



art news

Kyle J. Knoepfel

24 May 2018

getMany (ByType) behavior

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker value: 1 }  
    make7: { module_type: IntMaker value: 7 }  
    readIntsThenMakeSomeMore: { module_type: MoreInts }  
  }  
  p1: [make1, make7]  
  p2: [readIntsThenMakeSomeMore] // relies on getMany  
}
```

- According to *art*'s path-execution guarantees, the products made in p1 may not be available whenever p2 is executed.

getMany (ByType) behavior

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker  value: 1 }  
    make7: { module_type: IntMaker  value: 7 }  
    readIntsThenMakeSomeMore: { module_type: MoreInts }  
  }  
  p1: [make1, make7]  
  p2: [readIntsThenMakeSomeMore] // relies on getMany  
}
```

- According to *art*'s path-execution guarantees, the products made in p1 may not be available whenever p2 is executed.
- In MT execution, there are no trigger-path ordering guarantees at all.

getMany (ByType) behavior

- Consider the following configuration:

Options for *art* 3:

1. Retain unspecified (and nondeterministic in MT) behavior.
2. Remove getMany* interface.
3. For producers and filters, disable use of getMany* (at run-time).
4. For all modules, getMany* can only retrieve products from the input source.
5. For producers and filters, getMany* can only retrieve products from the input source.

- According to *art*'s path-execution guarantees, the products made in p1 may not be available whenever p2 is executed.
- In MT execution, there are no trigger-path ordering guarantees at all.

getMany (ByType) behavior

- Consider the following configuration:

Options for *art 3*:

1. Retain unspecified (and nondeterministic in MT) behavior.
2. Remove getMany* interface.
3. For producers and filters, disable use of getMany* (at run-time).
4. For all modules, getMany* can only retrieve products from the input source.
5. For producers and filters, getMany* can only retrieve products from the input source.
6. For producers and filters, getMany* can retrieve only products from (a) modules on the same path and (b) from the input source.

- Ac available whenever p2 is executed.
- In MT execution, there are no trigger-path ordering guarantees at all.

getMany (ByType) behavior – Ex. 1

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker value: 1 }  
    make7: { module_type: IntMaker value: 7 }  
    readIntsThenMakeSomeMore: { module_type: MoreInts }  
  }  
  p1: [make1, make7]  
  p2: [readIntsThenMakeSomeMore] // retrieves ints only  
                                     // from the source  
}
```

getMany (ByType) behavior – Ex. 2

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker  value: 1 }  
    make7: { module_type: IntMaker  value: 7 }  
    read1WithGetMany: { module_type: GetManyReader }  
    read7WithGetMany: { module_type: GetManyReader }  
  }  
  p1: [make1, read1WithGetMany] // only '1' retrieved  
  p2: [make7, read7WithGetMany] // only '7' retrieved  
}
```

getMany (ByType) behavior – Ex. 3

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker value: 1 }  
    make7: { module_type: IntMaker value: 7 }  
    readWithGetMany: { module_type: GetManyReader }  
  }  
  p1: [make1, readWithGetMany]  
  p2: [make7, readWithGetMany]  
}
```

getMany (ByType) behavior – Ex. 3

- Consider the following configuration:

```
physics: {  
  producers: {  
    make1: { module_type: IntMaker value: 1 }  
    make7: { module_type: IntMaker value: 7 }  
    readWithGetMany: { module_type: GetManyReader }  
  }  
  p1: [make1, readWithGetMany]  
  p2: [make7, readWithGetMany]  
}
```

- art* guarantees that `readWithGetMany` is executed only once.
- Therefore, the above configuration **is an error**, and it will trigger an exception at the beginning of the job ***if you use the consumesMany interface.***

Other consequences – Ex. 4

- Consider the following configuration:

```
physics: {  
  producers: {  
    m1: { module_type: ReadIntThenMake tag: "m2" }  
    m2: { module_type: ReadIntThenMake tag: "m1" }  
  }  
  p1: [m1]  
  p2: [m2]  
}
```

- This likely is a configuration error.
- If the input source is EmptyEvent, then one of these modules could successfully read a product, even though that probably wasn't intended.
- Both modules will now fail to read the product.

Other consequences – Ex. 4

- Consider the following configuration:

```
physics: {  
  producers: {  
    m1: { module_type: ReadIntThenMake tag: "m2" }  
    m2: { module_type: ReadIntThenMake tag: "m1" }  
  }  
  p1: [m1]  
  p2: [m2]  
}
```

- This likely is a configuration error.
- If the i *But can you catch the error sooner using CONSUMES?* ssfully
read a *ssfully*
- Both modules will now fail to read the product.

Other consequences – Ex. 4

- Consider the following configuration:

```
physics: {  
  producers: {  
    m1: { module_type: ReadIntThenMake tag: "m2" }  
    m2: { module_type: ReadIntThenMake tag: "m1" }  
  }  
  p1: [m1]  
  p2: [m2]  
}
```

- This likely is a configuration error.
- Even using consumes with this configuration wouldn't trigger an error.
- Because the process name has not been specified, the data-dependency checker assumes that the product could come from the input source.

Other consequences – Ex. 4

- Consider the following configuration:

```
physics: {  
  producers: {  
    m1: { module_type: ReadIntThenMake tag: "m2::current_process" }  
    m2: { module_type: ReadIntThenMake tag: "m1::current_process" }  
  }  
  p1: [m1]  
  p2: [m2]  
}
```

- This likely is a configuration error.

The solution is to use the “current_process” process name.

- Because the process name has not been specified, the data-dependency checker assumes that the product could come from the input source.

Other consequences – Ex. 4

- Consider the following configuration:

```
physics: {  
  producers: {  
    m1: { module_type: ReadIntThenMake tag: "m2::current_process" }  
    m2: { module_type: ReadIntThenMake tag: "m1::current_process" }  
  }  
  p1: [m1]  
  p2: [m2]  
}
```

----- Configuration BEGIN

The following represent data-dependency errors:

Module m1 on path p1 depends on

Module m2 on path p2

Module m2 on path p2 depends on

Module m1 on path p1

----- Configuration END

- This likely is a

The solution

- Because the p
- assumes that t

art 3 status

- Need to enhance the data-dependency checker to handle consumesMany
- Some final bow-tying on implementation
- Release/documentation

- Timescale: ~week until tag

- Next week's stakeholder meeting (5/31):
 - **art 3 presentation/show-and-tell**
 - Plan for full hour meeting (3-4 pm)