

## Database - Bug #7473

### rewrite toString() implementation for Dataset, StaticTempTable and TempTableBuilder to be pure-sql and don't rely on WRITE-JSON

06/30/2023 06:32 AM - Constantin Asofiei

<b>Status:</b>	Review	<b>Start date:</b>	
<b>Priority:</b>	Normal	<b>Due date:</b>	
<b>Assignee:</b>	Eduard Baci	<b>% Done:</b>	100%
<b>Category:</b>		<b>Estimated time:</b>	0.00 hour
<b>Target version:</b>		<b>case_num:</b>	
<b>billable:</b>	No		
<b>vendor_id:</b>	GCD		
<b>Description</b>			

#### History

##### #2 - 06/30/2023 06:34 AM - Constantin Asofiei

Current implementation relies on WRITE-JSON, which has side-effects on the default buffer for the temp-table. This makes debugging difficult if an instance is evaluated via 'toString' in the debugger while the default-buffer is holding a record (and needs that record).

A pure-sql implementation of toString will decouple this from the default-buffer, and avoid side-effects.

##### #3 - 06/30/2023 06:43 AM - Stefan Brands

Constantin Asofiei wrote:

Current implementation relies on WRITE-JSON, which has side-effects on the default buffer for the temp-table. This makes debugging difficult if an instance is evaluated via 'toString' in the debugger while the default-buffer is holding a record (and needs that record).

A pure-sql implementation of toString will decouple this from the default-buffer, and avoid side-effects.

Can we maybe raise the priority of this ? It is really giving us a hard time while debugging...

##### #4 - 07/02/2023 06:56 AM - Stefan Brands

Is this also why looking at the content of Exceptions changes the exception ? Because we just spend an entire week debugging a problem and in the end it turned out that the problem was not in the converted code but in our webui code. But we were misled by the behavior in the debugger while looking at the exceptions.

I really think this kind of behavior in the debugger should be avoided and should be fixed. Otherwise a lot of our developers will run into this problem, especially when the entire company will switch to the converted code.

#### #5 - 07/03/2023 01:31 PM - Ovidiu Maxiniuc

Please use `DataSet.getSqlData()` and `RecordBuffer.sqlTableContent(N)` instead. At least as a temporarily workaround. They use SQL to extract the content from database so the buffer(s) is(are) not altered.

You may delegate the `toString()` to these methods since the debuggers calls `toString()` as default renderer. If your debugger supports data renderers (IntelliJ Idea does, but I am not sure about Eclipse) you may want to configure `DataSet` and `RecordBuffer` objects it to use them.

#### Important note:

In case of `_temp` tables, the content is multiplexed, the table returned contain the records from ALL those tables. This was done intentionally at the moment the method was implemented. You need to use the `_multiplex` field to filter only the record of interest.

#### #6 - 07/03/2023 03:02 PM - Constantin Asofiei

Ovidiu, that looks interesting; I think is better to have the legacy field name instead of column label or SQL field name (as for dynamic temp-tables, the SQL field name is like `field#`).

Stefan: for object FWD vars (legacy OO variables/fields in 4GL), the Java object `toString()` delegates to `toLegacyString()` - which can be overridden by the 4GL class (as this is emitted for `Progress.Lang.Object.toString()`). The point here: I'll disable this `toString` to show only the object's actual type and maybe its internal ID. Otherwise, `toLegacyString()` can have any implementation which is outside of FWD's control, and will have side-effects. If you want to drill-down into the object, you will need to look at the individual fields.

#### #7 - 07/03/2023 03:16 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

Ovidiu, that looks interesting; I think is better to have the legacy field name instead of column label or SQL field name (as for dynamic temp-tables, the SQL field name is like `field#`).

The current result served a specific goal at the moment of the implementation (debugging a feature in development at that time). Usually, it is not difficult to identify the legacy name from their SQL counterpart. Except for dynamic temp-tables, where the columns are incremented values of a pattern (`field#`). But we can improve the signature of the methods with additional parameters, beside limiting of the number of returned records we currently have:

- to use legacy column name (boolean);
- to include (or not) the hidden before-table attributes (boolean);
- to filter the multiplexed records to current buffer/table only (boolean or more complex);
- to crop or not longer character values (they are cropped now at 32 chars) (int);
- other (?)

However, since the `toString()` has no parameters by default, we need to have default values for all above. And maybe define a resource (directory or file) to configure these values the developer needs at a certain moment.

## #8 - 04/08/2024 02:48 PM - Alexandru Lungu

- Assignee set to Eduard Baciu
- Status changed from New to WIP

Ovidiu, are you aware if the WRITE-JSON was finally rewritten without side-effects? Otherwise, Eduard B. can pick it up it fix it (for both trunk and 7156b).

## #9 - 04/08/2024 03:40 PM - Ovidiu Maxiniuc

WRITE-JSON will always leave the current/working buffer empty when it is finished, causing it content to be eventually flushed. This is documented in 4GL manual.

I guess you mean to rewrite toString() method to avoid the side-effects by NOT invoking WRITE-JSON runtime implementation?

I think the easiest way of inspecting a table/buffer SQL content is to call sqlTableContent(int) on its RecordBuffer, if any is present. For temp-tables, the default buffer can be used. The result is a tabular view of the SQL table.

If JSON is preferred, then it must be implemented. In fact, only the 'rendering' is necessary because the pulling of data from SQL should be identical.

Note the implementation of dataset. Its toString() behaviour can be switched at runtime (alter fullDebugString field from debugger) to JSON/schema only. The JSON variant will eventually need to be "redirected" to use the buffer safe implementation. There is also already implemented the getSqlData() which will return the tabular variant, but not wired to toString() in any way.

Also, see [#8183](#): I created a branch for a related purpose. My idea there was to extract debugging tools like this in a separate library which should not be packaged with the release binaries which get deployed to customers.

## #10 - 04/19/2024 09:48 AM - Eduard Baciu

- % Done changed from 0 to 70

I rewrote toString function for StaticTempTable and TempTableBuilder using a StringBuilder in which i added elements(parenthesis and spaces, in addition to the information from the tables) in order to imitate the result returned by writeJson().

This is toString method from StaticTempTable which is almost identical with the one from TempTableBuilder:

```
public String toString()
{
    String myName = (name == null) ? "unnamed" : name;
    StringBuffer jsonString = new StringBuffer("\n \"" + myName + "\" : [");
    ErrorManager.ErrorHelper eh = ErrorManager.getErrorHelper();
    boolean ws = eh.isSilent();
    boolean wp = eh.isPending();
    if (!ws)
    {
        eh.setSilent(true);
    }
    try
    {
        jsonString.append(defaultBuffer.buffer().sqlTableContentJson());
        jsonString.append(" ]\n");
    }
    catch (PersistenceException | SQLException e)
    {
        jsonString.append("Error!");
    }
    finally
    {

```

```

        if (!ws)
        {
            eh.setSilent(false);
        }
        if (!wp && eh.isPending())
        {
            eh.clearPending();
        }
    }
    return "STATIC " + myName + ", ID " + getUniqueID().getValue() + "\n" + jsonString;
}

```

"sqlTableContentJson()" is a function from the RecordBuffer class written by me where the json with our informations is created.

```

public String sqlTableContentJson() throws PersistenceException, SQLException
{
    String sql = "select * from " + dmoInfo.sqlTable;

    ResultSet res = persistence.getSession().getConnection().prepareStatement(sql).executeQuery();
    ResultSetMetaData md = res.getMetaData();
    int cc = md.getColumnCount();
    StringBuilder resultSB = new StringBuilder("");
    StringBuilder sb = new StringBuilder("");
    while (res.next())
    {
        sb.append(" {\n");
        for (int k = 1; k <= cc; k++)
        {
            String label = md.getColumnLabel(k).toLowerCase();
            if (!label.startsWith("_") && !label.equals("recid"))
            {
                String value = res.getString(k);
                sb.append("    \"" + label + "\":\" + "\"" + value + "\"\n");
            }
        }
        sb.replace(sb.length() - 2, sb.length() - 1, "");
        sb.append(" }");
        resultSB.append(sb + ",");
        sb.setLength(0);
    }
    resultSB.replace(resultSB.length() - 1, resultSB.length(), "");
    return resultSB.toString();
}

```

I compared the results(WriteJson() and the current method ) and there are no differences, so now i will look at the DataSet part.

## #11 - 04/19/2024 05:31 PM - Ovidiu Maxiniuc

Yes, I think this what we need. Some advices:

- do not involve `ErrorMessage` in `toString()`: even if you use it in brackets to restore the state, this method must not alter application's internal state. Also it is not necessary, if an exception occurs, just display it instead of the JSON. It will be more useful for the developer which uses it. After all, this method should be used only when debugging/logging;
- I see you skipped `!label.startsWith("_") && !label.equals("recid")`
  - "recid" is the default value for `Session.PK`, use that instead
  - I think these values are useful for the developers and should not be skipped. For example, when using a temp table, you may want to see record from a specific instance, but the query will return the content from all tables, multiplexed. In the absence of `_multiplex` field, this is not possible. Another example: when using before-tables, the links between the before/after images are maintained using `_peerRowid` field and the record's PK. More than that, this make the the meta tables not viewable, and, although we do not encourage the customer to use fields whose name starts with underscore, they insist on doing that;
- when using `StringBuilder`:
  - try not to use constructs like: `sb.append(a + b)`. Use `sb.append(a).append(b)` instead, to avoid an intermediary `StringBuilder` being constructed;
  - I see no point in using two `StringBuilder` instances. The `resultSB` can be used directly and drop additional operations for de/allocation of `sb`;
  - avoid `replace()`, `delete()` as much as possible. Plan ahead. For example, in case of inserting comma between the elements of a list but not at the end, use something like this:

```
boolean first = true;
while (/*loop condition*/)
{
    if (first)
        first = false;
    else
        sb.append(", ");
    sb.append(/*next item in list*/)
}
```

## #12 - 04/22/2024 09:01 AM - Eduard Baci

- % Done changed from 70 to 100

- Status changed from WIP to Review

I modified the code following your advices and managed to implement "getDataAsJson" method for "DataSet" as well.

toString method for StaticTempTable:

```
public String toString()
{
    String myName = (name == null) ? "unnamed" : name;
    StringBuffer jsonString = new StringBuffer("{\n  \"" + myName + "\" : [");
    try
    {
        jsonString.append(defaultBuffer.buffer().sqlTableContentJson());
        jsonString.append(" ]\n");
    }
    catch (PersistenceException | SQLException e)
    {
        jsonString.append("Error getting data: ").append(e.getMessage()).append("\n");
    }
    return "STATIC " + myName + ", ID " + getUniqueID().getValue() + "\n" + jsonString;
}
```

getDataAsJson method for DataSet:

```
private String getDataAsJson()
{
    StringBuilder sb = new StringBuilder();
    sb.append("{\n").append("  \"").append(name).append("\").append(": {");
    for (int i = 0; i < buffers.size(); i++)
    {
        try
        {
            BufferImpl buffer = buffers.get(i);
            String tableName = buffer.buffer().getDmoInfo().legacyTable;
            String res = buffer.buffer().sqlTableContentJson();
            sb.append("\n").append("    \"").append(tableName).append("\": [").append(res);
            if (i == buffers.size() - 1)
            {
                sb.append(" ]\n");
            }
            else
            {
                sb.append(" ],");
            }
        }
        catch (PersistenceException | SQLException e)
        {
            sb.append("Error getting data: ").append(e.getMessage()).append("\n");
        }
    }
    sb.append(" }\n");
    return sb.toString();
}
```

sqlTableContentJson method from RecordBuffer:

```
public String sqlTableContentJson() throws PersistenceException, SQLException
{
    String sql = "select * from " + dmoInfo.sqlTable;
    ResultSet res = persistence.getSession().getConnection().prepareStatement(sql).executeQuery();
    ResultSetMetaData md = res.getMetaData();
    int cc = md.getColumnCount();
    StringBuilder resultSB = new StringBuilder("");
    while (res.next())
    {
        resultSB.append(" {\n");
        for (int k = 1; k <= cc; k++)
        {
            String label = md.getColumnLabel(k).toLowerCase();
            String value = res.getString(k);
            resultSB.append("    \"").append(label).append("\": ").append("\"").append(value).append("\"");
        }
        ;
        if (k != cc)
        {
            resultSB.append(",\n");
        }
    }
}
```

```
        resultSB.append(", \n");
    }
    else
        resultSB.append("\n");
    }
    resultSB.append(" }");
    if (!res.isLast())
    {
        resultSB.append(", ");
    }
    }
    return resultSB.toString();
}
```

Thanks for the advices!