SURVOX:

Interfacing with ODBC

COMPUTERS FOR MARKETING CORPORATION
547 Howard Street • San Francisco, CA • 94105 U.S.A.

TABLE OF CONTENTS

SURVOX's ODBC-Compliant Software

Introduction

The sources of data in a questionnaire have been the phone record, a pre-loaded data record, or data from questions the respondent answers "on-the-fly." Consider, for example, a client who maintains a database of every customer that has purchased a product. This is a common practice so that clients can maintain warranty records, support contracts, upgrade status, billing and marketing efforts. Survent is an ideal tool to help maintain the information in the database, and it can be used to conduct periodic surveys to assess the customer satisfaction levels previously recorded.

With the ability to access a database record on-the-fly while the interviewer still has the respondent on the phone, Survent can make it easier to verify and update the information, and collect any new information the client deems important.

With this in mind, SURVOX has added the ability to perform calls to online relational databases on a prototype version of Survent (7.7). The feature makes use of ODBC (Open Database Connectivity) routines that provide a general-purpose access method for most relational database programs, including MySQL, MS-Access, MS SQL Server and others.

SURVOX's ODBC-compliant software has been developed so that Survent and Mentor applications can interface with ODBC-compliant databases. A special ODBC compile of the software (Survent, Mentor and the other the binaries) is created and special commands within Survent and Mentor are used for this purpose. SURVOX will support this with the release of its 7.7 software.

Here are some examples of how ODBC databases can be extremely useful in conjunction with SURVOX software:

- Market researchers can build and maintain panels.
- It can be used for updating sample files. The database that keeps track of call statuses parallel to the fone file can be updated. The file may be the source of the SURVOX .fon file. New sample can be added if a new phone record is generated.
- You can base survey logic on information contained in a client's database. Upon making a "transaction," a customer could be directed to a survey based on the specific transaction. Then, a new data record or other information can be added to the database.
- An invitation to a survey can be generated when a status within a client's database is updated.
- Complex, real-time tables that combine survey data and changing data within the database can be generated.
- A database can help you keep track of interviewer productivity numbers. Every time an interviewer finishes a call, the database is updated to show how productive the interviewer is.

Getting started

SURVOX Support policy:
Installation of the ODBC-compliant database as well as the ODBC API (application programming interface) will be the responsibility of the client. All knowledge of the ODBC-compliant database is the responsibility of the client.

ODBC-compliant databases installed on SURVOX Machines are:

MySQL
Microsoft Access (using EasySoft)

Other known ODBC-compliant databases that can be found at http://www.unixodbc.com/drivers.html) are:

AdabasD
Empress
IBM DB2
Informix
Interbase
Microsoft SQL
Mimer SQL
MiniSQL
News Server
Ovrimos Webbase
Postgres
RDBMS Linter SQL
SQL
Starquest DB2
Sybase
Yard SQL

ODBC API:
The ODBC API (application programming interface) serves as a bridge between SURVOX software and the ODBC-compliant software. It must be installed in order for the different software to communicate with each other.

For more details, http://www.unixodbc.com/

Additional environment variable:
LD_LIBRARY_PATH points to the location of the ODBC API

Add to .login:
```
setenv LD_LIBRARY_PATH
```

This example is for EasySoft:

```
setenv LD_LIBRARY_PATH
/usr/local/easysoft/lib:/usr/local/easysoft/oob/client:/usr/local/easysoft/unixODBC/lib
```

Troubleshooting:

Error: dynamic linker: mentor: error opening libodbc.so1

Explanation: The ODBC API cannot be found (in example, libodbc.so1 is the ODBC API)

Solution: As root, type "find / -name libodbc.so1 -print", add directory where the ODBC API is located to LD_LIBRARY_PATH environment variable for user.

If the ODBC API is not found: It has not been installed or it has not been installed properly.

Mentor’s interface with ODBC

The Linux and Max OS X Unix versions of Mentor 7.7 have the ability to read, add and delete “rows” from an ODBC database.

Connecting to the database

This new meta command is used to make a connection to a database:

>use_rdbms rdbms_name=<reference name> odbc=(<uname>, <pass>, <datasource>)

<reference name> is 1-14 characters; it may be called anything you want. This name is used as SURVOX's "tag" for this connection.
<uname> is the ODBC user name you want to connect as.
<pass> is the password for this user.
<datasource> is an ODBC Data Source name that is used to dictate which database you want to connect to.

You may have more than one connection by using multiple commands. But each connection needs to have a unique **<reference name>.**

Telling Mentor what data to use

Mentor is told what data to use via the **~input statement**. New subcommands have been added to the ~in syntax.

```
~input rdbms=<reference name>sql="select statement"&
        max_var_length=<size>max_text_length=<size>&
case_length=<value> work_length=<value> total_length=<value>&
precision_default=<digits> scale_default=<decimals>total_text_length=<tlen>&
text_location=<start>[.<tlen>] make_variables&
        rdbms_mapdots new_buffer number_input_buffers print_case_id
read_first_case&
        stop_after study_name= id= next_case_id= allow_update allow_new
```

rdbms= must be the first keyword on this type of ~in statement. This tells the program it is going to access an ODBC database.

<reference name> is used to indicate which **>use_rdbms** connection it is using for this selection.

sql="select..." is used to specify the SQL select used to access the data you want. You can continue this onto multiple lines using the standard long line syntax of:

```
"select" &&
" * from mytab"
```

The SQL statement must be a select type. It can be any valid SQL select statement. This statement is passed to ODBC so that it can be parsed.

Once the statement is parsed by ODBC Mentor calls ODBC routines to figure out how many "elements" are referred to in the select statement. Mentor gets the type of each "element". For now, the program deals with varchar2, number, char, and date types. For each number "element" the number of digits and number of decimal places is also determined. Date type "elements" are always put into the data as YYYYMMDDHHMMSS.

Mentor variables

max_var_length=<size> **(**default=70)
If a varchar2 or char type "element" is less than or equal to <size> characters, then it is treated as VAR data. If the number of characters is more than <size>, then it is treated as TEX data.

max_text_length=<size> (default=2000)
Dictates how much data from a varchar2 or char type "element" the software should use. The database will pass the whole contents of the "element" over to SURVOX software and then SURVOX software will put in the last three positions of data that is longer than **max_text_length.**
<size> can be from 1 - 2000 (2000 being the largest amount of text that can put into a text type question).

case_length=<value> (default=0)
This says how many columns to allocate for the data that is coming from the database. If you set this number smaller than what you actually need, then an error occurs.

work_length=<value> (default=0)
This is how many columns to allocate to the case's work area.

total_length=<value> (default=0)
This is how many columns to allocate to the whole in-core case. If **case_length** is not set, then use **work_length,** if set, otherwise use **total_length**. If **case_length** is still not set, then use the value calculated from the select information. If **work_length** is not set, then set to **total_length**, if set, else use **case_length**. If **total_length is** not set then set to **work_length**. **work_length** must be greater than or equal to **case_length. total_length** must be greater than or equal to work_length.

precision_default=<digits> (default=15)
If the number of digits for a number type "element" is unknown then this is used.

scale_default=<decimals> (default=2)
If the number of decimal places for a number type "element" is unknown then this is used.

total_text_length=<tlen> (default=0)
This is used to figure of how much area to allocate for text data. What you specify is the number of columns in the text area. If not set, then the program sums the total number of text characters that will be put into the data. The formula -- number of text questions times 4 + total text characters divided by 2) -- is used to set the number of text columns.

text_location=<start>[.<tlen>]
<start> must be at least where it would be if nothing was said. If .tlen is said then if total_text_length is also said they'd better match.
If either .tlen or total_text_length is said, this is the width of the text area. Otherwise, use case_length if said or work_length if said or total_length if said to figure out how long the text area should be. Otherwise, use the formula: number of text questions times 4 + total text characters divided by 2.

make_variables (default=yes)
Dictates whether or not variables should be created for each "element".

rdbms_map (default=yes)
Dictates if a map where each element is to be put into the case should be generated.

The following keywords have the same use and meaning as on a regular ~in statement:
dots new_buffer number_input_buffers print_case_id read_first_case stop_after study_name= id= next_case_id= allow_update allow_new

read_first_case defaults to yes.

next_case_id= defaults to 1 and is used when generating case IDs if the id= keyword is not used.

If **study_name** is not set, then the **<reference name>** is used if it is short enough.

The name of each variable is the same as the ODBC name. If this name is more than 14 characters, then the name is taken as a long variable name.

Mentor maps each "element," one behind the other, to figure out how many columns of data there are. It also figures out how many TEX-type variables there are. The case width is the number of columns to hold all the data plus the text area.

Once the SQL statement is parsed and Mentor figures out what the case will look like, the program is ready the tell ODBC to execute the SQL select statement.

- The software does not need to re-execute the SQL select statement for ODBC. ODBC is told to start again at the first “row.” Some database systems may not be able to do this. Therefore, an SQL statement might be re-executed to accomplish the rewind operation.

Retrieving a case

When Mentor retrieves a “case” from ODBC, the data is put into a “holding area. Mentor then looks at each "element" before putting it into the mapped columns. A varchar2 and char type “element” is put into the case either as a VAR or TEX variable, based on its maximum length. The data might be truncated, based on the value of max_text_length.

A number type “element” is received as a “string” and passed to SURVOX’s string to a number routine. The number is then passed to SURVOX'S number to string routine to format it with
the correct number of digits and decimal places. This formatted string is then put into the case. A date type “element” is received as 7 bytes of data in an ODBC format. This format is converted to a YYYYMMDDHHMMSS string and passed on to SURVOX's time cracking routine. This routine returns a SURVOX date, and this is then passed to the SURVOX routine that formats a string as YYYYMMDDHHMMSS. This string is what is put into the case.

This number and date work is done to make sure that the data that ends up in the case will be treated correct by our software.

These dump switches can be used to control displaying some information in the database.

F4 shows the original data fetched from the database
F5 shows the parsed data put into the data
J7 shows information about the particular SQL select
L2 shows more information than when J7 if used alone
L3 shows available data sources and drivers
L4 shows the first difference in the control area
L7 shows details about the current driver connected to
L8 show more information then when L7 is used alone

Survent's Interaction with ODBC

This run/setup demonstrates how, from Survent interviewing, the data goes both to a data file and is entered into a database, in this case MySQL. Key commands and options of in this setup are:

Compiler directives, key commands:

```
!odbc, open
!odbc, select
!odbc, put
!odbc, add

!NUM,X subtype
!EXP,X subtype
!SPC,A
```

This setup can be used only if unixODBC is installed and configured to access a MySQL database.

Note: It is very important to use "**Option=1**" in the **odbc.ini file** so that each field's full possible width is used, instead of the widest field that happens to appear in the current select.

The **!odbc, open** command logs the user onto the database.
The **!odbc select** statement should get either one record, or, as in this case, no records. If one record is selected, it is loaded into a work area that matches the schema (layout) represented by the select. If no one is selected, an empty work record matching the schema is created. Since this setup is creating only new database records, an empty work record is what is desired.
The **!odbc, put** commands put information into the work record.
The **!odbc, add** command adds the work record to the data base.

Normally, a Survent interview has no case id until the record is written to the data file. Because, in this run, we would like the database id and the interview id to match, the case id is acquired via an **!spc,a** and a hidden question that uses the case id columns is loaded into the database as the id.

IMPORTANT NOTE:
Notice that the **!odbc put** of any CAT question gets a **warn #7117**. This is because there is no database field type that corresponds to a multi-punched column. In this setup, the single-punch CAT questions happen to work but all of the multipunched CATs are wrong. (Compare odbcint3.prt with odbcint4.prt after running the example specs.)
When a column has been multi-punched, it usually appears as an asterisk in the database, just as would be the case in an ASCII data file. Other times a character may Appear, which is a representation of the data punches (e.g. 1& = A).

Sample spec files are as follows:

odbcint1.spx
odbcint2.spx
odbcint3.spx
odbcint4.spx

Here are the spec files and some comments within them:

Odbcint1.spx

```
~comment

" Refresh the database.
>system runmysql odbcint1

~prepare compile cmentor
[odbcint1]

{Q1:
Simple cat question with numeric response codes.
Legal answers are 1 or 2.
!cat
1 one
2 two
}

{Q2:
Simple cat question with alpha response codes.
Legal answers are Y or N.
!cat
Y Yes
N No
}

{Q3:
Simple fld question.
Legal answers are DIA or SFO.
!fld
DIA Denver
SFO San Francisco
}
```

```
{Q3m:
Multiple answer fld question.
Legal answers are DIA, SFO, AAA, BBB, CCC.
!fld,,5
DIA Denver
SFO San Francisco
AAA Ehhh
BBB Bee
CCC See
}

" Use subtype X so that numeric(2,0) in database works with this question. Without it,
" there will be a 2 <> 3 size mismatch.
{Q4:
Simple num question with a single numeric exception code.
Legal answers are 1-98, 99.
!num,X,,1-98,99
}

" This is defined as a CHAR in the data base because of the alpha exception.
{Q5:
Simple num question with a single alpha exception code.
Legal answers are 1-99,DK.
!num,,,1-99,,DK
}

" Use subtype X so that numeric(9,0) in database works with this question. Without it,
 " there will be a 9 <> 10 size mismatch.
{Q6:
" Simple expression.
!expr,X,2*10
}

{Q7:
Simple var question.
From 1 to 70 characters
!var,,70
}

{Q8:
Simple text question.
From 1 to about 700 characters
!tex
}
```

```
" Multi-response cat questions won't load into a database correctly.
{Q9:
Multiple response cat question, max of five answers, no exceptions.
Up to 5 answers.
!cat,,5
1 brand A
2 brand B
3 brand C
4 brand D
5 brand E
}

{Q10:
This is just a cat question with a somewhat long title and fairly long
response list.
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
Legal answers are 01-26
!cat
01 a
02 b
03 c
04 d
05 e
06 f
07 g
08 h
09 i
10 j
11 k
12 l
13 m
14 n
15 o
16 p
17 q
18 r
19 s
20 t
21 u
22 v
23 w
24 x
25 y
26 z
}

" A multi-response cat questions won't load into a data base correctly.
{Q11:
This is just a cat question with a somewhat long title and fairly long
response list. This question allows multiple answers.
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa aaaaa
Legal answers are 01-26 and there may be up to 26 answers.
!cat,,26
01 a
02 b
03 c
04 d
05 e
06 f
07 g
08 h
09 i
10 j
11 k
12 l
13 m
14 n
15 o
16 p
17 q
18 r
19 s
20 t
21 u
22 v
23 w
24 x
25 y
26 z
}

" ODBC-related codes start here:

" The first "SURVOXtest" is the database name. The next two positions are for user and
" password, which in this case are specified in the SURVOXtest entry in ~/.odbc.ini. The last
"SURVOXtest" argument is the entry name from ~/.odbc.ini.

{open:
!odbc, open, SURVOXtest, , , SURVOXtest }
{
!if open$ = "Y"
\aThe database opened just fine \|:open:|
!display }
{
!if open$ = "N"
\aThe database did not open. \|:open:|
!display }
{
!if open$ = "Q"
\aThere is no database support. \|:open:|
!display }

" SURVOXtest is the database name. sel1 is a name the user chooses as a label for this select
" statement. Since this run is intended to always create new records, "where id = NULL
" is used so that no existing records are selected. After this command has been executed,
" an empty work record having the schema or layout of the database is created and
 " waiting to be loaded and manipulated.

{select:
!odbc,select,SURVOXtest,sel1,"select * from odbcint1 where id = NULL"}
{
!if select$ = "Q"
Q if no database support \|:select:|
!display }
{
!if select$ = "S"
S if setup fails \|:select:|
!display }
{
!if select$ = "F"
F if find_text_area fails \|:select:|
!display }
{
!if select$ = "C"
C if control area not match \|:select:|
!display }
{
!if select$ = "E"
E if SQLExecute failed \|:select:|
!display }
```

```
{
!if select$ = "T"
T if more than one case selected \|:select:|
!display }
{
!if select$ = "N"
N if no cases selected \|:select:|
!display }
{
!if select$ = "Y"
Y if okay \|:select:|
!display }

" Put the answers from the questionnaire into the work record. SELL is the name of the
" previously executed select statement. The first Q1 is the destination name in the database,
" the second is the source question label from the Survent questionnaire.
{ !odbc, put,sel1,Q1,Q1 }
{ !odbc, put,sel1,Q2,Q2 }
{ !odbc, put,sel1,Q3,Q3 }
{ !odbc, put,sel1,Q3M,Q3M }
{ !odbc, put,sel1,Q4,Q4 }
{ !odbc, put,sel1,Q5,Q5 }
{ !odbc, put,sel1,Q6,Q6 }
{ !odbc, put,sel1,Q7,Q7 }
{ !odbc, put,sel1,Q8,Q8 }
{ !odbc, put,sel1,Q9,Q9 }
{ !odbc, put,sel1,Q10,Q10 }
{ !odbc, put,sel1,Q11,Q11 }

" Getting the ID as late as possible should help keep things in sequence.
{getid:
!spc,a
}
{id: 1.4 hide
!var }
{!odbc, put,sel1,id,id }

" Finally, add the work record to the database.
{add:
!odbc, add,sel1}
{
!if add$ = "Q"
Q add reports no database support
!display }
{
!if add$ = "N"
N add reports no such SELECT
!display }
{
!if add$ = "E"
E add reports error in update
!display }
{
!if add$ = "Y"
Y add reports okay
!display }

~end
```

Odbcint2.spx

```
odbcint1
dbug
y
rdg
y
0
10
1
n
n
n
0
n
n
odbcint2.prs
```

Odbcint3.spx

```
~set autotab
~specfile odbcint3
~def

&odbcint1.def

>printfile odbcint3
~input odbcint1
~exec
```

```
edit={: -coltna }
col=: total
maketables
~end
```

Odbcint4.spx

```
>use_rdbms rdbms_name=odbcint4 odbc=(,,SURVOXtest)
~input
        rdbms=odbcint4
        sql="select * from odbcint1"
        length=1500
        " text_location=161
        ;
~set autotab
~specfile odbcint4
~def

&odbcint4.def

>printfile odbcint4
~exec
edit={: -coltna }
col=: total
maketables
~end
```

Linux/Mac OS X Unix

Linux/Mac OS X Unix Version 7.7 Survent understands the following:

{!odbc, open, <Database ID>, <username>, <password>, <datasource name> }

{select:
!odbc, select, <Database ID>, <Select ID>, &
"Real SQL Select statement", &
"Test SQL Select statement" }

{get:
!odbc, get, <Select ID>, <ODBC Variable name>, <Survent location> }

{put:
!odbc, put, <Select ID>, <ODBC Variable name>, <survent location> }

```
{add:
 !odbc, add, <Select ID> }

{delete:
!odbc, delete, <Select ID> }

{update:
 !odbc, update, <Select ID> }
```

<err loc> or whatever your label is (select, update, etc.) is a one-column !var question that gets the result code from the requested operation.

Here are the various results or return values:

!odbc,open return values:
   Q if no database support
   Y if open
   N if not open

!odbc,select return values:
   Q if no database support
   S if setup fails
   F if find_text_area fails
   C if control area not match
   E if SQLExecute failed
   T if more than one case selected
   N if no cases selected
   Y if okay

!odbc,get or **put** return values:
   N if error in operation
   Y if okay

!odbc,add return values:
   Q if no database support
   N if no such SELECT
   E if error in update
   Y if okay

!odbc,delete return values:
   Q if no database support
   N if no such SELECT
   E if error in update
   Y if okay

!odbc,update return values:
Q if no database support
N if no such SELECT
E if error in update
Y if okay

The select sub-type ODBC question allows two Select SQL statements. The real one is used by Survent. The test one, if present, is used by Mentor during the compile. As the example below will show, it is likely that you have some variable that needs to be filled in to make the real select statement. As Mentor needs to know what data is being pulled from the database at compile time, an alternate select SQL statement is used. Mentor remembers the number of data elements and makeup of each one, and they must be the same when the real statement is parsed and executed by Survent.

A test .qpx file

Here is a test qpx file:

```
>purgesame
~prep compile
[odbc]

{ open: .1
!odbc, open, test2, "", "", mysqltest }

{
!if open$ <> "Y"
\aCan not open the database \|:open:|
!display }

{
!if open$ <> "Y"
!goto done }

{ recnum: .4
\aEnter the record number you want to deal with?
!num,x,,1,999 }

{ select: .1
!odbc, select, test2, sel1, &
"select * from junk where Record = \|:recnum:|", &
"select * from junk" }

{
!if select$ <> "Y"
\aCan not select from the database \|:select:|
!display }

{
!if select$ = "N"
```

```
!goto add }

{
!if select$ <> "Y"
!goto done }

{ odbc_num: .7 hide
!num,x,,-99999,999999 }

{ get1: .1
!odbc, get, sel1, Num2, odbc_num }

{
!if get1$ <> "Y"
\aCan not get Num2 from the database \|:get1:|
!display }

{
!if get1$ <> "Y"
!goto done }

{ odbc_txt: .1 hide
!text }

{ get2: .1
!odbc, get, sel1, Title, odbc_txt }

{
!if get2$ <> "Y"
\aCan not get Title from the database \|:get2:|
!display }

{
!if get2$ <> "Y"
!goto done }

{ rep_num: .7
\aWe just got \|:odbc_num:| from the database.\n
What do you want to put in its place?
!num,x,,-99999,999999 }

{ put1: .1
!odbc, put, sel1, Num2, rep_num }

{
!if put1$ <> "Y"
\aCan not put Num2 to the database \|:put1:|
!display }

{
!if put1$ <> "Y"
!goto done }

{ odbc_dsp: odbc_txt
!text,a }

{ rep_txt:
```

```
\aWe just got \:odbc_dsp: from the database.\n
What do you want to put in its place?
!text }

{ put2: .1
!odbc, put, sel1, Title, rep_txt }

{
!if put2$ <> "Y"
\aCan not put Title to the database \|:put2:|
!display }

{
!if put2$ <> "Y"
!goto done }

{ update: .1
!if rep_num >= 0
!odbc, update, sel1 }

{ delete: .1
!if rep_num < 0
!odbc, delete, sel1 }

{
\adone with \:odbc_num:
!var }

{!goto done }

{ add: .7
\aWe are creating a new case.\n
What do you want to put in for Num2?
!num,x,,-99999,999999 }

{ put3: .1
!odbc, put, sel1, Num2, add }

{ put4: .1
!odbc, put, sel1, Record, recnum }

{ add2: .1
!odbc, add, sel1 }

{ done:
\aWe are all done
!var }

~end
```

Please keep in mind that this is a preliminary document that will be revised as development and beta testing progresses. Please contact your usual support personnel at SURVOX by e-mailing support@SURVOX.com.