Add-ons performance analysis using .NET Profiler tool

Summary

In the article B1TE: B1 Test Environment tools a set of profiling tools for SAP B1 add-ons is presented. You can use these tools to profile the usage of B1 interfaces by an add-on solution.

In the present article we will concentrate in the .NET Profiler tool (one of the B1TE tools) and its potential in the performance tuning phase of development

By: SAP B1 Solution Architects
Date: December, 21 2005

Table of Content

Introduction

The SAP Business One SDK provides several programming interfaces to build your own add-on solutions. The use of these SDK interfaces needs you to follow some rules to guaranty the correct execution of your add-on. The .NET Profiler can help you to identify possible performance problems by tracing all UI-API and DI-API methods called from a .NET application and giving you the time spent on each SDK call.

In this article we will show you how to use the .NET Profiler to analyze the performances of your solution and how to interpret the results.

We assume you have B1TE already installed and you know how to run the .NET Profiler tool, if this is not the case please install B1TE and read carefully the sections presenting the .NET Profiler application.

Information profiled

.NET Profiler is able to profile all .NET calls and show very precious information about each call. The information profiled depends on the configuration applied, so let's have a look first to the configuration of the .NET Profiler.

Configuration

To configure .NET Profiler we can use filters and rules.

Filters

Filters allow you to decide which classes and methods are profiled by .NET Profiler.
Several filters are packaged with the application:

- Only DI API calls.filter profiling only DI API plus OleDb and Odbc classes
- Only UI API calls.filter profiling only UI API classes
- OnlyB1SDKcalls.filter profiling UI and DI API classes
- All B1 SDK calls.filter profiling UI and DI API classes plus OleDb and Odbc classes

You can also define new filters by selecting the classes/methods you are interested in (Figure 7).

Rules

.NET Profiler includes a rules file per each B1 version. This rules files check against API compatibility and some best practices programming rules. You can add new rules the existing ones by editing the rules files.
The default rules will be used by ICC for partner Add-on solution certification process. Please do not modify the rules except for specific or must cases.

By default "All B1 SDK calls" filter and the rules file corresponding to your version are set.

Execution

.NET Profiler profiles .NET applications while they are running; there is no need to instrument your add-on code. The only special configuration you need is to have .NET Framework 1.1 installed in your machine.

While running, .NET Profiler will list all calls corresponding to the classes and methods selected in the configured filter and will add comments regarding the application of the rules.

The information shown for each call is shown in Figure 1:

- The time spent since the beginning of the collection of data (“Time”)
- The Win32 Process Id of the process executing the profiled function call (“PID”)
- The Win32 Thread Id of the thread executing the profiled function call (“TID”)
- Whether the call belongs to DI or UI (“B1”)
- The name of the class to whom the called method belongs (“Type”)
- The name of the called method (“Call”)
- In case of a UI or DI method call, the elapsed time to complete the call (“Elapsed”)
- Description of a possible issue in the call related to a loaded set of rules when relevant (“Issue”)

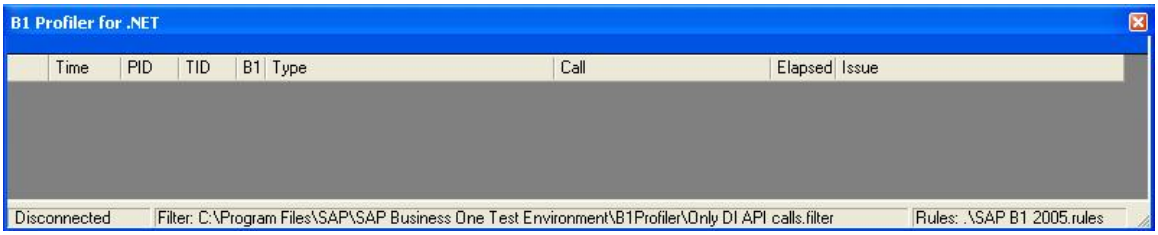


Figure 1: Add-on form

In this article we will focus our attention in the performances and we will mainly revise the information about the UI/DI classes and methods called, together with the elapsed time needed to complete the call.

By having a look to the elapsed time on the different called methods we can easily identify which methods take longer to be completed. We can use this information to observe possible programming mistakes and to try to optimize the time expend on each action.

Excel format

.NET Profiler offers the possibility of saving all profiled calls into csv files; you only have to call the option “Save all logs” and specify the format you want to have (comma separated values or semi colon separated values). Once the .csv file created you can use Excel to visualize it, easily analyze the data and customize the presentation of the results.

Let’s explain the usability of the .NET Profiler with a sample having bad performances; we observed this problem in a partner add-on solution implementation.

Sample

Description

Let's suppose we want to develop an application able to choose several Business Partners in a client database and update some properties of the chosen Business Partners.
Next figure shows our solution main form.

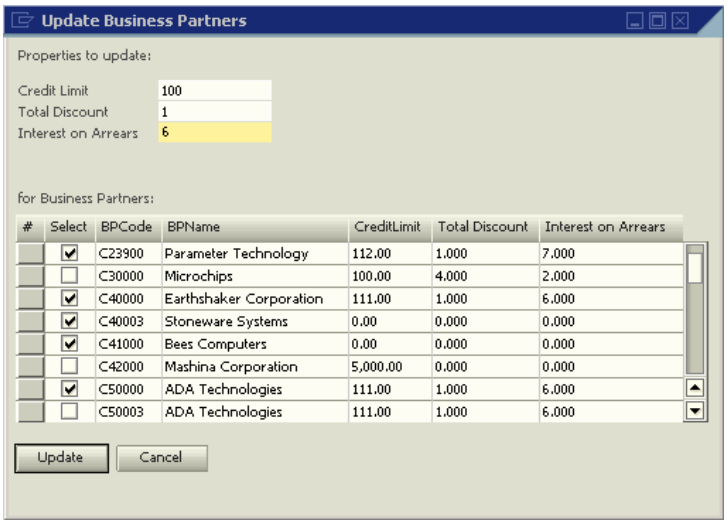


Figure 2: Add-on form

In the form we can see three static/edit texts presenting three properties of the Business Partners object (Credit Limit, Total Discount and Interest on Arrears) and a matrix with the list of Business Partners of the current database. The user can select one/several Business Partners and set the same Credit Limit, Total Discount and Interest on Arrears properties values for all of them by clicking on the Update button. After the user clicks on the Update button all the selected Business Partners will be updated in the database with the properties set in the header of the form (**Figure 8: Incorrect code sample** shows the code implementing the Update button action)

This solution is working fine; it does what it must do. But the performances are not good, when the user selects several Business Partners to update the time needed to complete the Update action is too long.
To try to find out where the problem can come from we will analyze the add-on by using the .NET Profiler tool.

How to profile our solution

We need to run the .NET Profiler application at the same time as our add-on.
Once the .NET Profiler running we set a specific filter in order to better record the calls we need to profile. In this case we know the performance problem is coming from the DI API, we will set then the filter "Only DI calls.filter" profiling only DI API calls (this filter is provided inside the .NET Profiler application).

Now that the filter is set we can start profiling our add-on, Start Profiling menu.

Once .NET Profiler activated we can call the Update function in our add-on to see which objects and methods are profiled by the .NET Profiler.

Results

Next figure shows the results obtained:

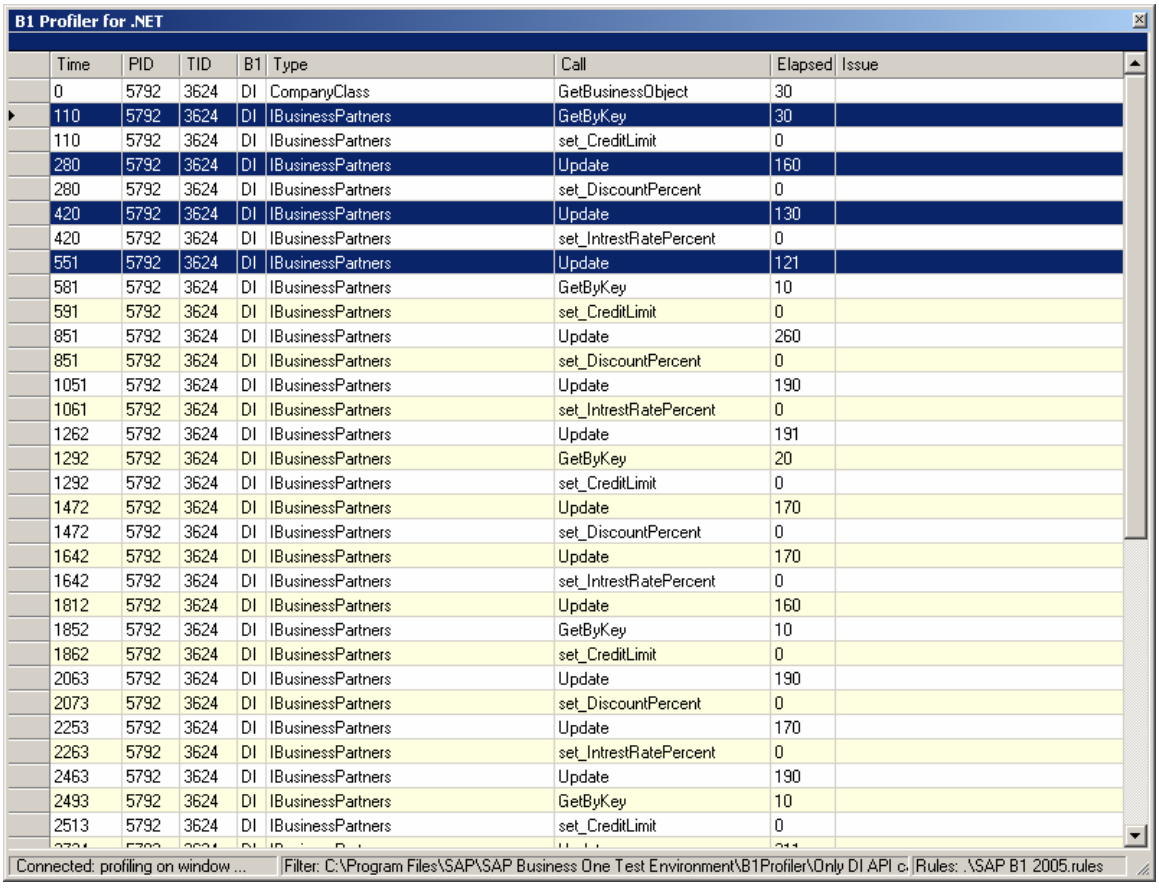


Figure 3: .NET Profiler window

In the list of calls we can see the methods consuming the most are GetBusinessObject, GetByKey and Update. We cannot avoid calling these methods according to SDK rules, but we can maybe optimize them.

If we look at the calls in more detail we can see that between two calls to the GetByKey method there are three calls to the Update method, one per property set. This implies for every Business Partner updated we call 3 times the Update method, why that? We only need to call the Update method once per Business Partner instance, the instance of the object keeps the properties modified in memory and calling Update after all properties are set in the object will update the information stored in the database. We can then reduce the time consumed by almost three times!

Next figure shows the execution of the same add-on but this time the add-on will only call Update once per each Business Partner instance (**Figure 9** shows the correct code of the Update action).

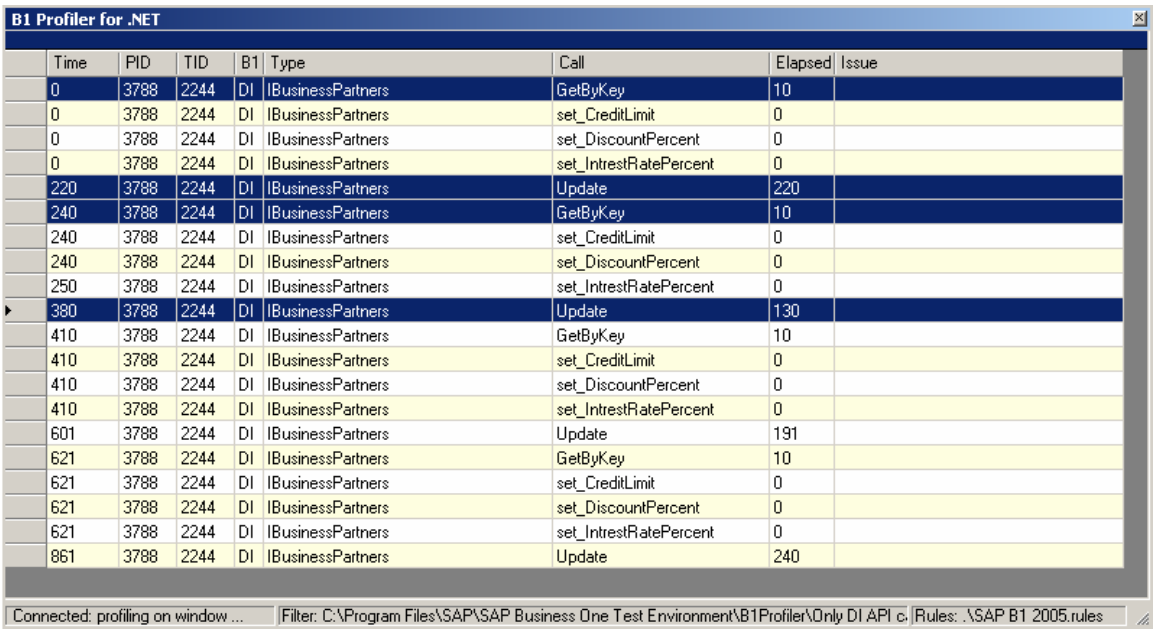
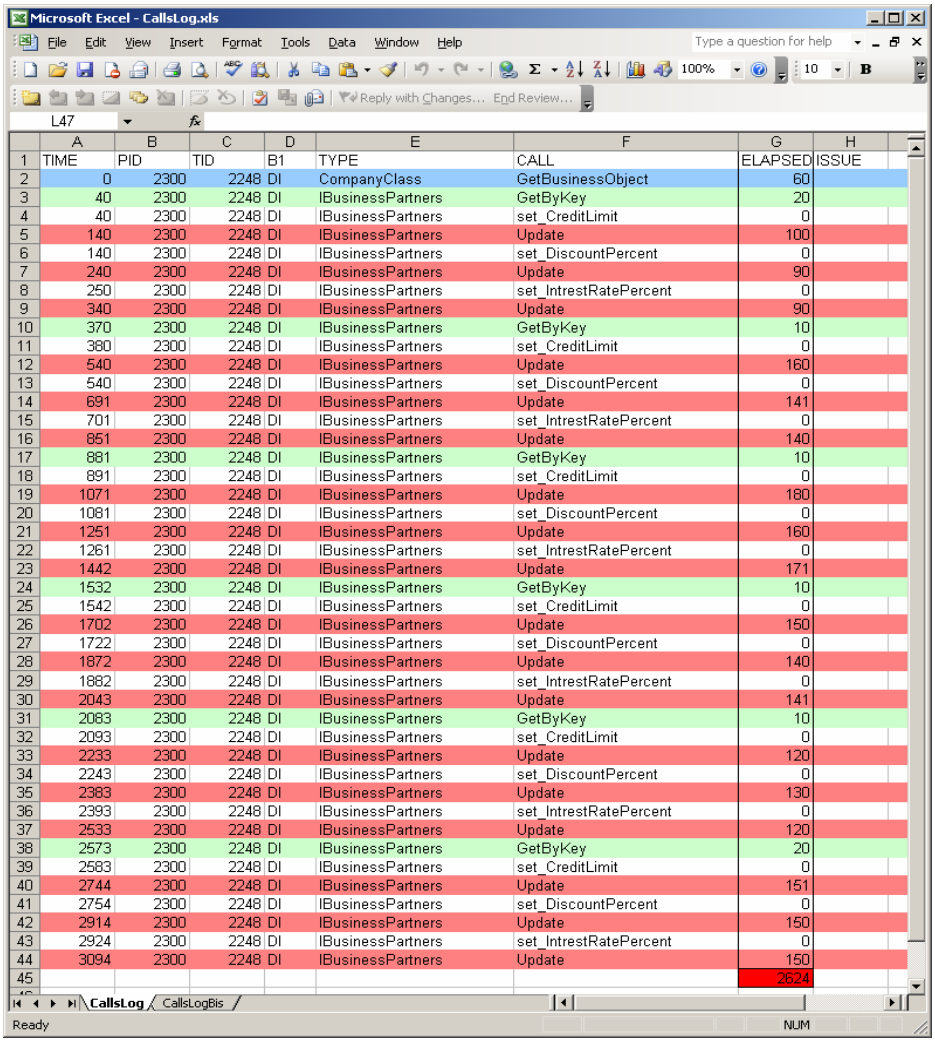


Figure 4: .NET Profiler window

We can clearly see that the Update call takes more or less the same time; we have then reduced the time consumed by three.

Excel format

We can also have a look in Excel format to the results obtained by using the option “Save all logs”. Next figure shows the same calls list in excel where we can customize the presentation.



Microsoft Excel - CallsLog.xls

File Edit View Insert Format Tools Data Window Help

Type a question for help

100% 10

Reply with Changes... End Review...

L47

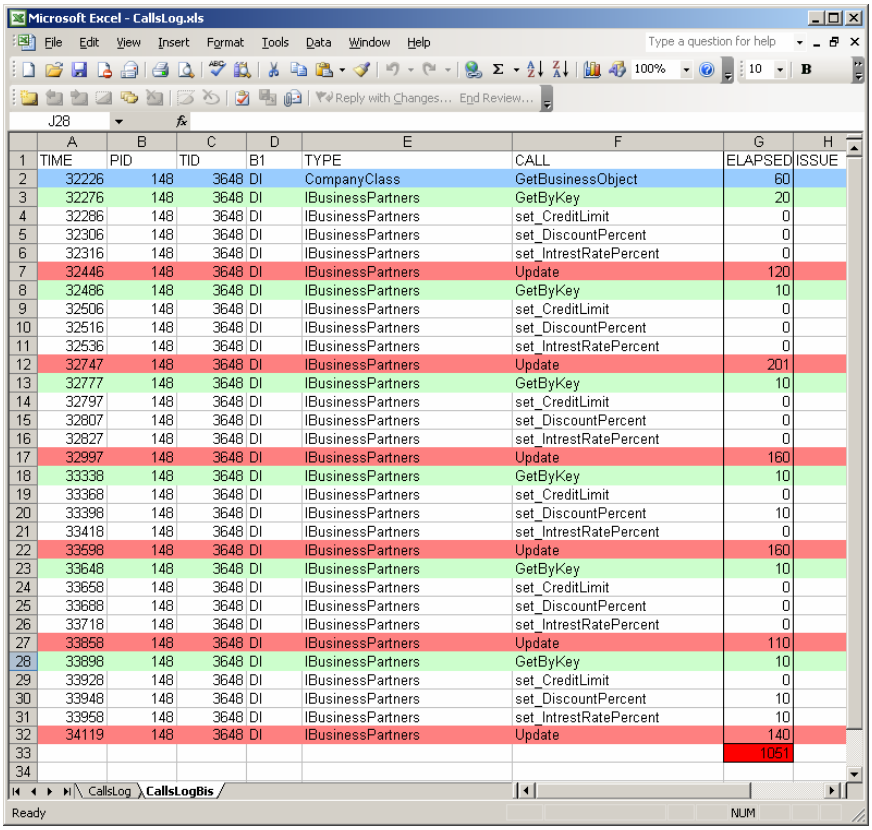
	A	B	C	D	E	F	G	H
1	TIME	PID	TID	B1	TYPE	CALL	ELAPSED	ISSUE
2	0	2300	2248	DI	CompanyClass	GetBusinessObject	60	
3	40	2300	2248	DI	IBusinessPartners	GetByKey	20	
4	40	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
5	140	2300	2248	DI	IBusinessPartners	Update	100	
6	140	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
7	240	2300	2248	DI	IBusinessPartners	Update	90	
8	250	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
9	340	2300	2248	DI	IBusinessPartners	Update	90	
10	370	2300	2248	DI	IBusinessPartners	GetByKey	10	
11	380	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
12	540	2300	2248	DI	IBusinessPartners	Update	160	
13	540	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
14	691	2300	2248	DI	IBusinessPartners	Update	141	
15	701	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
16	851	2300	2248	DI	IBusinessPartners	Update	140	
17	881	2300	2248	DI	IBusinessPartners	GetByKey	10	
18	891	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
19	1071	2300	2248	DI	IBusinessPartners	Update	180	
20	1081	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
21	1251	2300	2248	DI	IBusinessPartners	Update	160	
22	1261	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
23	1442	2300	2248	DI	IBusinessPartners	Update	171	
24	1532	2300	2248	DI	IBusinessPartners	GetByKey	10	
25	1542	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
26	1702	2300	2248	DI	IBusinessPartners	Update	150	
27	1722	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
28	1872	2300	2248	DI	IBusinessPartners	Update	140	
29	1882	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
30	2043	2300	2248	DI	IBusinessPartners	Update	141	
31	2083	2300	2248	DI	IBusinessPartners	GetByKey	10	
32	2093	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
33	2233	2300	2248	DI	IBusinessPartners	Update	120	
34	2243	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
35	2383	2300	2248	DI	IBusinessPartners	Update	130	
36	2393	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
37	2533	2300	2248	DI	IBusinessPartners	Update	120	
38	2573	2300	2248	DI	IBusinessPartners	GetByKey	20	
39	2583	2300	2248	DI	IBusinessPartners	set_CreditLimit	0	
40	2744	2300	2248	DI	IBusinessPartners	Update	151	
41	2754	2300	2248	DI	IBusinessPartners	set_DiscountPercent	0	
42	2914	2300	2248	DI	IBusinessPartners	Update	150	
43	2924	2300	2248	DI	IBusinessPartners	set_IntrestRatePercent	0	
44	3094	2300	2248	DI	IBusinessPartners	Update	150	
45							2624	

CallsLog CallsLogBis

Ready NUM

Figure 5: Excel format of profiled calls, wrong sample

With this format we can easily calculate the total time spent on B1 SDK calls, around 2600 milliseconds.
Let's have a look to the same sample but correctly using the SDK Update method, see next figure.



Microsoft Excel - CallsLog.xls

	A	B	C	D	E	F	G	H
1	TIME	PID	TID	B1	TYPE	CALL	ELAPSED	ISSUE
2	32226	148	3648	DI	CompanyClass	GetBusinessObject	60	
3	32276	148	3648	DI	IBusinessPartners	GetByKey	20	
4	32286	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
5	32306	148	3648	DI	IBusinessPartners	set_DiscountPercent	0	
6	32316	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	0	
7	32446	148	3648	DI	IBusinessPartners	Update	120	
8	32486	148	3648	DI	IBusinessPartners	GetByKey	10	
9	32506	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
10	32516	148	3648	DI	IBusinessPartners	set_DiscountPercent	0	
11	32536	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	0	
12	32747	148	3648	DI	IBusinessPartners	Update	201	
13	32777	148	3648	DI	IBusinessPartners	GetByKey	10	
14	32797	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
15	32807	148	3648	DI	IBusinessPartners	set_DiscountPercent	0	
16	32827	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	0	
17	32997	148	3648	DI	IBusinessPartners	Update	160	
18	33338	148	3648	DI	IBusinessPartners	GetByKey	10	
19	33368	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
20	33398	148	3648	DI	IBusinessPartners	set_DiscountPercent	10	
21	33418	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	0	
22	33598	148	3648	DI	IBusinessPartners	Update	160	
23	33648	148	3648	DI	IBusinessPartners	GetByKey	10	
24	33658	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
25	33688	148	3648	DI	IBusinessPartners	set_DiscountPercent	0	
26	33718	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	0	
27	33858	148	3648	DI	IBusinessPartners	Update	110	
28	33898	148	3648	DI	IBusinessPartners	GetByKey	10	
29	33928	148	3648	DI	IBusinessPartners	set_CreditLimit	0	
30	33948	148	3648	DI	IBusinessPartners	set_DiscountPercent	10	
31	33958	148	3648	DI	IBusinessPartners	set_IntrestRatePercent	10	
32	34119	148	3648	DI	IBusinessPartners	Update	140	
33							1051	
34								

Figure 6: Excel format of profiled calls, correct sample

In this case we only one call once the Update method per Business Partner instance (see Figure 6) we can see the time has been divided by almost 3 times!

Conclusion

In this article we have shown how to use the .NET Profiler tool to find out possible performance problems in add-ons developed in a .NET Environment.
You can also use .NET profiler and other tools provided in the B1TE package for additional analysis/tests of your add-on solution, please have a look to the B1TE package to have more information about the Business One Test Environment capabilities.

Figures

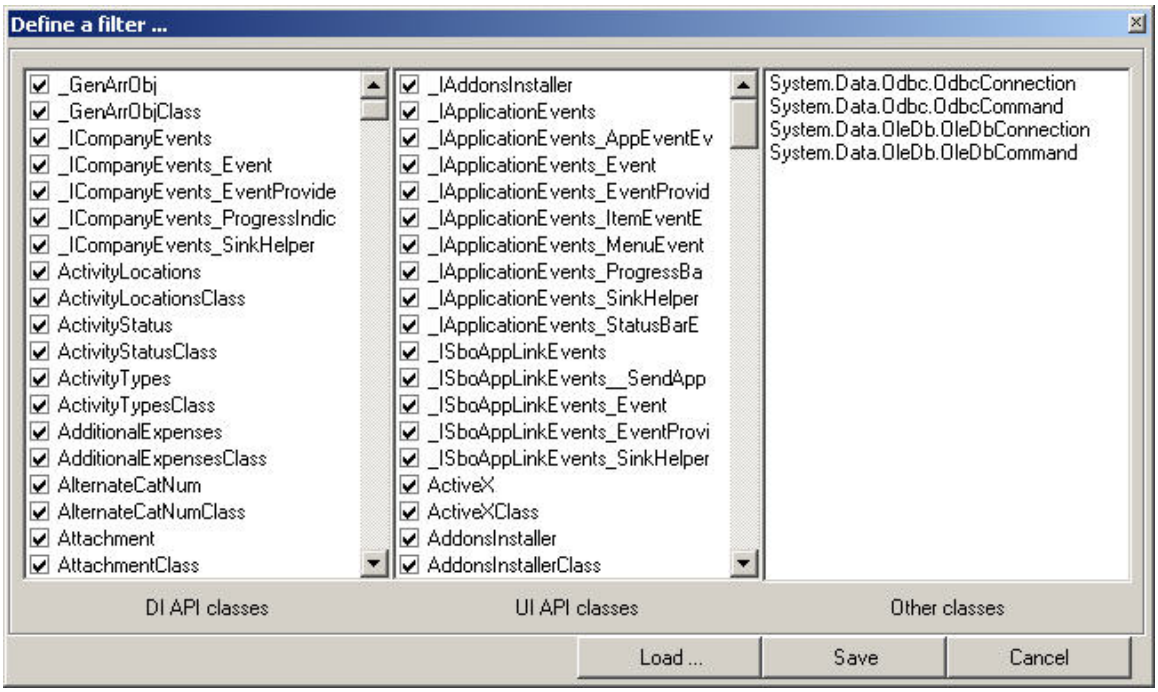


Figure 7: How to define a filter in the .NET Profiler application.

It is possible to select DI API methods, UI API methods and add other classes to the filter. In this case the filter contains also ODBC and OleDB classes used to access directly the database in order to track these illegal accesses in the add-ons.
Pay attention to the extra classes you add, classes called too often can block the .NET Profiler application.

```
// Please do not use this code as a model, it shows an incorrect use of the SDK!

BusinessPartners oBP;

oBP = B1Connections.diCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners);

// Go over the list of selected BPs
DBDataSource dbds = form.DataSources.DBDataSources.Item("OCRD");
for (int i = 1; i < dbds.Size; i++)
{
  //Get Business Partner
  cBPCode  = dbds.GetValue("CardCode", i - 1);
  oBP.GetByKey(cBPCode);

  // Update CreditLimit
  oBP.CreditLimit = double.Parse(cLimitVal);
  iErr = oBP.Update();
  if (iErr != 0)
    B1Connections.theAppl.MessageBox("Error BP " + cBPCode + ": " +
      B1Connections.diCompany.GetLastErrorDescription(), 1, "", "", "");

  // Update Total Discount
  oBP.DiscountPercent = double.Parse(cDiscPercent);
  iErr = oBP.Update();
  if (iErr != 0)
    B1Connections.theAppl.MessageBox("Error BP " + cBPCode + ": " +
      B1Connections.diCompany.GetLastErrorDescription(), 1, "", "", "");

  // Update Interest on Arrears
  oBP.IntrestRatePercent = double.Parse(cIntArrVal);
  iErr = oBP.Update();
  if (iErr != 0)
    B1Connections.theAppl.MessageBox("Error BP " + cBPCode + ": " +
      B1Connections.diCompany.GetLastErrorDescription(), 1, "", "", "");
}
```

Figure 8: Incorrect code sample

C# code used to implement the Update action on the selected Business Partners.
It calls the Update method per every property modified; please don't take it as a good sample!

```
BusinessPartners oBP;

oBP = B1Connections.diCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners);

// Go over the list of selected BPs
DBDataSource dbds = form.DataSources.DBDataSources.Item("OCRD");
for (int i = 1; i < dbds.Size; i++)
{
  //Get Business Partner
  cBPCode  = dbds.GetValue("CardCode", i - 1);
  oBP.GetByKey(cBPCode);

  // Update CreditLimit
  oBP.CreditLimit = double.Parse(cLimitVal);

  // Update Total Discount
  oBP.DiscountPercent = double.Parse(cDiscPercent);

  // Update Interest on Arrears
  oBP.IntrestRatePercent = double.Parse(cIntArrVal);

  // Call Update on BP only once per each BP instance
  iErr = oBP.Update();
  if (iErr != 0)
    B1Connections.theAppl.MessageBox("Error BP " + cBPCode + ": " +
      B1Connections.diCompany.GetLastErrorDescription(), 1, "", "", "");
}
```

Figure 9: Correct code sample

Correct C# code used to implement the Update action on the selected Business Partners.
It calls the Update method only once per each Business Partner instance.

Disclaimer & Liability Notice

This document may discuss sample coding, which does not include official interfaces and therefore is not supported. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing of the code and methods suggested here, and anyone using these methods, is doing it under his/her own responsibility.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of the technical article, including any liability resulting from incompatibility between the content of the technical article and the materials and services offered by SAP. You agree that you will not hold SAP responsible or liable with respect to the content of the Technical Article or seek to do so.