

RETURN TO MAIN MENU

Recital Universal ODBC Driver

Recital Universal ODBC Driver Documentation

Recital Corporation,
100 Cummings Center, Suite 318J
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.
COPYRIGHT ©1988-2006 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc.. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last Updated March, 2006

Index

Installation and Data Source Configuration	
Windows	2
Linux/UNIX	4
Features and Limitations	7
Supported SQL Grammar	9
ODBC Command Summary	11
Additional Functions	
ABS()	13
ASCII()	13
SIGN()	14
SUBSTR()	14
UPPER()	14
Bridging to Other File Formats	
Connecting to C-ISAM Files	15
Connecting to RMS Files	18
Connecting to Recital Database Gateways	22
Accessing a Resultset from a Stored Procedure	23

Installation and Data Source Configuration: Windows

Installation

The Recital Universal ODBC Driver for Windows is available as part of the Recital Database Server and Recital Client Drivers Windows Installer distributions. Run the Windows Installer distribution and follow the on screen instructions.

Data Source Configuration

Data Sources must be configured on the Client before data from the Server can be accessed. Data Sources are set up in the *ODBC Data Source Administrator* from the *Administrative Tools* section in the *Control Panel*.

To add a Data Source:

1. Select the required DSN type and click on *Add*
2. Select **Recital Universal ODBC Driver** and click *Finish*
3. Enter in the details as required:

Connection Information:

Field	Description
DSN	A unique name for the data source (minimum required configuration).
Description	A brief description for the data source. Clicking on the '...' button pops up an editor window.
Server	The server node hostname or IP address. A Recital Database Server must be installed and running on the target server in order to connect.
User	The login name for the remote server.
Password	The password for the specified login
Database Directory	The database name or directory where the tables reside

Note: Databases in Recital are implemented as directories containing files that correspond to the tables in the database. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory. The database catalog can contain information about a table's associated index files (single .ndx files and tagged .dbx files). It also contains path information, which allows tables in other directories to be accessed.

Advanced Configuration

For additional information on Gateway configuration, please also see the Connecting to Recital Database Gateways section of this guide.

Gateway:

Field	Description
Driver	The gateway driver type
Database	The gateway database name
User	The gateway login name
Password	The password for the specified gateway login
Options	Any additional options

Settings:

Field	Description
Database: Row ID	Check to enable the Row ID setting
Exclusive	Check to open the database exclusive
No Update	Check to open the database read-only
Login: Encryption	Check to enable DES3 Encryption

Logging:

Field	Description
Client: Logging	Check to enable client logging
Log To Window	Check to log to a Window
Log To File	Check to log to a File and enter the target log file name
Server: Logging	Check to enable server logging
Name	Name of server log file

Installation and Data Source Configuration: Linux/UNIX

Driver Manager

An ODBC Driver Manager must be installed to provide the essential bridge between and ODBC enabled application and the Recital Universal ODBC Driver. The unixODBC Driver Manager is distributed under GPL or LGPL and available for download from www.unixODBC.org. To install the GUI components of the unixODBC package, Qt/X11 GUI components is also required (Qt/X11 Free Edition for GUI components is available from <http://www.trolltech.com>). Please note that adding the Recital Universal ODBC Driver and configuring its data sources via the unixODBC GUI ODBCConfig tool is not currently supported.

Driver Manager Installation

The unixODBC package is available as a gzip tar file.

1. Download the unixODBC*.tar.gz file into a suitable directory
2. Gunzip the file:
gunzip unixODBC*.tar.gz
3. Extract the files from the tar archive:
tar xvf unixODBC*.tar
This will create a unixODBC sub-directory with all the source files
4. From the unixODBC sub-directory, build the libraries and programs:
./configure
make
make install

The unixODBC directory includes the files README and INSTALL with additional information on the installation and configuration of the unixODBC package and a doc directory with further documentation.

NOTE: To use the unixODBC Driver Manager with PHP, the `--with-unixODBC[=DIR]` option must be included when PHP is installed. For full information on this, and other PHP installation and configuration options, please go to <http://www.php.net>.

ODBC Driver Installation

The Recital Universal ODBC Driver for Linux/UNIX is included in the Recital Database Server and the Recital Database Server Client Drivers for Linux and UNIX distributions. Run the rpm (Linux) or extract the files from the tar archive and run the setup script (Linux/UNIX).

To make the Recital Universal ODBC Driver known to the unixODBC Driver Manager, the driver details must be set up in the *odbcinst.ini* file. The *odbcinst.ini* file is usually in */etc*, although this may be a symbolic link to */usr/local/etc/odbcinst.ini*.

To add the Driver, use the unixODBC odbcinst utility and the supplied Driver template file, issuing the following command at the OS prompt in the recital/odbc directory:

```
odbcinst -i -d -f odbcinst.ini
```

Or edit the odbcinst.ini file manually and enter the details as follows:

```
# Recital Universal ODBC Driver
[Recital]
Description = Recital Universal ODBC Driver
Driver      = /usr/recital/shared/libodbc_recital.so
Setup       = /usr/recital/shared/libodbc_recital.so
FileUsage   = 1
```

Data Source Configuration

Data Sources must be configured before data from the Server can be accessed. Data Sources are set up in the *odbc.ini* files. The system *odbc.ini* file is usually in */etc*, although this may be a symbolic link to */usr/local/etc/odbc.ini*. User *odbc.ini* files have a *'.'* prefix (*.odbc.ini*) and reside in the user's home directory.

To add a Data Source, use the unixODBC *odbcinst* utility and a DSN template file, issuing the following command at the OS prompt:

```
odbcinst -i -s -f <template file>
```

e.g., from the *recital/odbc* directory:

```
odbcinst -i -s -f odbc_test.ini
```

Note: This will modify the user's *.odbc.ini*. To add a system DSN, include the *'-l'* parameter:

```
odbcinst -i -s -l -f odbc_test.ini
```

Or edit the *odbc.ini* file manually and enter the details as follows:

```
[<unique DSN>]
Description  = <informative text>
Driver       = Recital
Database     = <URL>
```

Where the *<URL>* is in the following format:

```
ODBC:RECITAL:SERVERNAME=<node>;USERNAME=<user>;PASSWORD=<password>;DATABASE=<database>;
```

or

```
ODBC:RECITAL:SERVERNAME=<node>;USERNAME=<user>;PASSWORD=<password>;DIRECTORY=<directory>;
```

The following may also be included:

```
Trace       = <Yes | No>
TraceFile   = <filename>
```

e.g.

```
[Recital ODBC test]
Driver=Recital
DATABASE=ODBC:RECITAL:SERVERNAME=?;USERNAME=?;PASSWORD=?;DATABASE=odbc_test;
```

or

```
[Recital ODBC Free Form Tables]
Driver=Recital
DATABASE=ODBC:RECITAL:SERVERNAME=server1;USERNAME=user1;PASSWORD=pass1;DIRECTORY=/u/app/data/;
```

NOTES

- The <node> can be the hostname or IP address. A '?' can be used to signify the local host.
- Storing the user and password details in the DSN is optional and careful consideration should be given to security implications. A '?' can be used in place of the user and password for local host connections where the environment variable DB_LOCAL_LOGIN is set to true in the UAS/profile.uas file.
- The DB_TMPDIR environment variable in the profile.uas file in the UAS directory must be set to a valid writeable directory on the server.
- To test that the ODBC Driver is successfully installed and configured, use the odbctest program in recital/odbc:

`./odbc_test.exe`

Features and Limitations

- Supported ODBC SQL data types:

fSqlType	Recital/SQL Data Type
SQL_CHAR	CHAR(n)
SQL_VARCHAR	VARCHAR(n)
SQL_LONGVARCHAR	LONG VARCHAR
SQL_NUMERIC	DECIMAL(precision, scale)
SQL_NUMERIC	NUMERIC(precision, scale)
SQL_NUMERIC	SMALLINT
SQL_INTEGER	INTEGER
SQL_REAL	REAL
SQL_FLOAT	FLOAT
SQL_NUMERIC	DOUBLE PRECISION
SQL_BIT	BIT
SQL_TINYINT	TINYINT
SQL_NUMERIC	BIGINT
SQL_SHORT	SHORT(precision, scale)
SQL_DATE	DATE
SQL_BIT	LOGICAL
SQL_LONGVARCHAR	MEMO
SQL_NUMERIC	BYTE
SQL_LONGVARBINARY	OBJECT
SQL_DATETIME	DATETIME
SQL_CURRENCY	CURRENCY
SQL_ZONED	ZONED

- Full query optimization is supported on the server using "index restriction" query optimization. Make sure you have the required indexes set up.
- You can execute any Recital non user-interface command from the ODBC driver by prefixing the command in `SQLExecute()` or `SQLExecDirect()` with the keyword 'Recital'. For example:
`SQLExecDirect(hstmt, "recital do myproc", cbSqlStr);`
 - Any output can then be accessed using `SQLFetch()` calls. You can control shared access with this command. For example:
`SQLExecDirect(hstmt, "recital set exclusive off", cbSqlStr);`
- All Recital data types are supported (including memos). These can be specified in the `CREATE TABLE` command.
- Character and binary values supplied for parameterized queries are limited to 255 bytes.
- Qualifiers or owners are not allowed on databases or tables.
- Column and table names are not case sensitive. String data comparisons are case sensitive.
- By default all tables are opened shareable on the server. If you do not want this then:
 - Call `SQLExecute(hstmt, "recital set exclusive on")` before opening a table.
 - Set the 'useoptions' string in the `sysodbc.ini` file on the server to 'exclusive' for each table that you require non-shareable access to. See the UAS Installation and Configuration Guide for more information about `sysodbc.ini`.
 - Add the command 'set exclusive on' to the `config.db` file. This will cause all tables to be opened exclusively. See the UAS Installation and Configuration Guide for more information about `config.db`.
- Memo fields are seen as SQL 'LONG VARCHAR' columns, and default to a maximum of 64k characters. This can be altered by setting the 'maxvarcharsize' entry in `recital.ini` on the client to the

required value. Be careful when making this too large as performance will be unduly affected. The *recital.ini* file should reside in the <WINDOWS> directory.

- ODBC driver and Recital Universal Data and Application Server interaction can be debugged on the client by selecting the 'Trace RSI calls' checkbox in the SQLDriverConnect() dialog.
- ODBC server activity can be debugged by setting the 'logging' string in 'recital.ini' or see the UAS Installation and Configuration Guide for information about setting up logging on the server. The resultant file on the server can be viewed using a standard text editor.

Supported SQL Grammar

The grammar declarations are described with these notations:

Notation	Description
<blank>	No qualifier is required
<command> <command>	Use one of the specified commands
[qualifier]	Optional qualifier, may be omitted
(paramval)	Required parameter value
expression	Numeric or string calculation

- statement ::= ALTER [alter](#) | CREATE [create](#) | DELETE [delete](#) | DROP [drop](#) | INSERT [insert](#) | GRANT [grant](#) | RECALL [recall](#) | REVOKE [revoke](#) | SELECT [select](#) | UPDATE [update](#)

aggterm	COUNT (*) AVG (expression) MAX (expression) MIN (expression) MIN (expression) SUM (expression)
alias	AS aliasname
aliasname	identifier
alter	TABLE tablename ADD (createcols)
and	not not AND and
asc	<blank> ASC DESC
boolean	<blank> and and OR boolean
coldesc	<blank> DESCRIPTION string
colref	aliasname.columnname columnname
column	columnname
columnlist	identifier , identifier identifier
columnname	identifier
comparison	(boolean) colref IS NULL colref IS NOT NULL expression LIKE pattern expression NOT LIKE pattern expression IN { valuelist } expression NOT IN { valuelist } expression op expression
create	TABLE tablename (createcols) 0 [UNIQUE] INDEX indexname ON tablename (indexcols)
createcol	columnname datatype coldesc columnname datatype (integer) coldesc columnname datatype (integer, integer) coldesc
createcols	createcol , createcols createcol
cursorname	identifier
datatype	CHAR VARCHAR LONG VARCHAR DECIMAL NUMERIC SMALLINT INTEGER REAL FLOAT DOUBLE PRECISION BIT TINYINT SHORT DATE LOGICAL MEMO BYTE CURRENCY DATETIME LONG VARBINARY ZONED
date	a date in ODBC escape clause format (for example, {d'1996-04-04'} or --(*vendor(Microsoft), product(ODBC)d'1996-04-04'*)--
delete	FROM tablename where
drop	TABLE tablename INDEX indexname
expression	expression + times expression - times times
forupdate	<blank> FOR UPDATE FOR UPDATE OF columnlist
grant	privilege ON tablename TO usergroups
groupby	GROUP BY groupbyterms
groupbyterms	colref colref , groupbyterms
groups	integer , groups integer integer - integer , groups integer - integer *
having	<blank> HAVING boolean

identifier	an identifier (identifiers containing spaces must be enclosed in double quotes)
indexcols	columnname [ASC DESC] indexcols , indexcols
indexname	identifier
insert	INTO tablename insertvals
insertvals	(columnlist) VALUES (valuelist) VALUES (valuelist)
integer	a non-negative integer
join	INNER JOIN OUTER JOIN LEFT [OUTER] JOIN RIGHT [OUTER] JOIN CROSS JOIN tableref ON table.column = table.column
neg	term + term - term
not	comparison NOT comparison
op	> >= < <= = <>
orderby	<blank> ORDER BY orderbyterms
orderbyterm	colref asc integer asc
orderbyterms	orderbyterm orderbyterm , orderbyterms
pattern	string ? USER
privilege	ALL ALTER DELETE INSERT READ ONLY columnlist SELECT columnlist UPDATE columnlist
realnumber	a non-negative real number
recital	any recital command excluding user interface commands
revoke	privilege ON tablename FROM usergroups
select	selectcols FROM tablelist where groupby having orderby forupdate
selectallcols	<blank> ALL DISTINCT
selectcols	selectallcols * selectallcols selectlist
selectlist	expression , selectlist expression
set	column = NULL column = simpleterm
setlist	set setlist , set
simpleterm	string realnumber ? USER date time timestamp
string	a string (enclosed in single quotes)
table	tablename aliasname
tablelist	tableref , tablelist tableref tableref join viewname
tablename	identifier
tableref	tablename tablename alias
term	(expression) colref simpleterm aggterm
time	a time in ODBC escape clause format (for example, {t'10:19:48'} or --(*vendor(Microsoft), product(ODBC)t'10:19:48'*)--
times	times * neg times / neg neg
timestamp	a timestamp in ODBC escape clause format (for example, {ts'1996-04-04 10:19:48.529'} or --(*vendor(Microsoft), product(ODBC)ts'1996-04-04 10:19:54.529'*)
update	tablename SET setlist where
usergroup	user and group access control string in the format '[users,groups]'
usergroups	usergroup , usergroups PUBLIC
users	integer , users integer integer – integer , users integer – integer *
valuelist	NULL, valuelist expression , valuelist expression NULL
viewname	identifier
where	<blank> WHERE boolean WHERE CURRENT OF cursorname

ODBC Command Summary

The following functions are supported up to ODBC 3.5 conformance:

SQLAllocConnect()
SQLAllocEnv()
SQLAllocHandle()
SQLAllocStmt()
SQLBindCol()
SQLBindParam()
SQLBindParameter()
SQLCancel()
SQLCloseCursor()
SQLColAttribute()
SQLColAttributes()
SQLColumnPrivileges()
SQLColumns()
SQLConnect()
SQLCopyDesc()
SQLDataSources()
SQLDescribeCol()
SQLDescribeParam()
SQLDisconnect()
SQLDriverConnect()
SQLDrivers()
SQLEndTran()
SQLError()
SQLExecDirect()
SQLExecute()
SQLExtendedFetch()
SQLFetch()
SQLFetchScroll()
SQLForeignKeys()
SQLFreeConnect()
SQLFreeEnv()
SQLFreeHandle()
SQLFreeStmt()
SQLGetConnectAttr()
SQLGetConnectOption()
SQLGetCursorName()
SQLGetData()
SQLGetDescField()
SQLGetDiagField()
SQLGetDiagRec()
SQLGetEnvAttr()
SQLGetFunctions()
SQLGetInfo()
SQLGetStmtAttr()
SQLGetStmtOption()
SQLGetTypeInfo()
SQLMoreResults()

SQLNativeSql()
SQLNumParams()
SQLNumResultCols()
SQLParamData()
SQLParamOptions()
SQLPrepare()
SQLPrimaryKeys()
SQLProcedureColumns()
SQLProcedures()
SQLPutData()
SQLRowCount()
SQLSetConnectAttr()
SQLSetConnectOption()
SQLSetCursorName()
SQLSetEnvAttr()
SQLSetParam()
SQLSetPos()
SQLSetScrollOptions()
SQLSetStmtAttr()
SQLSetStmtOption()
SQLSpecialColumns()
SQLStatistics()
SQLTablePrivileges()
SQLTables()
SQLTransact()

Additional Functions

ABS()

Purpose

Returns the absolute value from numeric columns

Syntax

ABS(<n>)

Description

The ABS() function can be used in select statements for returning the absolute value of a numeric column. The <n> is the name of a numeric column from which to return the absolute value.

Example

```
SELECT ABS(company) FROM customer
```

ASCII()

Purpose

Returns the ASCII value of the first character from character columns

Syntax

ASCII(<char>)

Description

The ASCII() function can be used in select statements to return the ASCII value from character columns. The <char> is the name of a character column from which to return the ASCII value of the first character.

Example

```
SELECT ASCII(company) FROM customer
```

SIGN()

Purpose

Returns the sign type from numeric columns

Syntax

SIGN(<n>)

Description

The SIGN() function can be used in SELECT statements for returning the sign type from numeric columns. The <n> is the name of a numeric column from which the sign will be returned. If <n> < 0, the function returns -1, if <n> = 0 the function returns 0, if <n> > 0, the function returns 1.

Example

```
SELECT SIGN(balance) FROM customer
```

SUBSTR()

Purpose

Extracts portions of text from character columns

Syntax

SUBSTR(<char>, m [,n])

Description

The SUBSTR() function can be used in SELECT statements for extracting portions of text from character columns. The <char> is the name of a character column to extract, beginning at character m, n characters long (if n is omitted, to the end of char). The first position is of char is 1.

Example

```
SELECT SUBSTR(company,1,10) FROM customer
```

UPPER()

Purpose

Converts character columns into upper case

Syntax

UPPER(<char>)

Description

The UPPER() function can be used in SELECT statements for converting character columns into upper case. The <char> is the name of a character column from the selected table.

Example

```
SELECT UPPER(company) FROM customer
```

Connecting to C-ISAM Files

The Recital Universal ODBC Driver can access Informix compliant C-ISAM files.

Data access is achieved through a C-ISAM Bridge. This requires the creation of a Bridge file and an empty Recital table that has the same structure as the C-ISAM file.

NOTE: Users of Recital Terminal Developer on UNIX or Linux can create the Recital table and the Bridge file through the CREATE and CREATE BRIDGE work surfaces. Please see the Recital 4GL Commands Reference for instructions on using these work surfaces.

Creating the Recital Table

Create a Recital table with the same structure as the C-ISAM file. The fields/columns in the structure file must exactly match the data type and length of those in the C-ISAM file. The Recital table will have one byte more in total record length due to the Recital record deletion marker. To create the table, use the SQL CREATE TABLE command. The table should be given a '.str' file extension (rather than the default '.dbf') to signify that this is a structure file only.

Please see the end of this section for information on matching Informix and Recital data types. Sample code for table and Bridge creation is given at the end of the Connecting to RMS files section.

Creating the Bridge File

The Bridge File can be created in two ways: by using an 'ini' file, or by the SQL CREATE BRIDGE command.

'ini' file

Firstly, an 'ini' file should be created on the server in the data directory where the C-ISAM file is held. The ini file has the following structure:

```
[bridge]
bridgetype=<bridgetype>
externalname=<name of the C-ISAM file>
databasename=<name of the Recital structure table>
alias=<the name to use to access your file>
```

e.g. cisamdemo.ini

```
[bridge]
bridgetype=CISAM
externalname=cisam.dat
databasename=cisamdemo.str
alias=cisamdemo
```

There should be no white space either side of the '=' signs.

The Bridge file can now be created from the ini file. This can be given a '.dbf' file extension (rather than the default '.brg') so that it can be accessed like a normal Recital table. The following command is used to run the Recital/4GL CREATE BRIDGE <.brg filename> | (<expC>) FROM command:

e.g.

Recital create bridge cisamdemo.dbf from cisamdemo

CREATE BRIDGE (SQL)

The CREATE BRIDGE SQL command defines and creates the bridge in one step:

e.g.

```
exec sql
CREATE BRIDGE cisamdemo.dbf
TYPE "CISAM"
EXTERNAL "cisamdemo.dat"
METADATA "cisamdemo.str"
ALIAS "cisamdemo";
```

Using the Bridge

The Bridge can now be used. To access the C-ISAM file, use the 'alias' specified in the Bridge definition.

e.g.

```
Select * from cisamdemo
```

Data Types

Informix	Recital
Byte	Numeric
Char	Character
Character	Character
Date	Date
Datetime	Character
Decimal	Numeric
Double Precision	Float
Float	Real
16 Bit Integer	Short
Integer	Numeric
Interval	Character
32 Bit Long	Integer
Money	Numeric
Numeric	Numeric
Real	Numeric
Smallfloat	Numeric
Smallint	Numeric
Text	Unsupported
Varchar	Character

C-ISAM Error Messages

The following errors relate to the use of the Recital CISAM Bridge and can be received as an 'errno <expN>' on Recital error messages:

ERRNO()	Error Description
100	Duplicate record
101	File not open
102	Invalid argument
103	Invalid key description
104	Out of file descriptors
105	Invalid ISAM file format
106	Exclusive lock required
107	Record claimed by another user
108	Key already exists
109	Primary key may not be used
110	Beginning or end of file reached
111	No match was found
112	There is no "current" established
113	Entire file locked by another user
114	File name too long
115	Cannot create lock file
116	Memory allocation request failed
117	Bad custom collating
118	Duplicate primary key allowed
119	Invalid transaction identifier
120	Exclusively locked in a transaction
121	Internal error in journaling
122	Object not locked

Connecting to RMS Files

The Recital Universal ODBC Driver can access the following fixed length RMS File types:

- RMS Sequential
- RMS Indexed Sequential
- RMS Relative

Data access is achieved through an RMS Bridge. This requires the creation of a Bridge file and an empty Recital table that has the same structure as the RMS file.

NOTE: Users of Recital OpenVMS Developer can create the Recital table and the Bridge file through the CREATE and CREATE BRIDGE work surfaces. Please see the Recital 4GL Commands Reference for instructions on using these work surfaces.

Creating the Recital Table

Create a Recital table with the same structure as the RMS file. The fields/columns in the structure file must exactly match the data type and length of those in the RMS file. The Recital table will have one byte more in total record length due to the Recital record deletion marker.

To create the table, use the SQL CREATE TABLE command. The table should be given a '.str' file extension (rather than the default '.dbf') to signify that this is a structure file only.

Please see the end of this section for information on accessing VAX COBOL data types.

Creating the Bridge File

The Bridge File can be created in two ways: by using an 'ini' file, or by the SQL CREATE BRIDGE command.

'ini' file

Firstly, an 'ini' file should be created on the server in the data directory where the RMS file is held. The ini file has the following structure:

```
[bridge]
bridgetype=<bridgetype>
externalname=<name of the RMS file>
databasename=<name of the Recital structure table>
alias=<the name to use to access your file>
```

e.g. rmsseqdemo.ini

```
[bridge]
bridgetype=RMSSEQ
externalname=rmsseq.dat
databasename=rmsseqdemo.str
alias=rmsseqdemo
```

There should be no white space either side of the '=' signs.

The Bridge file can now be created from the ini file. This can be given a '.dbf' file extension (rather than the default '.brg') so that it can be accessed like a normal Recital table. The following command is used to run the Recital/4GL CREATE BRIDGE <.brg filename> | (<expC>) FROM command:

e.g.

Recital create bridge rmsseqdemo.dbf from rmsseqdemo

CREATE BRIDGE (SQL)

The CREATE BRIDGE SQL command defines and creates the bridge in one step:

e.g.

exec sql

CREATE BRIDGE rmsseqdemo.dbf

TYPE "RMSSEQ"

EXTERNAL "rmsseq.dat"

METADATA "rmsseqdemo.str"

ALIAS "rmsseqdemo";

Using the Bridge

The Bridge can now be used. To access the RMS file, use the 'alias' specified in the Bridge definition.

e.g.

Select * from rmsseqdemo

Accessing VAX COBOL Data Types

The following table provides details of the COBOL data types that can be directly accessed by RECITAL using the RECITAL RMS Bridge.

COBOL Picture Clause	COBOL Usage Clause	RECITAL Data type	Storage in bytes
PIC 9(n) [n <=18]	USAGE IS DISPLAY	(N)umeric	n
PIC 9(n) [n <=4]	USAGE IS COMP	(S)hort	2
PIC 9(n) [5 <=n <=9]	USAGE IS COMP	(I)nteger	4
PIC 9(n) [10 <=n <=18]	USAGE IS COMP	(Q)uad	8
PIC S9(n) [n <=4]	USAGE IS COMP	(S)hort	2
PIC S9(m) [5 <=n <=9]	USAGE IS COMP	(I)nteger	4
PIC S9(n) [10 <=n <=18]	USAGE IS COMP	(Q)uad	8
	USAGE IS COMP	(I)nteger	4
	USAGE IS POINTER	(I)nteger	4
	USAGE IS COMP-1	(R)eal	4
	USAGE IS COMP-2	(F)loat	8
PIC S9(n) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC 9(n) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC X(n) [n <=254]	USAGE IS DISPLAY	(C)haracter	N
PIC A(n) [n <=254]	USAGE IS DISPLAY	(C)haracter	n
PIC 9(n)V9(s)	USAGE IS DISPLAY	(S)hort	2
PIC S9(n)V9(s) [(n+s) <=4]	USAGE IS COMP	(S)hort	2
PIC S9(n)V9(s) [5<=(n+s)<=9]	USAGE IS COMP	(I)nteger	4
PIC S9(n)V9(s) [10<=(n+s)<=18]	USAGE IS COMP	(Q)uad	8
PIC 9(n)V9(s) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC S9(n)V9(s) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC S9(n) [n <=18]	USAGE IS DISPLAY	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS LEADING	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	

	SEPARATE		
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING SEPARATE	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	--not currently supported--	

NOTES:

The storage occupied by packed decimal data types is calculated as follows:

if (n+s) is odd then storage = ((n+s)+1)/2
else storage = ((n+s)+2)/2

When defining the “width” for binary data types, this value denotes the output display width. The storage occupied by the data type is as specified above.

When defining the number of decimal places for binary data types, this value represents the “scale” of the value. When the field is referenced, RECITAL scales it down by successive divisions of 10, as specified by “scale”, and evaluates all arithmetic in double precision floating point. When fields of this type are updated, then the result to be stored in the field is again re-scaled.

Connecting to Recital Database Gateways

The Recital Universal ODBC Driver allows gateway connectivity to the following data sources. Please note this may require the purchase of additional license options. Details of gateway availability for each platform are available from the Recital web site <http://www.recital.com/products.htm>.

ODBC data sources

Oracle
Informix
Ingres
Remote Recital
MySQL
PostgreSQL

The Database Gateway connection is specified in the 'Database' entry of the datasource configuration using the following syntax:

```
Database: odbc:Recital:SERVERNAME= servername;DIRECTORY=directory;  
USERNAME=username;PASSWORD=password;LOGGING=logging;LOGFILE=logfile;  
GATEWAY=type@node:dbms_username/dbms_password-database.protocol
```

or

```
Database: odbc:Recital:SERVERNAME= servername;DIRECTORY=directory;  
USERNAME= username;PASSWORD=password;LOGGING=logging;LOGFILE=logfile;  
GATEWAY=odbc:datasource
```

Parameter	Description
<i>servername</i>	The IP address of the Database Server. “?” connects to the local machine.
<i>directory</i>	The startup directory on the server.
<i>username</i>	Database Server Login username.
<i>password</i>	Database Server Login password.
<i>logging</i>	Set to ‘true’ or ‘false’ to turn logging on or off.
<i>logfile</i>	The name of the file to log to.
<i>gateway</i>	Gateway definition.
<i>datasource</i>	ODBC datasource (DSN).
<i>type</i>	Gateway type: e.g. ‘ora’; ‘inf’; ‘ing’; ‘db2’; ‘rec’.
<i>node</i>	The IP address (or hostname) of the data server.
<i>dbms_username</i>	The username for the data source, e.g. if connecting to Oracle, this must be the name of a valid Oracle user.
<i>dbms_password</i>	The password for the <i>dbms_username</i> above.
<i>database</i>	The database to connect to.
<i>protocol</i>	The network protocol, decnet (DECNET) or tcpip (TCP/IP). If the protocol is not specified, TCP/IP is assumed on Unix/Linux and DECNET on OpenVMS.

The LOGGING and LOGFILE settings are optional.

The USERNAME and PASSWORD settings are optional to the Database entry, and can be specified during connection.

Accessing a Resultset from a Stored Procedure

Stored procedures and user-defined functions are collections of SQL statements and optional control-of-flow statements written in the Recital 4GL (compatible with VFP) stored under a name and saved in a **Database**. Both stored procedures and user-defined functions are just-in-time compiled by the Recital database engine.

Stored Procedures can return a Resultset using the 4GL SETRESULTSET() function. The 'recital' command is used to execute the 4GL Stored Procedure and return the Resultset to the client application for processing. Resultset that are returned from Stored Procedures are read-only.

In this example, the getexamplecursor.prg stored procedure is in the 'southwind' database on the server. It accepts a parameter and runs a query, saving the results into a cursor. This cursor is then returned as a Resultset using the SETRESULTSET() 4GL function.

```
// Stored Procedure getexamplecursor.prg
lparameters lcAccountNo
exec sql
select account_no from example
where account_no = lcAccountNo
into cursor curExample;
return setresultset("curExample")
// end of getexamplecursor.prg
```

A DSN has been set up to access the southwind database as follows:

```
[southwind]
Driver = Recital
DATABASE = ODBC:RECITAL:SERVERNAME=?;USERNAME=?;
PASSWORD=?;DATABASE=southwind;
```

The C program establishes the ODBC connection, calls the Stored Procedure and displays the data.

```
/* Example C program to call stored procedure and display data from returned resultset */
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <malloc.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

static int get_vardata(
    SQLHSTMT hstmt);
static void show_error(
    int line,
    SQLSMALLINT HandleType,
    SQLHSTMT hstmt);

#define TRUE 1
#define NAME_LEN 6
```



```

int main(int argc, char* argv[])
{
    SQLHENV      henv;
    SQLHDBC      hdbc;
    SQLHSTMT     hstmt;
    SQLRETURN    retcode;

    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
        retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

            if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
                retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (void*)5, 0);

                retcode = SQLConnect(hdbc, (SQLCHAR*) "southwind", SQL_NTS,
                                    (SQLCHAR*) "?", SQL_NTS,
                                    (SQLCHAR*) "?", SQL_NTS);

                if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
                    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

                    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
                        retcode = get_vardata(hstmt);

                        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
                    }
                    SQLDisconnect(hdbc);
                } else {
                    show_error(__LINE__, SQL_HANDLE_DBC, hdbc);
                }
                SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
            }
        }
    }
    else {
        show_error(__LINE__, SQL_HANDLE_ENV, henv);
        SQLFreeHandle(SQL_HANDLE_ENV, henv);
    }

    return 0;
}

```

```

static int get_vardata(SQLHSTMT hstmt)
{
    SQLCHAR      szName[NAME_LEN];
    SQLINTEGER    cbName;
    SQLRETURN     retcode;

    retcode = SQLExecDirect(hstmt, "recital getexamplecursor('0001')", SQL_NTS);
    if (retcode == SQL_SUCCESS) {
        while (TRUE) {
            retcode = SQLFetch(hstmt);
            if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
                show_error(__LINE__, SQL_HANDLE_STMT, hstmt);
            }
            if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
                SQLGetData(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
                fprintf(stdout, "%s\n", szName);
            } else {
                break;
            }
        }
    }
    return(retcode);
}

static void show_error(int line, SQLSMALLINT HandleType, SQLHSTMT hstmt)
{
    SQLINTEGER NativeErrorPtr;
    unsigned char Sqlstate[6];
    unsigned char MessageText[512];
    SQLSMALLINT TextLengthPtr;

    SQLGetDiagRec(HandleType,
        hstmt,
        1,
        Sqlstate,
        &NativeErrorPtr,
        MessageText,
        512,
        &TextLengthPtr);
    fprintf(stdout, "%d:%s\n", line, MessageText);
}

```