

# IS Service Profiler



## Whitepaper

<b>Project</b>	ISSPROF
<b>Document File</b>	ISSPROF - Profiling Whitepaper.doc
<b>Version</b>	1.2.4
<b>Version Date</b>	28-09-2011
<b>Status</b>	FINAL
<b>Author</b>	António Abreu

## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b> .....	<b>1</b>
1.1	Abstract .....	1
1.2	Target Audience .....	1
1.3	What is Profiling? .....	1
1.4	When do you profile? .....	2
<b>2</b>	<b>PROFILING WEBMETHODS IS SERVICES</b> .....	<b>4</b>
<b>3</b>	<b>THE SERVICE PROFILER</b> .....	<b>6</b>
3.1	Main Features .....	6
3.2	Installation and Admin .....	7
3.3	Sample Analysis Tool .....	9
3.3.1	View Per Service .....	9
3.3.2	Code Coverage .....	10
3.4	Extensibility .....	11
<b>APPENDIX A</b>	<b>GLOSSARY</b> .....	<b>1</b>

## FIGURE INDEX

Figure 1 - When to profile .....	2
Figure 2 - webMethods IS Service Profiler .....	6
Figure 3 - Start/Stop the Profiler.....	8
Figure 4 - Managing Package Exclusion Patterns.....	8
Figure 5 - Freeze Snapshot setting.....	9
Figure 6 - The View Per Service report.....	9
Figure 7 - Timing display format .....	10
Figure 8 - The Code Coverage report .....	10
Figure 9 - The Code Coverage service usage detail .....	11
Figure 10 - The Snapshot as a Document structure .....	11
Figure 11 - Exporting Snapshot as a file .....	12

# 1 INTRODUCTION

## 1.1 Abstract

---

This document presents an overview of profiling as part of the development/maintenance cycle of **webMethods Integration Server** integrations and presents a tool developed by **Wrightia** to enable advanced **Integration Server** service profiling.

## 1.2 Target Audience

---

Being a **Profiler** a development and testing tool the targeted audience is mainly Developers for the **webMethods Integration Server v6.x**. However, Administrators and others that may face the responsibility of keeping a system running with acceptable performance may find the subject of interest.

## 1.3 What is Profiling?

---

The word **profiling** has its root in the ancient Latin language and mainly means: the drawing of a line or an outline of a shape.

The profiling concept is generic and commonly used in criminology and psychiatry, where a personality profile is created from formal summary or analysis of data, often in the form of a graph or table, representing distinctive features or characteristics.

**Profiling:** *Recording a person's behavior and analyzing psychological characteristics in order to predict or assess their ability in a certain sphere or to identify a particular group of people.*

WordNet © 2.0, © 2003 Princeton University

*In criminology, a **profiler** is a criminologist who studies a criminal's behavior for clues to psychology to aid in capturing them.*

wikipedia

The same concept, in generic terms, is applied in software development where the goal is to create an application runtime profile from where a performance optimizing strategic can be outlined and implemented.

*In computer programming, a **profiler** is a computer program that can track the performance of another program by checking information collected while the code is executed. A profiler can identify the time used by or frequency of use of various portions of the second program. Typically this information is used to identify those portions of the second program that consume the most time. These time-consuming parts are then targeted for optimization. Also a profiler can be useful for debugging. Like a debugger, it is often used with a front-end (front-ends to profilers are more specific to each profiler).*

www.reference.com

## 1.4 When do you profile?

---

In one word: always.

In every stage of the creation of a computer based automatic data processing solution, some form of profiling is automatically or intentionally being performed. Most of the times, the profiling is done with tools and techniques that are not identified as profilers and/or profiling.



Figure 1 - When to profile

In **Figure 1** (*above*), the main solution build phases are presented, together with the span of the profiling activity. These phases could be further subdivided, but because those details go beyond the objectives of the current document, and for the sake of simplicity, they have been collapsed to the presented four.

The profiling activity starts with the **Architecture** of the solution, at the hardware and at the software level. The **Architecture** will have to layout the adequate solution considering the involved hardware and software components that match the problem description and its environment.

From **Architecture**, through **Design** and **Implementation**, all need to base themselves on some solution profiling in order to cope with the specifications, which may incorporate concerns about performance, reliability, flexibility, management, etc.; and that may have to consider raw information such as data volumes, day time schedules, number of users, component distribution, etc.

In the **Architecture** phase, profiling may materialize by outlining the application/solution through the use of some platform's *out-of-the-box* components (*e. g.*, **Trading Networks**, **Broker**, **Broker Territory**, etc.), standard or widely used technology based implementations (*e. g.*, **HTTP**, **XML**, **JMS**, etc.), custom developed technology frameworks, more discrete custom developments (*e.g.*; database based information sharing) or even some radical and completely new approach.

**Design** profiling is usually accomplished by performing some architecture exercise at a smaller scale, identifying the components and cataloguing them individually and into groups by analyzing their commonalities and specificities. On **Design**, the profiling of a component will commonly establish a behavior pattern that matches one that is already known or a completely new or even one that is specific to the problem domain.

When making the **Implementation**, the profiling activity decreases very much because most of the behavioral intentions have already been established by the preceding phases. However, to some lower degree, the implementation of a piece of working software also include some architecting and designing which in their own terms will establish a usage profile for it.

In the **Testing** phase, the resulting implementation is tested for bugs, error handling and against the established specifications. When there is a discrepancy between the witnessed behavior and what is expected, some debugging activity will have to take place for diagnosis and uncovering of the reason(s) for the mismatch.

When the discrepancy is related to poor performance, a **Profiler Tool** comes handy because the current application running profile can only be determined by going under the hood while the application is actually running. This kind of runtime application profiling is the core subject of this document and is based on collecting information such as:

- Parts being executed;  
These can be procedures, methods, etc.
- Running count;  
The number of times a certain identified runtime element is accessed.
- Start timestamps;  
Timestamp of when the call started.
- End timestamps;  
Timestamp of when the call returned.
- Other available useful runtime information.  
This is dependent of the running environment.

The analysis of the profile data enables an educated selection for **Refactoring** of the **Services** that should contribute the most to the solution's overall performance improvement.

## 2 PROFILING WEBMETHODS IS SERVICES

An **Integration Server Service Profiler** is intended to help identifying the runtime profile of the **Services** in a particular server. Depending on its implementation characteristics, **Services** can be classified as:

- **FLOW Services;**  
Services developed using the **FLOW** language.
- **Java Services;**  
Services developed using **Java** language.
- **C/C++ Services;**  
Services developed using **C/C++** language.
- **Adapter Services;**  
Services that are generated by an **Adapter** through configuration of one of its **Service Templates**.
- **Web Services.**  
This is a subtype of a **FLOW** service that is generated by creating and configuring a new **Web Service Connector**.
- **Other.**  
In the evolution path of the **webMethods Integration Platform**, new types of **IS Services** may be introduced.

Without any dedicated tool, the gathering of information for the **IS Services** while running can be achieved by:

- Including statement that write the needed information to some predefined channel (a file, a database, the console, a document published to **Broker**, etc.);  
This solution is custom and still needs additional software to gather the collected data to allow you to analyze it.
- Setting the **Audit Properties** of the **Service**.  
This is a feature native to the **webMethods IS Services** and needs an **Audit Database** defined to persist the **Audit Information**, which is defined simply by configuring **Service** properties.  
Some **DB SQL** statements still need to be created to access the persisted information.

Because the usage of the above solutions has itself an impact in the **Service** performance, during development they are kept to a minimum or not included at all. This way, when the code is promoted to a **System Test** environment, or even a **Production** environment, and a performance issue happens, the use of the above solutions mean changing the **Services** physically. Not only does this break rules about making changes to code outside the **Development** environment, but may become a problem when it comes to removing those changes when the solution is found... how many of them are there... something may break and a new problem is created.

Ideally, the activity of profiling the **Services** in a running **Integration Server** should be a non-intrusive solution that:

- Is easily installed and removed;
- Can easily be switched on and off;
- Should have a small footprint in the **Integration Server** running environment;
- Not be itself a noticeable impact on the overall performance.

### 3 THE SERVICE PROFILER

The **Service Profiler** is a dedicated and extensible tool for profiling **IS Services** running in a **webMethods Integration Server**.

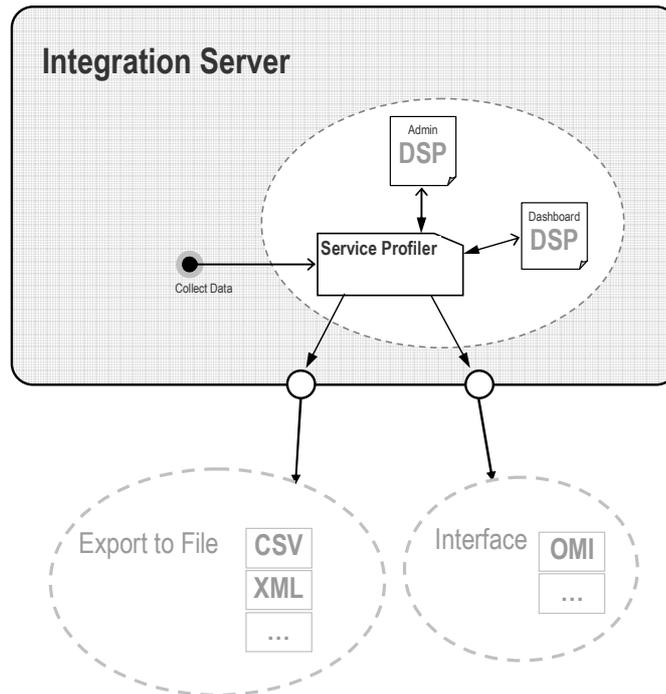


Figure 2 - webMethods IS Service Profiler

#### 3.1 Main Features

The **Service Profiler** has the following features:

- **Non-intrusive;**

No changes need to be made to the **Integration Server**, with exception to additional start-up parameters.

- **Easy Install & Uninstall;**

Installation is just:

- The standard installation of an **Integration Server** package;
- Copy of a folder contained in the installed package to the **Integration Server** installation folder; and
- Insertion of a sequence of predefined (copied from a separate provided file) shell script statements in a single specific section of the **IS** start-up shell script.
- These statements determine and set the additional parameters for the **JVM** command-line.

Uninstalling is a matter of:

- a) Deleting the installed package;
- b) Removing the extra **Integration Server** start-up shell script statements;
- c) Removing the folder created by copy on installation.

- **Small Footprint on the Integration Server;**

No changes need to be performed to the **Services** (audit or otherwise).

No need to set or use an **Audit DB, Broker, Monitor**, etc. All that is needed is the running **Integration Server**.

- **Very low impact on Integration Server resources/performance;**

The tool uses a very small amount of memory to gather data and negligible **CPU** time.

- **Fine-grained timing data;**

While running, **Service** code take turns to have a chance do any real work in the **CPU**.

The **Profiler** gathering of timings for the **Service** execution goes beyond the simple gathering of timestamps for elapsed time calculation. The time actually spent in the **CPU** is also collected.

- **Scheduled snapshots;**

The generation of profile data snapshots can be scheduled and the snapshots persisted for off-line analysis.

The saved snapshots can be combined to enable a flexible analysis on multiple time-boxes.

- **Simple set of operations;**

Start, Stop, Analyze (View a Report/Dashboard), Take Snapshot, Export Data.

- **Open to external analysis tools;**

Snapshots can be exported to a variety of standard exchange file formats for analysis by external tools.

Other **Integration Server Services** can access profile gathered data via a set of public **Services**.

- **Administration pages;**

The tools administration and configuration setting are accessed through **DSPs (Dynamic Server Pages)** that are part of the installed package, and can be used through a browser.

- **Dashboard.**

A different set of pages can be accessed to view and drill statistical data being gathered and saved as snapshots.

## 3.2 Installation and Admin

---

The installation must be made with the **Integration Server** shut down. After starting (*see Figure 3, below*) the **Integration Server** the **Profiler** must be started before using its functionality. The **Profiler** only gathers **Service** information when started, and when stopped all counters are reset.

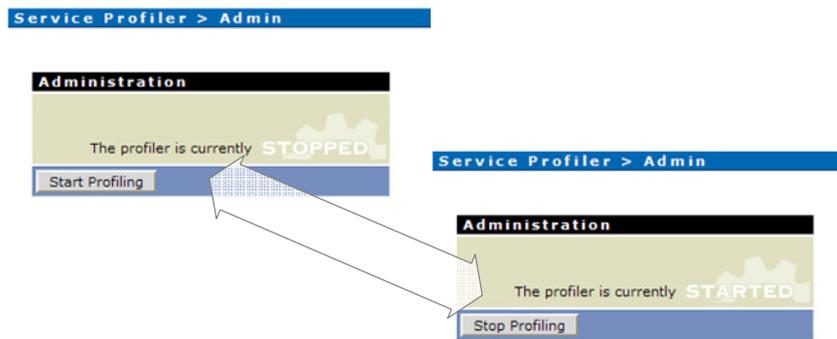


Figure 3 - Start/Stop the Profiler

When analyzing data, the number of services can be from just a few to a few hundred or even more. With a lot of services being viewed it can be cumbersome to visually analyze it when our attention should be focused on a subset of them.

Because all the services that are running in the **Integration Server** have their contribution to the overall performance, none of them is discarded or filtered out at the data gathering part of the tool and are included in the returned snapshot. However, not all of them may need to be shown in the dashboard pages. Some of them, because we cannot **Refactor** them (*e. g.*, those in packages distributed by **webMethods** as part of the platform, the one from the **Service Profiler** itself, 3<sup>rd</sup> party, etc.) may not need to be included in the displayed data.

For this purpose, the **Admin** page contains a global setting that allows defining which packages (and their included services) should be excluded from display when showing analyzed data. The exclusion is defined through a **Regular Expression** over the **Package Name**... and it can be tested before committing (*see Figure 4, below*).

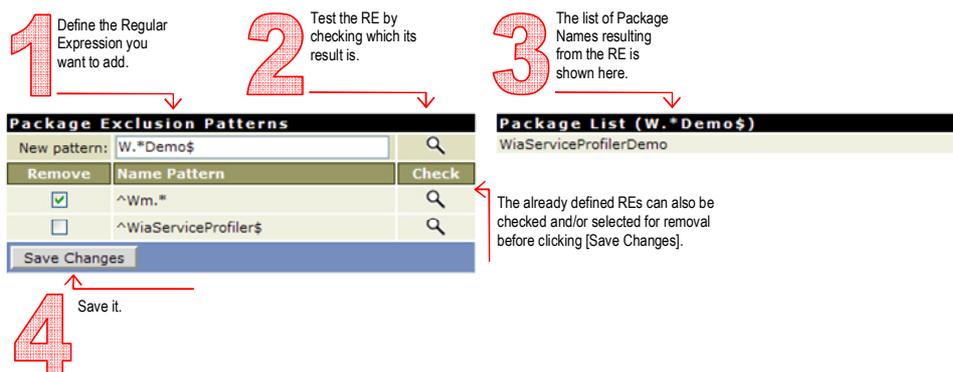


Figure 4 - Managing Package Exclusion Patterns

Once the **IS Service Profiler** starts the profiling activity, it is continuously collecting runtime data. Therefore, every time a **Snapshot** is requested it returns with the most recent counters. However, sometimes it is useful to exercise an analysis over a set of profile data frozen in time (*e. g.*, after running a predefined set of tests, after a period of time, etc.). For this purpose, the **Freeze Snapshot** administrative setting is available (*Figure 5, below*).

When turned on, the **IS Service Profiler** takes a snapshot of the current data and holds it as the response for the next **Snapshot** requests... while the setting remains on. While the **Snapshot** is

Frozen the  icon will appear on any Analysis Tools pages to give a visual hint of the *freshness* of the data shown.

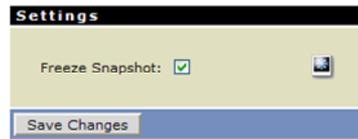


Figure 5 - Freeze Snapshot setting

The **Freeze Snapshot** is also used when the IS Service Profiler is stopped. As part of the **Stop Profiling** procedure, a **Snapshot** of the current counters is taken and frozen. This way, the rest of the functionality over the data can still be used over the last known **Snapshot**.

### 3.3 Sample Analysis Tool

From the data collected by the IS Service Profiler it is possible to build **Analysis Tools** that are views over that data with organization and functionality oriented for broad or specific use.

The IS Service Profiler already includes some of such tools, of which examples are presented in the next sections. The presented **Analysis Tools** are not the complete universe of those already available, and more can also be built.

#### 3.3.1 View Per Service

The **View Per Service** report (see *Figure 6, below*) is an example of the kind of analysis tool that can be implemented around the data gathered from a single snapshot. It reports totals for the data from the snapshot and presents it per service.

Dashboard > View Per Service

Snapshot Identification			
Sample Key	Server Name	Begin Time	End Time
WIA-AA:5555-20061108182045	WIA-AA:5555	08-11-2006 18:20:45.758	08-11-2006 18:52:38.138

Snapshot Details per Service								
Minimum Own Elapsed Time (D HH:MM:SS.m): <input type="text" value="0.100"/>								
Reported Times are averages: <input type="checkbox"/>								
<input type="button" value="Refresh"/>								
Package	Service	Type	Count		Own Code		With Child Code	
			Calls	Errors	Elapsed	Spent	Elapsed	Spent
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc2:createDocument		53	53	1:03.308	0.100	1:03.308	0.100
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc1:singleService		12	12	18.177	0.050	28.550	0.150
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc3:looping		1	0	1.811	1.452	22.132	16.374
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc3:createStringList		8412	0	2.245	1.602	5.429	3.856
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc1:looping		1	1	0.141	0.000	0.491	0.020
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc3:order		8412	0	10.444	8.012	20.341	14.921
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc2:twoLevels		53	53	53.207	0.170	1:56.545	0.270
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc3:orderStringList		8412	0	0.522	0.401	0.522	0.401
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc5.amabr_issue_nr9_svcPortType:queryEntity		15	0	0.130	0.010	2.151	0.090
WiaServiceProfilerDemo	wia.demo.issprofiler.ui.run_uc:getUseCasesMetadata		109	0	0.211	0.040	0.221	0.050
AA_Issue_9_DB_Connections	amabr.issue.nr9.DB:newInstance		1	0	0.581	0.010	0.581	0.010
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc3:orderNumericList		8412	0	1.394	0.991	1.394	0.991
AA_Issue_9_DB_Connections	amabr.issue.nr9.DB:getName		14	0	0.150	0.000	0.150	0.000
WiaServiceProfilerDemo	wia.demo.issprofiler.usecases.uc2:looping		1	1	0.251	0.000	0.541	0.010
WiaServiceProfilerDemo	wia.demo.issprofiler.ui.run_uc:triggerUseCase		108	0	0.170	0.040	0.210	0.080
WiaServiceProfilerDemo	wia.demo.issprofiler.util:randomString		168240	0	3.184	2.253	3.184	2.253
WiaServiceProfilerDemo	wia.demo.issprofiler.config:get		9697	0	0.120	0.120	0.120	0.120
WiaServiceProfilerDemo	wia.demo.issprofiler.ui.run_uc:getMenuMetadata		2	0	0.461	0.000	0.461	0.000
<b>Grand Totals:</b>			<b>2:36.507</b>	<b>15.252</b>	<b>4:26.331</b>	<b>39.697</b>		

Figure 6 - The View Per Service report

All timing values are displayed in the data format explained in [Figure 7 \(below\)](#) with the leading zeros suppressed with exception to values smaller than one second.

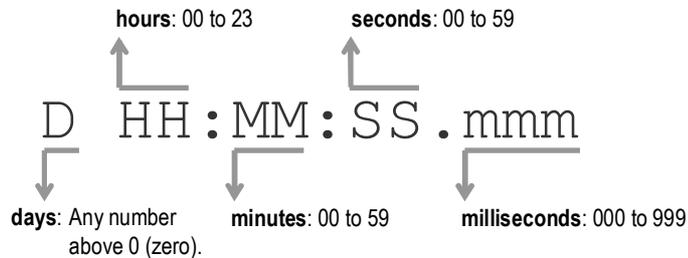


Figure 7 - Timing display format

The report can be refreshed with data for the most current snapshot by pressing the **[Refresh]** button. The amount of services listed can be further narrowed by setting a value in the accompanying text box to exclude **Services** for which the **Own Elapsed Time** is less than the value you entered. As all the other timing values the format explained in [Figure 7 \(above\)](#) should be used, and any leading zeros are still not needed.

By default, the presented timings are accumulated for the each presented service call count. However, these may be presented as **Averages Times** per call, by checking the appropriate checkbox.

The table can be sorted by any of the columns on the **Snapshot Details per Service** part of the report, including the **Service Type**.

### 3.3.2 Code Coverage

The **Code Coverage** report ([Figure 8, below](#)) is another example of an **Analysis Tool**.

This one presents a view over the **Snapshot** data reporting the percentage of **Services** in a **Package** that have effectively been run... while the profile was working.

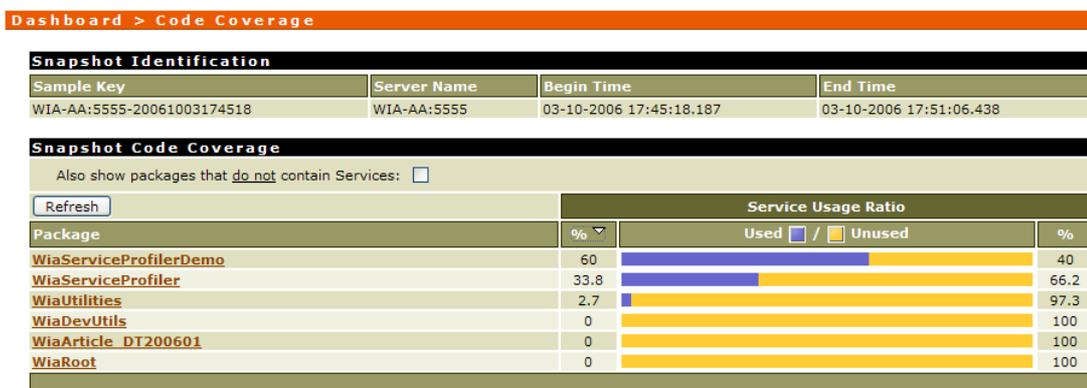


Figure 8 - The Code Coverage report

The table can be sorted by clicking on the corresponding column header on any of the detail columns, with exception to the percentage bar.

By default, only **Packages** that contain **Services** are listed. Even though this is a reasonable invariant, the complete list of **Packages** can be viewed by checking the appropriate checkbox and

clicking the **[Refresh]** button. All those that do not contain **Services** are so identified and their coverage percentage is suppressed. On the other hand, **Packages** which do contain **Services** can be further inspected by clicking on the link over its name. This will jump to a page (*Figure 9, below*) where all **Services** of the **Package** are listed with an indication of whether is has been run or not.

Dashboard > Code Coverage > Package Services

[Back to Code Coverage](#)

**Services From Package 'WiaServiceProfilerDemo'**

Service Name	Used?
wia.demo.issprofiler.usecases.uc1:looping	Yes
wia.demo.issprofiler.config:get	Yes
wia.demo.issprofiler.usecases.uc1:singleService	Yes
wia.demo.issprofiler.usecases.uc3:order	Yes
wia.demo.issprofiler.usecases.uc3:createStringList	Yes
wia.demo.issprofiler.util:randomString	Yes
wia.demo.issprofiler.usecases.uc3:orderStringList	Yes
wia.demo.issprofiler.usecases.uc3:orderNumericList	Yes
wia.demo.issprofiler.usecases.uc3:looping	Yes
wia.demo.issprofiler.ui.run_uc:getMenuMetadata	Yes
wia.demo.issprofiler.usecases.uc2:looping	Yes
wia.demo.issprofiler.usecases.uc2:twoLevels	Yes
wia.demo.issprofiler.usecases.uc2:createDocument	Yes
wia.demo.issprofiler.ui.run_uc:getUseCasesMetadata	Yes
wia.demo.issprofiler.config:list	Yes
wia.demo.issprofiler.ui.run_uc:triggerUseCase	Yes
wia.demo.issprofiler.config:set	Yes
wia.demo.issprofiler.util:doThreadInvoke	Yes
wia.demo.issprofiler.usecases.uc2:zip	No
wia.demo.issprofiler.admin:initialize	No
wia.demo.issprofiler.usecases.uc2:unzip	No
wia.demo.issprofiler.util:getThisPackageHomeDir	No
wia.demo.issprofiler.config:loadConfig	No
wia.demo.issprofiler.util:sleep	No
wia.demo.issprofiler.usecases.uc2.wsc:processEntry	No
wia.demo.issprofiler.usecases.uc2.wsc.uc2:processEntry	No
wia.demo.issprofiler.usecases.uc2:sleep	No
wia.demo.issprofiler.admin:startup	No
wia.demo.issprofiler.usecases.uc2:random	No
wia.demo.issprofiler.util:randomNumber	No

Figure 9 - The Code Coverage service usage detail

### 3.4 Extensibility

The **Service Profiler** core functionality is to gather raw information about the running **Services**. From that information, **Analysis Tools** may be (and are) implemented. However, to avoid limiting the analysis of the data to the tools included functionality, the **Export** of the **Snapshot** is provided.

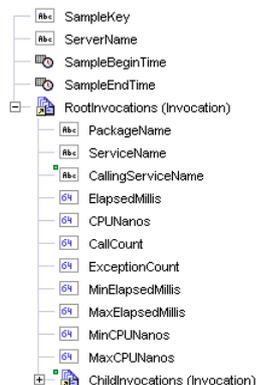


Figure 10 - The Snapshot as a Document structure

A Snapshot is a tree structure (see Figure 10, above) with common information on the root node plus the entire top-level Services, each of them being a node and with (possible) child nodes in them representing the Services they called. These child nodes can also have their own child nodes in them representing the Services they called... and so on.

The Snapshot can currently be exported as an XML file or a CSV (Figure 11, below).

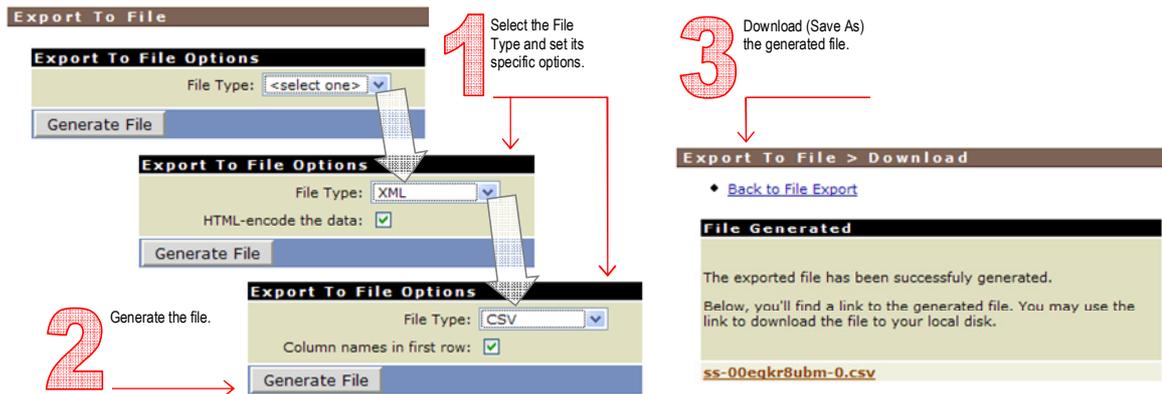


Figure 11 - Exporting Snapshot as a file

Beyond the export as file feature, and closer to the Integration Server environment, the Service Profiler provides a public interface (API) of Services and Document Types that can be used by any external IS Service to request Snapshots and process them.

## APPENDIX A GLOSSARY

Item	Definition
<b>Analysis Tool</b>	A dashboard view of the <b>Snapshot</b> data organized in a way that reveals relations on the collected data, either interactively or in the form of a report.
<b>Snapshot</b>	A Snapshot is an instant sample of the data structures internal to the <b>Service Profiler</b> .
<b>Time Elapsed</b>	This is the absolute time that has elapsed as measured by an external chronograph, or even on your wrist watch.
<b>Time Spent</b>	<p>In opposition to <b>Time Elapsed</b>, this value measures only the time the <b>Service</b> was in the <b>CPU</b> doing real work. The <b>CPU</b> is being used by more than one thread. The scheduler slices the availability of the <b>CPU</b> by giving each thread an opportunity to run. This value counts the time the <b>Service</b> spend effectively using the <b>CPU</b>.</p> <p>A <b>Service</b> may have a <b>Time Elapsed</b> much larger than the <b>Time Spent</b> if it is given little opportunity to run. This can be caused by:</p> <ul style="list-style-type: none"> <li>• A thread with much higher priority hogging the <b>CPU</b> availability;</li> <li>• A badly dimensioned system with not enough threads allocated, resulting in an activity of context switching;</li> <li>• Not enough resources (<b>RAM</b>, network bandwidth, etc.) causing the <b>Service</b> to spend most of the time waiting;</li> <li>• Etc.</li> </ul>
...	...