# Base Language - Bug #11897

## Can't resolve constructor requiring integer parameter with an int64 (eventually constant) argument

09/21/2026 04:43 AM - Alexandru Lungu

| | | | |
|---|---|---|---|
| **Status:** | Review | **Start date:** | |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | Artur Şcolnic | **% Done:** | 0% |
| **Category:** | | **Estimated time:** | 0.00 hour |
| **Target version:** | | | |
| **billable:** | No | **reviewer:** | Constantin Asofiei |
| **vendor_id:** | GCD | **production:** | No |
| **case_num:** | | **env_name:** | |
| **version_reported:** | | **topics:** | |
| **version_resolved:** | | | |

**Description**

**Related issues:**

| | |
|---|---|
| Related to Base Language - Bug #10004: NativeInvoker fails if an argument is ... | **WIP** |
| Related to Base Language - Bug #9060: Cache oftenly used values that are immu... | **Internal Test** |

### History

#### #1 - 09/21/2026 04:43 AM - Alexandru Lungu

*- Related to Bug #10004: NativeInvoker fails if an argument is of type integerConstant added*

#### #2 - 09/21/2026 04:43 AM - Alexandru Lungu

*- Related to Bug #9060: Cache oftenly used values that are immutable added*

#### #3 - 09/21/2026 05:16 AM - Alexandru Lungu

*- Subject changed from Can't resolve constructor requiring integer parameter with an int64 argument to Can't resolve constructor requiring integer parameter with an int64constant argument*

## Constructor look-up

I encountered the following issue with int64constant. In ControlFlowOps.resolveLegacyEntry, used when looking-up a suitable constructor when creating a new legacy object, there is:

```
if ((int64.class == sigType     && decimal.class == candidateType)     ||
                      (integer.class == sigType   && (decimal.class == candidateType     ||
                                                      int64.class == candidateType))     ||
                      (character.class == sigType && longchar.class == candidateType)    ||
                      (datetime.class == sigType  && datetimetz.class == candidateType)  ||
                      (date.class == sigType      && (datetime.class == candidateType    ||
                                                      datetimetz.class == candidateType)))
```

This doesn't account for int64constant or other *constant variants. There is a previous args[i] = normalizeType(arg, sigType, candidateType);, but I am not entirely sure it it is the responsibility of this method to resolve the *constant variants ... it may be, but currently it is not.

The code block above is from the INPUT mode, but OUTPUT mode has its own such conditional. Example to replicate:

```
def var value as int.
def var instance as MyClass.
instance = new MyClass(input value + 1).
```

```
class MyClass:

   constructor public MyClass (input p as integer):

   end.

end.
```

PS: The thingy here is that value + 1 yields **int64constant** instead of **integer**.

```
   public static int64 plus(integer op1, int op2)
   {
      return plus((int64) op1, int64.of((long) op2));
   }
```

from MathOps. I guess it is correctly **that it returns int64**, because it may overflow and OE is silently promoting the expression to int64 to hold the result.

## Audit the use of constants

I think we discussed in #9060 about auditing all places where syntax like <BDT>.class is used in order to upgrade them to account for *constant variants. This task is the second example where this audit doesn't seem sufficient; #10004 is the first one.

Artur, please leverage AI capability to generate a report on FWD of places where there are still <BDT>.class syntax which doesn't include *constant variants yet. Consider building a unit test suite in the process.

PS: Please mind the BaseDataTypeConstant decorator to simplify integration.

#### #4 - 09/21/2026 05:16 AM - Alexandru Lungu

*- Status changed from New to WIP*

*- Assignee set to Artur Școlnic*

#### #6 - 09/21/2026 05:21 AM - Alexandru Lungu

*deleted*

#### #7 - 09/21/2026 05:33 AM - Alexandru Lungu

Never mind, I think this is not about the int64constant. It is resolved by calculateType.
The if conditional however is too strict for #11897-3. Apparently, you can use an int64 argument for an integer parameter on constructor call as long as it is enough to fit.

#### #8 - 09/21/2026 05:54 AM - Alexandru Lungu

*- Subject changed from Can't resolve constructor requiring integer parameter with an int64constant argument to Can't resolve constructor requiring integer parameter with an int64 (eventually constant) argument*

Indeed, the problem on this is that there is no implicit coercion assumed by the resolveLegacyEntry method. One can use an int64 argument for an integer parameter. It is not consider a "narrowing".

| Argument | Ctor1 | Ctor 2 overload | Ctor chosen | FWD |
|---|---|---|---|---|
| int64 | integer | int64 | int64 | Good |
| integer | integer | int64 | integer | Good |
| integer + 1 | integer | int64 | integer **or** Value 2147483648 too large to fit in INTEGER. (15747) on overflow | plus always returns int64, the int64 ctor is always picked |
| int64 + 1 | integer | int64 | int64 | Good |
| integer | integer | - | integer | Good |
| integer | int64 | - | int64 (widen) | Good (we have widening works) |
| int64 | integer | - | Parameter 1 for CONSTRUCTOR Cls is not type compatible with its definition. (12905) or Cannot use NEW statement with class 'Cls' because no constructor found with matching signature. (13840) on overload | Ambiguous runtime method call. Could not resolve <class-name> reference. (13844) is thrown in FWD. |
| int64 | int64 | - | int64 | Good |

My concern is with the 3rd row. I think MathOps shall return int64, but the **actual instance** should be either int64.of() or integer.of() depending whether it actually fits within bounds. The type coercion is not done by the arithmetic operator, but by the function invocation / ctor lookup / assignment / etc.

This task is about a regression because I am most certain the tests I am running now were successful at some point. Thus, this is why I may be biased into relating it with #9060.

#### #11 - 09/21/2026 10:31 AM - Artur Școlnic

*- Assignee changed from Artur Școlnic to Eric Faulhaber*

*- vendor_id deleted (GCD)*

The issues are confirmed with test cases, also THIS-OBJECT(i + 1) / SUPER(i + 1)  results in the same error, working on the fixes.

#### #12 - 09/21/2026 10:32 AM - Artur Școlnic

*- vendor_id set to GCD*

*- Assignee changed from Eric Faulhaber to Artur Școlnic*

#### #13 - 09/21/2026 07:43 PM - Greg Shah

*- reviewer Constantin Asofiei added*

#### #14 - 09/22/2026 09:00 AM - Artur Școlnic

*- Status changed from WIP to Review*

In the 4GL, intVar + 1 is still an integer expression, so it should pick a constructor's integer overload — but FWD's arithmetic always produces an int64, so the int64 overload was chosen instead. This happened in two places: a NEW whose argument was written with an explicit INPUT keyword, and THIS-OBJECT/SUPER constructor chaining. Conversion now tags those arguments with their original integer type, so the right constructor is picked, and a new 29-test slice covers both cases — green on OpenEdge 12.8 and on FWD with 11897a.

Constantin, please review 11897a.