Runtime Infrastructure - Feature #3819

PROFILER system handle

11/30/2018 08:44 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	20%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 11/30/2018 08:54 AM - Greg Shah

- File psc_knowledge_base_article_01493_what_is_the_profiler_control_tool_20181130.pdf added

- File psc_knowledge_base_article_01426_profiler_object_attributes_and_methods_20181130.pdf added

- File psc_knowledge_base_article_22271_how_to_use_profiler_to_get_timing_about_running_external_procedures_20181130.pdf added

- File psc_knowledge_base_article_64682_profiler_editor_new_feature_in_the_11.6_progress_developer_studio_for_openedge_20181130.pdf added

PROFILER Object attributes and methods

How to use PROFILER to get timing information on running external procedures?

What is the Profiler Control Tool?

Profiler Editor - new feature in the 11.6 Progress Developer Studio for OpenEdge

#2 - 12/04/2018 04:34 PM - Greg Shah

Many customers use this, but at least one customer is using this in a production environment. The user enters a hotkey combination and a dialog opens to let them start and stop profiling for their session. The result is a text file that is generated with a combination of lists of the top level entry points which were executed and execution statistics. They combine this with other stats read from the VST (virtual system tables - for example, query stats), store all of that as files in a zip. The user provides the zip to the developers and they can analyze what was going on that was slow.

FWD could implement the same features by instrumenting the runtime, but we would have to match the format of the results.

Implement support for the PROFILER system handle, all attributes, methods and runtime features. As of trunk 11338, most features already have conversion support (and stubs). Everything else needs to be completed.

#3 - 07/18/2019 07:54 AM - Robert-Adrian Chelaru

For Profiler:Attributes we have the following structure

Files which are simply named by the attribute (ex: description.p) contain tests which throw errors followed by a tests with normal working parameters.

Files which are named by the attribute and test parameter (ex: description_binary.p) contain test cases with all possible types of parameters (binary, unknown, empty strings etc.) given to the attribute.

Some attributes have different default values as against the documentation an example is the :directory attribute which is set to "prof.out" instead of "profile.out" as the documentation reads. Tests have the correct values.

For Profiler:Methods we use the same structure, we also include a test in which we leave the Profiler:Enabled & Profiler:Profiling attributes on false, which makes Profiler not generate any file.

For procedure and class tests we have 4 files, 1. testing all procedure types (internal run, external run, persistent procedure and a block in a persistent procedure), 2. testing a simple iterative function and a recursive function defined in a procedure file, 3. testing the properties (the properties contain one with a custom getter and default setter and one with default getter and custom setter) and a variable stored in a class, 4. testing its methods (a static method and an object instance called method).

1. All procedures and the main block are registered by Profiler.

2. Both functions are registered by Profiler.

3. The default getters and setters are not registered by Profiler. The custom ones are registered.

4. Both the instanced and static methods are registered by Profiler.

In the same folder we have a test for dynamic objects, UI Triggers and Subscribe, these also are read by the Profiler.

#4 - 07/24/2019 05:11 PM - Greg Shah

Robert:

Can you please capture the recorded profiler output for each of the testcases? These should be checked in to the testcases project so that we can confirm that the FWD output matches the 4GL output.

In addition, is there documentation on the format of the output file?

#5 - 07/25/2019 02:58 AM - Robert-Adrian Chelaru

Greg Shah wrote:

> Robert:
>
> Can you please capture the recorded profiler output for each of the testcases? These should be checked in to the testcases project so that we can confirm that the FWD output matches the 4GL output.

Sorry, forgot to mention those, they are already up on bazaar. An output file for each test.

> In addition, is there documentation on the format of the output file?

There isn't any official documentation, but there are tools from Progress that can read the output files.

#6 - 07/25/2019 06:05 AM - Greg Shah

> Sorry, forgot to mention those, they are already up on bazaar. An output file for each test.

Good. Those must be in the profiler/out/*.log files.

> There isn't any official documentation, but there are tools from Progress that can read the output files.

Would you please use these tools to inspect the content and document the correspondence with the "raw" log files? The details can be posted here. If there is some content in the log files that is not decoded by the tools and it cannot be otherwise explained, please just make notes accordingly.

#7 - 08/30/2019 04:06 PM - Greg Shah

From testcases/profiler/README.txt (by Robert):

Profiler Raw Data Documentation

Profiler separates the results in sections with the help of a "." (dot) and white line, if two dots follow each other it means there is no information in that category. The profiler has a total of 11 "sections". If the code analyzed by the profiler contains an instantiated object from a class with CUSTOM

properties (note: not default properties) and/or methods those will be shown in the profiler in the 4th section.

1. Date followed by description, followed by time of profiling, followed by User ID.

2. ID of procedure/function/method, listed from the most executed to the least, followed by the name and then the source path from where it is called, followed by debug listing path(if any). Lastly we have the CRC of the external procedure.

3. Caller and called ID's
1st parameters are the ID of the caller.
2nd parameter is the line number (Also counts include statements).
3rd parameter is what procedure/function/method is called.
4th parameter is the number of times it is repeated.
example: 2 187 3 1 - means - Procedure with ID 2 is calling procedure with ID 3 at line 187, one time.

4. 1st parameter is the assigned id for the procedure/function/methods/properties,
2nd parameter is the line number,
3rd parameter is how many times it is executed,
4th parameter is total time taken to execute the statement,
5th parameter is total time take to execute the statement + callback's and exiting the statement.

5. 1st parameter is the ID,
2nd parameter is the line number,
3rd parameter is the time of execution for the line without any pauses,
4th parameter is the current total runtime before adding the 3rd parameters number to it (previous execution time + any pause).
If coverage is set on true, section 5 will also show an extra part with all custom properties and methods separated by a "." + white line,
along with how many different places it is called and the ID of the object from within it is called. (It is not needed)

6. It is not needed

7. 1st parameter is the ID of the procedure/function/method or code block that calls the action code,
2nd parameter is the corresponding number for the action code,
3rd parameter is the number of times action codes are executed,
4th parameter is the name of the action code.

Though Section 7 is not found in the the Profiler Viewer, it is needed to have at least one recording in it for the Profiler Viewer - AVM Information to work.
The information is not important as long as its in the basic format for section 7. An example that can be used everywhere
of what can fill section 7 is: 1 37 3 "ECONST"

8. It is not needed

9. It is not needed

10. Run configuration/parameters followed on the next line by database name and number of tables (system tables + defined tables).

11. 1st parameter is the time took to generate the profiler file,
2nd parameter is the user data (This only appears only if the "write-data" method isn't called and user-data method is set).

The profiler assigns an ID for each procedure/function, even if they are called multiple times, they all get unique ID's.
ID: 1 stands for internal statements, definitions and keywords from ABL.

Coverage - If set on false Profiler does not show the extra part of section 5 for custom properties and methods.
Listings - If set on true generates debug listings.
Profiling - If set on false section 5 does not record anything, nor do the other section record anything except internal statements.
Tracing - Must be set after the procedure to work.
Trace-Filter - The 5th section does not output anymore information in case of procedure/functions only for instantiated object and their custom properties and methods.

#8 - 08/30/2019 04:11 PM - Greg Shah

Robert, the following are my questions based on your "Profiler Raw Data Documentation".

1. If a section is listed as "It is not needed", what does that mean?

- Does it mean that OpenEdge always has an empty section? OR
- Does it mean that the meaning of the data is unknown but cannot be seen in the Profile Viewer?
- Something else?

2. What do you mean by "Also counts include statements"?

- Does this mean that the line numbers are based on the fully preprocessed file? OR
- Are the line numbers specific to the procedure, class or include file in which the code was originally written?

Are all line numbers using this same method?

3. What are debug listings? Do you have any examples of the content of these listings?

#9 - 09/02/2019 04:31 AM - Robert-Adrian Chelaru

Greg Shah wrote:

1. If a section is listed as "It is not needed", what does that mean?

It means that Profiler Viewer does not make use of those sections.

2. What do you mean by "Also counts include statements"?

   - Does this mean that the line numbers are based on the fully preprocessed file?

Yes

3. What are debug listings? Do you have any examples of the content of these listings?

Debug Listings are generated after Listings is set on true (Note: Directory Attribute changes the path of were debug listings will be generated). If it is set before Enabled, instantiation of an object, run statement w/ external procedure file it will generate a fully preprocessed file and source files for the external procedures/classes.

I have updated the README file with the mentioned information and also updated the Listings test so they generate debug files in /profiler/out/debug.

#10 - 10/14/2019 03:59 AM - Marian Edu

Hi Greg, is there anything that needs clarification on this topic?

#11 - 10/17/2019 09:16 AM - Greg Shah

Marian Edu wrote:

> Hi Greg, is there anything that needs clarification on this topic?

No, I think this is complete. Thank you.

#12 - 04/28/2020 10:09 AM - Constantin Asofiei

4231b rev 11493 added PROFILER:COVERAGE, TRACING, TRACE-FILTER and USER-DATA. All the profiler/ tests now convert and compile.

#13 - 06/20/2020 12:01 PM - Greg Shah

- % Done changed from 0 to 20

Task branch 4231b has been merged to trunk as revision 11347.

Files

psc_knowledge_base_article_01426_profiler_object_attributes_and_methods_20181130.pdf	109 KB	11/30/2018	Greg Shah
psc_knowledge_base_article_22271_how_to_use_profiler_to_get_timing_about_running_external_procedures_20181130.pdf	103 KB	11/30/2018	Greg Shah
psc_knowledge_base_article_01493_what_is_the_profiler_control_tool_20181130.pdf	92 KB	11/30/2018	Greg Shah
psc_knowledge_base_article_64682_profiler_editor_new_feature_in_the_progress_developer_studio_for_openedge_20181130.pdf	161 KB	11/30/2018	Greg Shah