

Profiling Runtime Generated and Interpreted Code using the VTune™ Performance Analyzer

User Guide

Copyright © 1998–2008 Intel Corporation

All Rights Reserved

Document Number: 319806-003US

World Wide Web: <http://www.intel.com>



Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting [Intel's Web Site](#).

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright (C) 1998-2008, Intel Corporation. All rights reserved.

Revision History

Document Number	Revision Number	Description	Revision Date
319806-001US	0.6	Initial Release	04 2008
319806-001US	0.74	<ul style="list-style-type: none"> • Tables are added in API descriptions • Intel Copyright notice added to the sample source code 	06 2008
319806-003US	0.75	<ul style="list-style-type: none"> • Updated the API section • Intel Copyright notice added to the sample source code 	06 2008



Contents

1	Introduction.....	4
2	Adding VTune™ Performance Analyzer JIT Profiling Support.....	5
	2.1 Instructions to include JIT Profiling Support	5
	2.2 Time and Event Based Sampling.....	6
	2.3 Call Graph Analysis.....	6
	2.4 Special Virtual Machine Events	6
3	API Description.....	7
	iJIT_NotifyEvent.....	7
	iJIT_GetNewMethodID	9
	iJIT_RegisterCallbackEx.....	11
	iJIT_IsProfilingActive.....	12
	FinalizeThread.....	12
	FinalizeProcess	13
4	Usage Example.....	14
	4.1 Sample Code	14
	4.2 Call Graph Analysis of the Sample Code	14



1 Introduction

The VTune™ Performance Analyzer's JIT (Just-In-Time) Profiling API provides functionality to profile runtime generated code. This API allows analysis of runtime generated code with both sampling and call graph profilers which are already available with VTune™ Performance Analyzer.

JIT Profiling API can also be used to analyze the virtual machines (VM) which interpret code. Analyzing a VM with sampling or call graph methods provides valuable performance data on how the VM functions and performs.

The VM (or any code that is generating code during runtime) can communicate with a profiler object through a statically linked library (Figure 1). The static library and the profiler object is part of the standard VTune™ analyzer installation. During runtime, the VM notifies the static library of specific events and provides the static library with the necessary data. The static library dynamically loads the profiler object (dll) and sends the data to the VTune analyzer, via the profiler object, to be formatted and displayed. If the VTune analyzer is not installed, profiling is disabled.

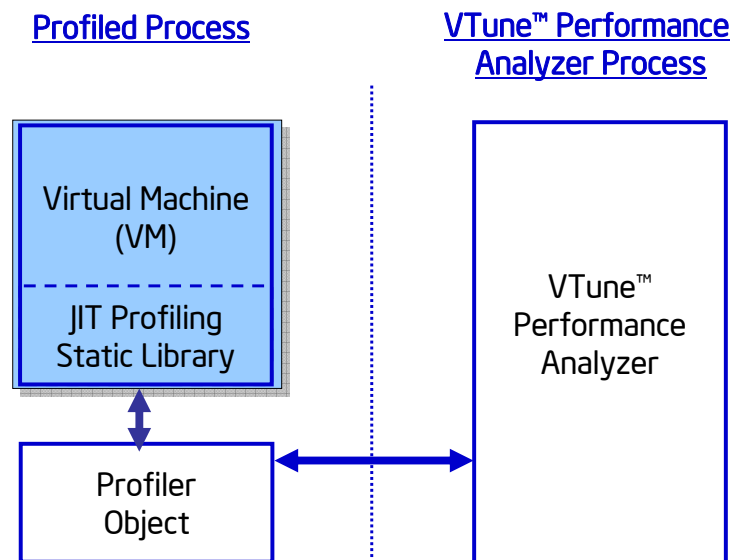


Figure 1: JIT Profiling Components



2 Adding VTune™ Performance Analyzer JIT Profiling Support

2.1 Instructions to include JIT Profiling Support

1. Include **JITProfiling.h** file located under "*C:\Program Files\Intel\VTune\Analyzer\include*" directory for Microsoft* operating systems and under */opt/intel/vtune/analyzer/include* for Linux* operating systems. This header file provides all API function prototypes and type definitions.
2. Link the Virtual Machine (or any code using these APIs) with **JITProfiling.lib** located under "*C:\Program Files\Intel\VTune\Analyzer\lib*" on Windows*, and with **JITProfiling.a** located under "*/opt/intel/vtune/analyzer/bin*" on Linux* operating systems. On Linux* please link with the standard libraries **libdl.so** and **libpthread.so**. **Note:** **JITProfiling.a** which comes with VTune analyzer is compiled with g++ and not with gcc, therefore either compile your code with g++ or compile with gcc and link with **-lstdc++** library.

In order to function properly, a VM that uses the JITProfiling API should implement a mode-change callback function and register it using **iJIT_RegisterCallbackEx**. The callback function is executed every time the profiling mode changes. This ensures that the VM issues appropriate notifications when mode changes happen.

To enable JIT profiling support, set the environment variable **ENABLE_JITPROFILING=1**.

On Windows:

```
set ENABLE_JITPROFILING=1
```

On Linux:

```
export ENABLE_JITPROFILING=1
```

On Linux JIT profiling can only be used with the command line interface (vtl) and *jitprofiling* option needs to be used.

For call graph analysis:

```
vtl activity jitcg -c callgraph -o jitprofiling -app  
./jitprof run
```

For sampling analysis:



```
vtl activity jitsamp -c sampling -o jitprofiling -app  
./jitprof run
```

If you wish to perform JIT profiling on a remote Linux OS system, define the **BISTRO_COLLECTORS_DO_JIT_PROFILING** environment variable in the shell where **vtserver** executes.

```
export BISTRO_COLLECTORS_DO_JIT_PROFILING=1
```

2.2 Time and Event Based Sampling

In order to profile run-time generated code with the sampling profiler, **iJIT_NotifyEvent** API function needs to be used in order to send the **iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED** notification. This function needs to be called after JIT compilation and before the first entry into the JIT compiled method.

2.3 Call Graph Analysis

To profile run-time generated code with the call graph profiler, use:

- **iJVM_EVENT_TYPE_ENTER_NIDS** notification upon function entry,
- **iJVM_EVENT_TYPE_LEAVE_NIDS** notification upon function exit,
- **iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED** notification upon function JIT compilation.

When an exception occurs, the VTune analyzer expects method leave events for each method that is unwound. When an exception occurs in the VM-code, the VTune analyzer expects the VM to issue the LEAVE callback when it “unwinds” the stack frame of the executing method. Otherwise, the generated call-graph will be incorrect.

Call **FinalizeThread** when a thread you are profiling exits. Call **FinalizeProcess** when a process you are profiling is about to exit. The JIT Profiling API automatically identifies new threads and handles thread related initialization.

Note: mixed mode call graph of VM functions and native IA-32, IA-32 with Intel® 64, or IA-64 architecture functions is not supported.

2.4 Special Virtual Machine Events

Use the **iJVM_EVENT_TYPE_SHUTDOWN** to terminate profiling, both for sampling and for call graph.



3 API Description

This section describes JIT Profiling API functions, their prototypes, and the data types associated with them.

iJIT_NotifyEvent

Sends an event notification to the VTune analyzer.

```
int iJIT_NotifyEvent (
    iJIT_JVM_EVENT event_type,
    void *EventSpecificData
);
```

Description:

The iJIT_NotifyEvent function sends a notification of event_type with the data pointed by EventSpecificData to the VTune analyzer.

Parameters:

Parameter	Description
iJIT_JVM_EVENT event_type	Notification code to send to the VTune analyzer. See a complete list of event types below.
void *EventSpecificData	Pointer to event specific data.

Return Values:

The return values are dependent on the particular **iJIT_JVM_EVENT**.

Event Types

The following values are allowed for event_type:

Event	Description
iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED	Send this notification after a JITted method has been loaded into memory, and possibly JIT compiled. Use the iJIT_Method_Load structure for EventSpecificData. The return value of iJIT_NotifyEvent is undefined.
iJVM_EVENT_TYPE_ENTER_NIDS	Send this notification at the entry point of a method, JITted or not. This notification



	is only used for call graph profiling. Use the <code>ijIT_Method_NIDS</code> structure for <code>EventSpecificData</code> . <code>ijIT_NotifyEvent</code> returns 0 on failure.
<code>ijVM_EVENT_TYPE_LEAVE_NIDS</code>	Send this notification at the exit point of a method, JITted or not. This notification is only used for call graph profiling. Use the <code>ijIT_Method_NIDS</code> structure for <code>EventSpecificData</code> . <code>ijIT_NotifyEvent</code> returns 0 on failure.
<code>ijVM_EVENT_TYPE_SHUTDOWN</code>	Send this notification to terminate profiling. Use NULL for <code>EventSpecificData</code> . <code>ijIT_NotifyEvent</code> returns 1 on success.

ijIT_Method_Load Structure

The `ijIT_Method_Load` structure has the following fields:

Field	Description
<code>unsigned int method_id</code>	Unique method ID. Method ID's may not be smaller than 0x100000. Either you use the API function <code>ijIT_GetNewMethodID</code> to get a valid and unique method ID, or you take care of ID uniqueness and correct range by yourself.
<code>char *method_name</code>	The name of the method, optionally prefixed with its class name and appended with its complete signature. This argument cannot be set to NULL.
<code>void *method_load_address</code>	The base address of the method code. Can be NULL if the method is not JITted.
<code>unsigned int method_size</code>	The code size of the method. Can be 0 if the method is not JITted.
<code>unsigned int line_number_size</code>	The number of entries in the line number table.
<code>pLineNumberInfo line_number_table</code>	Pointer to the line numbers info array. Can be NULL if <code>line_number_size</code> is 0. See <code>LineNumberInfo</code> Structure for a description of a single entry in the line number info array.
<code>unsigned int class_id</code>	Unique class ID. Can be 0.
<code>char *class_name</code>	Class name. Can be NULL.
<code>char *source_file_name</code>	Source file name. Can be NULL.
<code>void *user_data</code>	This field is obsolete.
<code>unsigned int user_data_size</code>	This field is obsolete.
<code>ijJDEnvironmentType env</code>	This field is obsolete.



LineNumberInfo Structure

The **LineNumberInfo** structure has the following fields:

Field	Description
unsigned int Offset	Opcode byte offset from the beginning of the method.
unsigned int LineNumber	Matching source line number offset (from beginning of source file).

iJIT_Method_NIDS Structure

The **iJIT_Method_NIDS** structure has the following fields:

Field	Description
unsigned int method_id	The method ID.
unsigned int stack_id	This field is ignored and filled automatically.
char *method_name	The name of the method, optionally prefixed with its class name and appended with its complete signature (see Method Signature section below for details). This field can be NULL.

Method Signature

The signature of a method is composed of its return value and the arguments it accepts in the following structure:

```
(<return value>(<param 1>, <param 2> ...))
```

The complete method_name argument structure is:

```
<class-name>.<method-name><signature>.
```

iJIT_GetNewMethodID

Generates new method ID upon each call.

```
unsigned int iJIT_GetNewMethodID(void);
```

Description:

The **iJIT_GetNewMethodID** function generates new method ID upon each call. You must use this function to assign unique and valid method IDs to methods reported to the VTune analyzer.

Parameters

None



Return Value:

A new unique method ID. When out of unique method IDs, this API function returns 0.



ijIT_RegisterCallbackEx

Registers a call back function.

```
void ijIT_RegisterCallbackEx(
    void *userdata,
    ijIT_ModeChangedEx NewModeCallBackFuncEx
);
```

Description:

The **ijIT_RegisterCallbackEx** function registers **NewModeCallBackFuncEx** as a call back function. When the profiling mode changes, the callback function is called.

Parameters:

Parameter	Description
void *userdata	Pointer to user data to be passed to callback function
ijIT_ModeChangedEx NewModeCallBackFuncEx	Pointer to callback function (see ijIT_ModeChangedEx prototype).

Return Values:

None

ijIT_ModeChangedEx Prototype

ijIT_ModeChangedEx has the following prototype:

```
void (void *UserData, ijIT_ModeFlags Flags)
void *UserData
```

The user data pointer passed to **ijIT_RegisterCallbackEx**

```
ijIT_ModeFlags Flags
```

Indicates what mode the VTune analyzer is in. It can have any of the following values OR'd together:

Value	Description
ijIT_NO_NOTIFICATIONS	The VTune analyzer is not running.
ijIT_BE_NOTIFY_ON_LOAD	Call ijIT_NotifyEvent with ijVM_EVENT_TYPE_METHOD_LOAD_FINISHED for every method that is loaded.
ijIT_BE_NOTIFY_ON_METHOD_ENTRY	Call ijIT_NotifyEvent with ijVM_EVENT_TYPE_ENTER_NIDS on entry of every



	method.
<code>ijIT_BE_NOTIFY_ON_METHOD_EXIT</code>	Call <code>ijIT_NotifyEvent</code> with <code>ijVM_EVENT_TYPE_LEAVE_NIDS</code> on exit of every method.

ijIT_IsProfilingActive

Returns the current mode of the profiler.

```
ijIT_IsProfilingActiveFlags  ijIT_IsProfilingActive (
void
);
```

Description:

Returns the current mode of the profiler: off, sampling, or call graph using the **ijIT_IsProfilingActiveFlags** enumeration.

Parameters:

None

Return Values:

Value	Description
<code>ijIT_NOTHING_RUNNING</code>	No profiler is running. Currently not used.
<code>ijIT_SAMPLING_ON</code>	Sampling is running. This is the default return value.
<code>ijIT_CALLGRAPH_ON</code>	Call graph is running.

FinalizeThread

Call this when the thread exits. Required for call graph only.

```
void FinalizeThread (
    void
);
```

Parameters:

None



Return Values:

None

FinalizeProcess

Call this after the process ends. Use this function only under call graph profiling.

```
void FinalizeProcess (  
    void  
);
```

Parameters:

None

Return Values:

None

Remarks:



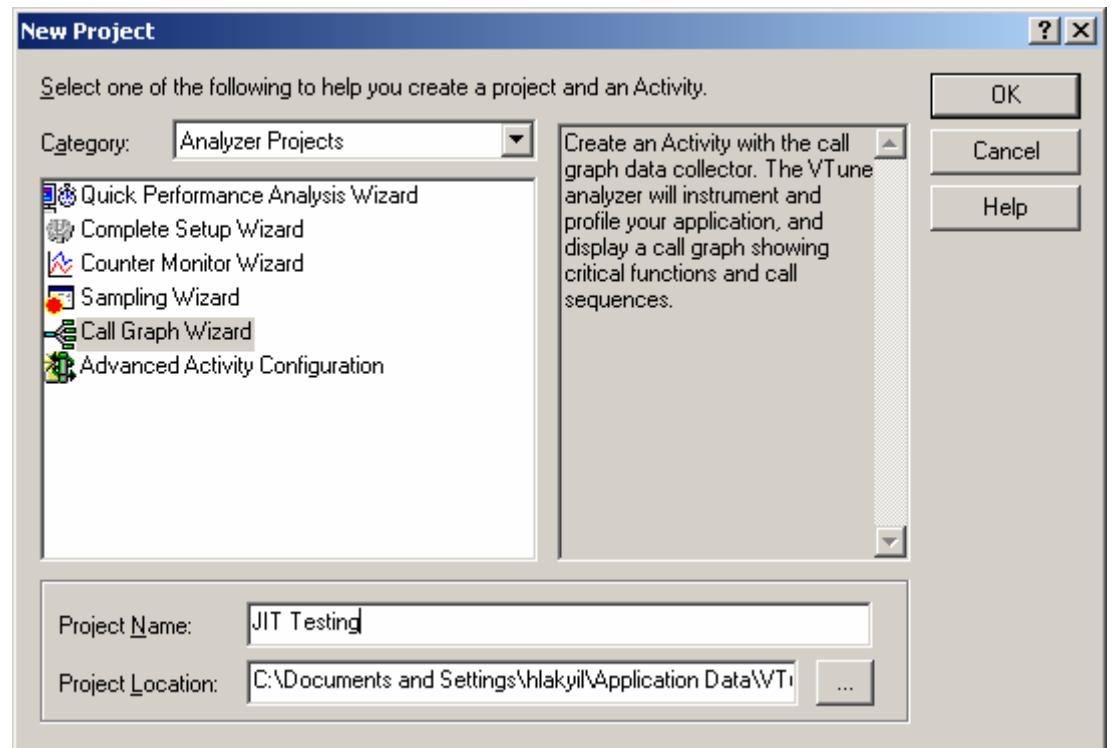
4 Usage Example

4.1 Sample Code

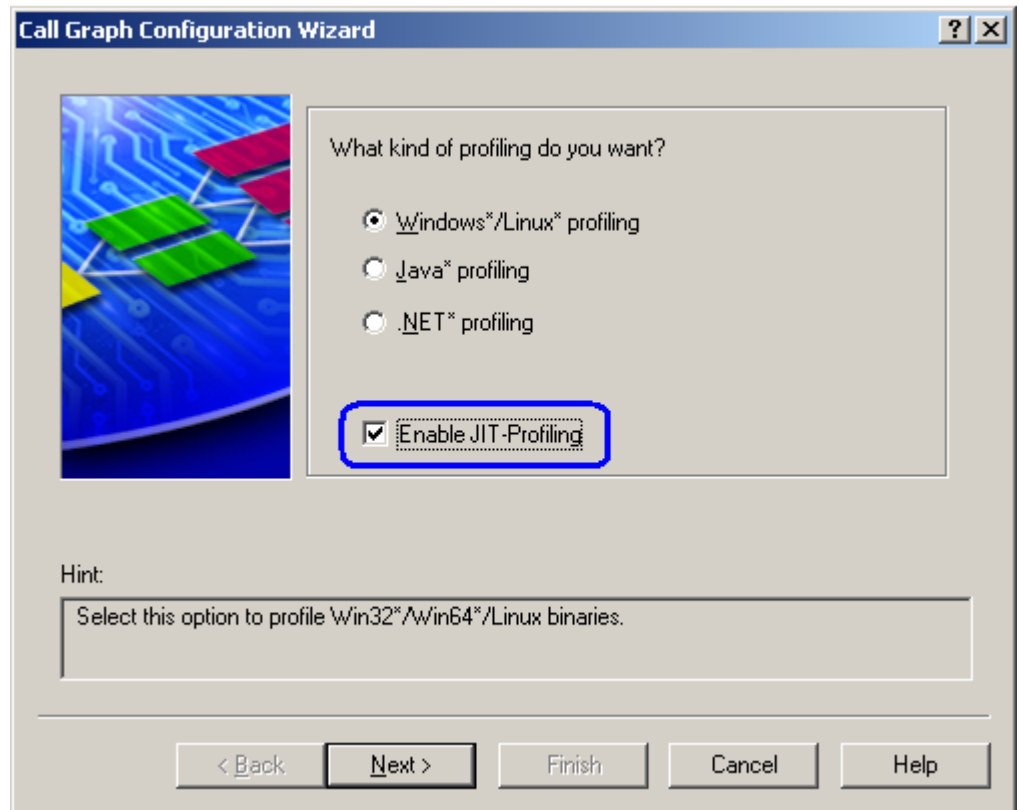
The sample code can be found under the *C:\Program Files\Intel\VTune\Examples* directory for Windows and under */opt/intel/vtune/samples* directory for Linux*.

4.2 Call Graph Analysis of the Sample Code

Create a new VTune analyzer project.



Select what kind of profiling you want to do and check “Enable JIT Profiling” option. Please note that “Enable JIT-Profiling” option won't appear unless **ENABLE_JITPROFILING** environment variable is enabled.



After following the wizard, the sample test code generates the following results.

