

Find All References in Photran - Documentation

1. Description and Goals of the Project

The goal of this project is to implement searching for all the references to a particular entity in Photran, as suggested on the "New Contributors" page at <http://www.eclipse.org/photran/newcontributors.php>. The functionality should be similar to the functionality found in JDT where you can click on Search and then References and choose from several search scopes, including workspace, project and file. At minimum, our goal for this project was to at least implement the workspace-wide searching capabilities, and implement the project and file capabilities if time permitted.

2. What we Modified and Description of Implementation.

Our development consisted of three primary segments, the execution UI, search engine, and search display UI. In addition to this, we implemented an internationalization interface, or *i18n* as the preferred method for the spelling impaired and those who wish to delay the onset of carpal tunnel. The execution UI consists of the three actions delegates that implement the main menu and context menu entries for the various scopes supported by our search engine, as well as a subclass that implements the common functionality shared between these classes. These classes are `FortranFindReferencesFileActionDelegate`, `FortranFindReferencesProjectActionDelegate`, `FortranFindReferencesWorkspaceActionDelegate`, providing searches scoped at the file, project, and workspace level respectively, and `FortranFindReferencesActionDelegate`, providing the base implementation for each of these classes.

Implemented Eclipse Interfaces

- `IAction` (`FortranFindReferencesFileActionDelegate`, `FortranFindReferencesProjectActionDelegate`, `FortranFindReferencesWorkspaceActionDelegate`)
- `IEditorActionDelegate` (`FortranFindReferencesFileActionDelegate`, `FortranFindReferencesProjectActionDelegate`, `FortranFindReferencesWorkspaceActionDelegate`)
- `IRunnableWithProgress` (`FortranFindReferencesFileActionDelegate`, `FortranFindReferencesProjectActionDelegate`, `FortranFindReferencesWorkspaceActionDelegate`)
- `IWorkbenchWindowActionDelegate` (`FortranFindReferencesFileActionDelegate`, `FortranFindReferencesProjectActionDelegate`, `FortranFindReferencesWorkspaceActionDelegate`)

In addition to the execution UI, we developed search results UI. This component is an extension point to the Eclipse search results UI, implementing the necessary interfaces and extension points for the search results view, as well as editor views, to display and manage the search results generated by our search engine.

Classes defining the search results UI:

- `ReferenceSearchViewPage` - `ISearchResultPage` implementation allowing Eclipse to access the other view classes.
- `IReferenceSearchContentProvider` - Interface defining the common functionality between the list and tree content providers.
- `ReferenceSearchListContentProvider` - Provides the search results maintained in the `ReferenceSearchResult` object to the list view.
- `ReferenceSearchTreeContentProvider` - Provides the search results maintained in the `ReferenceSearchResult` object to the tree view.
- `ReferenceSearchLabelProvider` - Common functionality for displaying the icon and text of a search result.
- `ReferenceSearchListLabelProvider` - Generates labels for search results when displayed in the list

- view.
- ReferenceSearchTreeLabelProvider - Generates labels for search results when displayed in the list view.

Implemented Eclipse Interfaces:

- ISearchResultPage (ReferenceSearchViewPage)
- IStructuredContentProvider (ReferenceSearchListContentProvider, ReferenceSearchTreeContentProvider)
- ILabelProvider (ReferenceSearchListLabelProvider, ReferenceSearchTreeProvider)

Backing the UI interfaces is the search engine implementation, which manages the results of a search, the bookkeeping associated with the search as well as performs the actual search itself. This is the entry point to the search from Eclipse's vantage point. The classes implementing this functionality are ReferenceSearch and ReferenceSearchResult.

Implemented Eclipse Interfaces:

- ISearchQuery (ReferenceSearch)
- ISearchResult (ReferenceSearchResult)
- IEditorMatchAdapter (ReferenceSearchResult)
- IFileMatchAdapter (ReferenceSearchResult)

There are two parts to the search itself. The first is done in the ActionDelegate code, and initializes the PhotranVPG with EnsureVPGIsUpToDate. The selected token is found, its declaration found, and this is passed to the SearchReferences code. From here we simply call the findAllReferences method on definition. Depending on the scope selected, we filter out unwanted matches and add the rest to our result set.

The internalization interface consists of the CSearchMessages class and the associated CSearchMessages.properties file. At this time only the default language is provided, and implemented in English, although additional languages may be implemented through the standard ResourceBundle i18n method.

3. Problems we Encountered

Two major problems encountered were limited documentation for the components related to the Search and Photran plugins and cross platform issues associated with Photran itself.

While Eclipse does have ample documentation for portions of its ecosystem, it seemed as though we were developing within segments that had almost no documentation. In particular, documentation on implementing a search engine and configuration of the plugin itself were more difficult that they could have been due to the lack of documentation. A big part of the difficulty is the number of abstract functions that must be overridden, but often were only vaguely documented. The plugin attribute descriptions suffered the same vagueness. The books and examples found in CDT and JDT were the only way we could get a sufficient understanding.

We also ran into issues with the simple configuration of Photran to allow the VPG to process references between projects in the workspace. While this was a relatively easy task to accomplish, once the correct parameter was determined, it was less than obvious which parameter associated projects from the perspective of the VPG. To complicate matters, there were many parameters, mostly introduced by the CDT environment, that seemed as though they would configure the functionality needed for our testing. In the end we had to step through the VPG initialization code until we were able to find a comment for a class definition that gave us a hint as to the necessary configuration.

The configuration parameters we're referring to are in the Project Properties. In order to make the

PhotranVPG work, we had to set the module directories in the project Fortran General -> Analysis/Refactoring. This is a project specific property, but it is listed under General, and it is not stored in the project metadata. Without these settings, the workspace search cannot work. Even loading each file individually into the VPG lets the VPG find the dependent modules, but when applying the actual search the referenced modules cannot be found. This particular setting is not particularly easy to find, and isn't mentioned anywhere in the Photran Developers guide, nor the VPG doc on jeff.over.bz site.

That was the "correct" parameter. There were a variety of misleading and incorrect parameters as well. One incorrect parameter would be the Fortran General -> Paths and Symbols settings page. There are two tabs, Library Paths and Source location, which both seemed more likely to work with the VPG than anything that might be found in "Analysis/Refactoring".

Inherited from CDT is the project specific Project References checkbox list. Apparently this has no effect in Photran, and as such is dangerously misleading. Yet another option we tried was the linker parameters in the compiler options. We thought that perhaps indicating a module was to be linked with the source would indicate a dependency to Photran, but it didn't. At the very least we think the Project References page should be removed or utilized. It should, if possible, be utilized since it's the first place users will look and matches with the CDT and JDT models.

Initializing the PhotranVPG itself turned out to be a somewhat daunting task. Even after we discovered the correct settings to make the PhotranVPG recognize dependences between projects, we often times got unexpected and unexplained Null Reference errors and failures to find references. It turned out that PhotranVPG was being loaded piecewise as files were opened in the workspace to generate the outline on the right pane. This was fine for a file scoped search, but unacceptable for a project or workspace scoped search.

Our first attempt to fix this was to visit all the files in the workspace and load them individually into the VPG. Unfortunately this still did not solve the references problem. Eventually we found the function `EnsureVPGIsUpToDate` being used in Refactoring and this function did the work we needed and resolved dependences correctly. This is apparently still buggy however, and if you repeat a series of code modifications and searches the function will not correctly update the VPG and sometimes not all references will be found.

Even though the limited documentation was an annoyance, the platform specific issues encountered throughout the development cycle were far more significant, as quite a few hours were lost as the team tried to install and configure Photran and all of the required libraries on the two Windows development machine used by the group. Even after the installations were completed, we kept running into issues as there are certain assumptions that were made in regards to cross platform issues that were not the case. The most notable example lies within the configuration of module lookup paths for the VPG. The selector UI does what is presumably the correct action and uses the native platform's path separator character, but under Windows this causes a failure as these paths are paths within the workspace, which only accept "/" as a path separator. The net result was that workspaces running under Windows would attempt to lookup modules in "\\lib_project" but only "/lib_project" will correctly dereference to the actual project. This discrepancy accounted for four man-hours of debugging that certainly could have been used to better ends.

4. Helpful References

Throughout the project finding documentation and examples was paramount to successful completion. These are some of the resources we found useful.

"Contributing to Eclipse: Principles, Patterns, and Plug-Ins", by Erich Gamma and Kent Beck.

This book was especially helpful during our Spike and early parts of our project. Setting up a new plugin is a somewhat daunting task, and one error in the XML can throw the whole project off without an error message. This book included step-by-step instructions on creating a new plugin and described with

reasonable detail each XML element.

Chapters 5 and 6 of "Eclipse: Building Commercial Quality Plug-ins", Second Edition by Eric Clayberg and Dan Rubel.

The book itself is an amazing warehouse of information. Classes and Interfaces are described along with their purpose, which can be difficult to decipher otherwise. In particular we used the chapters on Views and Menus/Actions. UI is difficult enough to implement with a standard toolkit. Extending someone else's UI can be a nightmare. These chapters provided us valuable insights, and while it didn't always give us all the answers it gave us great direction.

Chapter Three of Photran Developer's Guide (Parsing and Program Analysis), and <http://jeff.over.bz/software/vpg/doc/>, both by Jeff Overbey.

In our particular project we were concerned with finding references in Fortran code. Luckily, the Virtual Program Graph (VPG) simplifies the problem by parsing code into an internal model that can be searched. These two references were a great starting point for using the VPG. Both are slightly out of date and far from a complete reference, but pointed us to classes and functions that paved the way for progress.

JDT, CDT, and Photran source code.

As mentioned in class, reading a book is great but it doesn't really mean anything until you do something with it. JDT and CDT are perfect examples of eclipse plug-ins development in practice. Widely used and actively developed, they've stood the test of time and their implementations represent tried and true coding methods. Photran's strength as a reference is its integration with CDT. Photran worked somewhat like an index into CDT functionality that eased our implementation of menu contributions and search view objects. The vast number of tests in Photran also gave us an opportunity to re-use the code to load source-files for our own automated tests.