Conversion Tools - Feature #11733

Use of existing RuntimeJastInterpreter for dynamic execution of 4GL code

08/12/2026 07:57 AM - Ovidiu Maxiniuc

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

Related issues:

Related to Conversion Tools - Feature #11679: JIT conversion (a.k.a. RUN from...	**New**
Related to Conversion Tools - Feature #11528: implement COMPILE stmt	**New**

History

#1 - 08/12/2026 07:58 AM - Ovidiu Maxiniuc

- Related to Feature #11679: JIT conversion (a.k.a. RUN from Source) added

#2 - 08/12/2026 07:58 AM - Ovidiu Maxiniuc

- Related to Feature #11528: implement COMPILE stmt added

#3 - 08/12/2026 08:38 AM - Ovidiu Maxiniuc

Currently, FWD does not have an implementation of COMPILE 4GL statement.

In 4GL, this allows the programmer to dynamically build an external procedure and compile it to .r code, ready for execution. In FWD, this means to convert the .p source to something which we can execute in Java, using the FWD framework as base. ATM, there are to kind of solutions here:

- use the **static** conversion to transpile from 4GL to Java, then use a compiler to obtain the .class resource which can be class-loaded;
- use the **dynamic** conversion to create the JAST and execute it directly from memory.

Each of these solutions have pros and cons, which I will try to compare below.

Aspect	Static Conversion	Dynamic Interpretation
Workflow	Requires external Java compiler	Everything can happen **in memory**. There is no need to write intermediary files.
Persistence	The .class files can be stored persistently and a 'compile' server can provide cached instances from previous requests	JAST s can be saved to disc as well
Runtime Support	Minimal. .class can be loaded and executed.	ATM, the interpreter supports just a minimal set of syntax, limited to query execution. We need to extend that for any generic 4GL source. Conditional and looping statement are missing.
Conversion Performance	It might be a bit slower because of additional steps (Java brewing and compilation).	May be faster when performed in-memory. No file to write/read
Execution Performance	**Maximum speed** because the code is compiled.	Although the performance is not noticeable now because of the limited language support, some things like resolution (methods, constructors, classes) which are dynamically looked up will

		be more visible as the procedures get larger.
Security	There are a couple of **security holes**: * the .p code an be malicious (calls to FWD's Java native extension); * during the compile process the data gets out of our 'jurisdiction', and the compiler (or other process interfering the .class file) may provide malicious code instead of the .java we generated;	Beside all happening in memory, we can enforce checks on each method or variable accessed. We could limit access to FWD Java native extension, 4GL emulated networking, disc or even persistency, allowing creating a **true sandbox** for this sensitive code.

I will add other aspects as I will think of them.