# The Developer Panel


- Introduction
  - Terms
    - Developer Mode
    - Developer Settings
    - The Developer Panel
  - Entering Developer Mode
    - developer.properties file
    - Interactively entering developer mode
- The settings
  - General settings
  - The "General" tab
    - Freemarker settings
    - SPARQL Query settings
    - Page configuration settings
    - Language support settings
    - Links
  - The "Search" tab
    - Search query settings
    - Search indexing settings
    - Links
  - The "Authorization" tab
- What if the developer panel doesn't appear?


Diagnostic tools that can help you figure out what VIVO is doing.

# Introduction

Are you developing code for VIVO? Are you doing some extreme customization? You might benefit from VIVO's set of built-in diagnostic tools.

These tools help to reveal how VIVO operates behind the scenes. In general, they are only used during development, because they may have serious negative effects on the performance of the VIVO application. However, they may also be used (carefully) in production, to diagnose a particularly difficult problem.

The diagnostic tools are enabled and controlled by settings within VIVO. These settings may be changed interactively, without restarting VIVO. The settings may also be read from a file, so they will be in effect as VIVO is starting up.

## Terms

### Developer Mode

When the diagnostic tools are enabled, VIVO is said to be running in "Developer Mode".  This reflects the fact that all of the developer settings are ignored unless the tools as a whole are enabled.

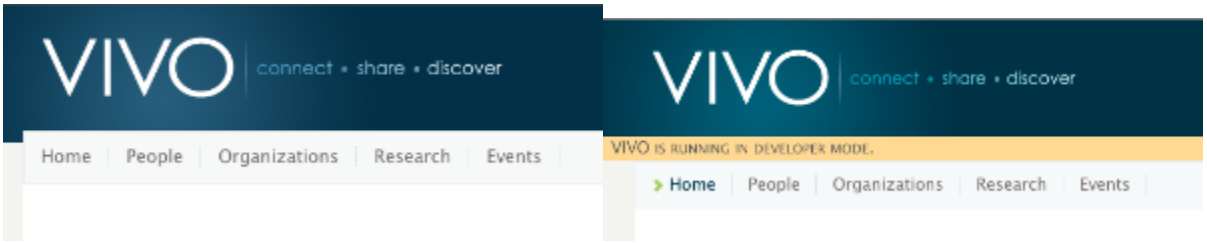
### Developer Settings

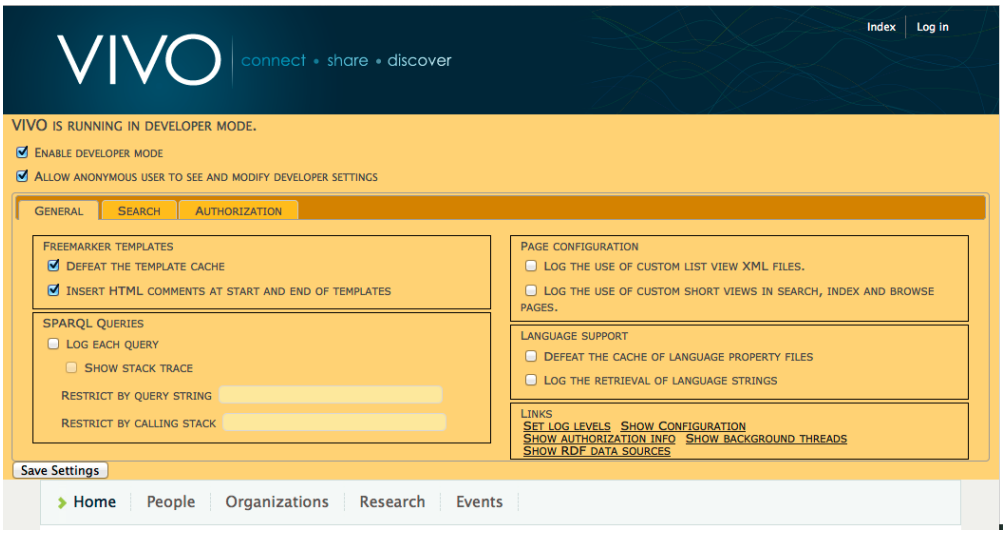
These are parameters that control the diagnostic tools. They may be set interactively, using the Developer Panel, or read from the `developer.properties` file at startup.

### The Developer Panel

When VIVO is in developer mode, the Developer Panel appears on every page. This serves two purposes:

1. It enables you to change the Developer Settings without navigating away from your current page.
2. It provides a visual reminder that VIVO is in Developer Mode. If a production instance were accidentally configured to run in Developer Mode, it would be easily noticed.






# Entering Developer Mode

## `developer.properties` file

When VIVO starts up, it looks for a file in the home directory, named `developer.properties`. If the file is found, the settings are read from it. Any settings that are not found in the file keep their default values. If the file is not found, then all settings keep their default values until set interactively.

**A typical developer.properties file**

```
developer.enabled=true
developer.permitAnonymousControl=true
developer.defeatFreemarkerCache=true
```

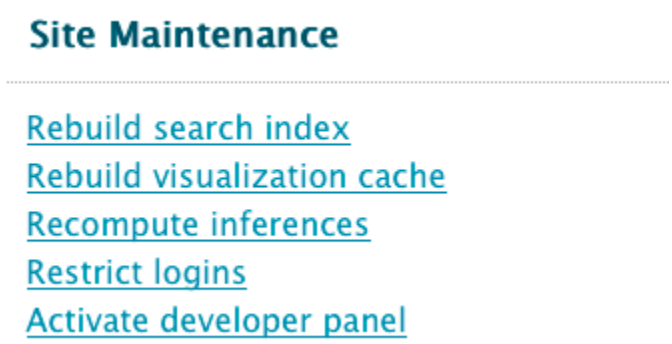
This example causes:

- Developer Mode is enabled immediately. The specified settings are in effect even while VIVO is starting.
- Users who are not logged in can manipulate the developer settings. Obviously, this should only be permitted on a development system.
- The Freemarker template cache is defeated. Each time a template is requested, it will be loaded from disk. This will cause VIVO to run more slowly, but it means that a developer can see the effects of a template change immediately, instead of waiting for the template to expire and be reloaded.

The VIVO distribution includes an `example.developer.properties` file. It contains descriptions of all of the settings, with examples. You can rename `example.developer.properties` to `developer.properties`, and uncomment the settings you want to use.

## Interactively entering developer mode

Log in as a system administrator (or root). Go to the `Site Administration` page. Click on `Activate developer panel`.




The Developer Panel will immediately appear below the page header. You may click on the panel to open it and change the settings. The Developer Panel will continue to appear in every page (except for some "back-end" pages for editing the ontology).

# The settings

The developer panel has two general settings, and three tabs of additional settings. The following tables show the meaning of each setting in the developer panel, and how to specify it in the `developer.properties` file.

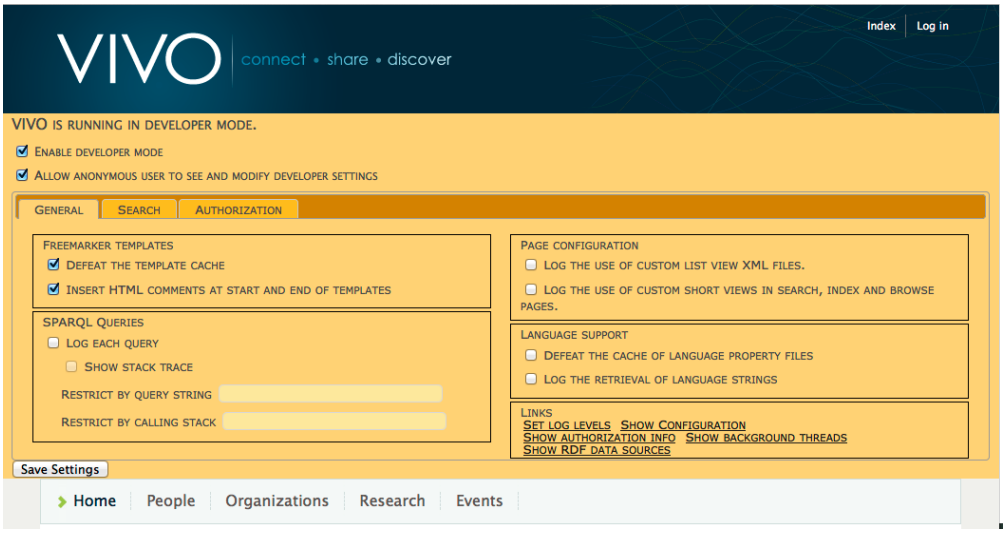
## General settings

These appear at the top of the panel, regardless of which tab is selected.

| | |
|---|---|
| **In the panel** | Enable developer mode |
| **In the file** | `developer.enabled` |
| **Effect** | Causes the developer panel to be displayed on each VIVO page. Enables all of the other developer settings. If this is `false`, other settings will retain their values, but will not take effect. |

| | |
|---|---|
| **In the panel** | Allow anonymous user to see and modify developer settings |
| **In the file** | `developer.permitAnonymousControl` |
| **Effect** | If `true`, any VIVO user may change the developer settings. If `false`, only a system administrator (or root) may change the settings. |

## The "General" tab




### Freemarker settings

| | |
|---|---|
| **In the panel** | Defeat the template cache |
| **In the file** | `developer.defeatFreemarkerCache` |
| **Effect** | If `true`, each Freemarker template is loaded from disk each time it is used. If `false`, a template change may be on disk for up to one minute before it is loaded. |

| | |
|---|---|
| **In the panel** | Insert HTML comments and start and end of templates |
| **In the file** | `developer.insertFreemarkerDelimiters` |

| | |
|---|---|
| **Effect** | If `true`, you may view the HTML source for a VIVO page to see which Freemarker templates were used to create each portion of the page. For example:<br><br>`...`<br>`<!-- FM_BEGIN view-search-default.ftl -->`<br>`<a href="/vivo/display/n2252" title="individual name">Oswald, Jeremiah</a>`<br>`<span class="display-title">Faculty Member</span>`<br>`<p class="snippet"></p><!-- FM_END view-search-default.ftl -->`<br>`...` |

## SPARQL Query settings

Full documentation for the logging RDF Service is available here, including detailed explanation of an example log excerpt.

| | |
|---|---|
| **In the panel** | LOG each query |
| **In the file** | `developer.loggingRDFService.enable` |
| **Effect** | Write an entry to the log for each SPARQL query, assuming that `INFO`-level logging is enabled for the `RDFServiceLogger`. Each entry includes<br><br>• The time spent executing the query<br>• The name of the method on `RDFService` that received the query<br>• The format of the result stream from `RDFService`<br>• The text of the query.<br><br>The remaining settings in this area can be used to restrict which queries are logged, or to include more information for each query.<br><br>The configuration models (display, user accounts) and the TBox models are memory-mapped. This means that any "read" operations are run against a cached copy of the model, and are not logged. For these models, only "write" operations are logged.<br><br>The ABox models are not memory-mapped; both "read" and "write" operations will be logged. |

| | |
|---|---|
| **In the panel** | Show stack trace |
| **In the file** | `developer.loggingRDFService.stackTrace` |
| **Effect** | Each log entry will include a stack trace. The trace is abridged so it starts after the `ApplicationFilterChain`, omits any Jena classes, and ends at the `RDFService`. |

| | |
|---|---|
| **In the panel** | Restrict by query string |
| **In the file** | `developer.loggingRDFService.queryRestriction` |
| **Effect** | Set this to a regular expression. A query will be logged only if the text of the query matches the regular expression, in whole or in part. |

| | |
|---|---|
| **In the panel** | Restrict by calling stack |
| **In the file** | `developer.loggingRDFService.stackRestriction` |
| **Effect** | Set this to a regular expression. A query will be logged only if the abridged calling stack matches the regular expression, in whole or in part. |

## Page configuration settings

| | |
|---|---|
| **In the panel** | Log the use of custom list view XML files. |
| **In the file** | `developer.pageContents.logCustomListView` |

| | |
|---|---|
| **Effect** | Write an entry to the log each time a property is displayed using a list view other than the default lists view. |

| | |
|---|---|
| **In the panel** | Log the use of custom short views in search, index and browse pages. |
| **In the file** | `developer.pageContents.logCustomShortView` |
| **Effect** | Write an entry to the log each time a search result is displayed using a short view other than the default view for that context. |

## Language support settings

| | |
|---|---|
| **In the panel** | Defeat the cache of language property files |
| **In the file** | `developer.i18n.defeatCache` |
| **Effect** | If `true`, the language property files are re-loaded each time they are called for. If `false`, the language property files are loaded only once, when VIVO starts up. |

| | |
|---|---|
| **In the panel** | Log the retrieval of language strings |
| **In the file** | `developer.i18n.logStringRequests.` |
| **Effect** | Write an entry to the log each time a language-specific string is retrieved from one of the language property files. |

## Links

The "General" tab also contains these links to special VIVO pages.

| | |
|---|---|
| **Link text** | Set log levels |
| **URL** | `/admin/log4j.jsp` |
| **The page** | Displays the logging levels of every Java class in VIVO, providing that it has an active `Log`. You must be logged in as a system administrator (or root) to use this page. Find the class you are interested in, set the logging level, then scroll to the bottom of the page to `Submit changes to logging levels`. |

| | |
|---|---|
| **Link text** | Show Configuration |
| **URL** | `/admin/showConfiguration` |
| **The page** | Displays a list of the `build.properties` and `runtime.properties`. Displays a list of the System properties in the Java virtual machine. |

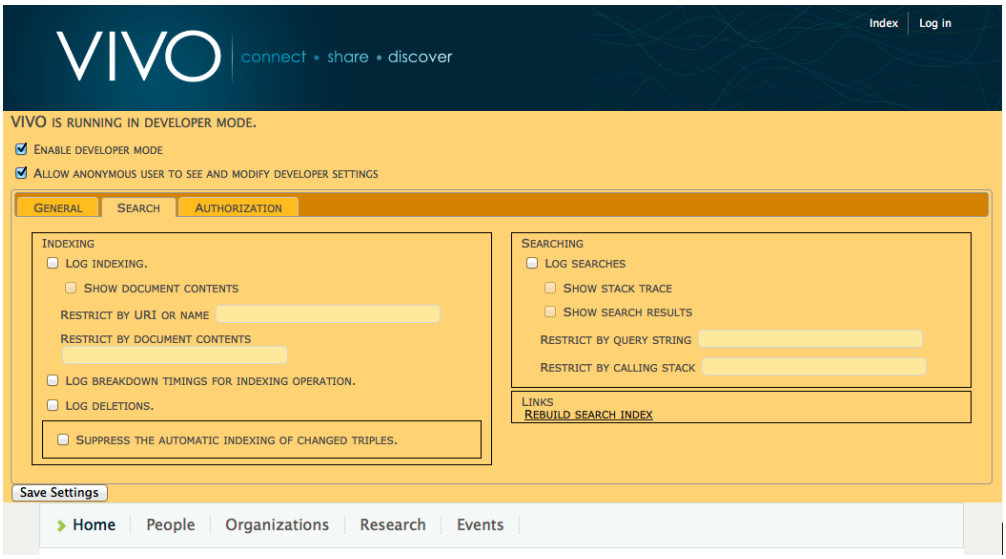
| | |
|---|---|
| **Link text** | Show authorization info |
| **URL** | `/admin/showAuth` |
| **The page** | Displays information about the user who is currently logged in, the identifiers associated with that user, and the permissions they have been granted. Display information about the configured `Policy` objects, and related objects. |

| | |
|---|---|
| **Link text** | Show background threads |
| **URL** | `/admin/showThreads` |
| **The page** | Displays information about the active background threads. These threads may be rebuilding the search index, re-inferencing the knowledge base, or rebuilding the Class Cache. |

| | |
|---|---|
| **Link text** | Show RDF data sources |

| | |
|---|---|
| **URL** | `/admin/showSources` |
| **The page** | Displays information about the triple stores, and the layers of adapters that the application wraps around them, including ModelMakers, OntModels, etc. |

## The "Search" tab




### Search query settings

| | |
|---|---|
| **In the panel** | Log searches |
| **In the file** | `developer.searchEngine.enable` |
| **Effect** | Write an entry to the log for each Search query, assuming that `INFO`-level logging is enabled for the `SearchEngineLogger`. Each entry includes<br>• The time spent executing the query<br>• The search query, including the query text, start row, row limits, search fields, return fields, facet fields, and filters.<br>• The number of results returned.<br><br>The remaining settings in this area can be used to restrict which queries are logged, or to include more information for each query. |

| | |
|---|---|
| **In the panel** | Show stack trace |
| **In the file** | `developer.searchEngine.addStackTrace` |
| **Effect** | Each log entry will include a stack trace. The trace is abridged so it starts after the `ApplicationFilterChain` and ends at the `SearchEngineWrapper`. |

| | |
|---|---|
| **In the panel** | Show search results |
| **In the file** | `developer.searchEngine.addResults` |
| **Effect** | Each log entry will include the search records that were returned from the query, as well as any facet fields. |

| | |
|---|---|
| **In the panel** | Restrict by query string |
| **In the file** | `developer.searchEngine.queryRestriction` |
| **Effect** | Set this to a regular expression. A query will be logged only if the text of the query matches the regular expression, in whole or in part. |

| | |
|---|---|
| **In the panel** | Restrict by calling stack |
| **In the file** | `developer.searchEngine.stackRestriction` |
| **Effect** | Set this to a regular expression. A query will be logged only if the abridged calling stack matches the regular expression, in whole or in part. |

## Search indexing settings

| | |
|---|---|
| **In the panel** | Log indexing |
| **In the file** | `developer.searchIndex.enable` |
| **Effect** | Write an entry to the log each time that documents are added to the Search index, assuming that `INFO`-level logging is enabled for the `SearchEngineLogger`. Note that documents are not changed in the Search index: instead, old documents are deleted and new documents are added. Each entry includes<br>• The time taken to add the documents.<br>• The number of documents added.<br>• The URIs of the individuals being indexed in the documents.<br>The remaining settings in this area can be used to restrict which queries are logged, or to include more information for each query. |

| | |
|---|---|
| **In the panel** | Show document contents |
| **In the file** | `developer.searchIndex.showDocuments` |
| **Effect** | Each entry will include the contents of the documents being added. This includes the document identifiers, the boost level, and the contents of each of the fields in the document. |

| | |
|---|---|
| **In the panel** | Restrict by URI or name |
| **In the file** | `developer.searchIndex.uriOrNameRestriction` |
| **Effect** | Set this to a regular expression. An addition will be logged only if the list of document identifiers matches the regular expression, in whole or in part. The document identifiers are the `URI` and the `Name` fields. |

| | |
|---|---|
| **In the panel** | Restrict by document contents |
| **In the file** | `developer.searchIndex.documentRestriction` |
| **Effect** | Set this to a regular expression. An addition will be logged only if the contents of the documents matches the regular expression, in whole or in part. |

| | |
|---|---|
| **In the panel** | Log breakdown timings for indexing operations |
| **In the file** | `developer.searchIndex.logIndexingBreakdownTimings` |
| **Effect** | When an indexing operation completed, write entries to the log showing how much time was taken by each indexing object: Excluders, DocumentModifiers, and UriFinders. Each entry includes<br>• The display label of the indexing object<br>• The number of times that the indexing object was invoked<br>• The total time required for the indexing object<br>• The average time for each invocation of the indexing object |

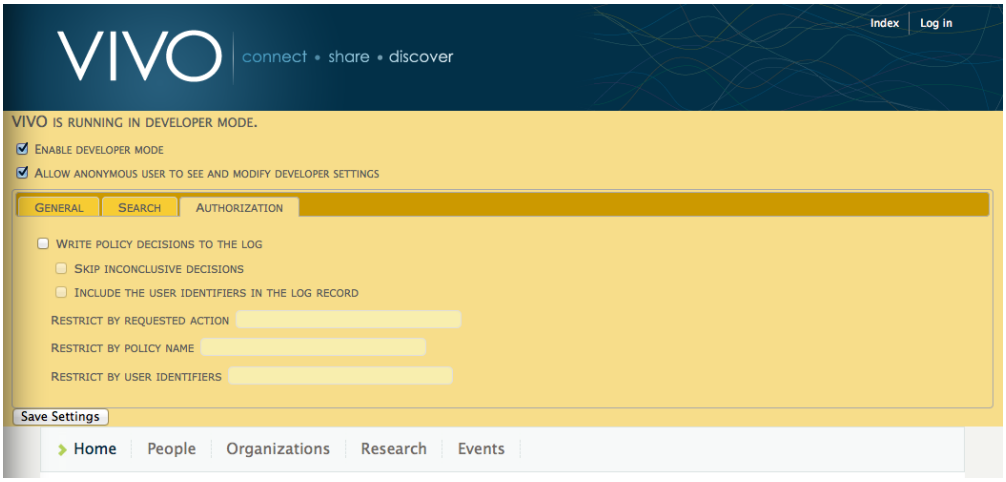
| | |
|---|---|
| **In the panel** | Log deletions |
| **In the file** | `developer.searchDeletions.enable` |
| **Effect** | Write an entry to the log each time documents are deleted from the Search index, assuming that `INFO`-level logging is enabled for the `SearchEngineLogger`. Each entry includes<br>• The time spent deleting the documents<br>• Either<br>  ◦ the list of `URIs` being deleted, or<br>  ◦ the search query that was used to find documents for deletion. |

| | |
|---|---|
| **In the panel** | Suppress the automatic indexing of changed triples. |
| **In the file** | `developer.searchIndex.suppressModelChangeListener` |
| **Effect** | If this is selected, the search indexer will not automatically adjust to changes in the data model. This means that you can ingest data much more quickly, but you must manually request a full re-indexing when your ingests are complete.<br>This doesn't really belong in the developer panel, since it changes the way VIVO operates. It was put here to answer an urgent requirement. |

### Links

| | |
|---|---|
| **Link text** | Rebuild search index |
| **URL** | `/SearchIndex` |
| **The page** | Allows you to request a rebuild of the search index, and to monitor its progress. |

## The "Authorization" tab




| | |
|---|---|
| **In the panel** | Write policy decisions to the log |
| **In the file** | `developer.authorization.logDecisions.enable` |

| | |
|---|---|
| **Effect** | Write an entry to the log for each policy decision that is made for any requested action, assuming that `INFO`-level logging is enabled for the `PolicyDecisionLogger`. Each entry includes<br><br>• The requested action<br>• The name of the policy<br>• The decision and message.<br><br>The remaining settings in this area can be used to restrict which queries are decisions are logged, or to include more information for each decision. |

| | |
|---|---|
| **In the panel** | Include the user identifiers in the log record |
| **In the file** | `developer.authorization.logDecisions.addIdentifiers` |
| **Effect** | Each log entry will include the identifiers assigned to the currently logged-in user. |

| | |
|---|---|
| **In the panel** | Skip inconclusive decisions |
| **In the file** | `developer.authorization.logDecisions.skipInconclusive` |
| **Effect** | Do not log `INCONCLUSIVE` decisions. If all policies return `INCONCLUSIVE` for a request, this is treated as `UNAUTHORIZED`, and will be logged. |

| | |
|---|---|
| **In the panel** | Restrict by requested action |
| **In the file** | `developer.authorization.logDecisions.actionRestriction` |
| **Effect** | Set this to a regular expression. A decision will be logged only if the string value of the requested action matches the regular expression, in whole or in part. |

| | |
|---|---|
| **In the panel** | Restrict by policy name |
| **In the file** | `developer.authorization.logDecisions.policyRestriction` |
| **Effect** | Set this to a regular expression. A decision will be logged only if the string value of the policy matches the regular expression, in whole or in part. |

| | |
|---|---|
| **In the panel** | Restrict by user identifiers |
| **In the file** | `developer.authorization.logDecisions.userRestriction` |
| **Effect** | Set this to a regular expression. A decision will be logged only if the list of user identifiers matches the regular expression, in whole or in part. |

# What if the developer panel doesn't appear?

If you are using a custom theme, and you created it from a VIVO release prior to 1.6, it's likely that your theme doesn't display the developer panel.

Confirm that the template `[vivo]/webapp/themes/[your_theme]/templates/menu.ftl` contains an `include` directive like this one:

**Excerpt from menu.ftl**

```
<#-- $This file is distributed under the terms of the license in /doc/license.txt$ -->

</header>

<#include "developer.ftl">

<nav role="navigation">

...
```