dbkey-expression |
field-expression |
first-from-expression |
literal-expression |
parameter-expression |
statistical-expression |
via-expression }
```

DESCRIPTION

A *value-expression* represents a symbol or string of symbols from which calculates a value. When the access method encounters a value expression in a statement, it computes the value associated with that expression and uses the value when it executes the statement.

A synopsis, usage description, and example of each of these clauses follows.

SEE ALSO

See the entries in this chapter for *boolean-expression* and **blr_rse**.

Syntax: arithmetic-expression

```
arithmetic-expression ::= { arithmetic-operator value-1 value-2 |
blr_negate value-expression }

arithmetic-operator ::= { blr_add |
blr_subtract  |
blr_multiply  |
blr_divide  |
blr_concatenate  }
```

Description: arithmetic-expression

An *arithmetic-expression* specifies an arithmetic relationship between two value expressions. The access method evaluates the expression and returns a value. If either value expression resolves to a missing value, the result of the arithmetic expression is also missing.

It is recommended that you send to the access method all arithmetic operations that involve database fields. This approach has the following advantages:

- It lets the access method perform an operation for which it has been optimized.
- It reduces the communications traffic between a calling program and the access method. Because the access method performs the calculations, it only has to pass back the result of the arithmetic operation. Otherwise, it would have to pass both value expressions to the calling program.
- It reduces the amount of code in the calling program.
- It lessens the dependence of the calling program on the current state of the database.

PARAMETERS

value-1
value-2 Value expressions in the relationship expressed by *arithmetic-operator*. Operations are performed left to right. For example, if you divide two value expressions, *value-1* is the dividend and *value-2* the divisor. The access method returns the quotient.

arithmetic-operator Specifies the desired arithmetic operation. The following table lists the arithmetic operators and what they do.

Table: Arithmetic Operators

Operator	Meaning
blr_add	Adds *value-1* to *value-2*
blr_subtract	Subtracts *value-2* from *value-1*
blr_multiply	Multiplies *value-1* by *value-2*
blr_divide	Divides *value-1* by *value-2*
blr_concatenate	Concatenates value-1 to value-2, resulting in an alphanumeric value

Example: arithmetic-expression

An example of the **blr_add** operator appears at the vertical line at the right margin. This request read records in the IDS relation, adding 1 to the ORDER_NUMBER field in each record. This assignment has an arithmetic operation as the *from* value and a database field as the *to* value.

```
.
        static char
          gds_$55 [56] = {
          blr_version4,
            blr_begin,
              blr_for,
              /* for loop, 1 input stream, no restrictions */
                blr_rse, 1,
                  blr_relation, 3, 'I','D','S', 0,
                blr_end,
              blr_modify, 0, 1,
              /* modify creates context for output */
                blr_begin,
                  blr_assignment,
                    blr_add,
                      blr_field, 0, 12,
                      /* field from input context */
                      'O','R','D','E','R','_','N','U','M','B','E','R',
                      blr_literal, blr_short, 0, 1,0,
                      blr_field, 1, 12,
                      /* field in output context */
                      'O','R','D','E','R','_','N','U','M','B','E','R',
                  blr_end,
            blr_end,
          blr_eoc
        };  /* end of blr string for request gds_$55 */
```

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: dbkey-expression

blr_dbkey *context-variable*

Description: dbkey-expression

The *dbkey-expression* uses an internal pointer to retrieve a record from the database.

You can use the **blr_dbkey** value expression to reference a record in:

- A **blr_fetch** statement
- A record selection expression where **blr_dbkey** is an argument in a **blr_eql** expression

Database keys are 8 bytes for records in primitive relations.

A dbkey, by default, is valid only during the transaction in which it was obtained. You can extend the scope of the dbkey to include the entire session during which a database is attached. You can specify this option in the database parameter block (represented as *parameter_buffer*) in a call to **gds_$attach_database** or **gds_$create_database**.

PARAMETERS

context-variable Identifies the record stream from which the access method obtains the dbkey for the current record. The notion of ‘‘current record’’ refers only to the record currently being processed in a stream.

Example: dbkey-expression

The following request includes a simple for loop that returns the dbkey for every record in CUSTOMERS:

```
blr_version4,
       blr_begin,
         blr_message, 0, 2,0, /* message with db_key and EOS flag */
           blr_text, 8,0,
           blr_short, 0,
         blr_begin,
           blr_for,
             blr_rse, 1,
               blr_relation, 9,
          'C','U','S','T','O','M','E','R','S', 0,
             blr_end,
             blr_send, 0,
               blr_begin,  /* for each record send back the dbkey */
                 blr_assignment,
                   blr_dbkey, 0,
                   blr_parameter, 0, 0,0,
                 blr_assignment,
                   blr_literal, blr_short, 0, 1,0,
                   /* and not EOS */
                   blr_parameter, 0, 1,0,
               blr_end,
             blr_send, 0,  /* at end send back EOS */
               blr_assignment,
                 blr_literal, blr_short, 0, 0,0,
                 blr_parameter, 0, 1,0,
               blr_end,
             blr_end,
       blr_eoc
```

SEE ALSO

See the entries in this chapter for:

- **blr_fetch**
- **blr_rse**

See also the entries in Chapter 10 for:

- **gds_$attach_database**
- **gds_$create_database**

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: field-expression

field-expression ::= { **blr_field** *context-variable count field-name* |
blr_fid *context-variable field-id* }

Description: field-expression

The *field-expression* provides the value of a field in a record:

- **blr_field** is the more general form of field reference that references a field by its name.
- **blr_fid** is a restricted form of field reference intended for use by calling programs that generate dynamic queries. It references the RDB$FIELD_ID field of the RDB$RELATION_FIELDS system relation. The value of RDB$FIELD_ID may change when you backup and restore the database with **gbak**. It is not recommended that you hard-code the field identifier if you want your program to work between backups of the database.

PARAMETERS

context-variable Identifies the record stream from which the access method retrieves the field from the current record. The notion of ‘‘current record’’ refers only to the record currently being processed in a stream.

count A one-byte counter containing the field name’s length in characters.

field-name The field name of the field to be retrieved from the current record.

field-id The field identifier of the field to be retrieved from the current record. This identifier is the value of the RDB$FIELD_ID field in the RDB$RELATION_FIELDS system relation.

Example: field-expression

Examples of the **blr_field** value expression appear at the vertical line at the right margin:

```
static char
         gds_$21 [89] = {
           blr_version4,
           blr_begin,
             blr_message, 0, 3,0,
             /* message number is 0, contains 3 params */
               blr_date, /* param 0: date */
               blr_long, 0, /* param 1: long scale 0 */
               blr_cstring, 6,0,
               /* param 2: null-terminated text 6 char */
             blr_receive, 0, /* receive message 0 */
               /* store a record into ORDER_ITEMS cxt = 0 */
               blr_store,
                 blr_relation, 11,
           'O','R','D','E','R','_','I','T','E','M','S', 0,
                 blr_begin,
                   blr_assignment,
                     blr_parameter, 0, 1,0,
                     blr_field, 0, 12,
            'O','R','D','E','R','_',
            'N','U','M','B','E','R',
                   /* msg 0, param 2 goes into */
                   blr_assignment,
                     blr_parameter, 0, 2,0,
                     blr_field, 0, 11,
            'I','T','E','M','_','N','U','M','B','E','R',
                   blr_assignment,
                     blr_parameter, 0, 0,0,
                     blr_field, 0, 9,
            'S','H','I','P','_','D','A','T','E',
                   blr_end,
           blr_end,
         blr_eoc
          };
```

See Chapter 6 for other examples.

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: first-from-expression

blr_from *rse value-expression*

Description: first-from-expression

The *first-from-expression* returns the value of a value expression from the first record in a record stream.

The access method:

- Finds the first record in the record stream that results from the evaluation of the record selection expression. If the stream is empty, the access method returns an error.
- Then evaluates the value expression for the first record in the record stream, returning the result as the value of the **blr_from** expression. The value expression may resolve to a field name. In this case, the access method returns the value of that field.

If the value expression in the **blr_from** expression does not match any fields in the first record of the record stream, the access method returns an error. If you want to provide a second value expression for the access method to return if there is no match, you should use the **blr_via** expression that is described below.

PARAMETERS

rse A valid record selection expression.

value-expression A valid value expression. This value is typically a field reference.

Example: first-from-expression

The following request returns the value of credit rating from the first customer record it encounters:

```
.
        blr_version4,
        blr_begin,
          blr_message, 1, 1,0, /* pass back the credit rating */
            blr_long, 0,
          blr_send, 1,  /* major action is the send */
            blr_assignment,
              blr_from,  /* from value expression */
                blr_rse, 1,
                  blr_rid, 12,0, 0,  /* relation id for customers */
                blr_end,
              blr_fid, 0, 0,0, /* field 0 is credit rating */
              blr_parameter, 1, 0,0,
          blr_end,
        blr_eoc
```

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: literal-expression

blr_literal *field-description literal-string*

Description: literal-expression

The *literal-expression* is a string constant.

Use the **blr_literal** value expression for static field values in a message and the **blr_parameter** for dynamic values.

PARAMETERS

field-description Describes the physical characteristics of the field.

literal-string A string of characters. The value of a numeric literal must be supplied in generic form. For binary values, this means reversing the byte order. For floating point, this involves so much computation that floating point values are typically passed as text (for example, ’-’, ’1’, ’.’, ’9’, ’7’).

Example: literal-expression

Examples of the **blr_literal** expression appear at the vertical line at the right margin:

```
blr_version4,
       blr_begin,
         blr_message, 1, 3,0,
         /* message number is 1, three parameters */
           blr_varying, 20,0, /* parameter 0: last_name */
           blr_short, 0, /* parameter 1: "missing parameter" for p0 */
           blr_short, 0,  /* parameter 2: EOS indicator */
         blr_begin,  /* Make block of FOR loop and terminating SEND */
           blr_for,    /* begin FOR loop */
             blr_rse, 1, /* rse is composed of one record stream */
               /* relation id for customers, context 0 */
               blr_rid, 12,0, 0,
               blr_boolean, /* restriction expression */
                 blr_missing, /* true for a missing value */
                   blr_fid, 0, 0,0,
                 /* of field id 0 in context 0 */
             blr_end,  /* end of the rse */
             blr_begin,
               blr_send, 1,
                 blr_begin,
                   blr_assignment,
                   /* value of last name or missing flag */
                     blr_fid, 0, 4,0,
                     blr_parameter2, 1, 0,0, 1,0,
                   blr_assignment, /* EOS is not true */
                     blr_literal, blr_short, 0, 0,0,
                     blr_parameter, 1, 2,0,
                   blr_end,  /* end of assignment list */
```

```
            blr_end,     /* end of FOR loop action */
            blr_send, 1, /* send an EOS message back */
              blr_assignment,
                blr_literal, blr_short, 0, 1,0,
                blr_parameter, 1, 2,0,
            blr_end, /* end the block of FOR and SEND */
          blr_end,
        blr_eoc
```

DIAGNOSTICS

See Chapter 8 for a discussion of errors and error handling.

Syntax: via-expression

blr_via *rse value-1 value-2*

Description: via-expression

The *via-expression* returns the value of a value expression from the first record in a record stream. Failing that, it returns the value of a second value expression.

The access method:

- Finds the first record in the record stream that results from the evaluation of the record selection expression. If the stream is empty, the access method returns an error. For example, the record stream may consist of all records with a field value within a specified range.
- Then evaluates *value-1* for the first record in the record stream, returning the result as the value of the **blr_via** expression. The value expression may resolve to a field name. In this case, the access method returns the value of that field. If *value-1* does not match any fields, the access method evaluates *value-2*. If it matches, the access method returns the value of *value-2* for the first record in the stream.

If the record stream is empty, the access method returns the value of *value-2*, which in this case is missing.

PARAMETERS

rse A valid record selection expression.

value-1 Typically specifies a field.
value-2 Typically a literal of the same type as the expected field.

Example: via-expression

The following example returns the value of credit rating from the first customer record it encounters:

.

```
DATABASE    DB = FILENAME "test_0.gdb";
static int
  *req_0 = 0;

struct  {
  int  credit;
    } msg_0;

static char blr_string[] = {
  blr_version4,
  blr_begin,
    /* pass back the credit rating */
    blr_message, 1, 1,0,
      blr_long, 0,
    /* major action is the send */
    blr_send, 1,
      blr_assignment,
        blr_via,
          blr_rse, 1,
            /* relation id for customers */
            blr_rid, 12,0, 0,
            blr_boolean,
              blr_gtr,
                blr_fid, 0, 0,0,
                blr_literal, blr_short, 0, 10,0,
          blr_end,
        blr_fid, 0, 0,0, /* field 0 is credit rating */
          blr_literal, blr_long, 0, -1,-1,-1,-1,
          blr_parameter, 1, 0,0,
    blr_end,
blr_eoc };

main ()
{
ready;
start_transaction;

if (!req_0)
  gds_$compile_request (*gds_$null, DB, req_0,
  (short) sizeof(blr_string), blr_string);
gds_$start_request (*gds_$null, req_0, gds_$trans, 0);

gds_$receive (*gds_$null, req_0, 1, sizeof(msg_0), msg_0 , 0);

printf ("credit rating is %d0, msg_0.credit);
}
```