Database - Feature #3549

implement RAW-TRANSFER statement

04/12/2018 05:52 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #3758: misc database features	**Closed**
Related to Database - Feature #2137: runtime support for FIELDS/EXCEPT record...	**New**

History

#2 - 04/13/2018 04:51 PM - Greg Shah

Useful links:

How is the order of fields in temp-table definitions specified?

4GL/ABL: How to use the RAW-TRANSFER method with a dynamic table buffer?

4GL, Signatures, RAW-TRANSFER, Temp Tables and How they Interact

RAW-TRANSFER error 4955 between temp-table and database table

https://www.progresstalk.com/threads/raw-transfer-database-update.121990/

#3 - 04/16/2018 01:22 PM - Greg Shah

The following are my early findings based on research and some initial testcases (see testcases/uast/raw_transfer/*). This analysis is not complete at this time.

RAW-TRANSFER is used in 3 possible modes:

- **BUFFER to RAW** - serialize the contents of a buffer into a stream of bytes.
- **RAW to BUFFER** - deserialize from a stream of bytes, create a record and assign the contents.
- **BUFFER to BUFFER** - a kind of buffer copy that can work whenever the signatures of two buffers match. When using RAW-TRANSFER from one buffer to another, it is similar in functionality to BUFFER-COPY, but there are some low level details that are different between them. Presumably, a worker in the runtime can just implement its own hidden RAW variable and call the BUFFER to RAW processing followed by the RAW to BUFFER processing.

For this reason, the implementation will most likely be focused on the first two forms.

Core to the RAW-TRANSFER statement is the concept that each record has a signature that involves 3 pieces of information:

- The list of fields in POSITION order (e.g. _FIELD._FIELD-RPOS).
- The data type for each field (e.g. _FIELD._DATA-TYPE).
- The extent value for each field (e.g. _FIELD._EXTENT).

The signature seems to be measuring the structure of the table itself. The names of the fields DO NOT MATTER. It is perfectly fine to transfer records between tables that use completely different names for the same structure.

The CRC-VALUE of the table also seems not to matter, though it tends to also change with the same metadata above, so it is a kind of related thing.

The POSITION value seems to be about the physical sequencing of the record buffer and it has implications for R-CODE. POSITION@ can be (but is not required to be) different from the ORDER which seems to be more about the DISPLAY sequencing. Both values are specified in the schema for permanent databases and are copied into TEMP-TABLES that are created LIKE a schema table. In order for POSITION to appear in the schema, the schema must be exported with a checkbox set for including the data for r-code compatibility. For temp-tables that are not defined LIKE another table, the ORDER is implicitly the same as the sequencing of the FIELD clauses in the 4GL source code. The POSITION value is implicitly set to be the same as the ORDER value. Neither can be set explicitly, only the sequencing of the source code can be controlled by the programmer.

This signature is written out in the serialized bytes and upon deserialization, it is compared to the signature of the target buffer before writing the changes into that buffer.

#4 - 04/16/2018 06:59 PM - Greg Shah

When I started looking at some testcases, I took a quick look at the serialized output to see how much of the structure could be figured out quickly.

testcases/uast/raw_transfer/raw_transfer_int_encoding.p outputs this:

```
                                                    Summary Data
CHARSET              Internal Codepage    CRC-VALUE   POSITION DATA-TYPE        EXTENT
-------------------- -------------------- --------- ---------- ------------ ----------
1252                 1252                 0xFFFFBA           2 integer               0
```

```
                                    Encoded Test Data
RECID       ROWID                   Record Length RAW Length Data Value                                Encode
d Record Hex Dump
----------- ------------------------ ------------- ---------- ---------------------------------------- ------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------
0x00000900  0x0000000000000900                   17         41 127 (0x0000007F)                         715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF017F00
0x00000901  0x0000000000000901                   19         43 4096 (0x00001000)                        715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF0210000101
0x00000902  0x0000000000000902                   20         44 32767 (0x00007FFF)                       715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF03007FFF0102
0x00000903  0x0000000000000903                   21         45 8388607 (0x007FFFFF)                     715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF04007FFFFF0103
0x00000904  0x0000000000000904                   21         45 2147483647 (0x7FFFFFFF)                  715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF047FFFFFFF0104
0x00000905  0x0000000000000905                   18         42 -1 (0xFFFFFFFF)                          715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF01FF0105
0x00000906  0x0000000000000906                   19         43 -128 (0xFFFFFF80)                        715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF02FF800106
0x00000907  0x0000000000000907                   20         44 -32768 (0xFFFF8000)                      715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF03FF80000107
0x00000908  0x0000000000000908                   21         45 -134217728 (0xF8000000)                  715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF04F80000000108
0x00000909  0x0000000000000909                   21         45 -2147483648 (0x80000000)                 715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF04800000000109
0x0000090A  0x000000000000090a                   17         41 0 (0x00000000)                           715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFF00010A
0x0000090B  0x000000000000090b                   17         41 ?                                        715302
00020002000000000000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFFFD010B
```

testcases/uast/raw_transfer/raw_transfer_int_array_encoding.p outputs this:

```
                                                                Summary Data
CHARSET              Internal Codepage    CRC-VALUE   POSITION DATA-TYPE        EXTENT
-------------------- -------------------- --------- ---------- ------------ ----------
1252                 1252                 0xFFFFFE           2 integer               3

                                      Encoded Test Data
RECID       ROWID                     Record Length RAW Length Data Value                                Encode
d Record Hex Dump
----------- ------------------------- ------------- ---------- ---------------------------------------- ------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------
0x00001100  0x0000000000001100                   25         53 127 (0x0000007F)                           715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA0007017F017F017FFF00
0x00001101  0x0000000000001101                   29         57 4096 (0x00001000)                          715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA000A021000021000021000FF0101
0x00001102  0x0000000000001102                   32         60 32767 (0x00007FFF)                         715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA000D03007FFF03007FFF03007FFFFF
0102
0x00001103  0x0000000000001103                   35         63 8388607 (0x007FFFFF)                       715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA001004007FFFFF04007FFFFF04007F
FFFFFF0103
0x00001104  0x0000000000001104                   35         63 2147483647 (0x7FFFFFFF)                    715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA0010047FFFFFFF047FFFFFFF047FFF
FFFFFF0104
0x00001105  0x0000000000001105                   26         54 -1 (0xFFFFFFFF)                            715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA000701FF01FF01FFFF0105
0x00001106  0x0000000000001106                   29         57 -128 (0xFFFFFF80)                          715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA000A02FF8002FF8002FF80FF0106
0x00001107  0x0000000000001107                   32         60 -32768 (0xFFFF8000)                        715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA000D03FF800003FF800003FF8000FF
0107
0x00001108  0x0000000000001108                   35         63 -134217728 (0xF8000000)                    715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA001004F800000004F800000004F800
0000FF0108
0x00001109  0x0000000000001109                   35         63 -2147483648 (0x80000000)                   715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA0010048000000004800000000048000
0000FF0109
0x0000110A  0x000000000000110a                   23         51 0 (0x00000000)                             715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA0004000000FF010A
0x0000110B  0x000000000000110b                   23         51 ?                                          715302
0002000200000001000200030000040000FFFF313235320000FA000B0001010103FDFDFDFDFDFDFFFA0004FDFDFDFDFF010B
```

testcases/uast/raw_transfer/basic_multi_column_raw_transfer_buffer_to_raw_to_different_buffer.p outputs this:

```
                                                                Summary Data
CHARSET              Internal Codepage    CRC-VALUE
-------------------- -------------------- ---------
1252                 1252                 0xFFFF91

                                                                                                  Field Details
  POSITION DATA-TYPE        EXTENT Value
---------- ------------ ---------- ----------------------------------------
         2 integer               0        49374
         3 int64                 0          3405691582
         4 decimal               0   1.111,11
         5 logical               0 no
         6 date                  0 09/01/04
         7 datetime              0 09/01/2004 08:30:00,000
         8 datetime-tz           0 12/01/2020 17:30:00,000-04:00
         9 raw                   0 ♦♦♦♦♦♦♦♦
        10 character             0 abcdefg
        11 handle                0        1139
        12 recid                 0    8448
        13 rowid                 0

 Encoded Test Data
RECID       ROWID                     Record Length RAW Length
Encoded Record Hex Dump
------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------------------------------------------------------------------
0x00002100  0x0000000000002100                    85        121
715302000D000D0000000000000042905030222808010A070D000000FFFF313235320000FA000B000109010DFDFDFDFDFDFF0300C0DE05
00CAFEBABE048211111100024D86064D8601D2EB400C049D07C0000064B5FFFFFF1008FFFFFFFFFFFFFFFF07616263646566702047302
2100080000000000002100
```

The encoded output always seems to have two bytes with 0x7153 at the beginning. This seems to have no obvious meaning. It was the same on OE 11.7 Windows as on OE 10.2B Linux.

After that seems to be a descriptor showing the number of fields:

```
0x0200020002  (single field)
0x02000D000D  (12 fields)
```

I think the 02 prefix is the number of following bytes to read. The single field case is then followed by 0002 and the 12 field case is 000D. In both cases, the value seems to be 1 greater than the number of fields. I think they have a partial encoding of the recid/rowid at the end of the data that may be the "extra" field. I think they then repeat the same number of fields value again, but without the "number of bytes to read" prefix. I'm not sure why they do this. So far, the two values are always the same. Anyway, this would be the first part of the record signature.

I believe that the next part is the rest of the record signature. I think you can read from there up to a delimiter of 0xFFFF (actually the delimiter is probably 0x0000FFFF):

```
000000000000040000FFFF (1 int field, extent 0)
0000000100020003000040000FFFF  (1 int field, extent 3)
0000000000000429050302222808010A070D000000FFFF (12 fields mixed data types, all extent 0)
```

I suspect that the 0x000000 is just some kind of spacer or empty section. It seems to be constant so far.

I think the next part is the extent specification. The cases where there are no extents seem to encode this as another empty section 0x000000. But the single int field of extent 3 has 0x01000200030000. I think this is read as:

- 01 is the length of the first part and 00 is the value (probably meaning that the extra or implicit field is not an array)
- 02 is the length of the second part and 0003 is the value (probably the extent of 3 for the "second" field which is the actual int field in this case)
- 0000 as the delimiter to end the section

If this interpretation is correct, then it explains why 000000 is an empty section. The leading 00 means there is no following data to read and the ending delimiter is still 0000.

After the extent section is something that I suspect is the field-level data types:

```
04 (1 int field, extent 0)
04 (1 int field, extent 3)
042905030222808010A070D (12 multi-data type fields, all extent 0)
```

In all 3 tables, the first field is an integer which may be encoded as 04. If this is the data types list in position sequencing, then the types for the 3rd table would be as follows:

04 integer
29 int64
05 decimal
03 logical
02 date
22 datetime
28 datetime-tz
08 raw
01 character
0A handle
07 recid
0D rowid

This is pretty plausible since the higher values are definitely the types added later. Even handle and rowid were added after the initial base types (01 through 08). Curiously, I wonder what type 06 is. As far as I know there are only COM-HANDLE, CLOB, BLOB and CLASS that are not yet tested

(all which would be outside of the first 8 values). Perhaps 06 is memptr and since they don't allow that as a field, it is a gap in the values.

From there the signature seems to end with 0000FFFF (for the single field cases) and 000000FFFF for the 12 field case. It is not clear why there is an extra 00 byte in the multi-field case.

Following the FFFF signature delimiter is the codepage. I guess this just affects the character data fields, but anyway, the codepage is encoded like this:

313235320000 (OE 11.7 Windows system using codepage "1252", 31 is the the ASCII encoding for the digit "1", 32 is "2", 35 is "5", 32 is "2")
49534F383835392D313500000000 (OE 10.2B Linux system using codepage "ISO8859-15", ASCII encoding is 49 for "I", 53 for "S", 4F for "O", 38 for "8", 35 for "5", 39 for "9", 2D for "-" and 31 for "1", 35 is "5")

Then there is an ending delimiter of 0000 and 00000000 respectively. I don't know why there is a difference.

Next is a section which I cannot explain, other than to say that I think it encodes some kind of array.

```
FA000B0001010103FDFDFDFDFDFF (single int field, extent 0)
FA000B0001010103FDFDFDFDFDFF (single int field, extent 3)
FA000B000109010DFDFDFDFDFDFF (12 fields multi-data types, no extents)
```

Based on other analysis, I think the FA indicates the start of an array and the final FF is the ending array delimiter. The 00 is how they encode a field of value 0 and the 0B is probably "read the next 11 bytes", which would be 0001010103FDFDFDFDFDFF or 000109010DFDFDFDFDFDFF respectively. As shown below the FD is likely to mean unknown value. I don't know what the other data is. There is 01 versus 09 and 03 versus 0D, but it is not obvious what these mean.

I believe that the rest of the encoding is the actual data. There is always an implicit field at the end which is a portion of the recid. Why they don't store the entire recid, I don't know. But the data is otherwise encoded in position order. I think that each data type will have its own way of being encoded.

For integer, there are four styles of encoding:

00 is the number 0
FD is the unknown value
<size_in_1_byte_value><variable_length_value> is a single scalar value
FA<total_following_bytes_including_ff_delimiter_in_2_byte_value><size_in_1_byte_value><variable_length_value><size_in_1_byte_value><variable_length_value><size_in_1_byte_value><variable_length_value>FF is a 3 extent array (BTW, this suggests that the overall size consumed by all elements of a given array is a maximum of 32Kbytes in size)

Recid seems to be encoded the same way as a single scalar integer.

Single integer field, extent 0:

Single Integer Value	RECID	Encoded (int field is separated from partial recid for clarity)
127 (0x7F)	0x00000900	017F 00
4096 (0x1000)	0x00000901	021000 0101
32767 (0x00007FFF)	0x00000902	03007FFF 0102
8388607 (0x007FFFFF)	0x00000903	04007FFFFF 0103
2147483647 (0x7FFFFFFF)	0x00000904	047FFFFFFF 0104
-1 (0xFFFFFFFF)	0x00000905	01FF 0105
-128 (0xFFFFFF80)	0x00000906	02FF80 0106
-32768 (0xFFFF8000)	0x00000907	03FF8000 0107
-134217728 (0xF8000000)	0x00000908	04F8000000 0108
-2147483648 (0x80000000)	0x00000909	0480000000 0109
0	0x0000090A	00 010A
unknown value	0x0000090B	FD 010B

Initially I assumed that certain data types would have simple fixed size encodings. For example, integer is a 32-bit signed value and I assumed that it would always be encoded in 4 bytes. CHARACTER and RAW types are naturally variable length (up to 32K or so), so for those I expected some kind of 2 byte length value followed by the encoded output. But clearly, there is a variable length used here even for very small fixed data types.

Integer array, extent 3 (in each example all 3 elements are assigned the same value):

Element Integer Value	RECID	Encoded (portions are separated for clarity)

127 (0x7F)	0x00001100	FA 0007 017F 017F 017F FF 00
4096 (0x1000)	0x00001101	FA 000A 021000 021000 021000 FF 0101
32767 (0x00007FFF)	0x00001102	FA 000D 03007FFF 03007FFF 03007FFF FF 0102
8388607 (0x007FFFFF)	0x00001103	FA 0010 04007FFFFF 04007FFFFF 04007FFFFF FF 0103
2147483647 (0x7FFFFFFF)	0x00001104	FA 0010 047FFFFFFF 047FFFFFFF 047FFFFFFF FF 0104
-1 (0xFFFFFFFF)	0x00001105	FA 0007 01FF 01FF 01FF FF 0105
-128 (0xFFFFFF80)	0x00001106	FA 000A 02FF80 02FF80 02FF80 FF 0106
-32768 (0xFFFF8000)	0x00001107	FA 000D 03FF8000 03FF8000 03FF8000 FF 0107
-134217728 (0xF8000000)	0x00001108	FA 0010 04F8000000 04F8000000 04F8000000 FF 0108
-2147483648 (0x80000000)	0x00001109	FA 0010 0480000000 0480000000 0480000000 FF 0109
0	0x0000110A	FA 0004 00 00 00 FF 010A
unknown value	0x0000110B	FA 0004 FD FD FD FF 010B

More work will be needed to figure out the encoding for every data type. The tricky part is ensuring that all possible values are tested.

If you look at the details above, you will see that the RECORD-LENGTH() for a buffer is smaller than the LENGTH() of the encoded value. For example, the table with a single int field (no extent) is 17 bytes but the encoded length is 41 bytes. Of course, this will vary with the variable encoding of the different data. Even the recid can change (a recid ending in 00 is encoded as 00 but a recid ending in 0A is encoded as 010A following the rules for integer encoding.

The signature is certainly a big part of the difference in sizes. I did not check to see if it is the only difference.

Here is a quick parsing of the 12-field record where each field had a different data type:

0300C0DE0500CAFEBABE048211111100024D86064D8601D2EB400C049D07C0000064B5FFFFFF1008FFFFFFFFFFFFFFFF0761626364656667
020473022100080000000000002100

03 00C0DE (integer 0xCoDE)
05 00CAFEBABE (int64 0xCAFEBABE, looks like an extended up to 8 bytes integer encoding)
04 82 111111 (decimal 1111.11 with DECIMALS 2)
00 (logical no)
02 4D86 (date 09/01/04 which is a julian day number)
06 4D86 01D2EB40 (datetime 09/01/2004 08:30:00,000 which is a julian day + milliseconds from midnight which is a max value of 86,400,000)
0C 049D 07C00000 64B5FFFFFF10 (datetime-tz 12/01/2020 17:30:00,000-04:00, same as datetime plus an offset in minutes from gmt?, 2 bytes would be enough so I'm not sure what the FFFFFF is for)
08 FFFFFFFFFFFFFFFF (raw where -1 was stored using PUT-INT64())
07 61626364656667 (character value "abcdefg")
02 0473 (handle value 1139 encoded as an integer)
02 2100 (recid 0x2100 encoded as an integer)
08 0000000000002100 (rowid encoded as a fixed size 8 byte int64)

This one doesn't have the partial recid/rowid field at the end. The recid and rowid fields that are there were explicitly put into the table (by me). I would have expected an extra 00 at the end of the encoded output so that is a bit confusing.

#5 - 04/16/2018 07:10 PM - Greg Shah

Do to a short term need to provide a working implementation, I am writing a "basic" implementation that works but which is not compatible. There will be error handling differences and most importantly, the serialized output will be different. This means that this implementation will definitely fail if provided with data serialized from the 4GL itself, but when using data serialized by FWD, it should work.

With this in mind, I'm not going to do any additional testcases at this time. When this work is restarted, the testcases need to be extended as follows:

- data type encoding
  - implement full testcases for all the data types which FWD supports (integer is the only one that is complete right now)
  - when FWD supports them in the database, add COM-HANDLE, CLOB, BLOB and CLASS
  - make sure to check:
    - all important variants of data values
    - different sized data/boundary conditions
    - unknown value for each data type
    - extent and non-extent
    - configuraton differences (DECIMALS, CASE-SENS...)
- confirm that:
  - field name does not matter
  - difference in code pages for character data will automatically convert
  - the same fields in a different order will fail (and what the error processing is)
  - there is no implicit type conversion (you can't write a field of int type into an int64 or decimal)
- check what happens when:
  - you read from a buffer that has no record avail
  - you write to a buffer that has no record avail
  - you reference a dynamic temp-table that is not prepared yet
  - in a copy TO a buffer, when a problem happens after the create occurs but before all the data is successfully written into the record, does any previous record get put back into the buffer?
  - there are some cases noted as TODO in the BufferImpl.rawTransfer(boolean, handle) method

The raw_transfer_test_driver.p should be extended to run the entire suite of tests non-interactively. The results can be stored in raw_transfer/results/.

#6 - 04/20/2018 11:41 AM - Greg Shah

As of branch 3507a revision 11264, the RAW-TRANSFER language statement (and the BUFFER method) has a complete "basic" implementation. It has been tested and works.

At this time it needs improvements to handle boundary conditions, to add proper error handling and the deserialization is not binary compatible with the 4GL version. Once the additional questions in #3549-5 are answered, the final implementation will not be difficult. All of the core functionality is already present.

#7 - 02/06/2019 09:50 AM - Greg Shah

- Related to Feature #3758: misc database features added

#8 - 09/05/2022 01:52 AM - Greg Shah

- Related to Feature #2137: runtime support for FIELDS/EXCEPT record phrase options added