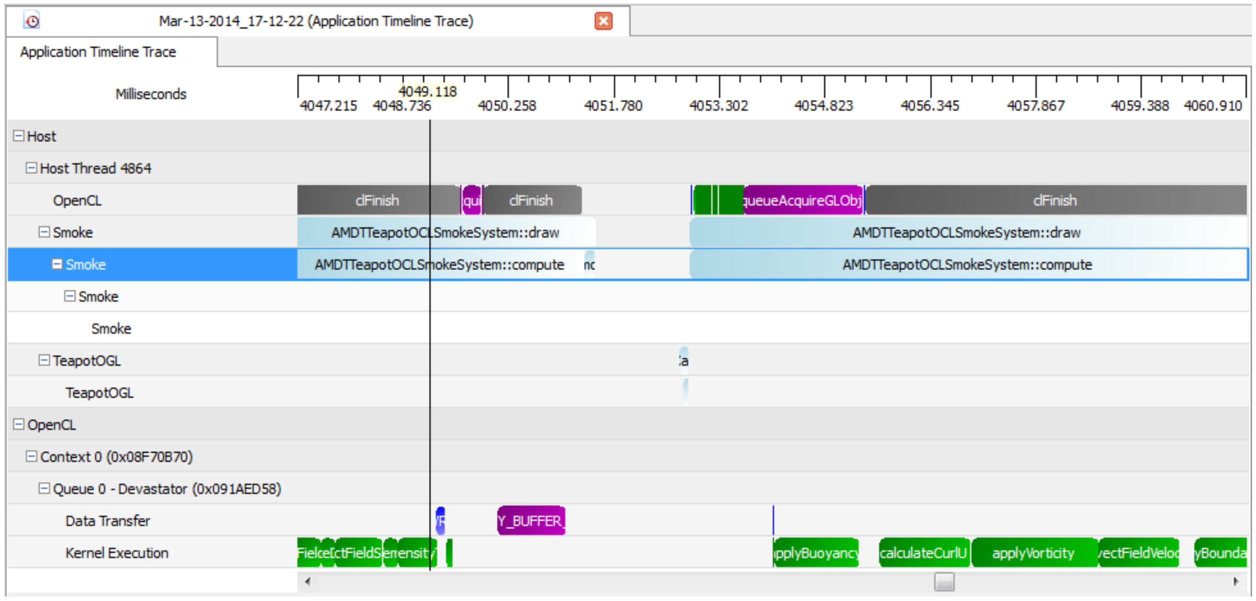
AMD Developer Tools Activity Logger API

Last updated: November 18, 2015

Introduction



The AMD Developer Tools Activity Logger (AMDTActivityLogger) library provides a simple instrumentation API that can help users to analyze their applications. It allows users to instrument their code with calls to amdtBeginMarker() and amdtEndMarker(). These calls are then used by CodeXL to annotate the application's timeline in a hierarchical way.

Each marker has a name, two timestamps which define the beginning and end of the code that the marker encloses and an optional group name. In CodeXL's Timeline view, markers from different threads are grouped separately under dedicated thread branches. All top-level markers are under a thread branch. There are two ways to create sub-branches under a thread branch. The first way is to make nested calls to amdtBeginMarker(). The second way is to create a top-level marker with a group name. Group names are ignored if markers are not directly under a thread branch.

All APIs are thread-safe.

C Functions Definitions

amdtInitializeActivityLogger()

Syntax:

```
int amdtInitializeActivityLogger()
```

Description:

This function initializes the AMDTActivityLogger library. It checks whether or not the log is enabled, and if not, it returns AL_APP_PROFILER_NOT_DETECTED. Depending on the CodeXL trace mode being used, AMDTActivityLogger may cache all the trace results and flush them to disk when amdtFinalizeActivityLogger() is called or it may flush trace result periodically and merge them when amdtFinalizeActivityLogger() is called.

Returns:

AL_SUCCESS	Returned if AMDTActivityLogger was successfully initialized.
AL_FINALIZED_PERF_MARKER	Returned if amdtFinalizeActivityLogger has already been called.
AL_APP_PROFILER_NOT_DETECTED	Returned if the application has not been run with the AMD CodeXL Profiler.

amdtBeginMarker()

Syntax:

```
int amdtBeginMarker(
            const char* szMarkerName,
            const char* szGroupName,
            const char* szUserString);
```

Description:

This function marks the beginning of an activity marker. If AMDTActivityLogger has not been initialized, AL_UNINITIALIZED_PERF_MARKER is returned. If the call is successful, a timestamp is internally recorded. The specified marker name is displayed on the timeline block. A nested amdtBeginMarker() call creates a new sub-branch with its group name ignored. A top-level amdtBeginMarker() call with a group name specified also creates a new branch.

Returns:

AL_SUCCESS	Returned if operation succeeded.
AL_UNINITIALIZED_PERF_MARKER	Returned if amdtInitializeActivityLogger() has not been called.

AL_FINALIZED_PERF_MARKER	Returned if amdtFinalizeActivityLogger() has been called.
AL_NULL_MARKER_NAME	Returned if szMarkerName is equal to NULL.

amdtEndMarker()

Syntax:

```
int amdtEndMarker();
```

Description:

This function marks the end of an activity marker. If AMDTActivityLogger has not been initialized, AL_UNINITIALIZED_PERF_MARKER is returned. If the call is successful, a timestamp is recorded. If amdtEndMarker() is called without sufficient amdtBeginMarker() calls, AL_UNBALANCED_MARKER is returned.

Returns:

AL_SUCCESS	Returned if operation succeeded.
AL_UNINITIALIZED_PERF_MARKER	Returned if amdtInitializeActivityLogger() has not been called.
AL_FINALIZED_PERF_MARKER	Returned if amdtFinalizeActivityLogger() has been called.
AL_UNBALANCED_MARKER	Returned if insufficient amdtBeginMarker() calls have been made to balance this call.

amdtFinalizeActivityLogger()

Syntax:

```
int amdtFinalizeActivityLogger();
```

Description:

This function finalizes AMDTActivityLogger. Unless this function is called, no AMDTActivityLogger file will be generated. After this call, no further activity markers will be recorded. The AMDTActivityLogger output file name is defined as the base output file name specified when running CodeXL Profiler plus .AMDTActivityLogger extension.

Returns:

AL_SUCCESS	Returned if AMDTActivityLogger was successfully

	finalized.
AL_UNINITIALIZED_PERF_MARKER	Returned if AMDTActivityLogger has not been initialized.
AL_FAILED_TO_OPEN_OUTPUT_FILE	Returned if AMDTActivityLogger failed to write to output file.

amdtStopProfiling()

Syntax:

```
int amdtStopProfiling(amdtProfilingControlMode profilingControlMode);
```

Description:

This function instructs the CodeXL profiler to stop profiling. Profiling will only be stopped if the current profile mode in CodeXL matches the mode specified by the profilingControlMode parameter. Possible values for profilingControlMode are “AMDT_TRACE_PROFILING”, “AMDT_PERF_COUNTER_PROFILING”, “AMDT_CPU_PROFILING”, and “AMDT_ALL_PROFILING”. “AMDT_ALL_PROFILING” indicates that the profiler should be stopped for either profiling mode. Together with amdtResumeProfiling (see below) these functions can be used to control the scope of the profile during runtime so that only relevant parts of the program are profiled.

Returns:

AL_SUCCESS	Returned if AMDTActivityLogger trace was successfully stopped.
AL_FAILED_TO_ATTACH_TO_PROFILER	Returned if AMDTActivityLogger has not been successfully initialized within the context of the profiled application.
AL_WARN_PROFILE_ALREADY_PAUSED	Returned if the profiler is already in pause state. If profiler is already in pause state, amdtStopProfiling() has no negative effect.

amdtResumeProfiling()

Syntax:

```
int amdtResumeProfiling(amdtProfilingControlMode profilingControlMode);
```

Description:

This function instructs the CodeXL profiler to resume profiling. Profiling will only be resumed if the current profile mode in CodeXL matches the mode specified by the profilingControlMode parameter. Possible values for profilingControlMode are “AMDT_TRACE_PROFILING”, “AMDT_PERF_COUNTER_PROFILING”, “AMDT_CPU_PROFILING”, and “AMDT_ALL_PROFILING”. “AMDT_ALL_PROFILING” indicates that the profiler should be resumed for either profiling mode. Together with amdtStopProfiling (see above) these functions can be used to control the scope of the profile during runtime so that only relevant parts of the program are profiled.

Returns:

AL_SUCCESS	Returned if AMDTActivityLogger trace was successfully resumed.
AL_FAILED_TO_ATTACH_TO_PROFILER	Returned if AMDTActivityLogger has not been successfully initialized within the context of the profiled application.
AL_WARN_PROFILE_ALREADY_RESUMED	Returned if the profiler is already in resume state. If profiler is already in resume state, amdtResumeProfiling () has no negative effect.

amdtStopProfilingEx()

Syntax:

```
int amdtStopProfilingEx(void);
```

Description:

This function is same as calling amdtStopProfiling(AMDT_CPU_PROFILING) and instructs the CodeXL CPU profiler to stop the profiling. Together with amdtResumeProfilingEx (see below) these functions can be used to control the scope of the CPU profile during runtime so that only relevant parts of the program are profiled.

Returns:

AL_SUCCESS	Returned if profiler was successfully stopped.
AL_FAILED_TO_ATTACH_TO_PROFILER	Returned if profiler not been successfully initialized within the context of the profiled application.
AL_WARN_PROFILE_ALREADY_PAUSED	Returned if the profiler is already in pause state. If profiler is already in pause state, amdtStopProfilingEx() has no negative effect.

amdtResumeProfilingEx()

Syntax:

```
int amdtResumeProfilingEx(void);
```

Description:

This function is same as calling amdtResumeProfiling(AMDT_CPU_PROFILING) and instructs the CPU profiler to resume the profiling. Together with amdtStopProfilingEx (see above) these functions can be used to control the scope of the CPU profile during runtime so that only relevant parts of the program are profiled.

Returns:

AL_SUCCESS	Returned if profiler was successfully stopped.
AL_FAILED_TO_ATTACH_TO_PROFILER	Returned if profiler not been successfully initialized within the context of the profiled application.
AL_WARN_PROFILE_ALREADY_RESUMED	Returned if the profiler is already in resume state. If profiler is already in resume state, amdtResumeProfiling () has no negative effect.

C++ Interface Definition

amdtScopedMarker class

The amdtScopedMarker class can be used for automatically creating a marker's beginning and ending, using the C++ object scoping rules similar to the Guard pattern. This class opens a marker in the constructor and closes it in the destructor. This saves the user the need to explicitly call amdtEndMarker and also handles user code with exceptions and multiple exit points correctly.

Example:

```
bool MyClass::initialize()
{
	amdtScopedMarker scopedMarker("MyClass::initialize", "NameOfMyGrup", "");
	// The rest of the function
	// …
}
```