
New and changed features for Unify DataServer Release 9.1

This document lists and explains the features that have been added and behavior changes that have been introduced for Release 9.1 of Unify DataServer. Some of these changes or enhancements may have been available in earlier releases or patches.

Contents

SHMOFFSET AND SHMMARGIN CAN NOW BE 64 BIT VALUES ON 64 BIT PORTS	3
NEW DATETIME DATA TYPE	4
DDL operations with DATETIME columns	4
CREATE TABLE	4
ALTER TABLE	4
Creating Indexes with DATETIME columns	5
DML operations with DATETIME columns	5
Specifying DATETIME constants	5
Selecting DATETIME values.....	5
ORDER BY	5
GROUP BY	6
Coercion.....	6
Comparisons of DATETIME values	6
Arithmetic operations involving DATETIME columns	6
Use of DATETIME columns in Embedded SQL/A	7
Using a string buffer.....	7
Using a UDTTM variable	7
When to use a string buffer or UDTTM	8
Using DATETIME columns in stored procedures, functions and triggers	8
Stored procedure/function parameters and return values.....	8
Database operations with DATETIME variables and columns in stored procedures, functions and triggers	8
4GL system functions supporting DATETIME types	8
Arithmetic operations involving DATETIME values.....	8
DATETIME values in triggers.....	9
NEW REPTALTALGF CONFIGURATION VARIABLE.....	10
NEW SHMSIGINTVL CONFIGURATION VARIABLE	11
NEW UNULLDATEFMT CONFIGURATION VARIABLE	12
NEW ALTER VOLUME STATEMENT	13
ENHANCED LOGGING FROM UPERF	14
CONFIGURABLE MAXIMUM NUMBER OF CONNECTION SLOTS FOR THE ODBCDMN	15
LOG FILES FROM ODBCDMN NO LONGER CREATED BY DEFAULT	16
NEW OHOST= COMMAND LINE OPTION FOR DBDMN.EXE ON WINDOWS.....	17
RPTDIVIDEBYZERO CONFIGURATION VARIABLE	18

SHMOFFSET and SHMMARGIN can now be 64 bit values on 64 bit ports

Systems with 64 bit processors offer greatly increased address space in comparison to the address space offered by 32 bit systems. While 32 bit processors are limited to a maximum of 4 gigabytes (32 bits) of address space of which only 2-3 gigabytes are available to user processes, current generations of processors based on the AMD64 instruction set provide over 130 terabytes (47 bits) of address space to user processes. Other 64 bit processors provide similar increases in the available address space.

In release 9.1 of Unify DataServer, the **SHMOFFSET** and **SHMMARGIN** configuration variables have been enhanced so that database administrators can take advantage of this increased address space. To simplify the specification of 64 bit SHMOFFSET and SHMMARGIN settings, a new suffix “**G**” has been implemented for these values, which multiplies the supplied value by 1,073,741,824. For example, the following settings:

```
SHMOFFSET=20480G
SHMMARGIN=10G
```

have the same effect as setting

```
SHMOFFSET=20971520M
SHMMARGIN=10240M
```

The new functionality, including the “**G**” suffix, is available only on 64 bit ports of Unify DataServer.

For more information about these variables, see the *Unify DataServer: Configuration Variable and Utility Reference* pages 105 (**SHMOFFSET**) and 100 (**SHMMARGIN**).

New DATETIME data type

Unify DataServer 9.1 introduces a new data type – **DATETIME**. The **DATETIME** data type can be used to store a moment in time with millisecond precision. **DATETIME** columns are stored in a time zone agnostic format and are converted to and from the current session’s time zone when displayed or input. This allows users in different time zones to be able to see and compare **DATETIME** values entered from other time zones in their own time zone.

DATETIME columns support dates after 1752 and through the year 9999. Attempts to use or create **DATETIME** values outside these limits will generate an overflow error.

DDL operations with DATETIME columns

CREATE TABLE

The **CREATE TABLE** statement (see *Unify DataServer: SQL/A Reference* page 77) has been enhanced to support creation of columns of type **DATETIME**. The syntax for the *data_type* clause for **DATETIME** columns is the following:

DATETIME [(3)]

The number between parenthesis is optional and specifies the precision of the **DATETIME**. The only value currently allowed is 3 (millisecond precision.) For example:

```
CREATE TABLE orders (  
    customer_id    NUMERIC(9) NOT NULL,  
    order_id      NUMERIC(9) NOT NULL,  
    order_time    DATETIME NOT NULL,  
    ...  
    fulfillment_time  DATETIME(3),  
    PRIMARY KEY(customer_id, order_id)  
);
```

As with **DATE** and **HUGE DATE** columns, **DATETIME** columns may allow NULLs or be defined as **NOT NULL**, may be part of the primary key, or may be included in other unique constraints for the table.

ALTER TABLE

The **ALTER TABLE** statement (see *Unify DataServer: SQL/A Reference* page 39) has been enhanced to support the creation of columns of type **DATETIME**. The same syntax is used as in the case of **CREATE TABLE** (see above.) For example:

```
ALTER TABLE events (  
    ADD COLUMN event_time DATETIME,  
    MODIFY COLUMN schedule_date DATETIME  
);
```

In the case of **ALTER TABLE...MODIFY COLUMN** the only allowed conversion to **DATETIME** is from either a **DATE** or a **HUGE DATE** column. When a **DATE** or **HUGE DATE** column is converted to **DATETIME**, the resulting **DATETIME** value will represent midnight of the specified date in the time zone of the database session executing the **ALTER TABLE**.

Creating Indexes with **DATETIME** columns

DATETIME columns may be included as part of B-tree, hash and link indexes. There is no special syntax required. Ordering in B-tree indexes will be chronological.

DML operations with **DATETIME** columns

Specifying **DATETIME** constants

DATETIME constants are specified as strings formatted to match the ISO 8601 standard without support for the optional timezone specification, for example **'2009-01-28 09:56:05.123'**. The format can be described as **'YYYY-MM-DD HH:II:SS.SSS'** where **MM** represents months and **II** represents minutes. The time portion is optional – if the time is not specified it will be interpreted as **'00:00:00.000'**. When specifying the time portion, the seconds and millisecond portions are also optional and will be interpreted as 0 if not specified. For example:

```
INSERT INTO events(event_time) VALUES ('2009-01-28 09:56:05.123');

SELECT * FROM orders WHERE order_time < '2011-08-26');
```

DATETIME constants will be interpreted as being in the time zone of the current database session and will be converted to the time zone agnostic internal format before being stored or compared.

The special value **''** (empty string) is interpreted as the current date and time. The precision of this value is dependent on the precision of the operating system's **gettimeofday()** system call. The following example will set the **fulfillment_time** column in the table **orders** to the current date and time:

```
UPDATE orders
SET fulfillment_time = ''
WHERE customer_id = 1203
AND order_id = 5;
```

Selecting **DATETIME** values

When selecting **DATETIME** values they will be displayed in the same ISO 8601 standard format that is used for specifying **DATETIME** constants. The **DATETIME** values will be converted from the time zone agnostic internal format into the time zone of the database session before display.

```
SQL> SELECT fulfillment_time FROM orders;
recognized query!

      fulfillment_time
-----
2009-01-28 09:56:05.123
2011-08-26 00:00:00.000
...
```

ORDER BY

Columns of type **DATETIME** can be specified in the **ORDER BY** clause of a **SELECT** statement. The ordering will be in chronological order, either ascending or descending.

GROUP BY

Columns of type **DATETIME** may be used in the **GROUP BY** clause of a **SELECT** statement. Comparisons between **DATETIME** columns will be done to the value up to the exact millisecond.

Coercion

Values of type **DATE** or **HUGE DATE** may be coerced to **DATETIME** values. This means it is possible to store date values directly into the database using the existing date formatting variables, as in the following statement (assuming **c1** is of type **DATETIME**):

```
INSERT INTO t1 ( c1 ) values ( 01/01/2008 );
```

In this case, the time portion of the field will be set to **00:00:00.000**.

Strings may also be coerced to **DATETIME** values, provided they are properly formatted (see *Specifying DATETIME constants* above).

Comparisons of DATETIME values

The following comparison operators are supported for **DATETIME** values:

=, !=, <>	Equality and inequality
<, >, <=, =>	Relational operators
BETWEEN	Range operator
IS [NOT] NULL	Null comparison

In all cases, comparisons are exact to the full precision (milliseconds) of the column. In the case of the relational and range operators comparisons are chronological. Due to the coercion described above, comparisons against **DATE** or **HUGE DATE** types will interpret the **DATE** or **HUGE DATE** value as having the time portion set to **00:00:00.000**.

Arithmetic operations involving DATETIME columns

Addition of a **DATETIME** and a floating point number or integer is supported. The number will be interpreted as a number of seconds and milliseconds. The following code adds a day to a **DATETIME** type:

```
UPDATE t1 SET c1 = c1 + 86400 WHERE...
```

To add a millisecond to every **DATETIME** value:

```
UPDATE t1 SET c1 = c1 + .001;
```

Subtracting a floating point number or integer from a **DATETIME** value is supported. The number will be interpreted as a number of seconds. To subtract a day from a **DATETIME** column:

```
UPDATE t1 SET c1 = c1 - 86400 WHERE...
```

Subtracting two **DATETIME** values will result in a floating point number representing the number of seconds between two given dates, expressed with 3 digits of precision beyond the decimal point. This result may be negative. For example, to display the difference in seconds between a **DATETIME** column and the current time:

```
SELECT '' - c1 FROM t1;  
recognized query!
```

```
          '-c1
-----
          0.104
1314393570.397
115220170.397
-34566666.256
115216509.297
...
```

Use of **DATETIME** columns in Embedded SQL/A

When using **DATETIME** columns in Embedded SQL/A, you may specify the host variable as a string buffer or as a variable of the new type **UDTTM** to select or provide a **DATETIME** value.

Using a string buffer

If selecting from a **DATETIME** column into a host variable that is declared as a string buffer, the buffer will contain a string representation of the value in the same ISO 8601 compliant format as specified for interactive operations. The **DATETIME** value will be converted to the time zone of the current database session before being formatted. The buffer should be declared with a size of at least 24 bytes (23 bytes for the value and one for the null terminator byte):

```
EXEC SQL BEGIN DECLARE SECTION ;
      char   strbuf[23 + 1];
EXEC SQL END DECLARE SECTION ;

EXEC SQL
      SELECT c1 INTO :strbuf
      FROM dtmhost
      WHERE n1 = 1000 ;
```

If using a host variable declared as a string buffer to provide a **DATETIME** value to an Embedded SQL/A statement, the buffer must contain a correctly formatted (ISO 8601 compliant) string representation of the desired **DATETIME** value. This value will be interpreted using the time zone of the current database session:

```
strcpy(strbuf, "2009-02-16 15:00:00.000");

EXEC SQL
      INSERT INTO dtmhost (c1) VALUES (:strbuf) ;
```

Using a **UDTTM** variable

The new **UDTTM** Embedded SQL/A data type is a 64 bit signed integer type. When used as a host variable, it provides access to the “raw” value of the **DATETIME** column. When a **UDTTM** host variable is used to fetch or provide a **DATETIME** value, the **UDTTM** variable will store the **DATETIME** value as the number of microseconds since **January 1st, 1970 00:00:00.000000 GMT**. **UDTTM** host variables are *not* converted to or from the time zone of the current database session.

```
EXEC SQL BEGIN DECLARE SECTION ;
      UDTTM   dtmvar1;
      UDTTM   dtmvar2;
EXEC SQL END DECLARE SECTION ;
```

```
dtm1 = 1234825200000000LL;

EXEC SQL
  SELECT dtm1 FROM t1
  INTO :dtmvar1
  WHERE dtm2 = :dtmvar2;

printf("UDTTM value selected = %lld\n", dtmvar1);
```

Although DataServer **DATETIME** columns currently have a precision of milliseconds, **UDTTM** variables are stored as a number of microseconds. For this reason, **UDTTM** values selected from the database will always be a multiple of 1000, and **UDTTM** values sent to the database will be truncated to the next lowest multiple of 1000.

When to use a string buffer or **UDTTM**

The decision of whether to use a string buffer or a **UDTTM** variable depends on the use for the selected or provided variable. In general, if the value is to be displayed to or provided by an end user, it is best to use the string buffer representation. In the case of doing interval operations on the **DATETIME** variable, it is easier to use the **UDTTM** representation.

Using **DATETIME** columns in stored procedures, functions and triggers

Stored procedure/function parameters and return values

DATETIME variables may be declared in stored procedure or function parameter lists or as function return values using the keyword **DATETIME**:

```
CREATE DATETIME FUNCTION dtmtodttm( RESULT DATETIME dtm ) AS...
```

The variables thus declared can be used within the stored procedure/function code.

Database operations with **DATETIME** variables and columns in stored procedures, functions and triggers

DATETIME variables and columns may be used in database operations in stored procedures, functions and triggers. When selecting a **DATETIME** column into an un-typed general variable, the variable will be set to the **DATETIME** data type. If selecting a **DATETIME** column into a variable that already has a **CHAR** data type, the string representation of the **DATETIME** will be selected. Either a **DATETIME** or **CHAR** variable can be used when providing data to a database operation. The string value stored in the **CHAR** variable will be interpreted as described in *Specifying DATETIME constants* on page 5.

4GL system functions supporting **DATETIME** types

The only 4GL functions currently supporting the **DATETIME** type are **to_string\$()** and **val_to_str\$()**. Both will return a string representation of the **DATETIME** value.

Arithmetic operations involving **DATETIME** values

Arithmetic operations within the context of a database operation are supported as described in the *DML operations with DATETIME values* section of this document. Arithmetic operations in the 4GL are not currently supported.

DATETIME values in triggers

The value of **DATETIME** columns of inserted, deleted or modified records are available as variables in triggers through the use of the **new:** and **old:** constructs in the same manner as for other data types.

New REPTALTALGF configuration variable

The alternate algorithm which is attempted by **CREATE LINK INDEX** when there is insufficient memory to use the regular algorithm has been found to be faster for links where the child table is very large. On one system where the child table contained over three million rows, the regular algorithm took over seventeen hours to complete, where using the alternate algorithm created the link in about one hour. The difference is most dramatic on systems where scanning backward through a large file is far slower than scanning forward.

A new configuration variable, **REPTALTALGF** (REPOinT ALTerNate ALGorithmn Factor), has been created to cause **CREATE LINK INDEX** to use the alternate algorithm without first attempting the regular one. The **repoint** program was enhanced. (SQL executes **repoint** to perform the actual build when creating a link index, and **repoint** can be run from the shell to rebuild existing links if repairs are needed.)

Setting **REPTALTALGF** to 100 (or greater) will cause **repoint** to use the alternate algorithm. If the value is less than 100, **repoint** will operate as it did before this enhancement was introduced--it will use the alternate algorithm only if the regular algorithm is not able to allocate the required memory. The default value is 0. (1 - 99 may become meaningful in a future release.)

REPTALTALGF is used when **repoint** is run from the shell to rebuild a link index, when the entire index will be rebuilt. Example (see the usage output for more information):

```
$ repoint -L158
```

An entry is written to the errlog (\$DBPATH/*.err) to record when the alternate algorithm has been selected due to this feature. The following example entry indicates **repoint** was executed by a **CREATE LINK INDEX** command (indicated by upper case REPOINT in the first line), and that **REPTALTALGF** was set to 100 (see the Errno line):

```
Program: REPOINT (15876)
Calling function: bldlnk - build link index
Offending function: fstlnk
Status: 0
Errno: 100
Notes: The (repoint) alternate algorithm has been selected.
```

When **repoint** is run from the shell to rebuild a link, and the output line "Rebuilding the entire link index" is displayed (as before), the line "The alternate algorithm will be used" has been added to indicate this feature has caused the alternate algorithm to be selected. For example:

```
$ repoint -L158
Repoint link index "steves_link" (lid=158).
Rebuilding the entire link index.
Start: Mon May 26 16:28:09 2008
The alternate algorithm will be used.
No progress messages will be displayed as it runs.
Finish: Mon May 26 17:21:03 2008
repoint completed successfully.
$
```

New **SHMSIGINTVL** configuration variable

When a shared memory latch is held, other database processes that wish to acquire the latch check to ensure that the process holding has not died without releasing it.

A new configuration variable **SHMSIGINTVL** has been added to the product. This new variable causes processes that wish to acquire a latch to only check the state of the process holding the latch every **SHMSIGINTVL/100** seconds. The default value, 500, makes the above check occur only once per 5 seconds, and should be suitable for most systems. Setting this variable to a larger value may cause the system not to detect a crashed state quickly. Setting the variable to a very low value may adversely affect performance.

New **UNULLDATEFMT** configuration variable

In Unify's Accell/IDS product, NULL **DATE**s were displayed as "****/**/****" (including the separators from **DATEFMT**). In Accell/SQL, the format for NULL **DATE**s was changed to "*********" (without the separators).

A configuration variable **UNULLDATEFMT** has been added to return to the Accell/IDS functionality. If **UNULLDATEFMT** is set to FALSE (the default) NULL **DATE**s will be displayed without separators (Accell/SQL behavior). If set to TRUE, NULL **DATE**s will be displayed with separators from **DATEFMT** (Accell/IDS behavior).

New **ALTER VOLUME** statement

An ALTER VOLUME statement has been added to Dataserver SQL. The syntax of **ALTER VOLUME** is as follows:

```
ALTER VOLUME volume_name (  
    [ TYPE [IS] ( FILE [, FORCE ] | DEVICE ) , ]  
    [ LENGTH [IS] integer , ]  
    [ PATH [IS] string ]  
);
```

As shown by the syntax description, any or all of the **TYPE**, **LENGTH** and **PATH** clauses can be used in a single operation to change multiple attributes of a volume.

The following restrictions are in place:

- You cannot change a volume's length to a value smaller than the already allocated space on the volume.
- You cannot change the length of a preallocated volume. However, you can remove the preallocated status from a volume and change its length in a single operation:

```
ALTER VOLUME myallocvol (  
    TYPE IS (FILE) ,  
    LENGTH IS 2147479552  
);
```

- Changing the type of a volume from DEVICE to FILE or FILE to DEVICE will only succeed if the physical file is of the correct type, i.e. a regular filesystem file or a device file.
- Changing the type of a volume from DEVICE to FILE will only succeed if the device volume was created with an offset of 0.

Enhanced logging from **uperf**

The uperf utility will now log most of the items displayed on the **uperf** screen. The format for the log file as specified with -L is now:

```
MM/DD/YYYY HH:MM:SS current time
HH:MM:SS last sync
N number of file manager users
N operations since last sync
N number of updaters
NORMAL database state(s) [FAILURE,BACKUP,SYNC-RUNNING,UPDATERS-SUSPENDED]
N transaction log free space
N number of active transactions
N logical reads this snapshot
N logical writes
N physical reads
N physical writes
N page hits
N page misses
N reclaims
N user page replacements
N.NNeN.NNeN.NNe transactions per sec: 1min 5min 15min averages (e=estimated)
N.NNeN.NNeN.NNe tps rate per second per user: 1/5/15 min avgs
Repeat for each shared memory segment:
  N % free
  N % used
```

Each column is separated by a pipe symbol. For example:

```
08/30/2011 15:41:03|15:32:42|3|0|0|NORMAL|990|4|6180|14|89|8|6102|93|6|0| 0.00e 0.00e
0.00e| 0.00e 0.00e 0.00e|362688|161416
08/30/2011 15:41:05|15:32:42|3|0|0|NORMAL|990|4|6180|14|89|8|6102|93|6|0| 0.00e 0.00e
0.00e| 0.00e 0.00e 0.00e|362688|161416
```

Usage example:

```
uperf -L uperf.log
```

Configurable maximum number of connection slots for the **odbcdmn**

A new property, **MaxConnectionSlots**, has been created for the **[DBIntegrator]** section of the *\$UNIFY/odbcdmn.ini* file. For example:

```
[DBIntegrator]
InstallDirectory=/u/ds91
MgrPort=1583
MaxConnectionSlots=2000
...
```

MaxConnectionSlots determines the number of connection slots available. If it is unset, or is set to an invalid value (less than or equal to 0) it will default to 1000, the internal value used prior to this enhancement.

If the root **odbcdmn** stops accepting new connections until others are released (also indicated by the message "Unable to reserve a connection slot in pLamReserveConnection"), increase the value for **MaxConnectionSlots** and restart the root **odbcdmn**. If **MaxConnectionSlots** was not previously set, use a value higher than the default shown above.

MaxConnectionSlots should be set to at least the number of simultaneous connections (across all data sources) required.

Each connection slot uses approximately 500 bytes of shared memory. A shared memory segment sized to store all of them is created by the root **odbcdmn** when it starts up. If the necessary amount of shared memory cannot be allocated by the OS, an error message will be generated showing the segment size, and the **odbcdmn** will not start.

Log files from **odbcadm** no longer created by default

The **odbcadm** no longer creates the event.log and connect.log files by default. Two new optional properties now exist for the **[DBIntegrator]** section of the `$UNIFY/odbcadm.ini` file which allow you to specify the locations of those files, and to change their names. If the new properties are not set, the files will not be created.

- **EventLogFile** - Specifies the location and filename of the odbcadm event log file previously named event.log.
- **ConnectionLogFile** - Specifies the location and filename of the connection log file previously named connect.log (if present).

Their values can be either absolute pathnames (starting with a '/'), or relative pathnames such as `./event.log` or `event.log`. When a relative pathname is used, the file will be created relative to the directory where the odbcadm is started.

You do not have to restart odbcadm processes when changing these properties in the odbcadm.ini file, and if you move or remove a log file while it is being used, a new file will be created to replace it (if permissions allow).

To get the same behavior as before this enhancement was added, set the properties as shown in the following example (replacing both instances of `/u/ds91` with the actual value of `InstallDirectory` in your odbcadm.ini file):

```
[DBIntegrator]
InstallDirectory=/u/ds91
ConnectionLogFile=/u/ds91/connect.log
EventLogFile=event.log
MgrPort=1583
...
```

The new functionality is illustrated by the following example:

```
[DBIntegrator]
InstallDirectory=/u/ds91
ConnectionLogFile=/logs/DBI_Connections.log
; Uncomment this line and the odbcadm will
; begin logging events again:
;EventLogFile=/logs/DBI_Events.log
MgrPort=1583
...
```

New **Ohost=** command line option for **dbdmn.exe** on Windows

This enhancement introduces the **Ohost=** option for the **dbdmn.exe** in the Windows release of DataServer. The **Ohost=** option can be set in one of two places:

- By changing the following Windows Registry value:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DataServer dbdmn\ImagePath
```

Here you can specify the parameter in the form **/Ohost=hostname** or **-Ohost=hostname**. For example:

```
C:\Unify\DataServer\bin\dbdmn.exe -Ohost=127.0.0.1
```

This will be the default value used for the interface to bind to, unless overridden by the following method.

- In the "Start Parameters" field of the Service Properties dialog for the "Datasever dbdmn" service, or on the command line of the "net start" command used to start the service. For example:

```
net start "Datasever dbdmn" /Ohost=127.0.0.1
```

In this case, you must use the **/Ohost=hostname** syntax. The **-Ohost=hostname** syntax will *not* work.

If you use either of these options, the default value set in the registry will be overridden only for this specific execution of the dbdmn. The only way to make a permanent change to the option is to edit the registry value shown above.

RPTDIVIDEBYZERO configuration variable

The RPT utility in ACCELL/SQL and DataServer releases prior to 8.1 did not complain when dividing by zero, but rather produced a zero result. This behavior changed in 8.1 as a side effect of new development. Division by zero began to result in a string of question marks (?????); an undefined value. Since reports written before 8.1 may fail due to that change, the previous behavior was restored, and configuration variable was created for reports which require the new behavior.

The new configuration variable also supports a compromise behavior. Use it for reports which depend on division by zero resulting in zero when the numerator is zero, but which must fail for division by zero calculations where the numerator is non-zero.

New configuration variable:

RPTDIVIDEBYZERO

Use this variable to specify division by zero behavior for RPT.

Valid values:

- **INDETERMINATE** - Division by zero is undefined, regardless of the value of the numerator.
- **ZIFZNUMERATOR** - Zero will be the result if the numerator is zero, otherwise the result is undefined.
- **ALWAYSZERO** - The result of division by zero is always zero.

Default value: **ALWAYSZERO**

Example: RPTDIVIDEBYZERO="ZIFZNUMERATOR"

If RPTDIVIDEBYZERO is set to an invalid string, the default will be assumed.