Conversion Tools - Feature #6320

sharded schemata

05/04/2022 10:51 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Conversion Tools - Feature #6256: improved profile support	**WIP**

History

#1 - 05/04/2022 10:55 AM - Greg Shah

We have multiple customers that split up a large 4GL application into multiple independent modules, each of which can stand on its own or run in combination with some other modules. This gives flexibility at development time (different teams can work on each module) and at deployment time (customers can choose which modules they use).

Evidently, OE allows splitting up the schema for a logical database into multuple standalone .df files (which we are calling "shards"). These can be somehow loaded together as a single database but they are maintained as separate files. Today, FWD can only load a single .df for a logical database. This task is intended to support a list of shards which will be loaded together.

Eric made some related comments in #6256-28.

#2 - 05/04/2022 10:56 AM - Greg Shah

- Related to Feature #6256: improved profile support added

#3 - 08/24/2022 04:52 PM - Greg Shah

One customer that uses this feature extensively organizes all activity in apps (each one like an independent module). This means that all development and compilation is done with code that only ever uses a specific sub-set of the overall schema shards. The customer reports that the same 4GL code can fail to compile with a merged schemata (that has all shards).

They found this during a test where they provided a merged schemata and all apps as a single code base. When they tried to compile it in the 4GL, they had to resolve a large number of ambiguous field names as a result. In the 4GL, each app can be compiled separately and then run in the same system with a single database that has all shards present. As long as the compiled code is unambiguous at the time it is compiled, runtime ambiguity will not exist.

FWD needs to support this capability. This means that apps/modules need to be able to be converted in pieces and executed at runtime as a larger system without issues.