OLAP Data Access Logging for Voyager, Web Intelligence, OLAP Intelligence and Crystal Reports

Applies to:

Voyager, Crystal Reports, Web Intelligence and OLAP Intelligence in versions XI Release 2, XI 3.0 and XI 3.1

Summary

Logging can be a key diagnostic tool when trouble shooting problems. Several BusinessObjects products share a data access component to access OLAP data. This document describes the logging facilities available in this data access component and how to configure them.

Author: Reuben Cox

Company: SAP

Created on: 18th November 2009

Author Bio

I have been an employee at Business Objects (previously Crystal Decisions) since 2001. I initially started off as a developer on the OLAP data access team working on data drivers for Microsoft Analysis services and later SAP BW and Essbase. I've have worked on the OLAP data consumption for various BusinessObjects products including OLAP Intelligence, Voyager, Web Intelligence and Crystal Reports. Since 2007, I have been working as a program manager writing feature specifications for upcoming releases and assisting customers with their OLAP deployments.

Table of Contents

The OLAP Data Access component

A number of BusinessObjects products all share a data access component to access OLAP data. Although in any one particular release of the BusinessObjects suite, different products may be using different versions of the OLAP data access component they all share a similar logging facility.

The OLAP Data Access component is written in C++ and all logging facilities are controlled through the registry. On a Windows platform, this in the 32-bit Windows registry. On a UNIX platform, this is through the pseudo-Windows registry that BusinessObjects products use. This is different than the MainWin registry that some Crystal Reports UNIX users may be familiar with.

In the XI R2, XI 3.0 and XI 3.1 versions of BusinessObjects software there are two different evolutions of the OLAP data access component in existence. There is the legacy version known as SOFA (Simple OLAP Framework Architecture) and the newer version known as MDA (Multidimensional Data Access). The table below shows which products use which versions.

	XI R2	**XI 3.0 and XI 3.1**
OLAP Intelligence	SOFA 11.5	N/A
Web Intelligence	SOFA 11.5	MDA 12
Crystal Reports	SOFA 11.5	SOFA 12
Voyager	MDA 11.5	MDA 12

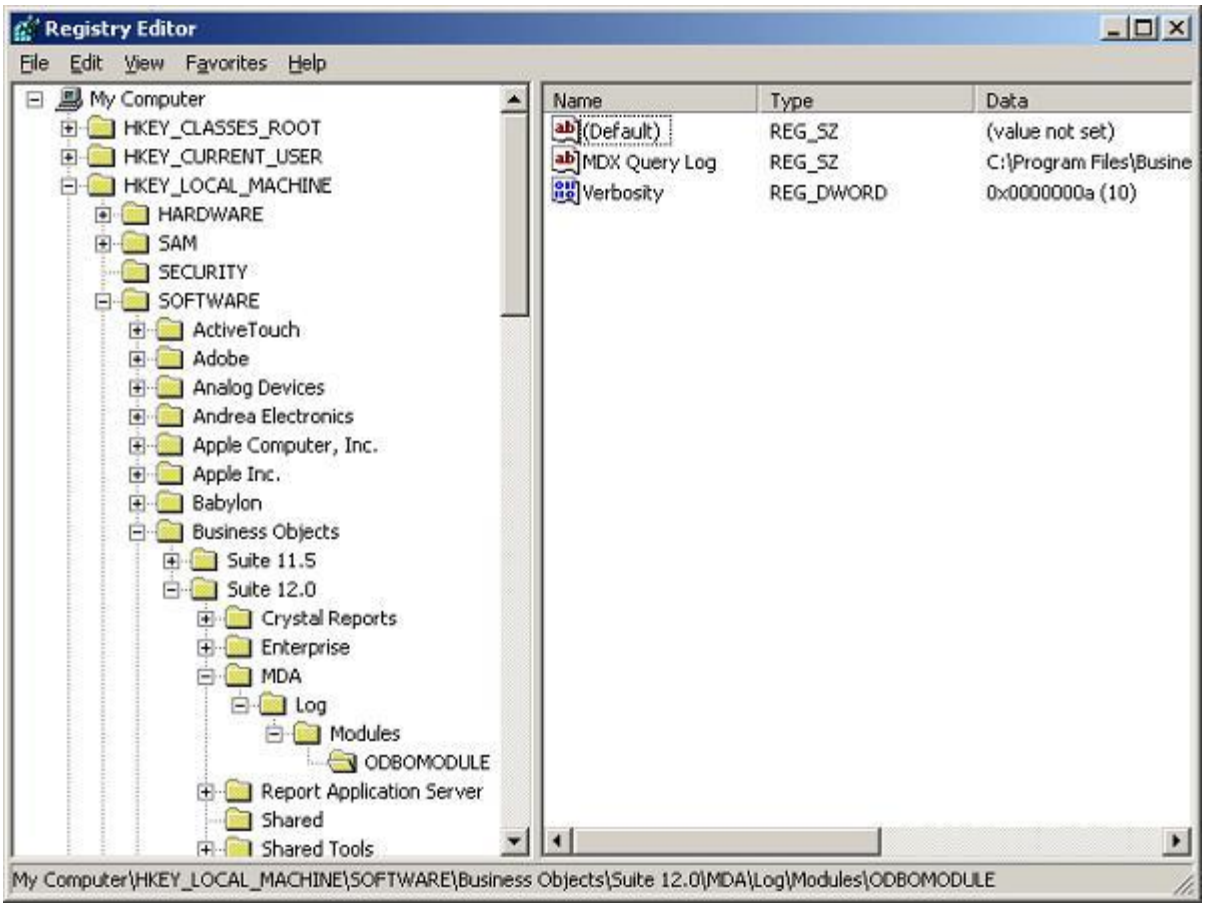
It is important to be aware of which version of the OLAP Data Access component is being used as it affects precisely where the registry settings are located and what logging facilities are available. It is also important to know that the settings for one version of the component will affect all products using that version of the component. For example, if logging is set up for SOFA 11.5, any instance of OLAP Intelligence, Web Intelligence or Crystal Reports on that particular machine will use those settings.

A note on example registry settings

As the OLAP Data Access logging is controlled through the registry this document contains many examples of registry settings. These examples are presented in the form they would appear in a **reg** file (a file ending with the extension ".reg") generated by exporting a section of the registry. The examples will consist of **keys** (identified by starting and ending with square brackets "[ ]") and **values** (name value pares with the name in quotes, followed by an equals, followed by the value in quotes). The keys correspond to folders (when viewing the registry through the **regedit** windows application) and the values are items in those folders. OLAP Data Access logging only makes use of **string** and **DWORD** value types.

The following example is shown as it would appear in this document and the corresponding screen shot of the registry.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"Verbosity"=dword:0000000a
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\ODBOMDX.log"
```



This example shows a key or folder called **ODBOMODULE**. There are two values. One string value **MDX Query Log** and a DWORD value **Verbosity**.

To quickly enter the example into the registry, the text of the example can be copied into a text file. The file name should be changed to end with ".reg". Then the file can be double clicked to enter the values into the registry.

In order for any changes to the registry to take effect, any processes that use the OLAP Data Access component have to be restarted.

Registry Location

Logging is controlled through the registry. The logging settings are located in and below a root registry key for the particular version of the OLAP Data Access component that is being used. The locations of the logging roots are given in the following table.

Version	Root location
SOFA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP Intelligence\OCCA(o)\Log]
SOFA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP Intelligence\OCCA(o)\Log]
MDA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\Log]
MDA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log]

Main settings

Underneath the root registry key are a number of registry values that control the main logging settings. Explanations are given here.

Values that are required to be present are indicated as being mandatory. Values not indicated as being mandatory are allowed to be absent from the registry.

Location

Value type: String

Mandatory: Yes

This indicates the location of a string resource DLL used by the logging system. This should be set up with a default install and should not need to be changed. The following table shows the values of this registry entry.

Version	Location
SOFA 11.5	"Location"="C:\\Program Files\\Business Objects\\OLAP Intelligence 11.5\\Bin\\"
SOFA 12	"Location"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise 12.0\\win32_x86\\"
MDA 11.5	"Location"="C:\\Program Files\\Business Objects\\common\\3.5\\bin\\"
MDA 12	"Location"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise 12.0\\win32_x86\\"

If this value is missing, contains the incorrect path, or the string resource DLL is missing then some logging statements may not appear correctly.

LogFile

Value type: String

Mandatory: Yes

This specifies name and location of the file to log to. The value of this setting can be changed to any valid file name and path. As an example here is the default value of this setting for MDA 12:

```
"LogFile"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\MDA.log"
```

This means that any logging will be done in the file **MDA.log** located in the folder **C:\Program Files\Business Objects\BusinessObjects Enterprise 12.0\Logging**

If this registry value is absent, then there will be no logging.

OverWrite

Value type: String

Mandatory: No

This determines whether a log file should be appended to or overwritten. A value of “Yes” overwrites the log file. A value of “No” appends to the log file. This value determines the behavior of a new process. If a process has been logging to a file already and has been idle for a while, the next time it logs it will append to the log file regardless of this setting. If the process was restated and the value was “Yes” then it would overwrite the log file.

Note If a process is already logging to the file and the value is set to ‘Yes’ and an additional process is started it may overwrite the current log.

An example entry would look as follows:

```
"OverWrite"="Yes"
```

AppendPID

Value Type: String

Mandatory: No

This setting appends the ID of the current process to the file name specified in the **LogFile** value. This value is useful if a machine has multiple processes that will be using the OLAP Data Access component. For example, you have multiple Voyager MDAS servers or a mixture of Voyager and Web Intelligence servers both accessing OLAP data. With the value of this property set to “Yes” all processes will log to their own log file. With the value set to “No” all processes will log to the same log file. If a server restarts and comes back with a different process ID and the setting is “Yes” it will now log to a different log file than it did before.

An example entry would look as follows:

```
"AppendPID"="Yes"
```

Verbosity

Value Type: DWORD

Mandatory: No

Verbosities are the controls which determine how much information is logged. Entering a verbosity value in the root **Log** folder specifies a global value for the entire OLAP Data Access component. Possible values are shown below.

Value	Description	Example registry entry
0	**Never**. Nothing is logged.	"Verbosity"=dword:00000000
1	**Critical**. Only critical errors are logged.	"Verbosity"=dword:00000001
2	**Serious**. All errors are logged.	"Verbosity"=dword:00000002
3	**Warning**. Errors and warnings are logged	"Verbosity"=dword:00000003
4	**Info**. In addition to errors and warnings, other useful information is logged.	"Verbosity"=dword:00000004
5	**Debug**. Error, warnings, information and some additional debugging information is logged.	"Verbosity"=dword:00000005
10	**Debug Extra**. Everything is logged out.	"Verbosity"=dword:0000000a

LogFormat

Value Type: String

Mandatory: No

It is possible to override the default format that entries are written to the log file. This is discussed in detail in a later section.

Modules and Components

The OLAP Data Access logging allows the level of information logged to be controlled two levels: modules and components. A module corresponds roughly to a single dll. A component is a class within that DLL. There are also non-DLL based modules that correspond to transversal functionality like memory management.

Each module or component has an associated verbosity. This means that different levels of logging can be set for different modules and components. Verbosity settings are optional for all modules and components. If no verbosity is set for a particular module or component the parent setting will be used. For example, if a particular component doesn't have a verbosity set, the setting used will be the parent module's setting. If a particular module does not have a setting it will use the global setting.

Finer grained verbosities take precedence over courser grained verbosities. So if there is a global, module and component setting, then the module setting takes precedence over the global, and the component setting takes precedence over the module setting.

Modules

To control logging using modules first a folder must be created under the root logging folder. The table below shows the folder for the different versions.

Version	Modules Folder
SOFA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP Intelligence\OCCA(o)\Log\Modules]
SOFA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP Intelligence\OCCA(o)\Log\Modules]
MDA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\Log\Modules]
MDA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules]

Under this folder, further folders need to be created: one for each module that needs a verbosity specified. The following table contains a non-exhaustive list of modules with a brief description of what the module corresponds to.

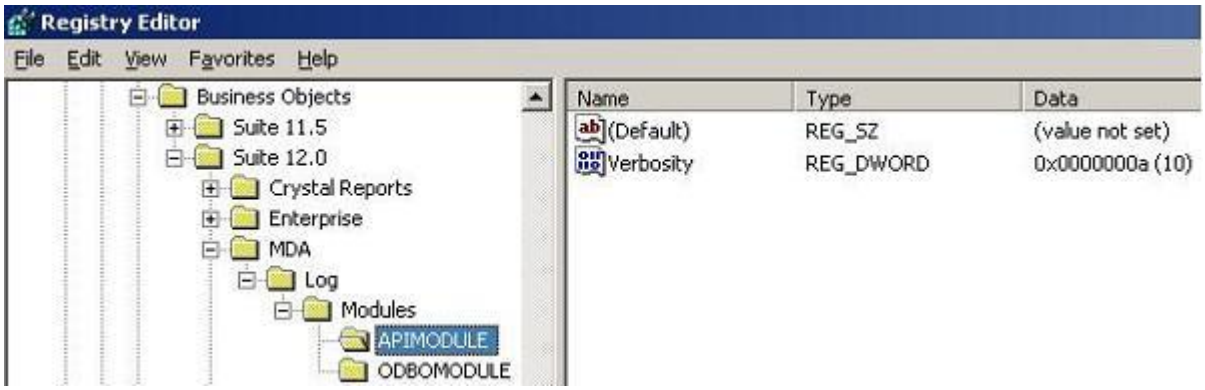
Name	Description	Relevant Versions
APIMODULE	Logs calls into the OLAP data access component's API	MDA 11.5 and 12
APISUPPORT	Information from the API support library	MDA 11.5 and 12
COMMONMODULE	Information common to all data providers	MDA 11.5 and 12
ESSBASEMODULE	Information from the Essabse data provider	All
JNIMODULE	Information from the component that links the OLAP data access components C++ API to its Java API via JNI	MDA 11.5 and 12
MDA_SUPPORT	Information from the general support library	MDA 11.5 and 12
ODBOMODULE	Information from the ODBO data provider for OLAP servers like Microsoft Analysis Services	All
ODBOPROVIDERMODULE	Logs errors encountered when trying to discover available ODBO provider on the system	MDA 11.5 and 12
ODBOSHAREDUTILITIES	Information about which ODBO providers are available on the system	All
RORMODULE	Web Intelligence/Universe specific component. Importantly logs out XML query specifications.	MDA 12

SAPMODULE	Information from the SAP BW data provider.	All
UTILITIES	Information about making connections	All
COMWRAP	Logs calls into the OLAP data access component's API	SOFA 11.5 and 12
COUTILITIES	Information from the data connection API	SOFA 11.5 and 12
RORFLATTENER	Web Intelligence/Universe specific component. Importantly logs out XML query specifications.	SOFA 11.5
SOFA_Common	Information common to all data providers	SOFA 11.5 and 12
support_dll	Information from the general support library	SOFA 11.5 and 12
MEMORY	Information about the creation, destruction and reference counting of objects	All
INTERFACE	Additional tracing information.	SOFA 11.5 and 12

To set the verbosity for a particular module a folder with the name of the module needs to be created under the **Modules** folder and then a DWORD value called verbosity. For example:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\APIMODULE]
"Verbosity"=dword:0000000a
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"Verbosity"=dword:00000005
```

The above will set the verbosity for the module **APIMODULE** to **10** and the verbosity for module **ODBOMODULE** to **5** for version MDA 12. Here is a screen shot of how this might look in the registry.



Components

Within each module there are a number of components. The components for each module are not listed here. However the names of particular components can easily be discovered by looking at an existing log file. For example, consider the three following lines from a log file:

```
2009-07-16T03:36:13.484: APIMODULE: CubeViewImpl: CubeViewImpl::toXML
2009-07-16T03:36:13.531: ODBOMODULE: ODBOMemberSet: Calling ODBOMemberSet::_getNumItems ...
2009-07-16T03:36:13.546: ODBOMODULE: ODBOCube: Query executed correctly
```

Each line starts with a timestamp followed by a colon and then a word that should be familiar from the list of modules given above. The first log line is from the `APIMODULE` and the second and third are from the `ODBOMODULE`. There is another colon after the module name. The next word is the name of the component. So the first line is being logged by the `CubeViewImpl` component, the second by the `ODBOMemberSet` component and the third by the `ODBOCube` component.

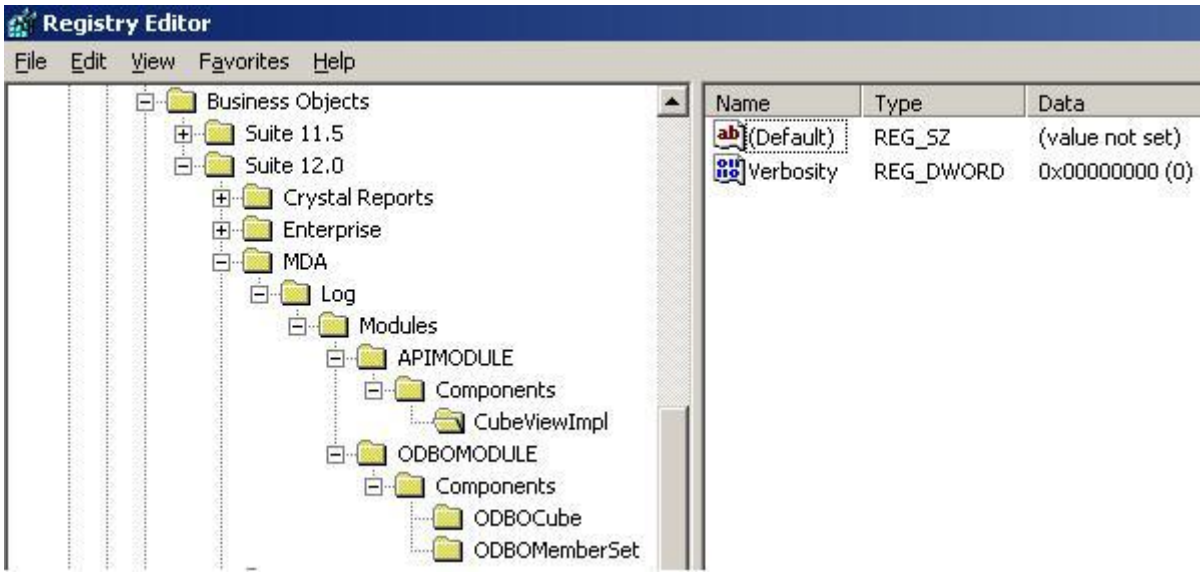
To specify the verbosity for the logging of a particular component first a folder called **Components** needs to be created under the specific module folder. Then a further folder with the name of the component needs to be created under the **Components** folder. Finally a DWORD value with the name Verbosity needs to be created in the folder named after the particular component.

Here is an example for the modules and components from the sample log lines above:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\APIMODULE]
"Verbosity"=dword:0000000a
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components]
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components\CubeViewImpl]
"Verbosity"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"Verbosity"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ODBOMODULE\Components]
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ODBOMODULE\Components\ODBOMemberSet]
"Verbosity"=dword:00000005
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ODBOMODULE\Components\ODBOCube]
"Verbosity"=dword:00000001
```

Here is `APIMODULE` is set to full verbosity but nothing will be logged for the `CubeViewImpl` component. There will be no logging from the `ODBOMODULE` module except for the `ODBOMemberSet` component which will log at a verbosity of 5 and the `ODBOCube` component which will log at a verbosity of 1.

Here is a screen shot of what this looks like in the registry:



API call timings

One of the pieces of information logged by the OLAP Data Access logging is the time it took for calls to the OLAP Data Access component API to complete. There are additional logging settings that can be utilized to control the logging of API call timings that make it easier, for example, to identify long running API calls which might point to potential bottlenecks.

The exact nature of the logging and the facilities to control it vary slightly between versions of the OLAP Data Access component.

MDA API Call Timing Behavior

Here is a list of points describing the behavior of API call timing logging in the MDA version of the OLAP Data Access component.

- API call timings are logged as part of the **APIMODULE**.
- They are logged at either a verbosity of **5** or **10**.
- Logging at a verbosity of **5** eliminates all API calls that complete in less than `0.0001` seconds.
- The API call timings are also logged to a specific component called **TIMER** so a log file could be configured to contain just the API call timings.
- Only times over a certain threshold are logged.
- By default the threshold is set to 0.000.

The threshold can be set by creating a DWORD called **Timer Threshold** in the **APIMODULE** folder and entering a value for the number of milliseconds. For example a value of 1000 will only log times that take over one second.

Note: In practice there is no difference to the amount of API calls logged out at verbosity 5 or 10. If the threshold it at its default setting of zero then times that are less than 0.0001 seconds tend to get rounded off to a value of zero so are not logged.

An example of what API call timing log statements look like is given here:

```
Tue Dec 01 16:40:42.698: APIMODULE: TIMER: Call to CubeViewImpl::query took 0.701 seconds

Tue Dec 01 16:42:07.675: APIMODULE: TIMER: Call to AxisResultSetImpl::getCount took 1.411
seconds

Tue Dec 01 16:42:09.090: APIMODULE: TIMER: Call to CellResultSetImpl::getCursor(UInt, UInt)
took 1.382 seconds
```

MDA Example registry settings

Here is an example for configuring the registry for capturing just API call timings for MDA 12.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\APIMODULE]

"Verbosity"=dword:00000000

"Timer Threshold"=dword:000001f4


[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components]


[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components\TIMER]

"Verbosity"=dword:00000005
```

This turns of all logging for **APIMODULE** at the module level. It sets the timer threshold to 500 milliseconds (500 is 1f4 in hexadecimal). It finally sets the **TIMER** component to log at verbosity 5.

If there are no other verbosity settings for other modules or components in the registry, this setting would log only OLAP Data Access API calls that took over half a second.

SOFA API Call Timing Behavior

Here is a list of points describing the behavior of API call timing logging in the SOFA version of the OLAP Data Access component.

- API call timings are logged as part of either the **COMWRAP** or the **COUTILITIES** modules
- They are logged at either a verbosity of **5** or **10**
- Logging at a verbosity of **5** eliminates all API calls that complete in less than `0.0001` seconds.
- Logging at verbosity **10** logs the start of an API call as well as the finish of a call.
- The API call timings are also logged to a specific component called **Timer** so a log file could be configured to contain just the API call timings.
- There is no threshold setting for SOFA API call times unlike MDA API calls

Note Unlike the MDA version of the API timer logging, logging at verbosity **5** or **10** does make a difference. Logging at verbosity **10** will log two lines for every API call regardless of how long the API call took to complete. The first line logged will indicate the beginning of the API call. For example:

```
Thu Dec 03 14:18:03.659: COMWRAP: Timer: Calling Level::CLevel::get_Caption ...
```

The second log line will indicate the completion of an API call. For example:

```
Thu Dec 03 14:18:03.675: COMWRAP: Timer: Call to Level::CLevel::get_Caption took 0.000
seconds
```

SOFA Example registry settings

Here is an example for configuring the registry for capturing just API call timings for SOFA 12.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COMWRAP]
"Verbosity"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COMWRAP\Components]
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COMWRAP\Components\Timer]
"Verbosity"=dword:00000005
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COUTILITIES]
"Verbosity"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COUTILITIES\Components]
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COUTILITIES\Components\Timer]
"Verbosity"=dword:0000000a
```

This turns off all logging for both the **COMWRAP** and **COUTILITIES** modules, but turns on logging for the **Timer** component for both those modules at a verbosity of **5** for **COMWRAP** and **10** for **COUTILITIES**. This will mean that there will be a single line logged for any **COMWRAP** API calls that take an amount of measurable time and two lines (start and end) will be logged for all **COUTILITIES** API calls.

Query Logging

One of the most useful diagnostic tools is looking at what the OLAP Data Access component is asking the OLAP server for. If the OLAP server is Microsoft Analysis Services or other ODBO based data access and SAP BW/Netweaver BI then this is in the form of MDX queries. Is the data source is Essbase then this is in the form of report scripts.

ODBO and SAP

The MDX executed by the ODBO and SAP data drivers, as well as being logged in the main log file, can be logged in a separate log file which just contains MDX queries. To enable MDX logging to a separate file an additional entry must be made in the registry. The additional entry is a string value with the name **MDX Query Log**. The contents of this string value are the path and name of a file to log the MDX to. The location of the key depends on the data source that MDX logging is required for. For logging MDX sent to an ODBO based OLAP server, the string value is located under **ODBOMODULE** folder. For logging MDX sent to an SAP BW OLAP server, the string value is located under the **SAPMODULE** folder.

For example here is what the registry settings look like for an MDA 12 system:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\ODBOMDX.log"

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\SAPMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\SAPMDX.log"
```

Adding this to the registry will log MDX queries for ODBO data sources to a file **ODBOMDX.log** and MDX queries for SAP data sources to a file called **SAPMDX.log**. Both these files will be located in the following directory:

```
C:\Program Files\Business Objects\BusinessObjects Enterprise 12.0\Logging
```

Essbase

Although it's not possible to log out the Essbase report scripts to a separate file it is possible to reduce the amount of logging in the main logging file to the point where the Essbase report scripts are one of the very few or only things logged out. The Essbase report scripts are logged under the **DATASOURCE** component under the **ESSBASEMODULE** module. There are logged out at verbosities of **4** (Info) and higher. The following is an example of the registry settings from an MDA 12 system that would log at this level.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ESSBASEMODULE]
"Verbosity"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ESSBASEMODULE\Components]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ESSBASEMODULE\Components\DATASOURCE]
"Verbosity"=dword:00000004
```

Query Timing

For the MDX query logging there is an additional setting that can be added to log out the time a query took to execute. This can be turned on by adding an additional DWORD to the registry called **MDX Query Clock**. This is located in either the **ODBOMODULE** or **SAPMODULE** folders. A value of **1** will log the query times and a value of **0** will turn the query time logging off. Here is an example for a MDA 12 system.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\ODBOMDX.log"
"MDX Query Clock"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\SAPMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\SAPMDX.log"
"MDX Query Clock"=dword:00000000
```

This example registry setting has the query clock turned on for ODBO MDX logging and off for SAP MDX logging.

Formatting Logs

The default format for a line in the OLAP Data Access logs contains a time stamp, the module name, the component name and then finally the information being logged. It is possible to configure the format of the logs and add additional information.

The format of log lines is controlled through the **LogFormat** string registry value located in the root **Log** folder. If the registry value is missing the format will revert to the default. The format of the log is defined by a series of tokens which can be wrapped in other text.

Formatting tokens

The following table shows a list of tokens available.

Token	Description
%C	Display the name of the component
%c	Display the name of the component
%D	Display the date
%d	Display the date
%F	Display the full path and filename of the source file where the information being logged originated.
%f	Display the filename of the source file where the information being logged originated.
%I	Display thread ID.
%i	Display thread ID.
%L	Display the line number of the source file where the information being logged originated
%l	Display the line number of the source file where the information being logged originated
%N	Display the name of the function logging the current information
%n	Display the name of the function logging the current information
%M	Display the logging message where non-alpha numeric characters are converted to character codes so that, for example, the message can be embedded in XML or HTML
%m	Display the logging message
%T	Display the date and time
%t	Display the date and time in ISO8601 format
%X	Display the name of the module
%x	Display the name of the module
%V	Display the verbosity at which the information is being logged at
%v	Display the verbosity at which the information is being logged at

So, for example, the following value for **LogFormat**

```
%t %I %X %C %V %n %M
```

which looks like the following in an MDA 12 registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log]
"LogFormat"="%t %I %X %C %V %n %M"
```

would produce a log output as follows

```
2009-12-04T14:02:13.720 7904 MEMORY String INFO ComponentLog::log Instance 0
(1F78B5E0): Created
2009-12-04T14:02:13.720 7904 APIMODULE ConnectionManagerImpl EXTRA
olap_api::ApiLogger::traceLog ConnectionManagerImpl::getDrivers
2009-12-04T14:02:13.728 7904 SUPPORT Synchronization DEBUG
support_dll::TasmanVerboseSynchronized::log_locking Thread ID 7904 : locking
ProviderBrowse::m_sProviderDetailsMutex[0] mutex
2009-12-04T14:02:13.732 7904 ODBOSHAREDUTILITIES ODBOPROVIDERENUMERATOR EXTRA
ComponentLog::log CoCreateInstance(CLSID_OLEDB_ENUMERATOR, NULL, CLSCTX_ALL,
IID_ISourcesRowset, pISrcRowset(0x00000000))
```

The token **%t** outputs the time and date in ISO8601 format. The token **%I** outputs the thread ID, in this case the thread ID is **7904**.

The token **%X** outputs the module name. This is **MEMORY** for the first line, **APIMODULE** for the second, **SUPPORT** for the third and **ODBOSHAREDUTILITIES** for the fourth line.

The token **%C** outputs the component name. This is **String** for the first line, **ConnectionManagerImpl** for the second line, **Synchronization** for the third line, and **ODBOPROVIDERENUMERATOR** for the fourth line.

The token **%V** outputs the verbosity. The values are **INFO**, **EXTRA**, **DEBUG** and **EXTRA** for the four lines respectively.

The token **%n** displays the name of the function doing the logging. The values are `ComponentLog::log`, `olap_api::ApiLogger::traceLog`, `support_dll::TasmanVerboseSynchronized::log_locking`, and `ComponentLog::log` for the four lines respectively.

As may be apparent, these function names have names related to logging rather than another functional aspect of the OLAP Data Access component. This is because much of the lines of code that actually do the logging have been factored into common functions. This means that facilities like logging the function name or logging the source file name and source code line will generally just log one of a handful of core logging function names and locations. This is obviously of limited diagnostic value, will take up additional disk space and add an additional performance overhead so they are not recommended settings to use.

Finally the token **%M** logs the message.

Adding additional text to the log format

You can add extra text around the log format tokens. This extra text could be used to turn the logging output into a valid XML document or a .csv file that could be then imported into a database.

For example, the following log format specification

```
<Log Time="%t" ThreadID="%I" Module="%X" Component="%C Verbosity="%V" Message="%M"/>
```

will output XML

```
<Log Time="2009-12-04T14:50:07.468" ThreadID="224" Module="MEMORY" Component="String
Verbosity="INFO" Message="Instance 0 (1E44B4D0): Created"/>
```

Non-alphanumeric character encoding

If the token used to display message is **%M** rather than **%m** then certain characters will be converted as illustrated below.

Character	Conversion
<	&lt
>	&gt
&	&amp
‘	&apos
“	&quote

Assertions

Assertions are used by developers to document assumptions about what state a program should be in when it is performing a particular operation. Assertions generally take the form of a logical test, for example the variable **A** should have a value greater than zero. The OLAP Data Access component has the facility to log when these assertions fail. This is generally a tool used by developers when a product is in development, but it can be a useful trouble shooting tool as it may highlight unusual or unexpected situations.

Turning on assertions

By default assertions are turned off. For the results of failed assertions to be logged assertions must be turned on. This is done by adding a new key **Asserts** at the same level as the **Log** key. The following table shows the location of this key for the different versions of the OLAP Data Access component.

Version	Asserts location
SOFA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP Intelligence\OCCA(o)\Asserts]
SOFA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP Intelligence\OCCA(o)\ Asserts]
MDA 11.5	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\ Asserts]
MDA 12	[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\ Asserts]

The **Asserts** can contain two DWORD values. The first value **Model** specifies the version of the assertion model that should be used. The next table shows the possible values for **Model** and explains what each model does.

Value	Description
0	Off. No assertion tests will be performed
1	Log and continue. If the appropriate logging is enabled, any failed asserts will be logged and the program will continue.
2	Log and abort. If the appropriate logging is enabled, any failed asserts will be logged and the program will terminate. **Caution** this setting will cause your program to stop if an assertion fails. This should not be used in a production environment.
3	Message box (windows only). This will log a failed assertion if logging is enabled and pop up a windows message box. **Caution** the program will pause until a user clicks on the message box. This should not be used in a production environment.
4	Windows event log (windows only). This will log a failed assertion to the windows event log and in the OLAP Data Access log if it is enabled.

The second DWORD value available is **StackDump**. This has two possible values: **0** and **1**. A value of one will log out the call stack for any failed assertions. If the value is missing or has a value of 0 then no stack dumps will be logged.

Caution: This setting has been associated with errors when the OLAP Data Access component is hosted by a JVM, for example when used by Voyager. These errors cause the JVM to crash so turning stack dumps on should not be used in a production environment.

An example registry for SOFA 12 would look as follows:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP Intelligence\OCCA(o)\Asserts]

"Model"=dword:00000003

"StackDump"=dword:00000001
```

This sets the assertion model to **3**, which pops up a message box. It also has stack dumps turned on.

Turning on assertion logging

The logging of assertions is controlled by the addition of an **ASSERTION** registry key underneath the **Modules** folder. As with other logging modules the **ASSERTION** key should contain a DWORD value called **Verbosity**. The only available verbosities for assertion however are **0** and **1**. That is to say assertion failures are either logged or not. An example registry setting for SOFA 12 would be as follows:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\ASSERTION]

"Verbosity"=dword:00000001
```

This will turn assertion logging on.

UNIX

UNIX deployments of BusinessObjects software will obviously not have access to a Windows registry in order to read the configurations for OLAP Data Access logging. However, BusinessObjects does have a pseudo registry on UNIX which plays the same role as the windows registry. BusinessObjects uses the file system to represent the windows registry. Folders that you observe in the Windows registry will appear as folders in the UNIX file system. Registry values will appear in a file called **.registry** (note the filename starts with a dot '.'). For example, the following Windows registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\log]

will represent the following folder on a UNIX system:

<INSTALLDIR>/bobje/data/.bobj/registry/software/business objects/suite 12.0/mda/log
```

In this folder there should be a file called **.registry**. The contents of this file should look similar to how a list of values would appear in a windows registry script. So, for example, if a verbosity value were required the contents of the file should look as follows:

```
"Verbosity"=dword:0000000a
```

The other file that should be in a folder that is part of the BusinessObjects registry is a file called **.keyname** (again, note the filename begins with a dot '.'). This file simply contains the name of the registry key. So if the folder was to represent the **ODBOMODULE** key, the contents of the file **.keyname** should be the text **ODBOMODULE**. The case of the text in the **.keyname** file should match the case of the name of the example key names given in this document. For example, **ODBOMODULE** is all capital letters. However, the names of the folders in the file structure need to be all lowercase.

To use the registry examples in this document on UNIX they must be converted to their UNIX form. This means creating the relevant folders and **.keyname** files for each new key that needs to be created and also adding the required values to the **.registry** file for each folder.

Known issue with UNIX logging

There is a known issue with the way modules and components are initialized on UNIX systems. The issue means that some information may not be logged under the correct module or component. For example a line that would be logged under the **SAPMODULE** on a Windows system may appear under **MDA_SUPPORT** on UNIX. To ensure the required information is being logged it is suggested that logging is turned on by setting the verbosity value in the root **Log** folder first to see which modules information is being logged under. Logging can then be incrementally turned off as desired by adding in unwanted modules and components and setting the verbosity to zero.

64-bit Windows

If an installation of BusinessObjects is on a 64-bit Windows operating system, then the entries in the registry will be in a slightly different place. This is because the BusinessObjects installation is still a native 32-bit installation (it will be installed by default under the `C:\Program Files (x86)` directory for example). Windows has a separate bit of the registry for applications that are natively 32-bit. The registry keys instead of being located under

```
[HKEY_LOCAL_MACHINE\SOFTWARE]
```

are instead located under

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node].
```

This means a registry key such as the following

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log]
```

becomes

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Business Objects\Suite 12.0\MDA\Log].
```

Therefore, any of the example registry settings given in this document must be modified to include the additional **Wow6432Node** folder if they are being applied to a 64-bit system.

Choosing the correct settings

When thinking about what level to set logging at the situation is very important to consider. Just running with all logging turned on at full verbosity is rarely what is required. It will also generate huge amounts (gigabytes) of information quite quickly and introduce a performance overhead to the software. The following are possible situations and suggested registry settings to match the situation.

Logging in concurrent environment

If it is expected that multiple users will be using the system concurrently at the same time then it is advisable to add the thread ID into the log format. This will help separate out individual user's activities in a situation where consecutive lines in the log file may be from two separate users doing totally separate things. The concurrent user case would be the expected situation for most deployments except if the tool being used is a desktop client tool like the Crystal Reports designer or the Web Intelligence rich client.

Monitoring a successfully running production system

If a system has been up and running successfully for a while it can be tempting to run the system with no logging at all, but it is useful to turn on minimal logging so that if an incident should happen there is at least some information available. Therefore it might be prudent to turn logging on at a verbosity of critical or serious for all modules and components. This can be done by deleting the **MODULES** folder and everything beneath it and just having a single **Verbosity** value set to **1** or **2** in the root **Log** folder. For example,

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log]
"Verbosity"=dword:00000002
```

Monitoring a system with intermittent problems

If a system is suffering intermittent problems such as users experiences errors or performance problems it might be advisable to turn up the logging to see if the source of the problems can be narrowed down. So if the problems are errors then the verbosity set in the above example could be raised to **Warning** or even **Info**. If there performance problems are being experienced, the API call timing logging could be turned on with a threshold set sufficiently high to only capture extremely long running API calls.

Narrowing down a critical issue on a production system and trouble shooting on a development environment

If critical but undiagnosed issues are occurring on a production system, or a system is under development and in need of regular trouble shooting then a rather more aggressive selection of logging may be advisable. It is still not advisable to turn everything on at the highest level but there are targeted segments of logging that generally give a good initial picture when troubleshooting.

The modules to turn on to full verbosity are **APIMODULE** or **COMWRAP** depending on the version in use. This gives an indication of what is being asked of the OLAP Data Access component by the layer above. The other module to turn on is the one associated with the particular data source being used. The modules are **ODBOMODULE**, **ESSBASEMODULE** or **SAPMODULE**. The next section gives some examples of good default registry settings for such a situation.

Example default settings

XI R2

Web Intelligence, Crystal Reports and OLAP Intelligence

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Asserts]
"Model"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log]
"Location"="C:\\Program Files\\Business Objects\\OLAP Intelligence 11.5\\Bin\\"
"LogFile"="C:\\Program Files\\Business Objects\\OLAP Intelligence
11.5\\Bin\\sofa.log"
"Overwrite"="Yes"
"AppendPID"="Yes"
"Verbosity"="2"
"LogFormat"="%T: %I: %X: %C: %m"

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\COMWRAP]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\COUTILITIES]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\RORFLATTENER]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\ODBOMODULE]
"Verbosity"=dword:0000000a
"MDX Query Log"="C:\\Program Files\\Business Objects\\OLAP Intelligence
11.5\\Bin\\odbomdx.log"
"MDX Query Clock"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\SAPMODULE]
"Verbosity"=dword:0000000a
"MDX Query Log"="C:\\Program Files\\Business Objects\\OLAP Intelligence
11.5\\Bin\\sapmdx.log"
"MDX Query Clock"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\ESSBASEMODULE]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\OLAP
Intelligence\OCCA(o)\Log\Modules\ASSERTION]
"Verbosity"=dword:00000001
```

Voyager

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\Asserts]
"Model"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\Log]
"LogFile"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
11.5\\Logging\\MDA.log"
"Location"="C:\\Program Files\\Business Objects\\common\\3.5\\bin\\"
"Overwrite"="Yes"
"AppendPID"="Yes"
"Verbosity"="2"
"LogFormat"="%T: %I: %X: %C: %m"

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.5\Voyager\MDA\Log\MODULES]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\APIMODULE]
"Verbosity"=dword:0000000a
"Timer Threshold"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\APIMODULE\Components]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\APIMODULE\Components\INFO]
"Verbosity"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\ODBOMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
11.5\\Logging\\odbomdx.log"
"MDX Query Clock"=dword:00000001
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\ODBOSHAREDUTILITIES]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\SAPMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
11.5\\Logging\\sapmdx.log"
"MDX Query Clock"=dword:00000001
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\MODULES\ESSBASEMODULE]
"Verbosity"=dword:0000000a
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
11.5\Voyager\MDA\Log\Modules\ASSERTION]
"Verbosity"=dword:00000001
```

XI 3.0 and XI 3.1

Crystal Reports

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Asserts]
"Model"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log]
"LogFile"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\SOFA.log"
"Location"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\win32_x86\\"
"Overwrite"="Yes"
"AppendPID"="Yes"
"Verbosity"="2"
"LogFormat"="%T: %I: %X: %C: %m"

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COMWRAP]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\COUTILITIES]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\ODBOMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\odbomdx.log"
"MDX Query Clock"=dword:00000001
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\SAPMODULE]
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\sapmdx.log"
"MDX Query Clock"=dword:00000001
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\ESSBASEMODULE]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\ODBOSHAREDUTILITIES]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\UTILITIES]
"Verbosity"=dword:0000000a
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\OLAP
Intelligence\OCCA(o)\Log\Modules\ASSERTION]
"Verbosity"=dword:00000001
```

Voyager and Web Intelligence

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Asserts]
"Model"=dword:00000001
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log]
"LogFile"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\MDA.log"
"Location"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\win32_x86\\"
"Overwrite"="Yes"
"AppendPID"="Yes"
"Verbosity"="2"
"LogFormat"="%T: %I: %X: %C: %m"

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\APIMODULE]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components]

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\APIMODULE\Components\INFO]
"Verbosity"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ASSERTION]
"Verbosity"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ESSBASEMODULE]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\ODBOMODULE]
"Verbosity"=dword:0000000a
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\ODBOMDX.log"
"MDX Query Clock"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite
12.0\MDA\Log\Modules\ODBOSHAREDUTILITIES]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\RORMODULE]
"Verbosity"=dword:0000000a

[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\MDA\Log\Modules\SAPMODULE]
"Verbosity"=dword:0000000a
"MDX Query Log"="C:\\Program Files\\Business Objects\\BusinessObjects Enterprise
12.0\\Logging\\SAPMDX.log"
"MDX Query Clock"=dword:00000001
```

Copyright