Base Language - Feature #1621

implement dynamic/unspecified EXTENT support

10/21/2012 09:28 AM - Greg Shah

Status:	Closed	**Start date:**	02/04/2013
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	16.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #1620: implement EXTENT parameter support	**Closed**	**02/04/2013**	
Related to Base Language - Bug #2133: fix precision for decimal, dynamic-exte...	**Closed**	**01/21/2014**	**01/27/2014**

History

#1 - 10/31/2012 01:44 PM - Greg Shah

- Target version set to Milestone 7

#2 - 02/04/2013 08:29 AM - Greg Shah

In Openedge 10.x, variables (and fields too?) can be defined as an indeterminate size that is "fixed" at runtime. This can be done for normal extent variable definitions (DEF VAR) as well as for extent parameters (see #1620). This support is implemented in var/parm definition statements and in the new EXTENT statement (not the function), which is what is used to set (and sometimes unset) the size.

#3 - 02/04/2013 08:30 AM - Greg Shah

- Assignee set to Constantin Asofiei

- Target version changed from Milestone 7 to Milestone 4

- Start date changed from 10/21/2012 to 02/04/2013

#4 - 02/04/2013 01:37 PM - Constantin Asofiei

Is there any case in P2J where something like i = v[1] where v is an extent variable gets converted to the Java equivalent i.assign(v[0]), without a ArrayAssigner.subscript wrapper? The idea is, for such dynamic-extent cases, we will need to always use the ArrayAssigner APIs to access/alter an element of an extent variable.

From testing, 4GL allows only variables to be dynamic-extent (at least for temp-table fields is not possible to set a field without the extent size, and the compile error message states explicitly that only variables can be used).

For the EXTENT statement, we will not be able to resize the array in place, thus we will need to convert it to something like v = ArrayAssigner.resize(v, new-size).

When passing a dynamic-extent variable to a fixed-size extent parameter (in a procedure or function call), 4GL validates the parameters on runtime and if incorrect a message is shown, but no ERROR condition is raised (this message is not shown if the NO-ERROR clause is present). We will have problems with the simple function calls (as they convert to Java method calls); in this case, we will need to validate the parameter in the function's body or init method and if validation fails, will not execute it. We can do this for procedures too (to have a common approach for both), but the problem with this approach is that we may end up in a case when the EXTENT parameter is invalid and its error message is not shown, as another parameter (listed after the extent parameter) was invalid and showed its error too soon (the correct error message being the one for the EXTENT parameter). Considering that simple function calls validate the type of the parameters at compile time, we can use the approach described and needed by #1971 (the function's worker versions) and validate the extent parameter inside Block.init only when this is a simple function call; for all other dynamic calls (both function and procedures), the validation will be done at ControlFlowOps, to validate all parameters in proper order.

More, it looks like when having dynamic-extent parameters, any EXTENT statement call within the procedure/function will fail at runtime, if the received extent array is fixed size or had its size set with the EXTENT statement:

```
procedure proc0.
   def output param p as int extent.
   extent(p) = 2. /* this fails as extent size is already set, regardless to what value was set.*/
end.
def var i as int extent.
extent(i) = 10.
run proc0(output i).
def var j as int extent 10.
run proc0(output j).
```

A peculiarity about passing dynamic-extent variables to fixed-size extent parameters is that, if the var's dynamic-extent size is not yet set and the parameter is in OUTPUT mode, then after the call the dynamic-extent variable is changed and its size set to the parameter's extent size:

```
procedure p0.
  def output param j as int extent 5.
  j = 10.
end.
def var i as int extent.
run p0(output i).
message extent(i) i[1] i[2] i[3] i[4] i[5]. /* this will display 5 10 10 10 10 */
```

More, if the parameter is dynamic-extent, the passed variable is dynamic-extent too but it hadn't had its extent size set yet, the following will set it:

```
procedure p0.
  def output param j as int extent.
  extent(j) = 5.
  j = 10.
end.
def var i as int extent.
run p0(output i).
message extent(i) i[1] i[2] i[3] i[4] i[5]. /* this will display 5 10 10 10 10 */
```

This shows that we might not be able to use Java array references in such cases, as array's length in Java is immutable. Thus, I think we will need to add our own Array implementation or, in cases when extent variables are passed to parameters, we should use a similar approach as when fields are passed: emit an i.e. IntegerVariable instance which takes as parameters the variable's name and the ExternalProgram.this instance (IntegerVariable will work similar to how IntegerField works).

#5 - 02/04/2013 01:40 PM - Constantin Asofiei

Later edit for this statement:

> or, in cases when extent variables are passed to parameters, we should use a similar approach as when fields are passed: emit an i.e. IntegerVariable instance which takes as parameters the variable's name and the ExternalProgram.this instance (IntegerVariable will work similar to how IntegerField works).

I don't think the IntegerVariable approach will work, as this will not be compatible with the actual parameter type (as the parameter is an array, it will expect to receive an array).

#6 - 02/04/2013 01:45 PM - Constantin Asofiei

- Status changed from New to WIP

#7 - 02/05/2013 04:08 PM - Greg Shah

```
Is there any case in P2J where something like i = v[1] where v is an extent variable gets converted to the Jav
a equivalent i.assign(v[0]), without a ArrayAssigner.subscript wrapper?
```

Not for variables anymore. See convert/variable_references.rules and search for "lbracket". There is some code there to decrement the value of the num_literal and emit it, BUT this is only used for fields now. If I recall properly, the issue is that there is a need to duplicate the error processing that comes from the range check on the subscript index. Now, with dynamic extents, this requirement is even more important.

```
For the EXTENT statement, we will not be able to resize the array in place, thus we will need to convert it to
 something like v = ArrayAssigner.resize(v, new-size).
```

This makes sense.

```
Considering that simple function calls validate the type of the parameters at compile time, we can use the app
roach described and needed by #1971 (the function's worker versions) and validate the extent parameter inside
 Block.init only when this is a simple function call; for all other dynamic calls (both function and procedures
), the validation will be done at ControlFlowOps, to validate all parameters in proper order.
```

OK.

```
I don't think the IntegerVariable approach will work, as this will not be compatible with the actual parameter
 type (as the parameter is an array, it will expect to receive an array).
```

Yes, this is a tricky problem. The FieldReference works because we can functionally delegate the set/get. In this case, there is a Java reference in the calling context to set. I do prefer to continue using Java arrays directly as much as possible. This means that there would have to be code in the caller that does the delegated assignment. We could promote the var to be a member and emit an inner class (like a validation expression) that handles assigning the real reference (on successful exit from the function/procedure). Perhaps in this case (an output extent parameter), we must use a class like "OutputExtentParm" which can be constructed taking an instance of the inner class and possibly the array reference itself (for the INPUT-OUTPUT case). Then the calling code would unpack the reference if needed, use it, then assign back the local changes upon exit using methods in the OutputExtentParm. This would change the type of the method, but the changes are less intrusive than implementing a completely new wrapper type (Array).

#8 - 02/06/2013 11:33 AM - Constantin Asofiei

Once a dynamic-extent array has its size set using the EXTENT statement, a second EXTENT statement on that array will fail. Thus, I think we can leave uninitilized dynamic-extent arrays to be set to length 0 (i.e. new integer[0]), and ArrayAssigner.resize will check if the array's length is zero or not, before doing the job. This is good news because if the size was really dynamic, then things would have got more complex. And I have a feeling that for OUTPUT extent parameters, if there is an EXTENT statement call for that parameter (or that parameter is fully assigned to something else using outputExtentParam = extentVar), we will have to call something like OutputExtentParam.set(extentParam) after each statement, to let your container know the new array to be set after the function/procedure ends.

PS: I will put all generic EXTENT info (dynamic or fixed size) in #1620, I will limit the information in this task to be specific to dynamic EXTENT and EXTENT statement.

#9 - 02/09/2013 12:33 PM - Constantin Asofiei

Dynamic-extent variables can not be used as widgets, like form x with frame f1 where def var x as int extent.

#10 - 02/13/2013 11:30 AM - Constantin Asofiei

- % Done changed from 0 to 70

#11 - 02/23/2013 11:44 AM - Greg Shah

- Target version changed from Milestone 4 to Milestone 7

#12 - 09/20/2013 09:50 AM - Greg Shah

This can be closed, right?

#13 - 09/20/2013 01:02 PM - Constantin Asofiei

Yes, it can be closed.

#14 - 09/20/2013 01:22 PM - Greg Shah

- % Done changed from 70 to 100

- Status changed from WIP to Closed

#15 - 11/16/2016 11:43 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features