

# Design Document CDT Remote Search/Index Framework

Chris Recoskie  
Vivian Kong  
Mike Kucera  
Jason Montojo

IBM Corporation

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to describe a design for a set of modifications to CDT that would introduce a framework for enabling ISVs to extend CDT with remote development capabilities. The modifications would be centered on making the search and index features of CDT extensible in such a way as to allow them to be used with resources that exist on a remote system in an efficient manner.

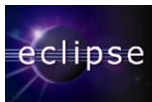
Motivation for this proposal is based on the following considerations:

- Users developing high performance computing applications targeting IBM POWER architecture on System p servers
- Systems are generally centralized at some remote location, and are often leased.
  - Latent network connectivity.
  - Large number of other simultaneous users on the same machine.
  - Bandwidth, CPU, and memory are metered and literally expensive.
- Running certain tools such as an Eclipse-based integrated development environment is sometimes not possible or is not feasible due to poor graphical user interface rendering performance.
- Parsing and indexing remotely are required for such users to make use of critical value adds of Eclipse/CDT such as content assist, navigation, search, etc., because the files do not all live locally.

The first part of this proposal involves decoupling the current parser and index subsystems from CDT so that they may be built as standalone JAR files. Once these JARs are available it will be possible to deploy the parser and indexer in an environment where eclipse/CDT is not available (such as on a remote system).

There are several uses for standalone jars:

- Deploy the parser and indexer remotely. This would be useful for tools that need to parse files remotely but currently do not have the luxury of having a CDT project in the workspace (such as TPF Toolkit).
- Provide a library for C/C++ parsing/indexing that would be usable by anyone interested in having this functionality but who does not need the Eclipse and CDT frameworks.



- Provide a tool that could be used to generate offline indexes for SDKs. This would be useful for the current offline indexing proposal. SDK vendors could provide CDT indexes by using a simple command line tool.

## 1.2 Scope

The scope of this document is to describe the design for a framework that would allow remote searching and indexing capabilities to be added to CDT. This document does not address the mechanism by which remote files would be accessed. This would be the responsibility of an ISV that would develop their remote development solution on top of the framework that is being proposed in this document. This design does not address remote build and remote debug capabilities.

## 1.3 Related Documents

There is a bugzilla created for this proposal:

[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=158975](https://bugs.eclipse.org/bugs/show_bug.cgi?id=158975)

This design is closely related to the project model subsystem within CDT.

[http://download.eclipse.org/tools/cdt/docs/summit2006/New\\_Project\\_Model\\_UI\\_06.09.25.ppt](http://download.eclipse.org/tools/cdt/docs/summit2006/New_Project_Model_UI_06.09.25.ppt)

In addition there are several Bugzilla entries related to the indexer and parser components of CDT that are closely related to this design.

[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=151846](https://bugs.eclipse.org/bugs/show_bug.cgi?id=151846)

[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=151847](https://bugs.eclipse.org/bugs/show_bug.cgi?id=151847)

[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=158975](https://bugs.eclipse.org/bugs/show_bug.cgi?id=158975)

PDOM paging scheme: [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=165451](https://bugs.eclipse.org/bugs/show_bug.cgi?id=165451)

## 1.4 References

Links are provided to supporting reference material.

EFS

<http://wiki.eclipse.org/index.php/EFS>

RSE (Remote System Explorer)

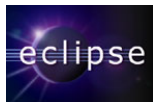
[http://www-304.ibm.com/jct09002c/isv/rational/remote\\_system\\_explorer.html](http://www-304.ibm.com/jct09002c/isv/rational/remote_system_explorer.html)

DAO Design Pattern

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

## 1.5 Stakeholders

A list of primary stakeholders, in no particular order:



- Beth Tibbits, IBM (PTP Project Committer)
  - Support for remote system development targeting POWER platform.
- Ankit Pasricha, IBM
  - CDT integration in TPF toolkit.
  - Parser and Indexer JAR files.
- Craig Rasmussen, Los Alamos National Laboratory (Photran & PTP Projects Committer)

## 1.6 Points of Contact

- Primary contact: Chris Recoskie, IBM Corporation, [recoskie@ca.ibm.com](mailto:recoskie@ca.ibm.com)
- Primary contact: Jason Montojo, IBM Corporation, [jmontojo@ca.ibm.com](mailto:jmontojo@ca.ibm.com)
- Primary contact: Mike Kucera, IBM Corporation, [mkucera@ca.ibm.com](mailto:mkucera@ca.ibm.com)
- Primary contact: Vivian Kong, IBM Corporation, [vivkong@ca.ibm.com](mailto:vivkong@ca.ibm.com)

## 1.7 Glossary

Term	Definition
CDT	The C Development Tools project hosted on Eclipse.org
ISV	Independent Software Vendor. Used as a general term to describe software developers which customize CDT, as opposed to someone who uses CDT as an end-user.
User	An end-user to CDT.
EFS	Eclipse File System. A virtual file system API used by the Eclipse platform. It abstracts away implementation details of how files in the workspace are stored.
JAR	Java Archive file. A ZIP file used to package and distribute a set of java classes. Stores compiled Java classes and associated metadata that can constitute a program.
Remote plug-in	A plug-in or set of plug-ins that would enable remote development using CDT. The design proposed in this document would enable a remote plug-in to be built on top of CDT.
Index	A data structure that can be stored on disk that provides fast lookup of identifiers within the source code of a program.
Parser	A program that translates a text file consisting of source code (for example C or C++) into a data structure (AST).
AST	Abstract Syntax Tree. A data structure that represents the structure of a source code file.
RSE	Remote System Explorer. An Eclipse plug-in that allows users to create connections to remote servers and explore their file systems.

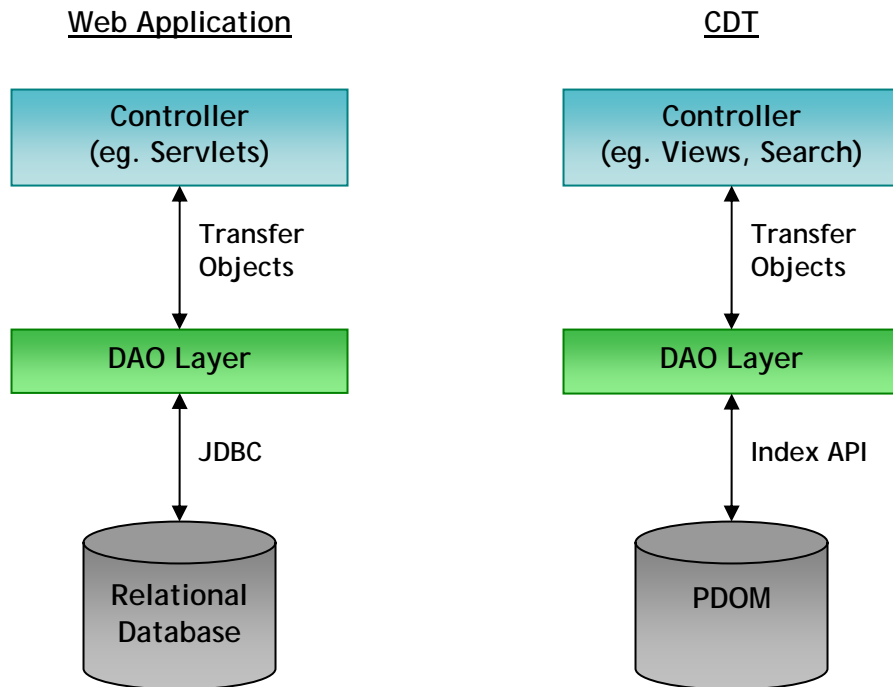


## 2 Design Overview

The central idea of the design involves significantly loosening the coupling between the index APIs and the UI controller code that uses them. This must be done in order to achieve the level of modularity that the design requires.

There exists an enterprise design pattern for this purpose, the Data Access Objects (DAO) design pattern. This pattern is commonly used in Java web applications to create an abstraction layer that encapsulates the implementation of persistent storage (such as a relational database). The front end of the application communicates with the DAO layer via transfer objects, which are value objects that contain the results of queries or contain data that is to be persisted.

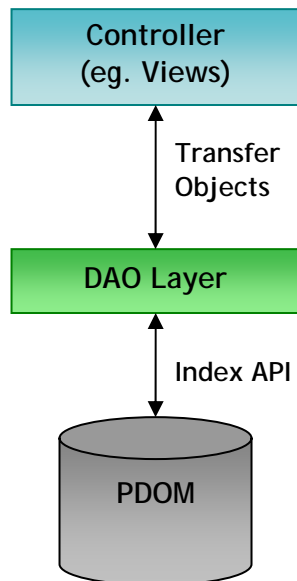
This pattern involves defining a clear separation between the code that accesses the index and the code that displays the results. The data access layer can be easily swapped out and replaced with another one without affecting the rest of the application. This pattern is most often used in the web development world when working with an architecture that involves a controller framework on the front end and a relational database on the back end. The parallel between this setup and the architecture of CDT is obvious.



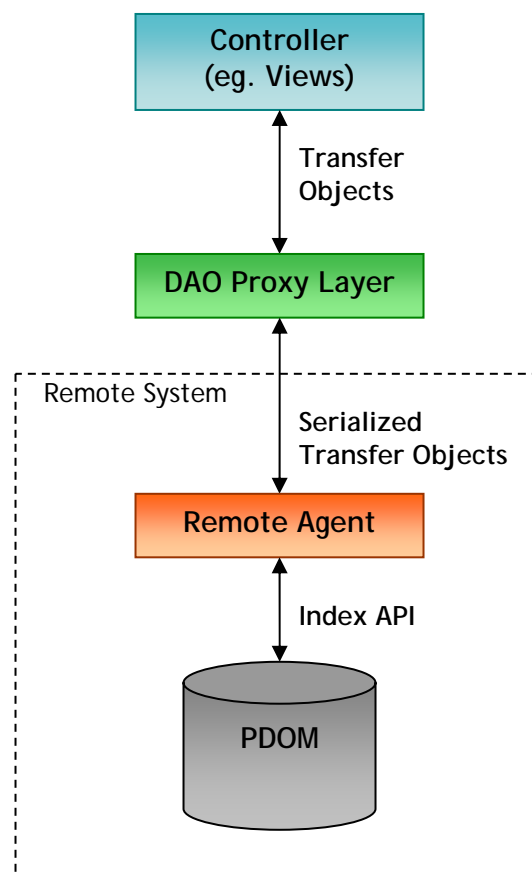
The proposed design includes the following:

- Migrating the CDT codebase such that all resource management is implemented on top of EFS. This will allow any resource to transparently exist on a remote system. Normal file operations, such as opening a file in the editor, would work as expected even for remote resources.
- Applying the DAO pattern to provide a layer that separates index related operations from clients that need the results of those operations. This will allow a 3<sup>rd</sup> party plug-in to enable remote index features on top of CDT by providing its own implementation of the DAO layer. Presumably this implementation will provide a DAO layer that is actually a proxy that communicates with the remote machine.

CDT Local Scenario



CDT Remote Scenario



- An extension point for adding custom implementations of the DAO layer.
- A default implementation of the DAO layer based on the PDOM.
- A manager for the DAO framework that would handle reading of the extension point and mapping of a particular DAO implementation to a particular resource. This would allow more than one DAO implementation

to coexist, allowing local resources to be indexed in a local PDOM and remote resources to be indexed on the remote system.

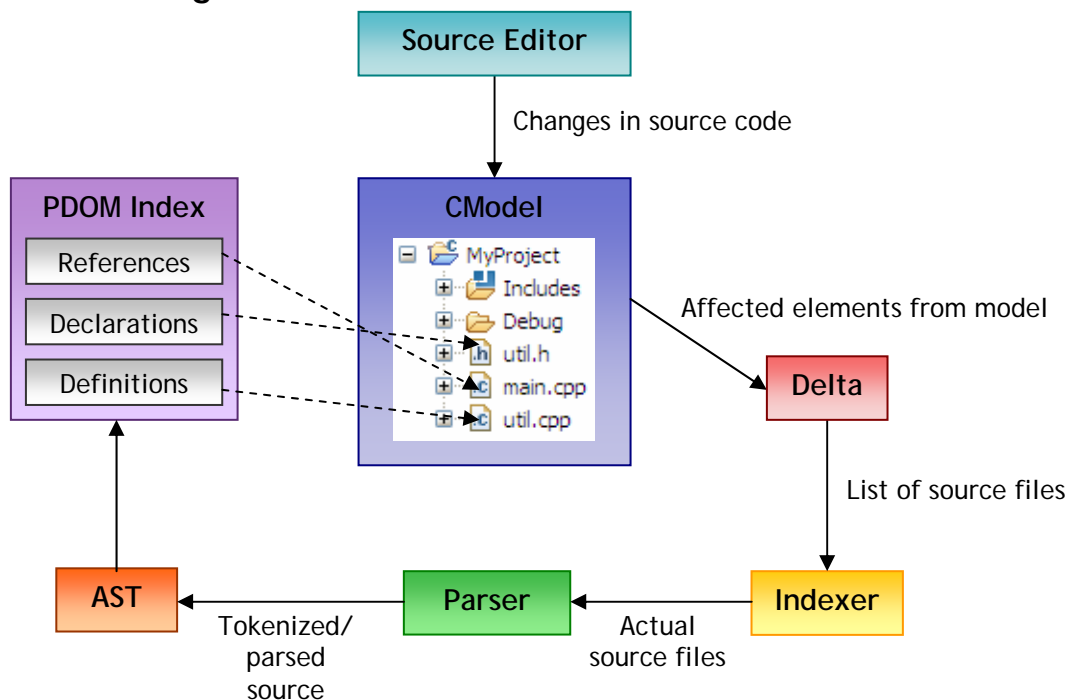
- Decoupling the current CDT parser and index subsystems to allow deployment as standalone components outside of CDT. This would allow a remote index agent to exist on a remote machine without Eclipse/CDT present.
- The PDOM based DAO layer could also be made part of a standalone JAR so that it may be reused on the remote machine by the remote agent.

### 2.1.1 Additional benefits of the DAO Design Pattern:

- Loose coupling between UI Controller code and the PDOM subsystem leads to improved maintainability.
- The DAO layer provides a well defined interface that can be documented.
- Much easier to replace the PDOM with a new Indexing technology in the future if it is ever needed.
- Makes unit testing UI Controller code easier as DAO objects can be easily simulated by mock objects.
- Makes testing index functionality easier via a test suite for the CModel layer. ISVs could reuse this test suite when implementing their own DAO layer.

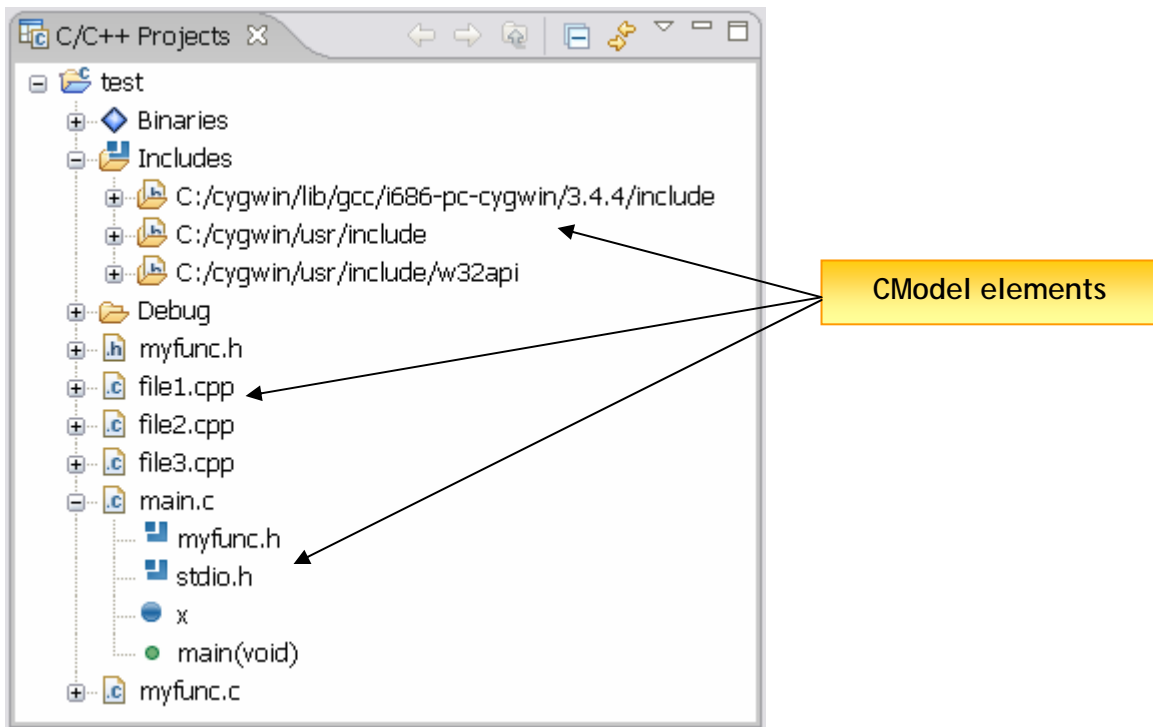
## 2.2 Background

### 2.2.1 Indexing and the PDOM



CDT uses a hierarchical model called the CModel to represent resources within CDT projects. This model can be viewed by users via the C/C++ Projects View.

The process of indexing source code in CDT happens in multiple phases. When a user makes changes to their source code, the changes are analyzed with respect to the core project model (CModel). Each element in the model is an ICElement. The corresponding ICElements for projects, folders and source files are ICProjects, ICContainers, and ITranslationUnits, respectively. The ICElements that are affected by the user's changes are collected and stored in a delta.



The indexer analyzes the delta to determine which files need to be indexed. The contents of these files are then passed to the parser. The parser extracts structural information from the source code and produces Abstract Syntax Trees (AST). Finally, the indexer extracts semantic information from the ASTs, such as class and variable references, declarations, and definitions, and stores that in the PDOM index.

### 2.2.2 Project Natures

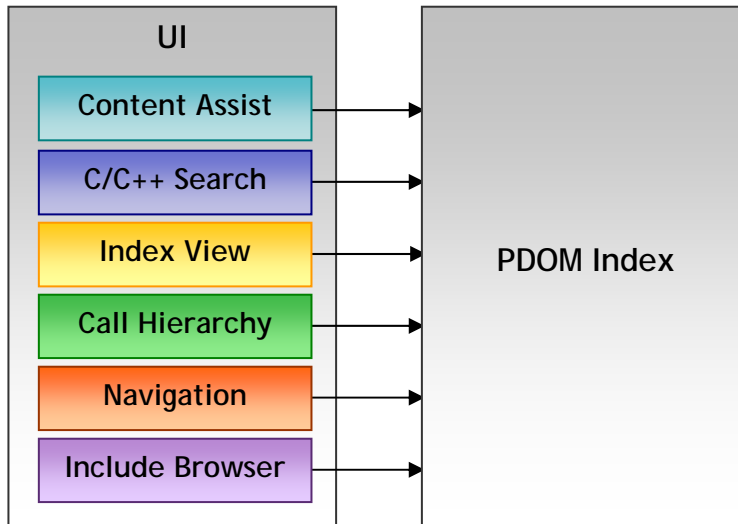
CDT provides project natures for C and C++. When the CModel is built all the projects in the workspace are scanned. Any project with the C or C++ nature is then added to the CModel (as a CProject element).





## 2.3 Current Behaviour

All search and index functionality is currently designed to be highly dependant on the PDOM subsystem. All of the UI controller code has been written to use the low level PDOM Index APIs directly. This means that the index functionality cannot be treated in a modular way and cannot be swapped out.



**Existing Usage Model**

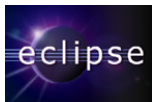
Currently there are two extension points that are directly related to the new design:

- Indexers are contributed via an extension point.
- Content assist is provided by completion contributors that are defined in an extension point.

## 2.4 Proposed Behaviour

We envision a scenario where remote development capabilities are built on top of CDT by a separate plug-in or set of plug-ins. CDT would provide the framework that would allow this to be possible. This set of 3<sup>rd</sup> party plug-ins will be henceforth referred to as the “remote plug-in” for the remainder of this document.

The new EFS API will enable remote resources to exist in the Eclipse workspaces alongside local resources. In order to take advantage of this CDT will need to be migrated to EFS. This has the additional benefit of allowing CDT to be



used with other EFS supported file systems, for example a Zip file could be mounted into the workspace using an EFS Zip file implementation.

Simply providing remote resources via EFS is not sufficient to provide the level of performance of remote development that may be required. The reason is that file operations on EFS remote files may cause the file to be transferred over the wire to a local cache folder. This is not desirable for performance reasons, for example indexing all the files in a project could cause all the files to be transferred locally.

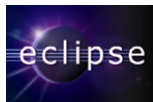
It is the responsibility of the remote plug-in to provide an implementation of EFS for accessing the remote files. The remote plug-in will probably also need to provide its own workspace projects and manage the resources in those projects. The C or C++ nature would then be applied to these projects allowing CDT to recognize them and add them to the CModel. This in turn enables all of the CDT functionality to work on the remote resources (by pretending they are local resources in the workspace). If the remote plug-in is not concerned with efficiency then this is all that it needs to provide. However, if the remote plug-in wants to provide a remote index as well, then it can participate in our proposed DAO service framework.

(There is an alternate usage scenario that does not involve the necessity to provide an EFS implementation. It will be possible for an ISV to use CDT components directly without actually using CDT as a framework. In this case an implementation of EFS may not be required.)

In order to enable remote search and index functionality in an efficient manner it would be necessary to have an agent on the remote system that provides an index for the files located there. In order to facilitate this we propose to provide stand-alone JAR files that contain the CDT parser and indexer subsystems. The goal is to make it possible to deploy the parser and indexer on a remote system where the Eclipse/CDT framework may not be available.

There will be an extension point for contributing implementations of our proposed DAO service framework. This implementation will provide implementations of search and index functionality that will communicate with the agent on the remote system.

There will be a DAO Manager component that will be responsible for mapping a workspace file to a DAO service. This mechanism will allow more than one DAO implementation to coexist (enabling local and remote projects to live side by side). If more than one project type is involved, for example if a local and a remote project are included in a search scope, then CDT must make use of each mapped DAO factory and combine the results.



A remote indexer would be contributed using the current extension point for indexers. The remote plug-in would then manually associate its indexer with its remote project using `IIndexManager.setIndexerId(ICProject project, String indexerId)`. This registers the remote indexer to receive deltas.

## 2.5 Transition Plan

The CDT codebase will need to be transitioned to use the EFS APIs exclusively for resource management.

The current PDOM based search/index functionality will be factored out into an implementation of the DAO service framework and then contributed via the new extension point.

## 2.6 Risks

- Re-architecting CDT to facilitate remote development may impact the performance of CDT for local development. We must ensure this change does not slow down CDT in any way.
- Since the majority of CDT users do not require remote development capabilities, the community may resist changing the existing architecture.
- It is possible that future clients of the index system will require full access to the index. This is an issue for remote development because the entire index may potentially be transferred to the local side.

## 2.7 Assumptions

- The CDT parser and indexer subsystems can be decoupled from CDT and packaged as standalone components.
- Clients of the index service extension point are responsible for providing the necessary bridging to connect their remote services with CDT. The CDT parser and indexer will be provided as reusable components for the remote systems.
- The remote system provides its own Java 1.4-compliant runtime environment.

## 2.8 Dependencies

- The proposed design introduces a dependency on EFS.

## 3 Requirements and Usage

### 3.1 Problem Statement

- Allow ISVs to implement remote development on top of CDT without significantly changing the user experience.

### 3.2 User Characteristics

Users of this system fall into two main categories: end users and ISVs.

#### 3.2.1 ISVs

For the purpose of this document an Independent Software Vendor is any software development firm that would be interested in extending CDT by adding remote development capabilities. ISVs are interested in the framework and public API that would be used to develop a remote plug-in.

#### 3.2.2 End Users

End users are users which are using CDT in order to work on some project. End users may also be using the remote plug-in to interact with C/C++ files that exist on a remote system. The user experience should not be significantly different between using a local project and a remote project. For example the C/C++ search dialog should be the same, and other actions such as find references should be triggered in the same way. End users expect a consistent user experience.

The goal of a consistent experience is ultimately a lofty one. It is likely that the remote plug-in will have to provide its own interface for interacting with the remote system. Furthermore, when working with remote resources there are problems with slow or unreliable connections. These will have to be dealt with.

### 3.3 Use Cases

The following are a set of use cases which define typical usage of the system from the end-user perspective. Since the use cases are relatively simple and straightforward, a brief prose description for each will be used rather than a detailed description involving primary actors, success scenarios, etc.

Ultimately it is the responsibility of the designers of the remote plug-in to determine the user experience of their software. The user interface and usage scenarios of CDT will not be significantly changed by this design. Usage scenarios will still be described here for reference.

### **3.3.1 C/C++ Search**

- 3.3.1.1 The user invokes the C/C++ search dialog via the Search menu or the toolbar.
- 3.3.1.2 The user enters search criteria into the dialog including the search string. Checkboxes are used to select the type of C/C++ language element to search for.
- 3.3.1.3 The user chooses the scope of the search by choosing a set of projects, then clicks the search button.
- 3.3.1.4 The system passes the scope to the IndexDAOManager which returns a search DAO which is used to perform the search.
- 3.3.1.5 If a remote project is included in the scope then the remote index agent will be contacted and the search will be performed on the remote system. The search results will be sent back from the remote system.
- 3.3.1.6 Search results are collected and displayed to the user in the C/C++ Search Results view. If there is a combination of local and remote search results they will be combined and displayed together in the view.

### **3.3.2 Find References/Declarations**

- 3.3.2.1 The user selects a one of the following: 1) An identifier in the source code editor, 2) an element in the outline view, 3) an element in the C/C++ projects view.
- 3.3.2.2 The user invokes the pop-up menu and chooses to search for references or declarations.
- 3.3.2.3 The system finds the project that contains the selected element and uses it as the search scope.
- 3.3.2.4 Go to 3.3.1.4.

### 3.3.3 Content Assist

- 3.3.3.1 The user types some source code into the source code editor. The user invokes content assist by clicking Ctrl-Space.
- 3.3.3.2 The system determines the project that contains the source file
- 3.3.3.3 The system obtains a Content Assist DAO from the IndexDAOManager . A Content Assist DAO is treated the same way as a Completion Contributor. All of the available completion contributors are invoked to provide the content assist.
- 3.3.3.4 If the project is a remote project then the remote agent will be contacted and the completions will be calculated on the remote system. A set of completions will then be sent back. These completions will be combined with any completions that were calculated locally (for example keyword or template completions) and displayed to the user.

### 3.3.4 Index View

- 3.3.4.1 The user chooses to display the C/C++ index view using the Window menu.
- 3.3.4.2 The index view displays a set of projects as its topmost elements. Remote projects will be displayed but will not be expanded by default.
- 3.3.4.3 The user chooses to expand a remote project.
- 3.3.4.4 The system retrieves the Index Structure DAO that is mapped to the project that is to be expanded. This will contact the remote agent and be used to display the content under the remote project in the index view.

### 3.3.5 Navigation - Open Declaration/Definition

- 3.3.5.1 The user selects a one of the following: 1) An identifier in the source code editor, 2) an element in the outline view, 3) an element in the C/C++ projects view
- 3.3.5.2 The user invokes the pop-up menu and chooses to open the declaration or the definition.
- 3.3.5.3 The system finds the project that contains the selected element and retrieves the corresponding Navigation DAO which is used to find the source code location of the definition or declaration.

- 3.3.5.4 If the selected file is under a remote project then the remote agent will be contacted and the source file and code location of the declaration/definition will be sent back.
- 3.3.5.5 The appropriate file is opened in the editor and the location of the declaration/definition is selected. If the file was located on a remote system then a copy is transferred locally so that it may be displayed in the editor.

### **3.3.6 Call Hierarchy**

- 3.3.6.1 The user selects a function or method in one of the following ways 1) as an identifier in the source code editor, 2) an element in the outline view, 3) an element in the C/C++ Projects View.
- 3.3.6.2 The user right clicks and chooses to open the call hierarchy view from the pop-up menu.
- 3.3.6.3 The system finds the project that contains the selected function/method and retrieves the corresponding Call Hierarchy DAO which is used to populate the Call Hierarchy view.
- 3.3.6.4 If the file is part of a remote project then the remote agent will be contacted and the call hierarchy will be computed on the remote system.

### **3.3.7 Include Browser**

- 3.3.7.1 The user opens the include browser view using the Window menu.
- 3.3.7.2 The user drags a source code file on to the Include Browser View
- 3.3.7.3 The system determines the project that contains the selected file. The corresponding Include Browser DAO, this is used to populate the view.
- 3.3.7.4 If the file is part of a remote project then the include dependencies will be computed by the remote agent and sent back.

### 3.3.8 Create New Project

- 3.3.8.1 The remote plug-in will have to provide its own new project wizard. Presumably this will create a remote enabled project in the workspace and apply a C nature to it. The resources in the remote project will need to be implemented on top of EFS
- 3.3.8.2 Any time the user chooses to perform an index or search related operation the DAO factory that is associated with the new project will be used by the IndexDAOManager.

### 3.3.9 Type Hierarchy

- 3.3.9.1 The type hierarchy view is not implemented yet.

## 3.4 Global Decisions and Tradeoffs

This design comes with the consequence that any search/index action may now be triggering communication with a remote system. Thus the problems of network communications become a concern in many places in CDT. Since the remote plug-in is contributing implementations of all the search and index features, and since CDT has no knowledge of the remote system or the communications protocol used, it is decided that all concerns of network communication errors be the responsibility of the remote plug-in. This means that the remote plug-in must be sufficiently robust as to not cause CDT to hang or crash.

Since there may be bandwidth limitations this design will take into consideration that the information transmitted between the local system and the remote system be kept as small as possible. Entire files should not be transmitted when it is not necessary. A search should result in only the results being sent back from the remote system and not entire files. Indexing should take place on the remote system, indexing of existing remote files should not require them to be transferred to the client machine. These requirements have been taken into consideration with this design.

It has been identified that the on-disk index can be up to three times larger than the corresponding source code. For this reason it is desirable to avoid transmitting index nodes over the wire when not necessary. All index access will be encapsulated within the various Providers that are part of the IIndexDAOServiceFactory interface.



## 3.5 Functional Requirements

The new framework will provide the following functionality to the developers of the remote plug-in.

- The ability to provide an implementation `IIndexDAOFactory` through an extension point.
- The ability to specify a mappings between a DAO Factory and a resource. This mapping would be maintained by the DAO Factory itself via a method that takes an object representing a resource and returns true only if the factory applies to that object. A default DAO factory will apply if none of the other factories applies.
- In the case that more than one `IIndexDAOFactory` is involved in a search query, CDT will be responsible for combining the results into one consistent search results view.
- The ability to programmatically associate an indexer with a project. This already exists in `IIndexManager.setIndexerId()`. This is a concern because we are not changing the extension point for providing indexers.
- CDT will provide deltas of file changes that will be passed to the remote indexer.

## 3.6 Performance Requirements

Since there may be memory restrictions on the remote system the index will implement a memory paging algorithm. This will allow the amount of memory that is used by the index to be tuned.

## 3.7 Security Requirements

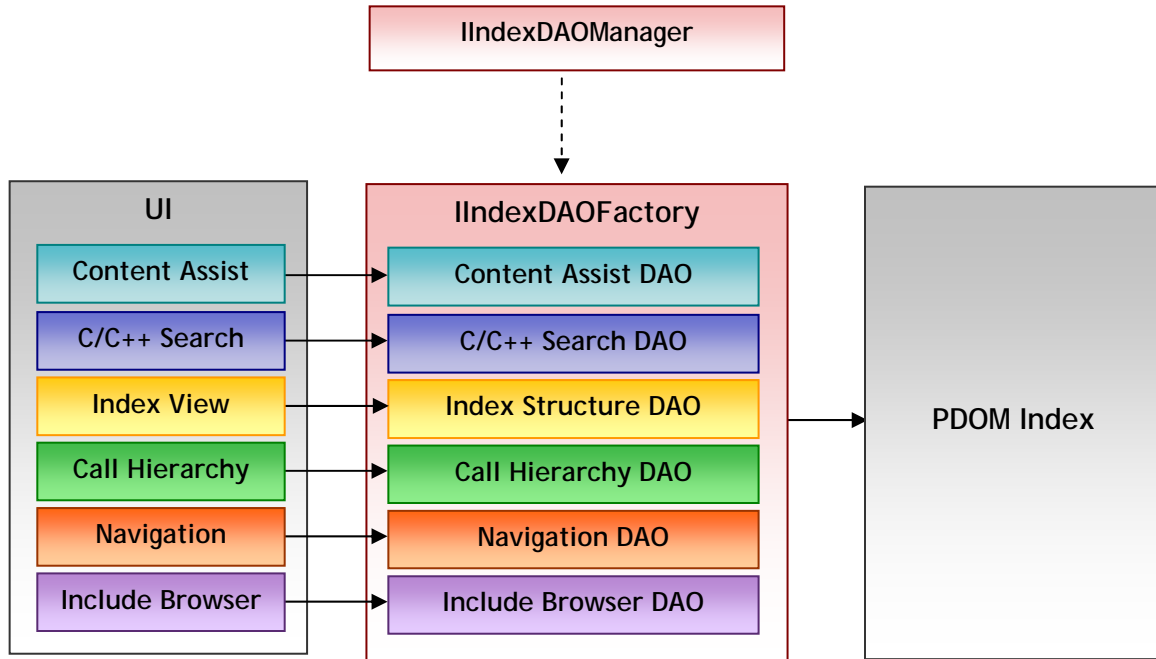
Security requirements are the responsibility of the remote plug-in.

# 4 High Level Design

## 4.1 Architecture Overview

The `IIndexDAOFactory` is designed to decouple knowledge of the index implementation from the UI layer. Instead of querying the index directly, the UI uses DAO objects that it obtains from the `IndexDAOManager` to provide content

that it will ultimately be displayed to the user. Each DAO provided by an IIndexDAOFactory corresponds to a UI component.



**Proposed Usage Model**

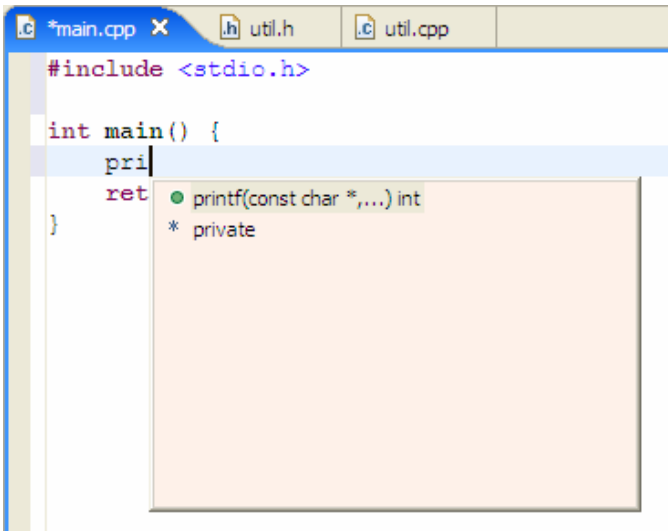
## 4.2 Component Listing

### 4.2.1 IIndexDAOFactory

The IIndexDAOFactory provides access to a number of index-related services. Each service is specific to a user operation and is handled by one of the Data Access Objects listed below. DAO objects are stateless.

### 4.2.2 Content Assist DAO

The Content Assist DAO supplies a list of possible completions for the current cursor position within a source editor. The completions may be type names, variables, functions, keywords, or even freeform text.



```

#include <stdio.h>

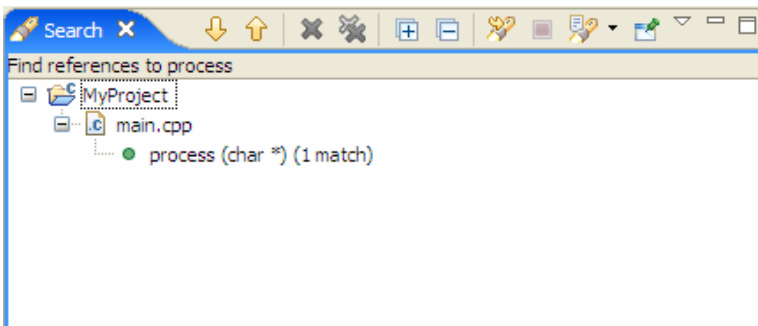
int main() {
    pri
    ret
}

```

• printf(const char \*,...) int  
 \* private

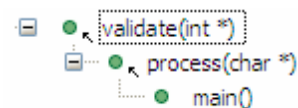
### 4.2.3 C/C++ Search DAO

The C/C++ Search DAO performs syntax- and semantic-sensitive searches on source code. The provider supplies a structured set of search results to the UI. Each search result contains the location of the match so the UI can display and highlight it.



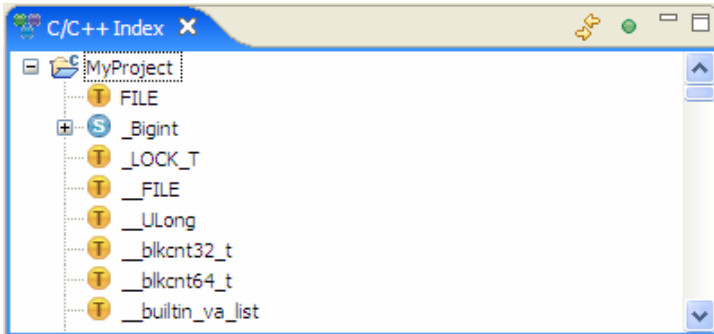
### 4.2.4 Call Hierarchy DAO

The Call Hierarchy DAO finds and returns all the callers (functions which call it) or callees (functions which it calls) of a given function or method. Multiple calls to the provider can produce a complete call tree.



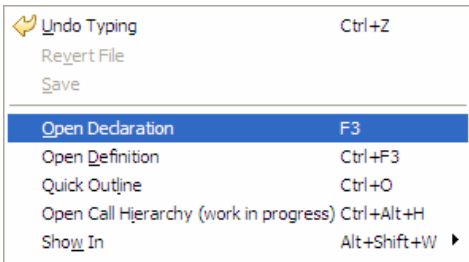
## 4.2.5 Index Structure DAO

The Index Structure DAO gives its clients access to an ICElement hierarchy representing the contents of the index. This provider may return the entire hierarchy, or a proxy which incrementally fetches portions of the hierarchy on demand as it is traversed.



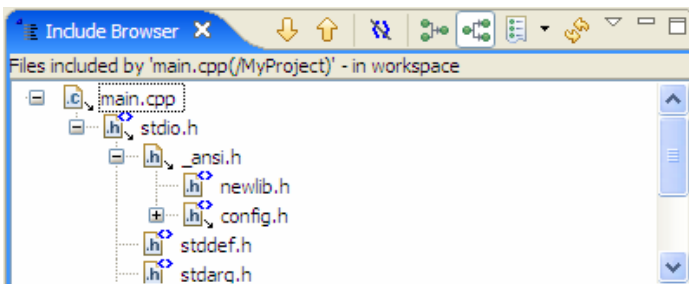
## 4.2.6 Navigation DAO

The Navigation DAO performs lookups for the declaration or definition of a named type or variable. The provider returns the location of the named element so the UI can display and highlight it to the user.



## 4.2.7 Include Browser DAO

The Include Browser DAO provides its clients with a list of files included by a particular source file. By making multiple calls to the provider, a client can construct complete include trees for source files.



## 4.2.8 IndexDAOManager

The IndexDAOManager provides the following:

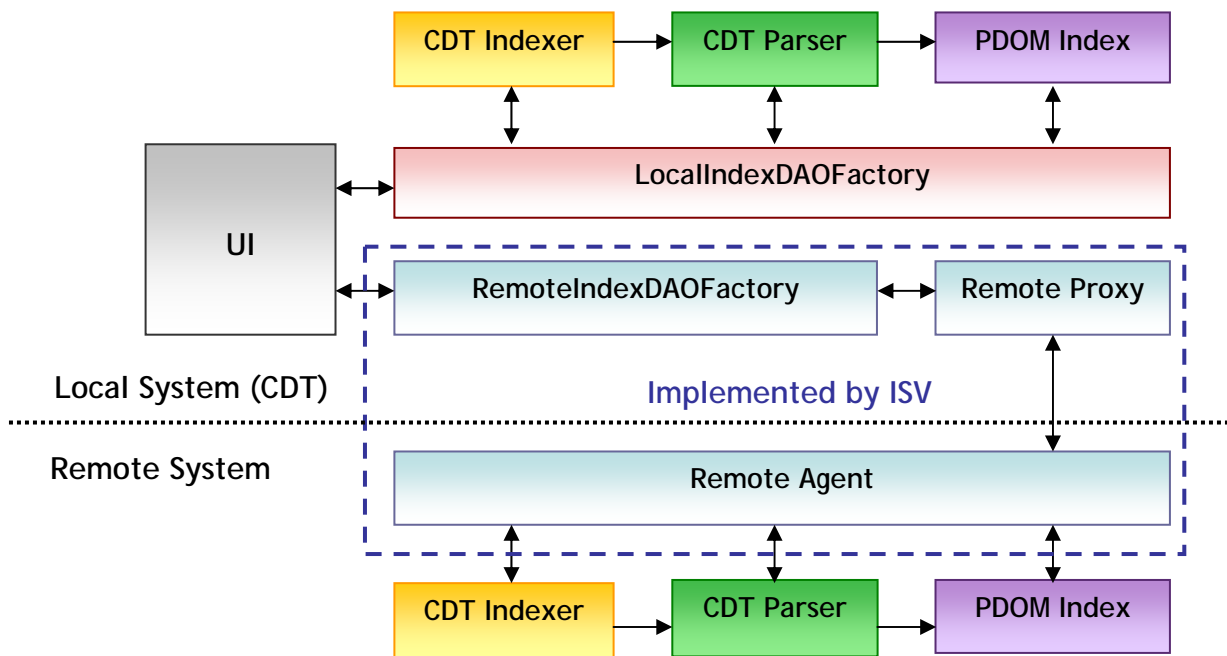
- Reads the extension point registry to discover implementations of IIndexDAOFactory.
- Maintains a list of contributed DAO Factories. There will be one DAO factory that is considered the default (presumably this will be the PDOM based implementation).
- Provides methods for getting DAOs, for example getSearchDAO(). The Manager will use its list of DAO factories to create the DAOs. Composite DAOs may be built in the case where more than one factory applies to a given set of resources.
- Determines when a DAO Factory applies to a given resource.

## 4.3 Data Structures

Not applicable.

## 4.4 Example Usage

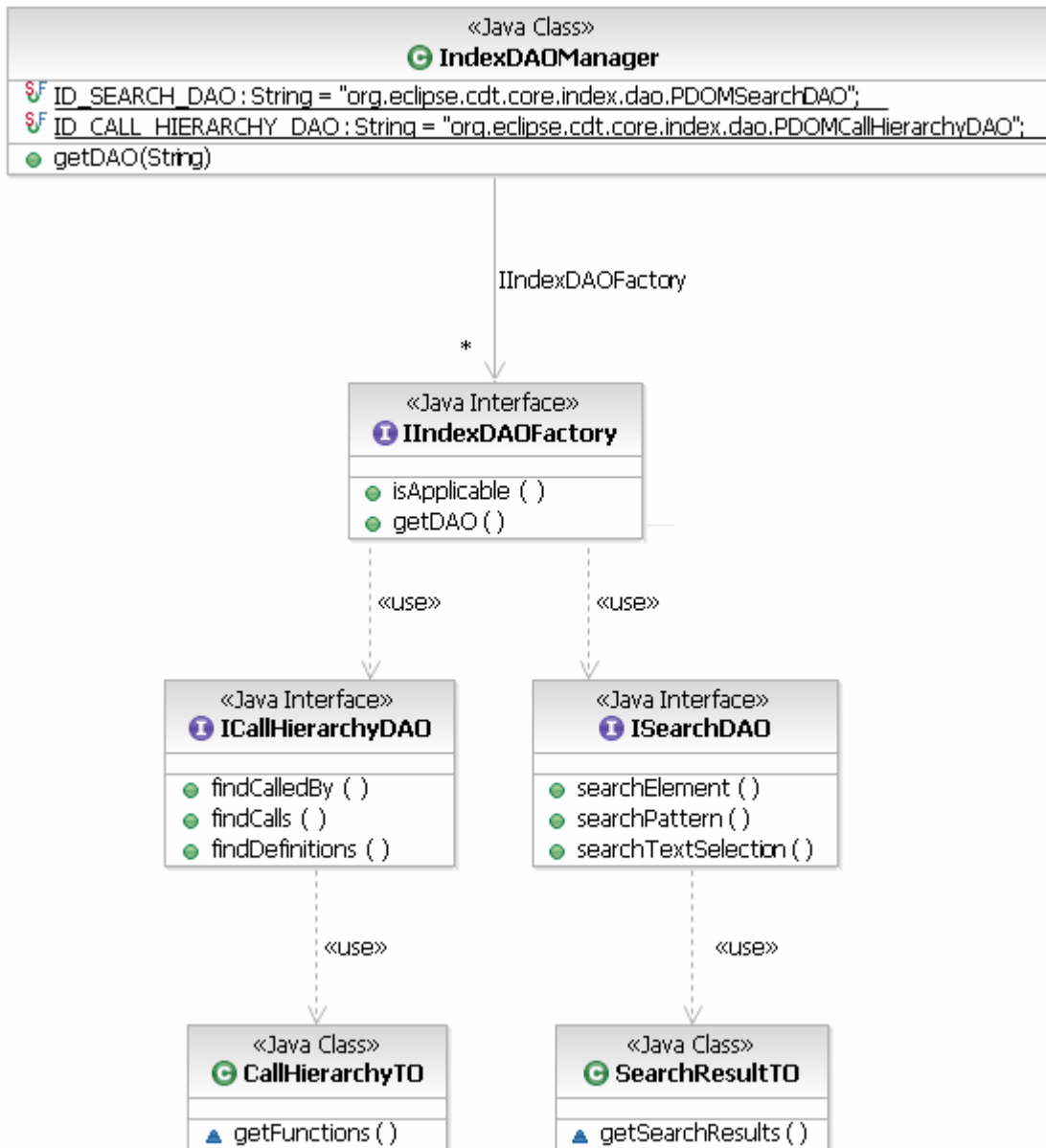
The following diagram illustrates how an ISV can architect a solution for enabling CDT development on a remote system:



## 5 Detailed Design

### 5.1 Component Interrelations

The following diagram displays a subset of the components that are involved in the design. The IndexDAOManager is used by client code to gain access to DAO objects. The IndexDAOManager uses the IIndexDAOFactory to create the DAO objects. Each DAO has associated transfer objects for returning results. Only the Call Hierarchy and Search related DAOs are given in the diagram, other DAOs will work in a similar fashion.



The above diagram is not complete, it is meant to show a high level idea of how the DAO API will work.

The various get methods of `IIndexDAOServiceFactory` are allowed to return null, this way a remote plug-in may selectively implement a subset of the potential functionality.

## 6 Interface Design

### 6.1 API Evolution

The public API has been designed so that it may be easily evolved in the future without breaking clients. For this reason the `IndexDAOManager` class and the `IIndexDAOFactory` interface both have a single method for gaining access to all types of DAO object:

```
Object getDAO(String daoID)
```

`IndexDAOManager` will contain static final fields for the default set of PDOM DAOs. Example usage would look like:

```
ISearchDAO dao = (ISearchDAO)  
    IndexDAOManager.getDAO(IndexDAOManager.ID_SEARCH_DAO);
```

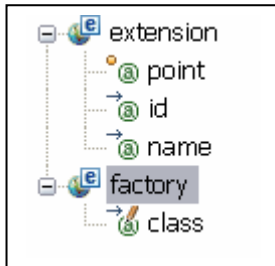
Adding new DAO interfaces in the future will not require any changes to the public method interface of `IndexDAOManager` or `IIndexDAOFactory`. To add a new DAO interface it would be necessary to add a new static final field for the DAO's ID.

In order to add specific index-related functionality it suffices to add a new DAO interface. For example, if the call-hierarchy view needs new functionality from the Call Hierarchy DAO this would be done by adding a whole new DAO interface with the required method instead of begetting a new subinterface of `ICallHierarchDAO`. This would not affect existing clients as they are not required to implement the new interface.

### 6.2 Extension Points

The design impacts upon three extension points:

- 1) A new extension point for contributing implementations of `IIndexDAOServiceFactory` will be added. Below is the proposed extension schema.



```
<!ELEMENT extension EMPTY>
<!ATTLIST extension
  point CDATA #REQUIRED
  id    CDATA #IMPLIED
  name  CDATA #IMPLIED>
```

```
<!ELEMENT factory EMPTY>
<!ATTLIST factory
  class CDATA #IMPLIED>
```

- **class** - Must implement `IIndexDAOFactory`.

- 2) There currently exists an extension point for contributing indexers. This would simply be reused (without changes) for contributing a remote enabled indexer. The remote plug-in would be responsible for mapping the appropriate remote indexer to a remote project. There currently exists a way to do this programmatically through `IIndexManager`. In order to facilitate this it is proposed that `setIndexerId()` method be included in the `IIndexManager` interface.
- 3) There currently exists an extension point for content assist. This extension point will remain untouched because it is used to provide completion contributors that apply to any type of project, such as the keyword completion contributor. It will also be possible to contribute completion contributors via new the new `IIndexDAOFactory` extension point. It may be necessary to make all completion contributors return the same kind of transfer object regardless of which extension point is used to provide the completion contributor.

## 6.3 Public API Listing

This section contains a summary JavaDoc of a subset of the proposed API.

## 6.4 Class `IndexDAOManager`

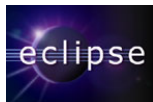
`java.lang.Object`

└ **`IndexDAOManager`**

---

```
public class IndexDAOManager
  extends java.lang.Object
```





Singleton used by client code to gain access to DAO objects, will return the appropriate DAO based on the given scope.

Field Summary	
static java.lang.String	<a href="#">ID_CALL_HIERARCHY_DAO</a>
static java.lang.String	<a href="#">ID_SEARCH_DAO</a>

Constructor Summary	
	<a href="#">IndexDAOManager()</a>

Method Summary	
java.lang.Object	<p><a href="#">getDAO</a>(java.lang.String id) Returns a DAO for the given DAO ID, or null if no DAO Factory is applicable to the given ID.</p> <p>Example Usage: ISearchDAO dao = (ISearchDAO) IndexDAOManager.getDAO(IndexDAOManager.ID_SEARCH_DAO);</p>

## 6.5 Interface IIndexDAOFactory

```
public interface IIndexDAOFactory
```

A factory for creating DAO objects.

Method Summary	
java.lang.Object	<p><a href="#">getDAO</a>(java.lang.String id) Returns a DAO for the given DAO ID, or null if no DAO Factory is applicable to the given ID.</p>
boolean	<p><a href="#">isApplicable</a>(java.lang.Object element) Returns true only if the given scope element should be handled by this DAO factory.</p>



## 7 Revision History

Revision Number	Date	Authors	Notes
1.0	Dec 7, 2006	Chris Recoskie Jason Montojo Vivian Kong Michael Kucera	Initial design.
2.0	Dec 14, 2006	Michael Kucera	Changed to include DAO pattern terminology. Removed dependency on ICElement from design. Added new diagrams.
2.1	Dec 20, 2006	Michael Kucera	More generic getDAO() methods. Added section on API evolution.