| Misc | | |
| --- | --- | --- |
|  |  |  |
| Integration with other tools |  | Updating This Document |

# Misc

## Performance

### Retrieval of repository configuration

In order to present the valid options for attributes in the Task Editor, the repository 'configuration' must be retrieved from the repository. The default connector implementation requests a new configuration every 24hrs. In the case of Bugzilla, this is further refined by first performing a head request on the configuration to see if the Last-Modified header has changed. If so, the repository configuration is retrieved. Additionally, Eclipse.org's webmaster has redirected the config.cgi request to a static page, eliminating the processing overhead server side. For details on server side optimization of Bugzilla see [Mylyn\_FAQ#Tips\_for\_server\_administrators](http://wiki.eclipse.org/Mylyn_FAQ#Tips_for_server_administrators).

## Command Line

The system properties below can used to change the behavior of Mylyn. To set a property pass it on the command line when starting Eclipse: eclipse -argument.

| Argument | Mylyn Version | Description |
| --- | --- | --- |
| -no-activate-task | 3.1 | Disables task activation on startup. The last active task is not re-activated on startup if a workspace crash is detected. |

## System Properties

The system properties below can used to change the behavior of Mylyn. To set a property pass it on the command line when starting Eclipse: eclipse -vmargs -Dorg.eclipse.mylyn.property=value.

| System Property | Mylyn Version | Default | Description |
| --- | --- | --- | --- |
| org.eclipse.mylyn.linkProviderTimeout | 3.1 | 5000 | Number of milli-seconds before link providers are timed out. Set to -1 to disable link providers timing out. |
| mylyn.discovery.directory | 3.2 | <http://www.eclipse.org/mylyn/discovery/directory.xml> | URL for the discovery directory. |

## How can I report a dead-lock or a problem about a stalled UI?

The recommended way is to file a bug with a full thread dump using jstack which is part of the Java Development Kit 6. These wiki pages have more details on using jstack on different platforms:

* [How to report a deadlock](http://wiki.eclipse.org/How_to_report_a_deadlock)
* [Debugging Mylyn](http://wiki.eclipse.org/Mylyn_Contributor_Reference#Debugging)

## How do I enable debugging output for network communications?

Add the following lines to the eclipse.ini file in your eclipse directory to enable tracing of header information for HTTP requests. Log output is written to the console eclipse was started from. Make sure to add these after the line that says **-vmargs**. Add a -vmwargs line if there is none in the file.

-Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog  
-Dorg.apache.commons.logging.simplelog.showlogname=true  
-Dorg.apache.commons.logging.simplelog.defaultlog=off  
-Dorg.apache.commons.logging.simplelog.log.httpclient.wire.header=debug  
-Dorg.apache.commons.logging.simplelog.log.org.apache.commons.httpclient=off  
-Dorg.apache.commons.logging.simplelog.log.org.apache.axis.message=debug

To also enable logging of message content, also add this lines:

-Dorg.apache.commons.logging.simplelog.log.httpclient.wire.content=debug

## How do I enable debugging output for plug-ins?

Create a **.options** file in your eclipse directory and enable tracing by including the corresponding lines which set the tracing setting for a particular concern to true:

**JIRA**

org.eclipse.mylyn.jira.core/debug/connector=true  
org.eclipse.mylyn.jira.core/debug/dataHandler=true  
org.eclipse.mylyn.jira.core/debug/repository=true

**Trac**

# trace authentication requests  
org.eclipse.mylyn.trac.core/debug/authentication=true  
# trace XML-RPC calls  
org.eclipse.mylyn.trac.core/debug/xmlrpc=true

Then start eclipse with **-debug**. You can also optionally pass a filename to -debug if your .options file is not located in the eclipse directory.

See also [FAQ\_How\_do\_I\_use\_the\_platform\_debug\_tracing\_facility%3F](http://wiki.eclipse.org/FAQ_How_do_I_use_the_platform_debug_tracing_facility%3F).

## Which usage monitoring framework should I use?

Three usage data collection frameworks have been created for Eclipse:

1. [Eclipse Instrumentation Framework](http://dev.eclipse.org/viewcvs/index.cgi/platform-ui-home/instrumentation/index.html?revision=1.12): created as part of Platform UI, not released, not currently maintained
2. [Mylyn Monitor UI Usage Reporting](http://wiki.eclipse.org/Mylyn_Integrator_Reference#Monitor_API): created in 2004, maintained as part of the Mylyn project, first released in 2005
3. [Usage Data Collector](http://www.eclipse.org/epp/usagedata/index.php): maintained as part of the EPP project, first released in 2008

While (1) and (3) are work in a similar way, there is a significant difference between the approach used by the Mylyn Monitor and the other usage data collectors. Instead of gathering statistics, the Mylyn UI Usage Reporting component uses the interaction history that is captured by the Mylyn Monitor. This is the same interaction history stream that is used to determine the interest level of elements in Mylyn’s Task-Focused UI and has been refined by long-term use of Mylyn’s Task-Focused UI. The UI Usage Monitor is extensible, de-coupled from the other parts of Mylyn and can be used independently.

There are several main benefits to the approach of capturing a full interaction history stream instead of reporting on particular statistics.

* Since all of the interaction information is captured on the client, server-side reporting facilities can report on usage statistics that were not defined at deployment time. For example, during one of the Mylyn studies we did not decide to monitor which views were opened within which perspective. Since we had the full interaction histories this reporting was easy to add.
* Interaction histories make it possible to report not just on the usage of the UI, but on the state of the UI at any given time. For example, in one study we wanted to know what actinos the user did after invoking a build and which views were visible while the invoked that action.
* An explicit interaction history makes it easier to avoid and debug privacy problems. The Mylyn Monitor has a robust facility for filtering, encrypting and obfuscating data from interaction histories.

For an example study see: <http://kerstens.org/mik/publications/mylar-ieee2006.pdf>

## How does Mylyn relate to IBM’s Jazz?

At the EclipseCon and JavaONE 2006 conferences IBM demonstrated previews of Jazz, a collaborative team server and Eclipse-based client. Articles have remarked on the similarities between Mylyn and Jazz because both integrate tasks into Eclipse (Jazz’s “work items” and Mylyn’s “tasks”), and both provide change sets grouped by task. But there are both significant differences and complementary aspects to the two approaches. A key goal of Mylyn is to provide an open source framework to support integration of task management with *existing issue trackers and source repositories*. According to the presentations, components that come with Jazz include include a *next-generation issue tracker and source repository* and related lifecycle management tools such as project health. In addition, a driving and unique goal of Mylyn is to focus the UI around a usage-based and degree-of-interested weighted task context, which is complementary to the Jazz platform. Just as it integrates with Mylyn’s Task List, Mylyn’s Task Focused UI and task context model components are possible to integrate with other kinds of task management systems, such as the one provided by Jazz.

Update: at EclipseCon 2007 the IBM Jazz team showed a Mylyn connector for Jazz.

[Category:FAQ](http://wiki.eclipse.org/Category:FAQ) [Category:Mylyn](http://wiki.eclipse.org/Category:Mylyn)

|  |  |  |
| --- | --- | --- |
| Integration with other tools |  | Updating This Document |