Database - Feature #1992

add support for double-colon (::) syntax

02/13/2013 09:53 AM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Conversion Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

History

#1 - 02/13/2013 10:03 AM - Eric Faulhaber

The double-colon (::) syntax is a shorthand for extracting information from a handle to a buffer, query, or ProDataSet (not in scope). See the ":: Punctuation" section in the ABL Reference.

So, this:

```
myBufferHandle::customer-number
```

is equivalent to:

```
myBufferHandle:buffer-field("customer-number"):buffer-value
```

Probably, the best way to handle this is to expand the former form to the latter form early in the annotations phase of conversion, very similar to what we do with the INSERT statement. Then, conversion will occur naturally on the expanded form.

We're not sure how the query form of this works, but it does not appear to be in use.

#2 - 02/13/2013 12:51 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

#3 - 02/13/2013 01:21 PM - Greg Shah

From Ovidiu:

```
The :: (DB_REF_NON_STATIC) is declared and used there but seems to be faulty as it is not recognized by lexer
at all.
```

Please provide more details of your finding. DB_REF_NON_STATIC is definitely a top level lexer rule and I see these nodes in ASTs, so it seems to work for me.

#4 - 02/14/2013 12:47 PM - Ovidiu Maxiniuc

By chance, I used the NAME as the field to be evaluates like: bufh::name.
Here is the error:

```
line 7:5: unexpected token: ::
    at com.goldencode.p2j.uast.ProgressParser.display_stmt(ProgressParser.java:25818)
    at com.goldencode.p2j.uast.ProgressParser.stmt_list(ProgressParser.java:21182)
    at com.goldencode.p2j.uast.ProgressParser.statement(ProgressParser.java:5299)
    at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:4273)
    at com.goldencode.p2j.uast.ProgressParser.block(ProgressParser.java:4036)
    at com.goldencode.p2j.uast.ProgressParser.external_proc(ProgressParser.java:3962)
    at com.goldencode.p2j.uast.AstGenerator.parse(AstGenerator.java:1416)
...
```

Looking into the lexer intermediary file I notice that the lexer identified it as <KW_NAME> instead of a normal <SYMBOL>. I am investigating fixing this and allowing fields that are named with progress keywords to be parsed (in this case NAME).

For the converted code I was thinking/investigated implementing directly the :: as a member function (dynamicFieldValue) directly in handle object.

#5 - 02/15/2013 09:57 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> For the converted code I was thinking/investigated implementing directly the :: as a member function (dynamicFieldValue) directly in handle object.

This shouldn't be a member of handle.

But before we finalize an approach, please write some test cases to see exactly what types of information/properties can be referenced in this way, from buffer, query, and even ProDataSet handles. We don't want to limit our design to work with buffer handles only.

An important part of these tests is to determine how chaining works. We understand that it's possible with ProDataSet, for instance, to dereference the buffer handle of a temp-table using this syntax, and then dereference a field within that buffer. What are the limits of this chaining? Is anything like this possible with other handle types (e.g., query)? What types of properties can be dereferenced from a query handle? A buffer handle?

Once we have an idea what the limits/capabilities of this syntax are, we can make sure we are designing the right conversion approach.

#6 - 02/16/2013 04:40 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Ovidiu Maxiniuc wrote:
>
>> For the converted code I was thinking/investigated implementing directly the :: as a member function (dynamicFieldValue) directly in handle object.
>
> This shouldn't be a member of handle.

I should elaborate. Once we know exactly the types of properties that can be referenced this way from a buffer, query, and ProDataSet handle, we can determine what interface(s) we need to create to represent these resources. We then treat this like any other handle.unwrapXXXX situation.

#7 - 02/18/2013 01:13 PM - Ovidiu Maxiniuc

Taking into consideration a dataset handle dh that contains a buffer handle bh (named bh-name) with a character field named Name, all the below forms are legal in OE10 and refer the same character (in the particular case) value:

```
bh::Name
dh::bh-name:BUFFER-FIELD("Name"):BUFFER-VALUE()
bh:BUFFER-FIELD("Name"):BUFFER-VALUE()
dh:GET-BUFFER-HANDLE("bh-name"):BUFFER-FIELD("Name"):BUFFER-VALUE()
dh:GET-BUFFER-HANDLE("bh-name")::Name
dh::bh-name::Name
```

As you can see all kind of chaining is possible as long as they make sense. From my research:

- ProDataSet handle name cannot be in the right side of the :: because this is a top-level container.
- buffers can occur both in left side of the :: as a container for a named field and also in the right as a member of a dataset.
- named fields can only appear in the right side of the ::.
- only tables/buffers can appear after :: of a dataset (12785: "table not found in dataset" warning is issued otherwise).
- only buffer-fields can appear after :: of a buffer (7351: "buffer-field was not found in buffer warning is issued otherwise).
- I found no way to access queries using ::. They are unnamed, not contained into dataset, don't have named fields.

In summary, if :: is applied to a dataset it will return a buffer handler on which you can apply a method, or access an attribute or even chained another ::. If :: is applied to a buffer it will simply return the value of field specified.

The progress.g expects chaining of only one kind of punctuation and the parser it's eager to consume only : or ::. This will result in an unexpected token exception:

```
line 43:43: unexpected token: ::
    at com.goldencode.p2j.uast.ProgressParser.display_stmt(ProgressParser.java:25818)
    at com.goldencode.p2j.uast.ProgressParser.stmt_list(ProgressParser.java:21182)
    at com.goldencode.p2j.uast.ProgressParser.statement(ProgressParser.java:5299)
    at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:4273)
    at com.goldencode.p2j.uast.ProgressParser.then_clause(ProgressParser.java:32380)
    at com.goldencode.p2j.uast.ProgressParser.if_stmt(ProgressParser.java:26624)
    at com.goldencode.p2j.uast.ProgressParser.stmt_list(ProgressParser.java:21242)
    at com.goldencode.p2j.uast.ProgressParser.statement(ProgressParser.java:5299)
    at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:4273)
    at com.goldencode.p2j.uast.ProgressParser.block(ProgressParser.java:4036)
    at com.goldencode.p2j.uast.ProgressParser.external_proc(ProgressParser.java:3962)
```

The lvalue rule in grammar should not expect chaining but treat this more like a function composition.

Another issue is the scoped field name. I believe that a new lexical item should be defined for the right side of the ::. For the particular cases when the right side is in conflict with a reserved word. This token can be generared at runtime, for example:

```
    th:TEMP-TABLE-PREPARE("bh" + "-name").
```

so it cannot be 'gathered' during the process of the conversion. I'm thinking of letting the lexer handle this as a keyword (if the case) and in early stages of the conversion, maybe in parser, to change the token type to <symbol> if it appears just after the ::.
I need some guidance here, especially because the changes in progress.g in a rather sensible places: lvalue/SYMBOL.

#8 - 02/19/2013 12:58 PM - Ovidiu Maxiniuc

First, as I wrote above in notes 4 and 7, the second operand can be almost anything (as it is the name of a field from a table or the name of a buffer) including keywords. My question is, how can I capture the next token, whatever type it is?
I intend to change it's type to SYMBOL or something that can be further processed.

The second problem is the chaining. As you can see from my experiments, yesterday I noticed that practically, any combination of :: and : could be valid.
I intend to alter the chained_object_members_core rule so that chaining could be done with either :: or :, the difference being in an annotation that will be used later on to create the correct java call. Is this a good way to implement this ?

#9 - 02/19/2013 01:23 PM - Greg Shah

> First, as I wrote above in notes 4 and 7, the second operand can be almost anything (as it is the name of a field from a table or the name of a buffer) including keywords. My question is, how can I capture the next token, whatever type it is?
> I intend to change it's type to SYMBOL or something that can be further processed.

This:

```
         | DB_REF_NON_STATIC^ SYMBOL
           (
              options { generateAmbigWarnings = false; }
              :
              DB_REF_NON_STATIC! SYMBOL
           )?
```

can be changed to this:

```
         | DB_REF_NON_STATIC^ symbol
           (
              options { generateAmbigWarnings = false; }
              :
              DB_REF_NON_STATIC! symbol
           )?
```

Please see the symbol rule (around line 25642). It matches any SYMBOL or unreserved keyword and it returns a node that is set to type SYMBOL. This is exactly what you are looking for.

> I intend to alter the chained_object_members_core rule so that chaining could be done with either :: or :, the difference being in an annotation that will be used later on to create the correct java call. Is this a good way to implement this ?

I don't think an annotation would be needed here. The tree structure itself (with the COLON or DB_REF_NON_STATIC) as the "operator" will tell us what we need.

BUT for now, this is beyond the scope of the task. The current project has no chaining needed. Instead of trying to fix this (which will be tricky). Please put documentation into progress.g in 2 places:

1. In the "* Open TODOs/Issues:" part of the class javadoc.
2. In the lvalue rule where the DB_REF_NON_STATIC is matched.

Make sure it is clear that the chaining cases are not fully implemented and in particular, the mixing of COLON and DB_REF_NON_STATIC (which is valid in the 4GL) is not supported yet.

#10 - 02/20/2013 10:28 AM - Eric Faulhaber

OK, so without chaining, the scope of this feature should be much smaller. As you suggested earlier, let's go with a runtime function approach (let's call it dereference(String)). That method should be in an interface that is extended by both Buffer and P2JQuery (and eventually ProDataSet, but not yet). So, let's create a new interface (say, Dereferenceable) that both both Buffer and P2JQuery extend. That interface has the single dereference method, which both BufferImpl and AbstractQuery implement. We then add an unwrapDereferenceable method to handle. We also need conversion rules which convert instances of:

```
buffer-handle::some-name
```

and

```
query-handle::some-name
```

to

```
handle.unwrapDereferenceable({buffer|query}Handle).dereference("someName")
```

much in the same way as we do for methods and attributes. Make sense?

#11 - 02/20/2013 11:19 AM - Ovidiu Maxiniuc

I already have an implementation here but, reading your previous note, made me turn back to OE installation.
The reason is that :: does not work with Query objects. At least not in my test.
Doing something like:

```
CREATE TEMP-TABLE th.
th:ADD-FIELDS-FROM("Customer").
th:TEMP-TABLE-PREPARE("bh-name").
bh = th:DEFAULT-BUFFER-HANDLE.

CREATE QUERY qh.
qh:SET-BUFFERS(bh).

CREATE DATASET dh.
dh:ADD-BUFFER(bh).

DISPLAY dh::bh-name.
DISPLAY qh::bh-name qh::name.
```

The very last line will fail twice at runtime:

```
**? is not a queryable attribute for QUERY widget. (4052)
**? is not a queryable attribute for QUERY widget. (4052)
```

Is there a reason why you mention the QUERY related to :: ?

#12 - 02/20/2013 12:13 PM - Greg Shah

Some thoughts:

1. Queries are explicitly mentioned as supported by :: in the 4GL language reference. It wouldn't be the first time the docs were wrong, but I suspect that it is supported in some way. Is "name" a field name of Customer?

2. In regard to Eric's point about the target result:

```
handle.unwrapDereferenceable({buffer|query}Handle).dereference("someName")
```

I think this is slightly off. The string should not be "someName" (a converted name), it should be the legacy "some-name" so that we can handle any ambiguity at runtime. I'm sure that if we tried for a static name, that it would not be 100% determined at conversion-time, since the handle could be used for many different tables, which could have common names.

3. Use the methods_attributes.rules to add the handle unwrapping for this case. This will be especially important since eventually we will have to properly chain : and :: anyway.

4. The emitting of the SYMBOL as a String literal can be easily done in literals.rules.

#13 - 02/20/2013 12:39 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Some thoughts:
>
> 1. Queries are explicitly mentioned as supported by :: in the 4GL language reference. It wouldn't be the first time the docs were wrong, but I suspect that it is supported in some way. Is "name" a field name of Customer?

Yes, seems that they (docs) are wrong again. I did the tests and the result is that QUERY is not supported by ::. I also had the idea of obtaining the query using its name (putting it at the right of the :: punctuation). Yes, name is a field in table Customer, DISPLAY bh::NAME. prints the current field content.

> 2. In regard to Eric's point about the target result:
> [...]
> I think this is slightly off. The string should not be "someName" (a converted name), it should be the legacy "some-name" so that we can handle any ambiguity at runtime. I'm sure that if we tried for a static name, that it would not be 100% determined at conversion-time, since the handle could be used for many different tables, which could have common names.

This could be one of the hardest things to implement. If the field name has changed during the conversion process it might be impossible to lookup the actual field name as there may exist multiple tables with the same field name and converted different. Am I wrong?
Also, dereference() returns a BaseDataType. This is because at conversion time I cannot predict what kind of result :: will return. It can be a POLY (BUFFER-VALUE) or can be another handle to a Buffer.

> 3. Use the methods_attributes.rules to add the handle unwrapping for this case. This will be especially important since eventually we will have to properly chain : and :: anyway.

At this moment I extracted the converting code for :: into a separate dereference.rules file. Should I merge it into methods_attributes.rules ?

> 4. The emitting of the SYMBOL as a String literal can be easily done in literals.rules.

I do this inside dereference.rules when building the small deference tree.

#14 - 02/20/2013 01:50 PM - Greg Shah

> This could be one of the hardest things to implement. If the field name has changed during the conversion process it might be impossible to lookup the actual field name as there may exist multiple tables with the same field name and converted different. Am I wrong?

For dynamic database we are going to need full mapping information for legacy <--> converted schema names, so we will already have it available.

At runtime, dereference() will take the legacy name and the resource being accessed and use that resource's mapping info to determine the target converted name. Then it will have to obtain that value and return it. It should not be so bad, considering that the mapping data will already be there.

> Also, dereference() returns a BaseDataType.

Yes, that is correct.

> At this moment I extracted the converting code for :: into a separate dereference.rules file.

That is fine. Leave it for now.

#15 - 02/20/2013 01:59 PM - Ovidiu Maxiniuc

- File om_upd20130220b.zip added

- File om_upd20130220a.zip added

Here is the implementation and some testcase.
I'm afraid this is not built on the last source-code, I will work on that tomorrow, but you can make an idea of how the current implementation is. In fact there are two of them:

- in the first (disabled in dereference.rules by a AND FALSE in rule) approach I tried to resolve simple derefs and chaining in two separate static methods in handle - this code will be removed.
- in the second the chaining is done at conversion time. Because the dereference() returns a base type, a cast needs to be inserted in between. The converted code looks like this:

```
handle.unwrapDereferenceable(dh).dereference("name");
handle.unwrapDereferenceable((handle) handle.unwrapDereferenceable(dh).dereference("bh-name")).dereference
("NAME");
```

#16 - 02/21/2013 09:51 AM - Constantin Asofiei

Greg, Ovidiu found an interesting case of BUFFER-VALUE usage (which applies to any ATTR_POLY or METH_POLY). If is used in a statement like:

```
def var h as handle.
display h:buffer-value.
```

conversion fails because, at the time the frame definition is built, the type of this attribute is not fixed (can be any value). In 4GL, it seems like if such *_POLY cases are used in a frame (DYNAMIC-FUNCTION case included), the associated widget is built as a 8-char widget. From this, there are at least two questions for which we should know the answer, for the current project:

1. are there any ATTR_POLY/METH_POLY used directly in a frame-related statement?
2. in assignment case, when the rvalue is a ATTR/METH_POLY, are there cases when the lvalue is of a different type than the type of the value returned by the rvalue?

#17 - 02/21/2013 09:57 AM - Ovidiu Maxiniuc

Yes, this is the case for the ::, too, as the final attribute in the rewritten form is buffer-value() attribute when applied to a buffer/temp-table.

#18 - 02/22/2013 02:57 PM - Greg Shah

> are there any ATTR_POLY/METH_POLY used directly in a frame-related statement?

In regard to the FUNC_POLY/ATTR_POLY/METH_POLY usage, here is what I have found:

```
APPLY IF some-logical-expr THEN LASTKEY - 32 ELSE LASTKEY.

DISPLAY (IF some-logical-expr THEN an-expr ELSE other-expr).

MESSAGE "text" SKIP my-field-handle:buffer-value.
MESSAGE "blah" SKIP dynamic-function("func-name" IN TARGET-PROCEDURE).

DISPLAY (ACCUM TOTAL some-expr).
```

I suspect we already handle these cases.

#19 - 02/22/2013 03:00 PM - Greg Shah

> in assignment case, when the rvalue is a ATTR/METH_POLY, are there cases when the lvalue is of a different type than the type of the value returned by the rvalue?

I'm not sure I understand what is being asked here. By definition, the ATTR/METH_POLY type is not easily known. But since it will be assigned using <wrapper_class>.assign(BaseDataType), it seems that these cases should work naturally.

#20 - 02/22/2013 03:09 PM - Constantin Asofiei

All the specified cases work nicely.

> I'm not sure I understand what is being asked here. By definition, the ATTR/METH_POLY type is not easily known. But since it will be assigned using <wrapper_class>.assign(BaseDataType), it seems that these cases should work naturally.

The problem is not at conversion time, but at runtime. In 4GL, following is valid:

```
def var i as int.
function func0 returns char.
return "1000".
end.

/* i = func0(). this is compile-time error */
i = dynamic-function("func0"). /* this executes and sets i to 1000 */
message i.
```

Idea is, if we have the ATTR/METH_POLY in an expression or as a rvalue, then 4GL does some automatic conversion of the "poly" rvalue to the lvalue's expected type. I know that for DYNAMIC-FUNCTION we disambiguate the return type based on the function's real return type, but this case that the lvalue can be assigned a not-compatible type.

#21 - 02/23/2013 10:48 AM - Greg Shah

> Idea is, if we have the ATTR/METH_POLY in an expression or as a rvalue, then 4GL does some automatic conversion of the "poly" rvalue to the lvalue's expected type. I know that for DYNAMIC-FUNCTION we disambiguate the return type based on the function's real return type, but this case that the lvalue can be assigned a not-compatible type.

It seems like we will have to do the following:

1. Test the limits/capabilities of the 4GL polymorhpic return transformations, especially in regard to assignments. I suspect there are real limits in what silent conversions are allowed.
2. Determine our exposure and plan for a resolution. My hope is that the 4GL has a standard set of conversions and we can implement those in each wrapper's constructor that takes a BaseDataType (which you recently added for dynamic-function return values). Then we can wrapper any assignment of a *_POLY rvalue with the proper constructor.

Am I missing anything important here? My plan would be to do this for milestone 7 but not for milestone 4.

#22 - 02/23/2013 10:55 AM - Constantin Asofiei

For milestone 4, I don't think there is anything we need to do about the *_POLY rvalues, all ~~will~~ can be in the runtime (LE: I was thinking to hide all this .assign(BaseDataType) APIs).

#23 - 02/23/2013 11:11 AM - Greg Shah

OK, please add a suitable task to the Base Language project and target it to milestone 7. Please put all the relevant details there as needed.

#24 - 02/25/2013 06:19 AM - Ovidiu Maxiniuc

- File om_upd20130225a.zip added

- File om_upd20130225b.zip added

Latest :: conversion merged with bzr revision 10199 and updated testcase.

#25 - 02/25/2013 10:43 AM - Eric Faulhaber

Code review 20130225a:

Nice work! I am running it through conversion regression testing now.

There was one change to progress.g that seemed unnecessary: after discussion with Greg, I removed the actions in the lvalue rule where you explicitly set the token type to symbol ({ #sym1.setType(SYMBOL); }). These appear to be redundant with what the reserved_or_symbol already does. Your test case seemed to convert OK without these.

#26 - 02/25/2013 10:55 AM - Ovidiu Maxiniuc

> There was one change to progress.g that seemed unnecessary: after discussion with Greg, I removed the actions in the lvalue rule where you explicitly set the token type to symbol ({ #sym1.setType(SYMBOL); }). These appear to be redundant with what the reserved_or_symbol already

does. Your test case seemed to convert OK without these.

I just wanted to be sure, just in case that reserved_or_symbol will set some other dedicated type in the future.

On the other hand, the current implementation uses recursivity to allow more that two levels of dereference. However, at this moment the parser does not allow it but, from my investigations, the parsing of :: can be loosen to allow the left-side of the punctuation to be any handle-type expression, and the code should be converted correctly. I am not sure if this was too explicitly documented in the sources.

#27 - 02/25/2013 11:38 AM - Eric Faulhaber

- File ecf_upd20130225a.zip added

Your update (including my change to progress.g) has passed conversion regression testing. I am attaching my updated progress.g version.

If you feel you need to clarify the documentation, please do so, but **do not** change any of the logic/code at this point. Then please commit your update to bzr ASAP as rev 10201, distribute the updated archive to the team, and post the final update and revision number here. Thanks.

#28 - 02/25/2013 12:35 PM - Ovidiu Maxiniuc

I have chosen not to alter in any way the update that have passed the testing (om_upd20130225a.zip) and committed it to bzr as revision 10201 and then distributed it to team.

#29 - 02/25/2013 01:01 PM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#30 - 11/16/2016 11:07 AM - Greg Shah

- Target version changed from Milestone 4 to Conversion Support for Server Features

Files

om_upd20130220a.zip	281 KB	02/20/2013	Ovidiu Maxiniuc
om_upd20130220b.zip	1.01 KB	02/20/2013	Ovidiu Maxiniuc
om_upd20130225a.zip	304 KB	02/25/2013	Ovidiu Maxiniuc
om_upd20130225b.zip	1.16 KB	02/25/2013	Ovidiu Maxiniuc
ecf_upd20130225a.zip	265 KB	02/25/2013	Eric Faulhaber